

A Hierarchical Task Network Planner for Pathfinding in Real-Time Strategy Games

Munir Naveed¹, Diane Kitchen¹ and Andrew Crampton¹

Abstract. In this paper, we propose an automatic mechanism of Hierarchical Task Networks (HTNs) creation for solving the problem of real-time path planning in Real-Time Strategy (RTS) Games. HTNs are created using an abstraction of the game map. A real-time heuristic search approach called Learning Real-Time A* (LRTA) is applied to execute the primitive tasks of the HTNs. The main purpose of using a HTN based real-time path planner is to restrict the real-time search to a shorter part of the problem space while keeping it in the direction of the actual goal position. Results show that the hierarchical approach reduces the suboptimality of the LRTA and speeds up the convergence process.

1 INTRODUCTION

Real-time heuristic search is also known as agent-centered search [1]. This kind of search method interleaves plan making and plan execution. Another notable characteristic of this group of search algorithms is that it keeps the planning process within the neighbourhood of the current state. This means that a real-time heuristic search method looks at the part of the domain world that is directly relevant (i.e. successor states) in its present situation, determines what actions to execute within this part and then executes them. This episode of action selection and execution is done in a fixed time interval that is independent of the size of the domain world. This process continues until the goal state is reached.

HTNs are not built with the objective of achieving a set of goals but to carry out some set of tasks [2]. These tasks can be either primitive or nonprimitive. A nonprimitive task is the abstract task which can be decomposed into sub-tasks (either primitive or non primitive). Primitive tasks are the ground tasks which can be executed using the planning operators. A HTN contains methods to decompose the nonprimitive tasks into subtasks.

One of the motivational examples for real-time pathfinding is Real-Time Strategy (RTS) Games [3]. In RTS games, solving the pathfinding problem becomes a challenging and complex task due to concurrency (i.e. various characters can start the pathfinding task simultaneously) and hard constraints on CPU time (e.g. BioWare Corp. allows 1-3 milliseconds for the pathfinding of all units [3]).

LRTA [4] is an adaptive learning method that satisfies all the essential properties of the real-time heuristic search. It uses a breadth-first approach to find all successor states of a current state in a look-ahead search of a constant depth. In the look-ahead search of depth d , it selects a successor state with

minimum cost, d actions away from the current state. A cost is the sum of two distances; a distance from the current state to the successor state and an estimated distance (also called heuristic value) from the successor state to the goal state. The heuristic value of the current state is modified with that of the selected successor state and the path from the current state to the successor state is executed. This process continues until the goal state or a stopping condition is reached.

LRTA suffers from two main problems: it works with a fixed look-ahead depth and produces poor solution quality due to convergence of the heuristic values into local minima (also known as heuristic depression [5]). Ishida proposed to use A* [6] to rectify the results of LRTA when it is trapped in a heuristic depression.

In this paper, we explore real-time path planning using HTNs and LRTA with the following two contributions. Firstly we demonstrate a method of automatic HTN creation. The main job of HTNs is to use LRTA search for actions selection in a smaller part of the problem search space. We have used a sector-based abstraction of the game map and a simple A* to automatically create the HTNs. The abstraction is built by dividing the game map into large blocks. Our second contribution is to propose an HTN planner by integrating the hierarchical information of the game map with LRTA. We also provide an empirical evaluation of the proposed hierarchical approach.

We use Open Real-Time Strategy (ORTS) [7] environment as a platform to explore real-time path planning in RTS games. It is an open source tool that provides an RTS game programming environment to explore AI (Artificial Intelligence) in RTS games. It also provides options to create new game definitions and user-friendly widgets (interfaces) for human interaction with the game AI during the game play.

The rest of the paper is organised as follows. In section 2 we discuss the algorithmic details of a procedure to build the game hierarchies. The third section describes the details of the automatic HTN creation. Section 4 discusses the proposed HTN planning model for path finding with LRTA. In section 5 we describe the existing relevant work. The section “Experimental Setup” provides the details about the experimental design of the model and evaluation parameters. Section 7 gives the experimental outcomes and their analysis. The final section concludes the work.

2 GAME MAP HIERARCHIES

To build the game map hierarchies, we use a sector-based partitioning of the map. In this partition, a game map is divided into distinct large blocks. Each block is called a zone. Figure 1 shows the partitioning of a game map into four blocks. A zone is represented by a structure with three coordinates and two

¹ Dept. of Informatics, Univ. of Huddersfield, HD1 3DH, UK.
Email: {m.naveed, d.kitchen, a.crampton}@hud.ac.uk.

strategic parameters. Figure 2 shows a C++ like representation of a zone. Three coordinates are used to represent the geographical area covered by a zone. These coordinates are the start, end and center points of a rectangular area covered by a zone as shown in figure 3.

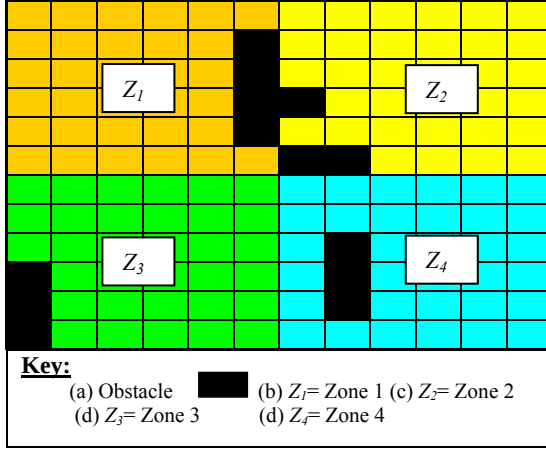


Figure 1. Partitioning a game map into four zones.

```

Struct Zone
{
    startPoint(int xo, int yo);
    endPoint(int xe, int ye);
    centerPoint(int xm, int ym);
    double obsDensity[4];
    double enemyDensity[4];
    Region region[4];
    int index;
}

```

Figure 2. A structure of zone representation.

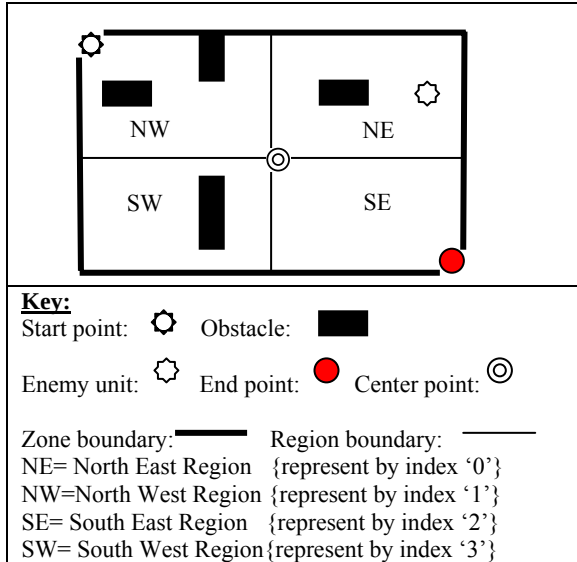


Figure 3. Graphical representation of a zone and its regions.

The center of a zone is moved to the nearest empty cell (point) if it lies on an obstacle. A zone is divided into four sub blocks called regions. The boundaries of each region are

calculated using three coordinates of a zone. After creation of four regions, the strategic parameters are calculated. The strategic parameter *obsDensity* represents the amount of obstacles that a region contains while *enemyDensity* is used to represent the strength of the enemy units in a region. Equation (1) shows the calculation of *obsDensity* in a region *R* and *enemyDensity* is measured using equation (2). The algorithmic details of the construction of the game map hierarchies are given in figure 4.

$$obsDensity(R) = \frac{\alpha}{\eta} \quad (1)$$

$$enemyDensity(R) = \frac{\beta}{\eta} \quad (2)$$

In above equations, α represents the number of cells covered by the obstacles in a region, β represents the number of cells covered by the enemy units in a region and η is the total number of cells in a region.

*/*Map is a structure representing 2D game map. N is a positive whole number such that N>1 and represents number of zones in a map. */*

```

Build_mapHierarchies( Map &map, int N)
{
    1-Zone allZones[N]=CreateZones(map, N);
    2- AdjustzoneCenters(map, allZones); /*if theoretical
        center lies on an obstacle*/
    3- CalculateRegionBoundarys(allZones);
    4-CalculateCenterofRegions(allZones)
    5- CalculateObstacleDensity(allzones);
    6-CalculateEnemyDensity(allzones)
}

```

Figure 4. Procedure of building map hierarchies.

3 CONSTRUCTION OF HTN

To solve the problem of moving a game object (a worker in our case) from one location to another location on an actual game map, we propose a solution by dividing the main problem into tasks and sub tasks. We assume the main task (of moving an object between two locations) is the planning of a route from one zone to another zone and the sub-task (and primitive task) is to find a path at the regional level. The primitive task is executed by using a variation of LRTA. In the presence of hierarchical information of a game map, we propose to create the task tree automatically (and initially offline) and then convert it in the form of an ordered task network. A representation scheme of the main task, given in figure 5, is used to create main tasks for path planning between the zones. According to this scheme, a main task of moving an object from zone A to zone B is different than moving the same object from zone B to zone A.

```

Struct MainTask
{
    /*Precondition parameters*/
    Zone sZone; // zone of start position
    Zone tZone; // zone of target location
    Boolean min_obst; /* path with regions of minimum
                      obstacle density. */
    Boolean noenemy; /* path with regions of minimum
                     enemy density*/

    /*Subtasks*/
    Region *list_of_regions;
    /*For future use in heuristic function*/
    int eucDistance;
    int actualDistance;
}

Struct TaskNetwork
{
    int index_sZone; /* index (zone number) of the zone of the
                     start position*/
    int index_tZone; /* index (zone number) of the zone of the
                     target position*/
    MainTask *list_of_tasks;
}

```

Figure 5. Main task and task network representation.

To create a list of all regions within a main task, we have used a modified version of a simple A* algorithm (implemented by Michael Buro and Sami Wagiaalla) which is a part of the free ORTS software. Algorithmic details of the creation of the Hierarchical Tasks Networks are given in figure 6.

```

TaskNetwork htn[N][N]; // A lookup table
HTN CreateHTN(Map &map, Zone* allZones, int N)
{
    For i=0:N
        For j=0:N
            1- Set the htn zone indices
            /* htn[i][j].sZone=allzone[i].index;
               htn[i][j].tZone=allzone[j].index;*/

            2- Create a new Main Task for path finding with
               min_obst =false
            3- Set the preconditions
            4- Calculate list of regions
            5- Create a new Main Task for path finding with
               min_obst =true
            6- Set the Preconditions
            7- Calculate list of regions
            8- Create a list of main task
            9- Attach the list to htn array
        End For
    End For
}

```

Figure 6. Procedure of automatic HTN creation.

4 HTN PLANNING

Like other HTN Planners e.g. [8], our proposed HTN Planning model, called HTN-LRTA, also performs both planning activities (i.e. task decomposition) and instantiation of operators

(i.e. finding of a sequence of actions to execute the primitive task). HTN-LRTA decomposes a main task into subtasks in a sequential way rather than a recursive one. However, the selection of a main task (non-primitive task) relevant to the current planning problem is performed in a random access way as the HTN structure is in the form of a lookup table. Once a primitive task is found, then LRTA is applied to execute this task. An algorithmic detail is given in figure 7.

HTN-LRTA has one method, called *Decomp*, which is a simple forward search with a linkedlist traversing mechanism. The procedural details of *Decomp* (in a C++ style) are given in figure 8. To implement LRTA in ORTS, we have modified a simple A* with a look-ahead of 1-ply.

```

/*HTN-LRTA path planner*/
// P1 is the start position and P2 is the goal position
// d is the depth of look-ahead task in LRTA

HTN_LRTA( P1, P2, htn, Map &V, d)
{
    1- Zone z1=Zoneof(P1);
    2- Zone z2=Zoneof(P2);
    3- MainTask T=htn[z1.index][z2.index].list_of_regions.
    4- Regionlist l1=Decomp(T, z1, z2, true)
    If l1 is Null
        6- Print "Target out of Reach"
        7- Return
    Else
        While l1→next!=NULL
            8- P1=LRTA(P1, l1→Center, V, d)
            If P1=l1→Center
                9- l1=l1→next
            End if
        End While
    End if
}

```

Figure 7. HTN-LRTA procedure.

```

/* z1 is the zone of the start position, z2 is the zone of target
position, mp is the minimum obstacle path and bool
represents boolean data type*/

```

```

Region* Decomp(MainTask *T, Zone z1, Zone z2, bool mp)
{
    1-Region *list=NULL;
    If T=NULL
        2- Return list;
    Else
        /*use pre-conditions to select the relevant subtask*/
        While T→next!=Null
            If T→sZone=z1 && T→tZone=z2 && T→ min_obst =mp
                3- list=T→list_or_regions
            End if
            4- T=T→next;
        End While
    End if
    5- Return list;
}

```

Figure 8. HTN Method

5 RELATED WORK

The use of game map abstraction for speeding up the pathfinding process has been explored in Hierarchical Path-finding A* (HPA*) [9] and Partial Refinement A* (PRA*) [10]. In both approaches, A* is applied to make a path plan on a high-level abstract graph. This abstract path is then refined to a path with coordinates of the actual map. However, they both use different abstraction and refinement methods. HPA* divides the game map into large blocks and connects the neighbouring blocks using entrance points. The set s of entrance points, along a border line (say) l , of a block is identified using four conditions: (i) each point of s lies on l , (ii) for each member of s , there is a symmetrical point (on the boundary of a neighbouring block) that is adjacent to the corresponding point of s , (iii) all points of s are obstacle free and (iv) s is extended along l as long as all the previous three conditions are met.

The optimal paths between all entrance points of a block are computed offline using A* and cached in a look-up table for their use in the refinement task during an online path search. These entrance points of each block are also used to build an abstract graph of the game map. The abstract graph is built around transitions between the blocks. The number of transitions between two blocks is dependent on the number of entrance points between two neighbouring blocks e.g. there are two transitions if the set's size is 6, one transition if the set is of size 3 and no transition if the set is empty. During an online search, both start and target positions are inserted (temporarily) into the abstract graph by attaching them to the corresponding borders of their blocks. A* search is applied on the new abstract graph for computing an abstract path. In the refinement process, each block crossing is replaced by the pre-computed paths (stored in the look-up table). After refinement, HPA* also applies path smoothing (as a post-processing step) to improve the solution quality (i.e. path length). The results show that the abstraction provides a huge reduction in the number of nodes explored during the path planning, especially when the solution length is large. However, for short paths, A* performs faster than HPA*. At higher levels of abstraction, the integration of start and target positions with the abstract graph takes more time than inserting them at a low level of an abstract graph.

Our work has one similarity with HPA* i.e. the use of block (rectangular) shaped regions for the creation of game map hierarchies. However, we use a different form of block representation in HTN-LRTA than that of HPA*.

PRA* [10] uses an abstraction approach that is based on two local features of the game map i.e. cliques and orphans. A clique is a group of nodes which are connected to each other. In this work, a maximum 4-clique is used. An orphan is a node that is connected to only one node. The abstraction process moves through the game map and finds all the cliques. Then each clique is represented by one abstract node. An orphan node is attached to the abstract node which it neighbours. PRA* uses A* with a modified heuristic function i.e. it uses the *octile distance* to estimate the distance between two abstract nodes. It dynamically selects an abstraction level to start the search from. PRA* refinement process breaks the abstract path into parts (of fixed length) and keeps refining the parts by finding a path in a low level abstraction using A* until it finds any node that is abstracted by the target state. The representation scheme of PRA* is also applicable for regions of irregular shape.

Minimal-Memory (MM) abstraction [11] is a hybrid of PRA* and HPA*. An MM abstraction is built by dividing the game map into large sectors and then each sector is divided into regions. A region in the MM abstraction is an obstacle free area of the sector, so a sector can have more than one region. Each region in the abstraction is represented by a single point e.g. the center point of the region. The results of [11] show that the amount of memory required to store the abstract maps decreases as the sector size increases. The MM abstraction and refinement also finds optimal paths with the larger sector sizes. This approach has been used in the commercial game Dragon Age™ (BioWare Corp.). Our abstraction building scheme has two similarities with MM abstraction. First, it uses blocks of regular shape (i.e. rectangular) to partition a game map. Second, our method uses the center point of the regions for the online path planning. However, we use a different representation scheme and a path planning mechanism.

LRTA offers many design choices such as how to find the successors states (of a current state), how to modify the heuristic values, and which successor states' values should be updated [12]. This means there are many variations of LRTA in the literature. It is not possible to discuss all of them in this paper, therefore, we mention here the variations of LRTA that have been explored with the abstraction of game maps.

PR LRTS [3] is a hybrid of LRTS [13] and the abstraction scheme of PRA*. LRTS (Learning Real-time Search) is a variation of LRTA. LRTS makes three modifications to the simple LRTA: (i) it uses a weighted heuristic function, (ii) it introduces backtracking in LRTA by putting an upper bound on the learning amount, and (iii) it applies a max-min learning rule (to modify the heuristic value of the current state). For these three modifications, LRTS introduces three control parameters which are look-ahead depth, optimality weight and learning quota. In fact, LRTS is a combination of three other variations of LRTA which are SLA* [14], weighted LRTA [15] and γ -Trap [16].

In PR LRTS, the path planning process starts by applying LRTS at a fixed level of the abstract graph. The partial path found by LRTS is refined by Local Repair A* [17]. This refined abstract path is used to create a sub goal and a corridor for the path search (or refinement) in the next iteration. At ground level, the procedure performs execution instead of refinement. The purpose of this corridor is to keep the LRTS or A* search limited to the states which are part of the corridor. However, if the corridor is empty then search is applied on the entire graph.

The results of PR LRTS [3] are obtained on different pathfinding problems (generated over three maps) and are compared with A*, LRTA and LRTS. The results are compared using five performance measures which are convergence planning, convergence memory, first-move lag, convergence travel and suboptimality. The convergence planning shows the planning effort (in terms of CPU time and number of states touched) during the convergence process. The convergence memory is the total number of heuristic values that are stored during convergence process and first-move lag is the planning effort to make the first move. The convergence travel is the total cost of traversing all edges during the convergence process while suboptimality is the difference (in percentage) between the calculated path and the shortest-path (calculated by running the A*). The results show that A* performs the best in convergence travel and convergence memory while LRTA shows the best

performance in the first-move lag. PR LRTS (at level 3 and 2 of abstraction) with look-ahead depth of 1-ply shows the best performance in convergence planning. Suboptimality also increases with increasing the abstraction level, therefore, A* and LRTA (without abstraction) produce fewer suboptimal results than others. In our work, we apply LRTA at ground level rather than on top level hierarchies of the game map.

A variation of PR LRTS [18] uses a dynamic approach to select the look-ahead depth during the planning process. This approach uses the decision tree to dynamically determine look-ahead depth for solving a given problem.

A hybrid of an HTN-based planner SHOP [8] and the FF (Fast Forward) planner [19] has been proposed by McCluskey et al.[20] as *HyHTN*. The results of hybrid HTN are compared with two planners, SHOP and EMS [21]. The results show that *HyHTN* is faster than EMS and takes almost same time as SHOP does to solve the benchmark problems.

6 EXPERIMENTAL SETUP

Experiments are performed using the ORTS game programming environment. We have modified an existing game of ORTS called game-1 to create a testbed for our experiments. We changed the game-1 script file and its server definition such that it has only one unit with one base center and three workers. In the modified game-1, a base center and three workers are the only Non Player Characters (NPCs). We applied our model in game-1 to move the workers from one location to another on the game map. A 2D display of this game world is shown in figure 9. Base center is a static object of the game and is used as a reference point for the creation of hierarchies.

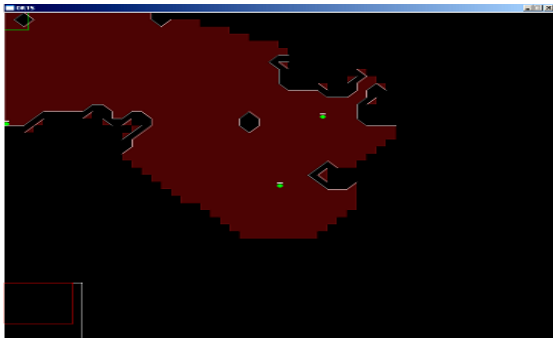


Figure 9. A map used in the modified version of game-1.

In the modified definition of the game-1, a game world has the static obstacles (i.e. hills or cliffs) only. The default position of three workers and a base center is set on the top right corner of the map for each simulation. To implement the game map hierarchies and HTN planning procedure, we have used “Sample AI project” which is a part of the ORTS game programming environment.

Three maps of 64X64 tiles are used to generate 35 pathfinding problems where each worker is given one problem to solve in a simulation. Three evaluation methods are used to compare the performance of HTN-LRTA and LRTA. These three evaluation methods are suboptimality, convergence memory and convergence cost. The suboptimality is measured using equation (3). This measure is taken from [9]. In equation

(3), l_m is the length of the path found by a method and l_o is the length of the optimal path. The optimal path is calculated by using A*.

$$\text{suboptimality} = (l_m - l_o / l_o) \times 100 \quad (3)$$

The convergence memory is the total number of heuristic values saved during the planning process and the convergence cost is the total number of nodes touched during the planning process.

7 RESULTS AND ANALYSIS

The results are obtained by running ORTS game-1 simulations on three different computers of 3.0 GHz clock speed. In all simulations, both ORTS server and client applications are run on the same computer i.e. with localhost connectivity. The hierarchies of three maps are created with four different zone sizes which are 3, 6, 9 and 12. Average suboptimalities of HTN-LRTA with respect to these zone sizes are shown in figure 10.

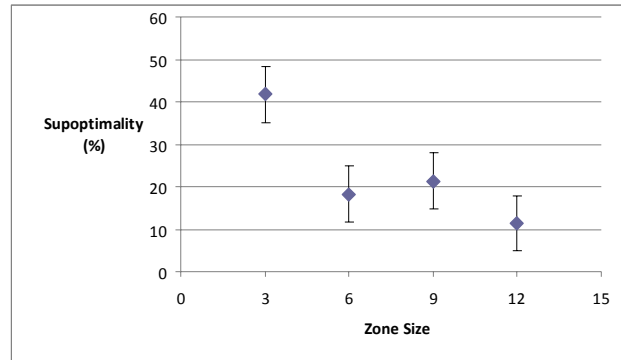


Figure 10. Suboptimality profile of HTN-LRTA with zone sizes.

The sub-optimality of HTN-LRTA is dependent on the zone size and it finds short paths with higher zone sizes. Figure 10 also reveals that HTN-LRTA finds better paths with zone size of even numbers than the paths with odd ones.

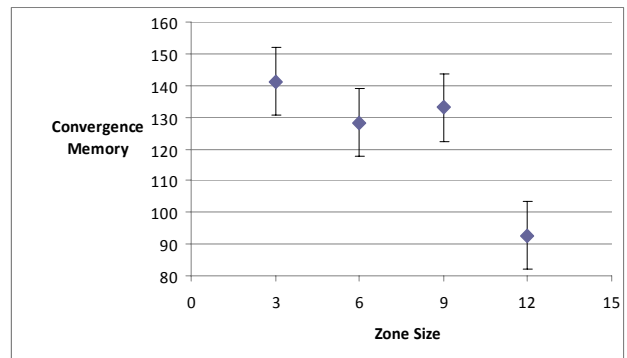


Figure 11. HTN-LRTA convergence memory Vs zone size.

HTN-LRTA consumes a very small amount of memory during the online path planning if a game map is partitioned with a larger zone size. Figure 11 shows that the memory consumption of the hierarchical task network planner remains

high when using the game map abstraction with smaller zone sizes.

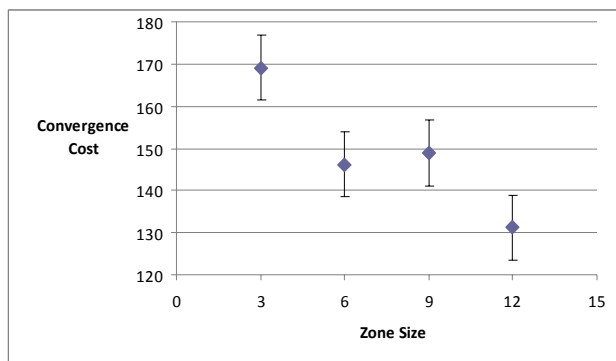


Figure 12. HTN-LRTA convergence cost Vs zone size.

Figure 12 shows that HTN-LRTA explores a small number of states if a game map abstraction is created with a high zone size. However, with small zone sizes, HTN-LRTA consumes high convergence cost. The convergence cost is directly related to the CPU usage, so HTN-LRTA with large zone size finds a path using less CPU cycles as compared to its planning with a smaller zone size. In HTN-LRTA with a higher zone size, the adjacent regions have a shorter distance between their centers than the adjacent regions of the smaller zone sizes. As LRTA gives better performance in the problems where the actual path between the start and goal states is short [12], therefore, with large zone sizes HTN-LRTA gives better performance than with small zone sizes.

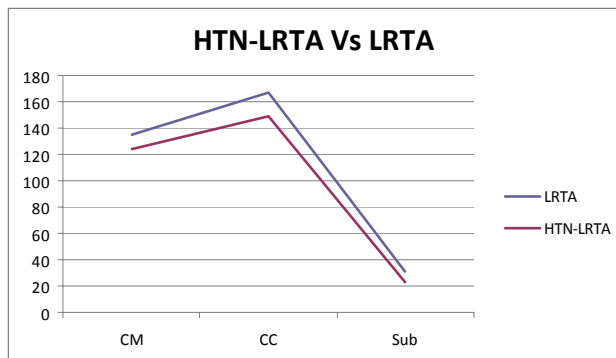


Figure 13. A comparison of HTN-LRTA and LRTA w.r.t. convergence memory (CM), Convergence Cost (CC) and sub-optimality (Sub).

The average results of HTN-LRTA over all four zone sizes are compared to the average results of LRTA for all 35 problems. This comparison is shown in figure 13. These results show a clear advantage of HTN-LRTA over LRTA with respect to all three validation parameters. On average, HTN-LRTA consumes 8% less memory than LRTA and it makes approximately 11% less search efforts (i.e. convergence cost) than LRTA. HTN-LRTA also improves the quality of the solutions and provides 25% less sub-optimal results than LRTA.

8 CONCLUSIONS

A preliminary work is presented in the direction of using an automatic hierarchical task network approach in the domain of real-time strategy games. A detailed description of the proposed model is provided and the novelty of the work has been highlighted. The results show that the effectiveness of the automatic creation of HTN depends on a user-defined parameter i.e. number of zones. The number of zones directly influences the processing time, memory consumption and the quality of the solution. The proposed hierarchical approach achieves a better performance than the LRTA on average.

As a future work, the hierarchical model can be modified to automatically determine an appropriate number of zones for a given map using an evolutionary approach. We also aim at using the hierarchical information for tuning the heuristic function of the path planner. In future experiments, the proposed model would be studied with the dynamic settings of the game i.e. the enemies and other movable obstacles in a game map. The future work also includes an empirical analysis of the proposed hierarchical approach with respect to the existing relevant approaches.

REFERENCES

- [1] S. Koenig. Agent-Centered Search. *Artificial Intelligence Magazine*, 22(4): 109-131 (2001).
- [2] M. Ghallab, D. Nau and P. Traverso. *Automated Planning and Practice*. Morgan Kaufmann Publishers, Elsevier (2004).
- [3] V. Bulitko, N. Sturtevant, J. Lu and T. Yau. Graph Abstraction in Real-time Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 30(1): 51-100 (2007).
- [4] R. Korf. Real-Time Heuristic Search. *Artificial Intelligence*, 42(2-3): 189-211 (1990).
- [5] T. Ishida. Moving Target Search with Intelligence. In: *Procs. 10th National Conference on Artificial Intelligence (Problem Solving: Search and Expert Systems)* AAAI Press, pp. 525-532 (1992).
- [6] P.E. Hart, N.J. Nilsson and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2): 100-107 (1968).
- [7] M. Buro. Real-Time Strategy Games: A New AI Research Challenge. In: *Procs. International Joint Conference on AI*, (2003).
- [8] D. Nau, Y. Cao, A. Lotem and H. Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In: *Procs. 16th International Joint Conference on Artificial Intelligence*, 2:968-973 (1999).
- [9] A. Botea, M. Müller and J. Schaeffer. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 1(1): 7-28 (2004).
- [10] N.R. Sturtevant and M. Buro. Partial Pathfinding Using Map Abstraction and Refinement. In: *Procs. 20th National Conference on Artificial Intelligence* AAAI Press, pp. 1392-1397 (2005).
- [11] N.R. Sturtevant. Memory-Efficient Abstraction for Pathfinding. In: *Procs. AIIDE-2007*, June 6-8, Stanford, California, USA, pp.31-37 (2007).
- [12] S. Koenig. A Comparison of Fast Search Methods for Real-Time Situated Agents. In: *Procs. 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2: 864-871 (2004).
- [13] V. Bulitko and G. Lee. Learning in Real-Time Search: a Unifying Framework. *Journal of Artificial Intelligence Research (JAIR)*, 25(1): 119-157, (2006).

- [14] L-Y. Shue and R. Zamani. An Admissible heuristic search algorithm. *Lecture Notes in Computer Science (LNCS)*, (procs. 7th International Symposium on Methodologies for Intelligent Systems), 689(1993): 69-75 (1993).
- [15] M. Shimbo and T. Ishida. Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, 146(1): 1-41 (2003).
- [16] V. Bulitko. Learning for Adaptive Real-Time Search. *Tech Report*. <http://arxiv.org/abs/cs/0407016>, *Computer Science Research Repository (CoRR)* (2004), accessed date: 14th Nov, 2009.
- [17] B. Stout. Smart Moves: Intelligent Pathfinding. *Game Developer Magazine* (1996).
- [18] V. Bulitko, M. Luštrek, J. Schaeffer, Y. Björnsson and S. Sigmundarson. Dynamic Control in Real-Time Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 32(1): 419-452 (2008).
- [19] J. Hoffmann. A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-Climbing Algorithm. *Lecture Notes in Computer Science (procs. 12th International Symposium on foundations of Intelligent Systems)*, 1932: 216-227 (2000).
- [20] T.L. McCluskey, D. Liu and R.M. Simpson. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In: *Procs. ICAPS-2003*, Trento, Italy, pp. 92-101 (2003).
- [21] T.L. McCluskey. Object Transition Sequences: A New Form of Abstraction for HTN Planners. In: *Procs. 5th International Conference on Artificial Intelligence Planning Systems (AIPS-2000)*, pp. 216-225 (2000).

A Flexible Bio-Affective Gaming Interface

Jorge Arroyo-Palacios¹, Daniela M. Romano¹

Abstract. Affective bio-feedback can be an important instrument to enhance the game experience. Several studies have provided evidence of the usefulness of physiological signals for affective gaming. However, due to discrepancies in the findings of psychology studies, the pattern matching models employed are limited in the number of emotions they are able to classify. This paper presents a bio-affective gaming interface (BAGI) that can be used to customize a game experience according to the player's emotional response. Its architecture offers important characteristics for gaming such as: flexibility, scalability and multi-category discrimination in real time. These features are important not only because they allow the classification with pattern matching and machine learning models but also because they make possible the reusability of previous findings and the inclusion of new models to the system.

In order to prove the effectiveness of BAGI two different types of neural networks have been trained to recognize emotions. After that, they were incorporated into the system to customize in real-time the computer wallpaper, according to the emotion experienced by the user. The best results were obtained with a probabilistic neural network with accuracy results of 84.46% on the training data and 78.38% on the validation for new independent data sets.

1.- INTRODUCTION

Physiological signals offer a promising medium to interact in a natural and intuitive way with the game environment. In addition of being reliable, sensible and provide real time feedback, physiological signals offer an insight into human's physical and mental state which can be used to enrich the game interaction. Common physiological measures already used in gaming or Human Computer Interaction (HCI) include: cardiovascular, electrodermal, muscular tension, ocular, skin temperature, brain activity and respiration measures. The game industry has already seen their potential and several research studies have been conducted to investigate the best way to use them.

One of the earliest commercial examples of a bio-adaptive interface used in commercial games can be seen in "Tetris 64"², released for the Nintendo 64 platform. In "Tetris 64" a sensor is placed at the player's ear to monitor the heart rate, the game speeds up or slows down depending on the player's cardiovascular activity. A more modern commercial example is "Relax & Race"³ by Vyro games. In "Relax & Race" the game takes the form of a race between two characters: stress and the player. The game senses the electrodermal activity (EDA) of the player and increases the speed of the character as the player relaxes.

There are also several research studies that explore physiological signals in gaming. Toups et al. [1] for example use electrodermal and electromyographic (EMG) activity as an indication of increased attention, effort and stress in team game play. In their game, the computer opponents sense the physiological signals and pursue players with higher activation levels. One interesting feature of this study was that the physiological information was provided to the team coordinator thus he could modify the game strategy. Another study that also uses EDA and EMG activity was carried out by Predinger et al. [2] where the emotional response of a character was adapted according to physiological responses of the player in a virtual card game.

Gilleade et al. [3] proposed that the identification of emotions, through physiological signals, can provide means to the Artificial Intelligence (AI) of the game for automatically: i) assist (e.g when the player is frustrated), challenge (e.g. when the player is bored), emote (to enhance the game experience of the player). In their studies they developed a game that monitors the cardiovascular activity to increase or decrease the number of threats in a game.

Educational games and tutoring systems have also employed the use of physiological signals to provide affective feedback to the system and adapt according to the student emotional state [4].

Research in brain activity has lead to the creation of brain-computer interfaces (BCI) which have been used as a game interface control for an immersive 3D gaming environment [5] and for moving a ball on a table in Brainball [6]. Ko et al. presents in [7] the emotion recognition capabilities for games of three currently available portable BCI devices: NIA⁴, Emotive EPOC⁵ and Mind Set⁶.

Respiration measures have also been used to interact with games. Arroyo and Romano [8] developed a simple balloon game where the player blows up a virtual balloon with his chest expansion and contraction while breathing. Table 1 presents a summary of some games that recognize emotions by the use of physiological signals.

Game	Signal	Detect	Emotion	Action
Tetris 64	HR	Increase	Anxiety	Game slows down (normal mode) Game speeds up (reverse mode)
		Decrease	Relaxation	Game speeds up (normal mode) Game slows down (reverse mode)

¹ Dept. of Computing Science, Univ. of Sheffield
Email: {j.arroyo, d.romano}@dcs.shef.ac.uk

² <http://uk.ign64.ign.com/articles/154/154081p1.html>

³ <http://www.vyro-games.com/products/relaxandrace.php>

⁴ http://www.ocztechnology.com/products/ocz_peripherals/nia-neural_impulse_actuator

⁵ <http://www.emotiv.com/>

⁶ <http://www.neurosky.com/>

Relax & Race	SR	Increase	Relaxation	Character slows down
		Decrease	Anxiety	Character speeds up
PhysiRogue [1]	EDA, EMG	Increase	Attention, effort, stress.	Opponents pursue players with highest activation levels
Card game [2]	EDA, EMG	Increase	High arousal, negative valence	If player makes progress agent shows: -joy (empathic mode) -fear (negative empathic mode)
Similar to Missile Comm and [3]	HR	Increase	Over stimulation	Reduce number of threats.
		Decrease	Lack of engagement	Increase number of threats
Prime Climb [4]	EMG, EDA, HR	Specific changes	Reproach, shame and joy	Depending on emotion, agent can intervene in game.
Brain ball [6]	Brain activity	Specific changes	Relaxation	Ball is moved
Blow up balloon [8]	RSP	Increase and decrease	None	Virtual balloon is blown up
Relax to Win [9]	SR	Increase	Relaxed	Character slows down
		Decrease	Anxious	Character speeds up

HR = Heart rate
RSP = Respiration

SR = Skin Resistance

Table 1.- Use of physiological signals in games.

There are two important observations from the literature reviewed:

i) Most of the emotion recognition systems (ERS) follow an ad hoc strategy (i.e. they provide a solution designed for a specific problem or task, non generalizable, and which is not easily adaptable to other purposes).

ii) There are very few multi-category discrimination emotion recognition systems implemented in real-time. All the emotion recognition systems of the games presented in table 1 are aimed to identify the presence or absence of a particular emotion, or to discriminate among small sets of opposite emotions.

Nowadays, games are being produced for a great variety of domains including: entertainment, training, medical therapies, education and socialization. Users are more demanding in their expectations, and current games involve more photographic realism, cinematic sequences and new interaction methods. Emotions recognition systems can contribute to enhance the emotional experience, however their current construction strategy is expensive, time-consuming, limited (in discrimination capabilities) and does not allow the reusability of tools and previous findings.

In this paper the architecture, implementation and validation of a flexible multi-category emotion recognition system is presented. Specifically, this paper contributes to the research efforts in affective gaming by presenting:

- Implementation of a real-time multi-category emotion recognition system.
- A flexible and scalable architecture that facilitates the integration of findings from other studies.
- Real-time classification results from two machine learning algorithms.
- A simple application that adapts the desktop wallpaper according to the emotional state of the user.

2.- VARIABLE PARAMETERS IN EMOTION RECOGNITION SYSTEMS

2.1.- Set of emotions.

Depending on the type of application or game, different sets of discrete emotions need to be considered. For instance, adventure/action games might need the identification of emotions such as: enjoyment, frustration, anxiety, boredom. In learning/training games shame might need to be considered too (as suggested in [4]). Social games such as Second Life⁷ could find useful to include in the recognition emotions such as: love, anger and sadness. For horror games such as Dead Space⁸, FEAR⁹ and Resident Evil¹⁰, fear and anxiety might need to be recognized.

Besides the use of discrete emotions, researchers have also proposed the use of dimensional models for representing emotions in games (e.g. [10]). In this way a set of emotion for a game could be composed by the following 4 affective states: 1) positive valence – high arousal, 2) positive valence – low arousal, 3) negative valence – high arousal, and 4) negative valence – low arousal.

2.2.- Measures and features to analyze

Common physiological measures already used in gaming or HCI include: cardiovascular, electrodermal, muscular tension, ocular, skin temperature, brain activity and respiration measures. Each of these measures can be more or less convenient depending on the type of game and emotions to be recognized. From these measures, several features can be calculated from various analysis domains including: statistics, frequency domain, geometric analysis, multiscale sample entropy and sub-band spectra. Reviewing some relevant studies in HCI it can be seen that they have privileged the use of different physiological measures over others, and have extracted different numbers of features to analyze. For instance, Kim and Andre [11] extracted a total of 110 features from 4 sensors: EMG, ECG, SC and RSP; Picard et al. [12] extracted 40 features from EMG, BVP, RSP and SC; Nasoz et al. [13] used GSR, temperature and HR signals.

2.3.- Times to analyze

Another variable parameter among the emotion recognition systems in the literature concerns to the time window used to

⁷ <http://secondlife.com/>

⁸ <http://www.visceralgames.com/us/page/deadspace>

⁹ <http://www.whatisfear.com/>

¹⁰ <http://www.residentevil.com/index.php>

analyze the physiological signals. For instance, Picard et al. [12] used a time window of 100 seconds from the end of each emotion segment to analyze all the signals in their first data set; Kim and Andre [11] segmented their data in samples of 160 seconds by taking the middle part of each signal; Nasoz et al. [13] used a time frame of 7 seconds when the emotion was most likely to be experienced.

2.4.- Approaches for physiological pattern classification

2.4.1.- Pattern matching

The emotion recognition systems in the games listed in table 1 are implemented following a pattern matching approach. Following this strategy, the system detects the presence of the constituents of a rigidly specified pattern in the physiological data. The classification process can be based on existing models describing the relationship between psychology and physiology, like for example the model proposed by Cacioppo & Tassinary [14] or use empirical findings from literature, for example, Stemmler [15]. The dilemma is to decide which model and physiological definitions, the system should be based on.

Table 2 illustrates some patterns of heart rate associated to emotions from different studies that a computer system could use to discriminate a set of three emotions composed by fear, joy and anger.

Author	Signal	Fear	Sadness	Happiness
Christie [16]	Heart rate	Increase	Decrease	Decrease
Ekman et al. [17]	Heart rate	Increase (high)	Increase (high)	Increase (low)
Frederickson [18]	Heart rate	Increase	Decrease	No significant difference

Table 2.-Patterns associated to emotions from different studies.

A vast number of studies in psychology document the autonomic responses to emotions. However not all of them followed the same methodology. Unfortunately, the differences are present in critical elements of the study such as: model and set of emotions investigated, stimuli used to evoke the emotions, physiological measures, features extracted, and type of analysis on the data (already highlighted in [19]). As a result some discrepancies remain among their findings, as it can be observed in Table 2 and, in more detail, in the compilation of physiological responses to specific emotions in [20, 21].

Despite the discrepancies among psychological studies, in the gaming literature pattern matching techniques have demonstrated to be practical especially for: i) determining the presence or absence of a particular emotion, or ii) discriminating small sets of opposite affective states. Additionally pattern matching offers important characteristics for a real time interaction such as simple implementation and very fast computation.

2.4.2.- Pattern recognition

Another approach that can be used to implement emotion recognition for gaming is pattern recognition. It implies the

training of a system to recognize a set of affective states corresponding to a particular emotional model. The machine learning model receives as input labelled sets of physiological data and then it is trained to detect the patterns associated to each emotion. The model is then able to classify new sets of physiological data into specific emotions. As a result, a physiological definition for different affective states has been obtained through empirical data. The advantage of this approach is that it offers ecological validity on the emotions recognized (the settings during the pattern recognition process can be identical or similar to the classification settings) and provides good classification rates among several emotions (as showed in several offline HCI studies [11-13, 22-24]).

Despite the advantages of this approach it has not been fully exploited by the game industry, perhaps due to concerns that might arise as consequence of:

- The time that a pattern recognition process takes (depending on the processing power, algorithms and quantity of data, it can take up to several hours or days).
- Difficulties to reuse the system for other purposes or to reuse the finding from other studies.
- Few studies show good results from user independent and real-time implementations (e.g. [25]).
- Lack of guided principles for its application
- Lack of physiological affective databases to train the systems.

In the research presented in this paper some of these concerns have been addressed.

2.5.- Importance of flexibility in ERS

There is a wide range of applications where emotion recognition systems can be very useful however the development of such system for a particular situation is complex, multifaceted and very time-consuming. It requires taking into consideration theoretical, methodological, technical and some other aspects.

The ad hoc strategy followed so far has proven good classification results; nevertheless, the reusability of systems and findings is very difficult and the comparison of results among experiments carried out in different labs is almost impossible.

The flexibility in an emotion recognition system would facilitate its use in different type of applications, allow the reusability of findings and facilitate the comparison of results. It would be also possible to use the classification strategy that best suit to our application. Moreover, the research community is continuously proposing new theories and providing new discoveries. A flexible architecture on the set of emotions, physiological signals, time windows and classification strategy would also allow scalability for the inclusion of advances in emotion research.

Table 3 presents some machine learning based emotion recognition studies where different parameters were considered for the classification of emotions.

Study	Emotions	Time window	Measures	Analysis technique
Kim & Andre [11]	High arousal, low arousal, positive valence, negative valence	160 seconds	EMG, ECG, SC, RSP (110 features)	LDA, Binary tree model

Picard et al. [12]	No emotion, anger, hate, grief, platonic love, romantic love, joy, reverence.	100 seconds	EMG, BVP, RSP, SC (40 features)	SFFS, FP, SFFS-FP
Nasoz et al. [13]	Sadness, anger, fear, surprise, frustration, amusement	7 seconds	EDA, ST, HR	KNN, DFA, MBP
Kim et al. [22]	Fear, neutral, joy	50 seconds	ECG	K-means based expectation maximization
Haag et al. [23]	Valence and arousal	2 seconds	ECG, BVP, SC, EMG, RSP, ST	NN trained with resilient propagation
Kim et al. [24]	Sad, stress, anger, surprise	50 seconds	ST, EDA, ECG, BVP	SVM

ST = Skin Temperature
LDA = Linear Discriminant Analysis
MBP = Marquardt Backpropagation
SFFS= Sequential Floating Forward Selection

SC = Skin Conductance
NN = Neural Network
SVM = Support Vector Machine
FP = Fisher Projection

Table 3.- Different parameters in offline emotion recognition studies

3.-BAGI – DESIGN AND IMPLEMENTATION

One of the most important characteristics of BAGI is its architecture which allows: flexibility, scalability, multi-category discrimination and real time affective feedback. The innovative aspect resides in the classification module which provides two important characteristics: i) flexibility to receive different configurations in the parameters mentioned in section 2, and ii) ability to incorporate different models obtained through a machine learning process for a real time multi-category discrimination. Figure 1 presents the architecture of BAGI.

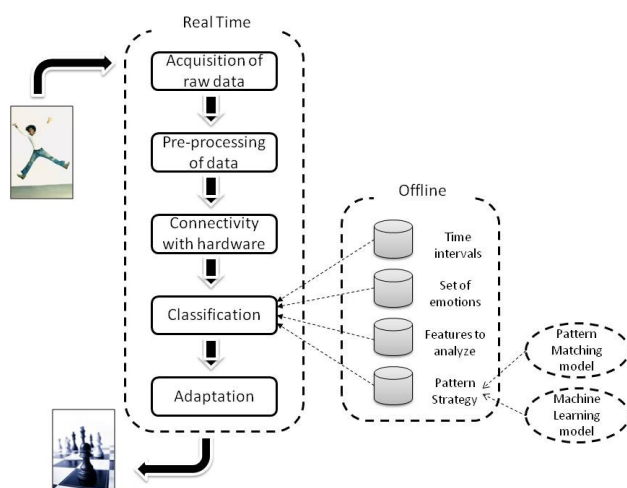


Figure 1.- Architecture of BAGI

The following subsections briefly present the modules involved in BAGI, with a more detailed description of the classification module.

3.1.- Acquisition of physiological data

The equipment selected for the acquisition of the physiological data was Procomp Infiniti by Thought Technology¹¹. It offers important characteristics such as: user-friendly sensors, real time feedback, and connectivity with other applications. The Procomp Infiniti is an eight channel, multi-modal device for real time computerized biofeedback and data acquisition. It has 8 sensor inputs with two channels sampled at 2048 samples/second (s/s) and six channels sampled at 256 s/s.

3.2.- Pre-processing

The pre-processing module focused in two main tasks: i) the application of noise reduction algorithms to clean the signals, and ii) the construction of vectors with the values of the different physiological measures.

Although there are several techniques that could be applied to clean the signals and facilitate the analysis, simple and fast algorithms that could contribute to the effectiveness of the system without compromising its real-time capabilities had to be considered. For this reason, the noise in the signals was reduced using a moving average filter. A moving average filter smoothes the data by replacing the data points of a signal by averaging the value of a selected number of d adjacent points, Equation (1) illustrates a 3-point smooth:

$$S_j = \frac{Y_{j-1} + Y_j + Y_{j+1}}{d} \quad (1)$$

where S_j is the j^{th} point in the smoothed signal, Y_j the j^{th} point in the original signal, and d is a positive integer called *smooth width* or *averaging factor*. The higher the number of d , the less variability will be observable in the resulting signal.

This filter was computed directly on the signals by using the smoothing algorithm function provided by Biograph Infiniti software.

The construction of vectors was carried out to facilitate the analysis and classification process. The first step was the extraction of 5 statistical features from the signals: mean, standard deviation, coefficient of variation, maximum and minimum.

The next step involved the normalization of the signals, a common practice in the emotion physiology literature, as a consequence of the physiological differences among individuals. The relation of change (ρ) was obtained by comparing the extracted features of each signal during the stimuli (f_{stimuli}) with the extracted features of each signal during baseline (f_{baseline}):

$$\rho = \frac{f_{\text{stimuli}}}{f_{\text{baseline}}} \quad (2)$$

¹¹ <http://www.thoughttechnology.com/proinf.htm>

In this way, the normalization facilitates the identification of physiological changes from the baseline to one particular emotional state: If $\rho > 1$ it means an increase in the signal, if $\rho < 1$ it means a decrease in the signal and if $\rho = 1$ it means no change in the signal.

Finally, each vector is composed of one emotional label and the normalized values corresponding to the signals considered for the analysis.

3.3.- Connectivity with hardware

The connection to the Procomp equipment was actually done through the Biograph Infiniti with the aid of a Software Development Kit provided by Thought Technology. There were three components involved in the implementation of the connectivity with Biograph Infiniti: C++ classes, XML file and Connection Instrument.

3.4.- Classification

First, at the initial setup, the classification module receives the intervals of time, set of emotions, features to analyze and the pattern strategy for classification. For the pattern strategy this module can receive models from two types of sources: i) existing rigid pattern matching models from literature and ii) models obtained through a machine learning process. The system allows the user to browse through different folders and select the data that BAGI will use for the classification.

Once the system started, the classification module receives in real-time the relation of change obtained by the pre-processing module. Finally with the information provided at the initial configuration the module classifies the physiological patterns and gives as an output one emotion.

BAGI allows the incorporation of new sets of emotions, intervals of times and pattern matching models to the system. The only constraint is that this information must be in a specific but simple format. A simple text editor can be used for this purpose. A brief description of the implementation of the flexibility features of the system in the classification module is provided below.

3.4.1.- Different set of emotions

In BAGI the user specifies the set of emotions to classify; this set can vary in the number and labels of affective states. In order to do this, a simple text file with the emotions can be created, and saved into the system. Figure 2 presents two sets of affective states that can be used for classification.

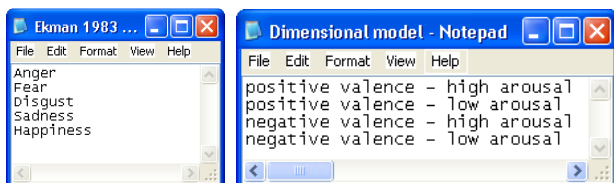


Figure 2.- Two sets of emotions for the BAGI system

3.4.2.- Flexible time window

Following the same strategy, BAGI enables the user to specify the time window for the analysis. The user can create a file in simple text editor with the time intervals, indicating the starting and ending time (in milliseconds) for each section to analyze. The system reads the file and let a timer know about the intervals, the first two values in the file represent the start and ending time for the baseline and the following pairs of values represent the start and ending time of the sections to analyze.

The calculated values of the first interval are stored as the baseline for a user and at the end of the following intervals the system processes the physiological signals and gives as an output an emotion. Figure 3 presents an example of a file containing the time intervals of a session.

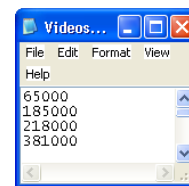


Figure 3.- Time intervals for the analysis of signals

3.4.3.- Pattern Matching models

Different pattern matching models can be used with BAGI. They just need to follow a specific format as for example the one illustrated in Table 4.

	CH1		CH2	
ID	HR		SC	
Anger	1.2	2	0.9	2
Fear	1.2	2	1.2	5
Disgust	0.5	1	1.2	2
Sadness	0.5	1	0.9	1.1
Joy	0.9	1.1	1	1.3

Table 4.- Example of pattern matching model

The values presented in table 4 correspond to normalized values obtained by comparing a feature of a signal during the stimuli with the same feature during the baseline, as explained previously in equation (2).

The two values in each channel (CH1 and CH2) in figure 4 represent the ranges in the physiological signals that will characterize a particular emotion. For instance, the pattern of fear is specified in this model as 20% to 100% increase in the HR signal, and a 20% to 500% increase in the SC signal.

The pattern matching files can be easily created in a program like for example Microsoft Excel and saved with a .csv extension.

3.4.4.- Multi-category discrimination in real-time

The key to achieve effective and real-time emotion recognition is the ability of the system to incorporate information obtained from an offline process. In this way, it is possible to use machine

learning models for multi-category discrimination and use them for real-time classification.

Different types of neural networks were trained, wrapped into .dll and .h files and integrated in the system. In this way, BAGI calls trained neural networks and performs real-time classification.

Following this procedure new pattern recognition models can also be included in the system.

3.5.- Game adaptation

The adaptation module receives as an input the emotion provided by the classification module and matches the emotion with a predefined action in the game. After that it establishes a connection and, if required, instructs the game to perform any adjustment.

4.- EXPERIMENTAL VALIDATION OF BAGI

The validation of BAGI consisted in three stages. First the multi-category discrimination in real-time was tested. For this purposes an experiment was carried out to collect physiological and emotional data, to train our machine learning model and classify physiological signals in real-time.

In order to test the process of game adaptation, a computer application that customizes the computer desktop wallpaper according to the emotion of the user was developed.

Finally, in order to validate the flexibility of BAGI a series of black box tests were carried out to evaluate different: set of emotions, times to analyze the signals, physiological measures and pattern strategies.

The three stages of the validation are described in the following subsections.

4.1.- Real-Time Multi-category Discrimination

As a first step on the validation of the multi-category discrimination in real-time of BAGI, a series of experiments to gather physiological and emotional data were conducted. After that, machine learning models were trained and implemented to achieve real-time emotion classification.

The target emotions to be recognized by the system in this experiment included: relaxation, joy, sadness and fear. They were selected because each of the emotions are localized in one different quadrant in the circumplex model of emotions [26]. Figure 4 illustrates the target emotions in the valence and arousal dimensions.

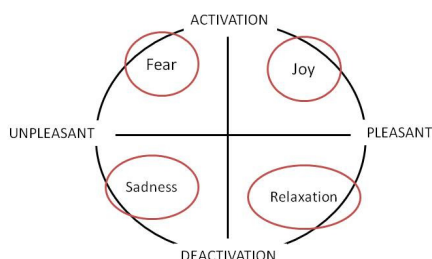


Figure 4.- Target emotions in valence and arousal dimensions

The physiological measures considered for this study included cardiovascular, electrodermal and respiration measures. From these measures a total of 17 signals were extracted using the Biograph Infiniti software.

To create a database of physiological responses to emotional stimuli, highly evocative video clips from movies were used.

Video clips have been widely used as stimuli in Psychology studies (e.g. [27, 28]) and in HCI (e.g. [13]). They have shown effectiveness to elicit emotional responses and offer good control for the experiment. Films also, have the desirable properties of being already categorized and being dynamic rather than static. There can be found several sources of information about movies and plenty websites where thousands of users vote and rate films and video clips from different genres (e.g. The Internet Movie Database, Yahoo movies, Youtube).

In order to select the most evocative video clips for the participants, the following procedure was undertaken. First, three types of sources that could provide titles of highly evocative video clips for our four target emotions were considered: i) previous studies; ii) websites with massive rates of video clips; and iii) recommendations from our research group and one media expert. Some criteria used for the selection include: self explanatory clips, highly evocative, short dialogues, length of clips. At the end, a corpus of 30 clips that fulfilled these conditions was gathered. Finally a pilot study was carried out with a group of 17 participants to select the best 12 video clips (3 clips for each target emotion).

4.1.1.- Experiment setup

At the beginning of the session the participant was briefed about the purpose of the experiment and signed a consent form. The electrodermal and cardiovascular sensors were placed in the fingers and the respiration sensor around the chest of the participant. During the experiment the participant was sat down in front of a desk with a questionnaire. Two minutes of baseline were recorded from the participant while he was watching a relaxing video, then 12 more video clips were projected in a 10ft by 8ft screen with the lights off. It was decided to use this big screen, instead of a common PC monitor, in order to get greater emotional responses. Studies recompiled in [29] support that more emotional responses are obtained when larger screens are used to present the stimuli.

The videos had an average of 3 minutes length each. After each video presentation the participants had 30 seconds to report his emotional experience in the questionnaire. The session lasted for an hour in total and there was just one participant per session. The session was video recorded to corroborate the emotions reported with the facial expressions of the participants. A total of 61 participants volunteered for this experiment.

4.1.2.- Feature selection

There were different motivations to carry out a feature selection process in our study. One of the most important was due to a restriction from the Biograph Infiniti software when connecting with third-party applications in real-time. In other words, the software that comes with the Procomp equipment, can create up to 8 connections through which only 8 signals can be passed from the Biograph Infiniti to the emotion recognition system. Having extracted a total of 17 signals for analysis, it was

necessary to select 8 signals that discriminate better the target emotions. Additionally, the feature selection process can improve the performance of the leaning model and recognition system by:

- Enhancing the generalization capabilities.
- Speeding up the learning and classification process.
- Improving the model interpretability.

The feature selection process was carried out offline, using a stepwise algorithm to select the best 8 features. Table 5 presents the 8 most important variables obtained from our analysis.

No.	Variable	Importance
1	Raw signal from BVP sensor	100.00
2	Amplitude from BVP	97.900
3	Respiration Period	81.486
4	Heart Rate	67.175
5	Raw signal from respiration sensor	34.112
6	Skin Resistance	23.702
7	Low Frequency % power	18.646
8	Ratio LF/HF	18.281

Table 5.- Variables obtained from feature selection analysis.

4.1.3.- Training results

With the selected variables, machine learning models were designed to automatically recognize the physiological patterns from the emotions experienced by participants. A supervised learning approach was followed to deduce a function from our training data and predict a class label. Two types of models were trained to recognize the patterns of the target emotions: i) Neural Network (NN) trained with a backpropagation algorithm and ii) Probabilistic Neural Network (PNN).

From the 61 participants some interruptions in the recording of the physiological data were observed in 2 participants. Although the information could be recovered and reconstructed, it was decided to discard it from the analysis to reduce any external bias.

A total of 708 training vectors were constructed from the 59 participants and 12 video clips considered for the analysis. The video clips were grouped in 4 categories, according to the target emotions, and there were 177 vectors for each category. Each vector contained the emotion reported by the participant and the normalized values for the 8 physiological signals presented in Table 6.

From the training process we achieved a total classification accuracy of 62.42% with the backpropagation NN and 84.46% with PNN. Tables 6 and 7 present the confusion matrices of the NN and the PNN.

Actual category	Predicted category			
	Relaxation	Joy	Sadness	Fear
Relaxation	100 (56.49%)	28	35	14
Joy	15	137 (77.4%)	17	8
Sadness	28	18	93 (52.54%)	38
Fear	13	13	39	112 (63.27%)

Table 6.- Confusion matrix for the backpropagation NN

Actual category	Predicted category			
	Relaxation	Joy	Sadness	Fear
Relaxation	151 (85.31%)	15	8	3
Joy	14	156 (88.13%)	3	4
Sadness	14	6	139 (78.53%)	18
Fear	5	3	17	152 (85.87%)

Table 7.- Confusion matrix for the PNN

In order to assess the generalization capabilities of our predictive models on new independent data sets, a leave one out cross validation was used. From the evaluation of the trained models a total classification accuracy of 54.94% on the NN trained with backpropagation and a 78.38% on the PNN was obtained. The confusion matrices of the validation of these two models are presented in tables 8 and 9.

Actual category	Predicted category			
	Relaxation	Joy	Sadness	Fear
Relaxation	102 (57.62%)	26	33	16
Joy	27	123 (69.49%)	16	11
Sadness	36	21	65 (36.72%)	55
Fear	17	15	46	99 (55.93%)

Table 8.- Confusion matrix for the validation of the backpropagation NN

Actual category	Predicted category			
	Relaxation	Joy	Sadness	Fear
Relaxation	143 (80.79%)	18	13	3
Joy	22	145 (81.92%)	5	5
Sadness	18	8	127 (71.75%)	24
Fear	8	8	21	140 (79.09%)

Table 9.- Confusion matrix for the validation of the PNN

4.2. Real-Time bio-affective feedback

For testing the real-time recognition and adaptation capabilities a very simple application that adapts the desktop wallpaper according to the emotion recognized by the system was built. The replay capability of Procomp was used to simulate the real-time acquisition of physiological signals from 59 participants of our experiment. Then, the BAGI was connected to the application that customizes the desktop wallpaper. Two types of models for real time classification were tested: one pattern matching model based on the findings from Ekman et al. [17] and one machine learning model trained with the PNN. Both models were able to successfully classify physiological signals in real-time and adapt the computer wall paper to the emotion

recognized. Table 10 presents a description of the wallpapers assigned to each emotion and an example of BAGI performing real-time adaptation is presented figure 5.

Emotion recognized	Wallpaper colour
Relaxation	White wallpaper
Joy	Blue wallpaper
Sadness	Gray wallpaper
Fear	Yellow wallpaper

Table 10.- Customization of the wallpaper

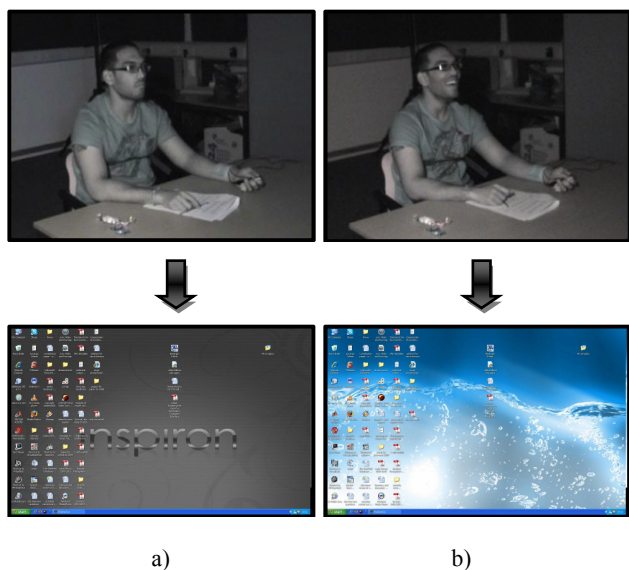


Figure 5.- a) participant at beginning of experiment; b) BAGI customizing wallpaper according to emotion

Figure 5 a) shows the default computer wallpaper when the participant is starting the experiment; figure 5 b) illustrates how BAGI customize the computer wallpaper as soon as it recognizes joy.

4.3.- Flexible Architecture

The flexibility of BAGI has been tested using the black box testing method. The detailed tests carried out on BAGI are presented in table 11.

Test	Details
Set of emotions	System tested with 3 different sets of emotions with different number of emotions and labels.
Time intervals	System tested using 3 different time intervals for the analysis of the signals: 10, 120 and 180 seconds.
Physiological measures	System tested using 3, 5 and 8 different physiological measures.
Pattern matching models	System tested with 3 different pattern matching models.
Machine learning models	System tested with 1 NN trained with backpropagation algorithm and 1 PNN.

Table 11.- Flexibility black box tests on BAGI

The results obtained from the black box tests demonstrated the flexibility of BAGI to work with different affective sets, time intervals, measures and psycho-physiological models.

5.- CONCLUSION AND FUTURE WORK

At the present time, when the game industry is looking for more immersive interactive techniques and ways to improve the artificial intelligence, affective recognition systems can play an important part as an instrument to enhance the game experience.

Physiological signals have been already used to recognize emotions in games; nevertheless the pattern matching approach commonly followed is not practical for the recognition among several emotions.

The feasibility of the use of physiological signals with machine learning techniques has already been proved by several offline HCI studies. However the implementation of such systems in a real-time application is not a straightforward task.

In this study a user independent real-time emotion recognition system that could be used for gaming was presented. The architecture of BAGI facilitates the reusability of previous findings and the integration of new models.

An experiment to gather emotional and physiological data was carried out with 61 participants. The results from the questionnaires showed that the video clips used as a stimuli effectively evoked the four target emotions in the most of the participants.

From the three sensors used to record the physiological signals a total of 17 signals were extracted. And due to the real-time connectivity limitations of the equipment with third-party computer applications the best 8 signals that discriminate our data were selected.

Two machine learning models were trained to recognize the physiological patterns of 4 emotions, having the best results with a PNN with classification accuracy of 84.46% on the training data and 78.38% when it was cross-validated. The two machine learning models were implemented on the system for a real time classification. Finally, a simple application that customizes the desktop wallpaper to the emotional state of the user was developed. The results from our study provide evidence of the feasibility of the use of BAGI as an interface for gaming.

Although the training models can be claimed to apply just to classify emotions experienced while watching video clips, the methodology used in this study can be used to train models for different types of stimuli. And the architecture of BAGI allows the inclusion of these new trained models to perform a real-time classification.

An interesting future study would be the utilization of a model trained with stimuli to classify emotional responses evoked by other type of stimuli (e.g. use a model trained with video clips to classify emotions evoked by pictures). There are already some differences in the physiological patterns, for the same emotions using different stimuli, in the psychology literature; however there is no empirical evidence in the literature of machine learning models attempting to classify emotions evoked by different types of stimuli.

Our future work will involve the training of a machine learning model in a game context and its inclusion to BAGI. Hopefully, with the affective feedback provided by BAGI it will be possible to reinforce the AI of the game and enhance the emotional experience.

ACKNOWLEDGEMENTS

J.A.P. would like to thank CONACYT and SEP for their financial support.

REFERENCES

- [1] Z. O. Toups, R. Graeber, A. Kerne, L. Tassinari, S. Berry, K. Overby, and M. Johnson, "A Design for Using Physiological Signals to Affect Team Game Play," in *Augmented Cognition International*, 2006.
- [2] H. Prendinger, C. Becker, and M. Ishizuka, "A study in Users' Physiological Response to an Empathic Interface Agent," *International Journal of Humanoid Robotics*, vol. 3, pp. 371-391, 2006.
- [3] K. M. Gilleade and A. Dix, "Affective Videogames and modes of Affective Gaming: Assist Me, Challenge Me, Emote Me," in *Digital Games Research Association (DiGRA)*, 2005.
- [4] C. Conati, "Probabilistic Assessment of User's Emotion in Educational Games," *Applied Artificial Intelligence*, vol. 16, pp. 555-575, 2002.
- [5] E. C. Lalor, S. P. Kelly, C. Finucane, R. Burke, R. Smith, R. B. Reilly, and G. McDarby, "Steady-State VEP-Based Brain-Computer Interface Control in an Immersive 3D Gaming Environment," *EURASIP Journal on Applied Signal Processing*, vol. 19, pp. 3156-3164, 2005.
- [6] S. I. Hjelm and C. Browall, "Brainball - using brain activity for cool competition," in *1st Nordic Conference on Human-Computer Interaction*, 2000.
- [7] M. Ko, K. Bae, G. Oh, and T. Ryu, "A Study on New Gameplay Based on Brain-Computer Interface," in *Digital Games Research Association (DiGRA)*, 2009.
- [8] J. Arroyo-Palacios and D. Romano, "Exploring the Use of a Respiratory-Computer Interface for Game Interaction," in *IEEE Consumer electronics Society Games innovation Conference*, London, 2009, pp. 154-159.
- [9] D. Bersak, G. McDarby, N. Augenblick, P. McDarby, M. Daragh, M. Brian, and R. Karkun, "Intelligent Biofeedback using an Immersive Competitive Environment," in *Designing Ubiquitous Computing Games Workshop*, Atlanta GA, 2001.
- [10] C. Peter and A. Herbon, "Emotion representation and physiology assignments in digital systems," *Interacting with Computers*, vol. 18, pp. 139-170, 2006.
- [11] J. Kim and E. Andre, "Emotion Recognition Based on Physiological Changes in Music Listening," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 2067-2083, 2008.
- [12] R. W. Picard, E. Vyzas, and J. Healey, "Toward Machine Emotional Intelligence: Analysis of Affective Physiological State," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 1175-1191, 2001.
- [13] F. Nasoz, K. Alvarez, C. L. Lisetti, and N. Finkelstein, "Emotion recognition from physiological signals using wireless sensors for presence technologies," *International Journal of Cognition, Technology and Work*, vol. 6, pp. 4-14, 2004.
- [14] J. T. Cacioppo and L. G. Tassinari, "Inferring Psychological Significance from Physiological Signals," *American Psychologist*, vol. 45, pp. 16-28, 1990.
- [15] D. Stemmler, "The autonomic differentiation of emotions revisited: Convergent and discriminant validation," *Psychophysiology*, pp. 617-632, 1989.
- [16] I. C. Christie, "Multivariate Discrimination of Emotion-Specific Autonomic Nervous System Activity," vol. MSc Thesis: Virginia Polytechnic Institute and State University, 2002.
- [17] P. Ekman, R. W. Levenson, and W. V. Friesen, "Autonomic nervous system activity distinguishes among emotions," *Science*, vol. 221, pp. 1208-1210, 1983.
- [18] B. L. Fredrickson, R. A. Mancuso, C. Branigan, and M. M. Tugade, "The Undoing Effect of Positive Emotions," *Motivation and Emotion*, vol. 24, pp. 237-258, 2000.
- [19] J. Arroyo-Palacios and D. Romano, "Towards a Standardization in the Use of Physiological Signals for Affective Recognition Systems " in *Measuring Behavior 2008*, Maastricht, The Netherlands, 2008, pp. 121-124.
- [20] R. B. Zajonc and D. N. McIntosh, "Some Promising Questions and Some Questionable Promises," *Psychological Science*, vol. 3, pp. 70-74, 1992.
- [21] J. T. Cacioppo, D. J. Klein, G. G. Berntson, and E. Hatfield, "The Psychophysiology of Emotion," in *Handbook of Emotions*, M. Lewis and J. M. Haviland, Eds., 1993, pp. 119-142.
- [22] T. Kim, D. Shin, and D. Shin, "Towards an Emotion Recognition System based on Biometrics," in *International Joint Conference on computational Sciences and Optimization*, 2009, pp. 656-659.
- [23] A. Haag, S. Goronzy, P. Schaich, and J. Williams, "Emotion Recognition Using Bio-sensors: First Steps towards an Automatic System," in *Lecture Notes in Computer Science. Affective Dialog Systems*, vol. 3068/2004: Springer Berlin / Heidelberg, 2004.
- [24] K. H. Kim, S. W. Bang, and S. R. Kim, "Emotion recognition system using short-term monitoring of physiological signals," *Medical & Biological Engineering & Computing*, vol. 42, pp. 419-427, 2004.
- [25] E. Leon, G. Clarke, V. Callaghan, and F. Sepulveda, "A user-independent real-time emotion recognition system for software agents in domestic environments," *Engineering Applications of Artificial Intelligence*, vol. 20, pp. 337-345, 2007.
- [26] J. A. Russell, "A circumplex model of affect," *Journal of Personality and Social Psychology*, vol. 39, pp. 1161-1178, 1980.
- [27] J. R. Averill, "Autonomic response patterns during sadness and mirth," *Psychophysiology*, vol. 5, pp. 399-414, 1969.
- [28] J. J. Gross and R. W. Levenson, "Emotion elicitation using films," *Cognition and Emotion*, vol. 9, pp. 87-108, 1995.
- [29] M. Lombard and T. Ditton, "At the Heart of It All: The Concept of Presence," *Journal of Computer Mediated Communication*, vol. 3, 1997.

Improving Games AI Performance using Grouped Hierarchical Level of Detail

David Osborne and Patrick Dickinson¹

Abstract. Computer games are increasingly making use of large environments; however, these are often only sparsely populated with autonomous agents. This is, in part, due to the computational cost of implementing behaviour functions for large numbers of agents. In this paper we present an optimisation based on *level of detail* which reduces the overhead of modelling group behaviours, and facilitates the population of an expansive game world.

We consider an environment which is inhabited by many distinct groups of agents. Each group itself comprises individual agents, which are organised using a hierarchical tree structure. Expanding and collapsing nodes within each tree allows the efficient dynamic abstraction of individuals, depending on their proximity to the player. Each branching level represents a different level of detail, and the system is designed to trade off computational performance against behavioural fidelity in a way which is both efficient and seamless to the player.

We have developed an implementation of this technique, and used it to evaluate the associated performance benefits. Our experiments indicate a significant potential reduction in processing time, with the update for the entire AI system taking less than 1% of the time required for the same number of agents without optimisation.

1 INTRODUCTION

Computer games AI is becoming increasingly complex, with players expecting increasingly realistic behaviour from non-player characters. At the same time, the size and detail of game environments has increased dramatically, with games such as *Crysis* [5] having effective view distances of over 8 kilometres. Often these environments are largely uninhabited, with the game agents centred around the player's location. If agents are present in other areas, they will sometimes use simplified behaviour models to reduce computational overheads. However, unless these are carefully constructed, problems can arise when an agent's simplified model is noticeably inconsistent with its full behaviour.

Increasingly sophisticated methods of performance optimisation have been investigated. For example, attempts have been made to dynamically adjust the amount of CPU time given to each agent [17]. Others have taken the concept of *level of detail* (LoD), a system used extensively in computer graphics [4, 16] to dynamically adjust the complexity of visual representation, and transposed it to AI behaviour functions.

For example, work by Brockington [1] presents a LoD system which was used in the commercial game *Neverwinter Nights*. This made use of reduced behaviour complexity, based on 5 LoDs, with

characters at the highest level using a complete set of combat and pathfinding algorithms. Those at lower LoDs had simplified versions of the same functions. For example, at the highest level, characters could use a full pathfinding system, whereas at the lowest level no pathfinding was used: agents were simply moved in a straight line to their destination with a time delay equivalent to the travel time. Whilst effective, Brockington's implementation is designed around one specific game environment: many of the pathfinding abstractions in his implementation rely on the structure of the levels within the *Neverwinter Nights* game engine.

Other methods have made use of hierarchical representations of behaviour models. Champandard's [3] Hierarchical State Machines (or HSMs) makes use of nested finite state machines. Each state of a machine could contain another machine or a set of child states: transitions from the parent state are effectively connected to all the sub-states, and transitions to the parent state can be redirected to the nested states. The key to HSMs is refinement and abstraction, with abstraction allowing for a parent state to be used as a regular state, rather than using the nested states or machines. Refinement allows for the states contained within the parent state to be used instead. This system applies LoDing to individual behaviours; however, designing and visualising the HSM is difficult as complex behaviour functions have to be decomposed into a set of simpler sub-states, while also providing appropriate behavioural elements at the parent level.

Brom *et al.* [2] take a comparable approach by applying hierarchical models to goal-based task representation. Non-player character behaviours are represented as tasks, which are further decomposed into atomic actions. Brom *et al.*'s proposed LoD system controls the detail with which individual actions are modelled and executed, and is coupled with an analogous LoD representation for the environments inhabited by those agents.

1.1 Level of Detail and Group Behaviours

Whilst the above methods have explored LoD type representations of the behaviour of individual game agents, our interest is the simulation of large numbers of agents acting in groups. Games have often made use of groups of agents working together, and one of the best known examples is Reynold's flocking algorithm [11]. Agents are controlled using a small number of steering behaviours that are applied individually, based on their neighbourhood, in order to create a believable emergent dynamic. Reynold's example of *bird* flocks, that is, flocks of artificial bird like creatures, made use of three very basic components (alignment, cohesion and separation). Later work [12] introduced other components for features such as path following and obstacle avoidance.

Other systems have been based on human behaviour. For example, van der Sterren presented two methods based on military com-

¹ University of Lincoln, Lincoln, email: david@digital-clouds.com, pdickinson@lincoln.ac.uk

bat. These compared emergent, decentralised command structures [13], where agents passed information to each other and acted on their local knowledge of the world, with centralised command [14] where one agent made decisions for an entire group. Both of these approaches provide good results: however, due to the high complexity of the behaviours, they are better suited to smaller sized groups.

Aside from military oriented models, a significant body of related work has developed around the simulation of human crowd behaviour. The objective of these systems is to create believable emergent group dynamics in everyday situations which are recognisable to the player or user. In most cases, such systems are comparable with Reynold's steering behaviours but employ considerably more complex behavioural models coupled with typically human environmental influences. Early work by Musse and Thalmann [6] considered the use of emotional and sociological models of human behaviour for simulating crowd activities in simple contexts, such as visiting a museum. Later work by Sung *et al.* [15] focussed on situation dependent behaviour functions, and scalability, whilst Pelechano *et al.* [9] developed a model of interaction which generates interesting and realistic emergent behaviours in queues and crowded spaces.

Works by Brockington, Champandard, and Brom *et al.* which considered the optimisation of behaviour functions for individual agents has already been mentioned. Similar approaches to the optimisation of crowd simulations have also been proposed, such as that of Pettre *et al.* [10] which used reduced update times for non-visible agents, combined with simplified individual behaviour functions. However, systems comprising group-based agents offer the potential to further consider both interactions between agents, and representations of the entire group. Some existing research has approached LoDing from this perspective.

For example, Niederberger and Gross [7], presented a hierarchy of agents in a group, where each individual was assigned to a discrete LoD value. This determined both how each agent's behaviour was implemented as an individual, and also how computational resources were distributed through the group. For example, an agent could opt to pass its scheduled computation resources to another, superior, agent in the hierarchy.

Work by O'Hara [8] is of particular interest to us. This considered the representation of a group of agents engaging in Flocking behaviour (using Reynolds steering behaviours [11]), and sought to reduce the number of entities required to maintain a persistent model. Agents with stable trajectories and similar spatial characteristics are dynamically clustered into sub-groups called *stable groups*. These were then abstracted to a single entity which represents the stable group's dynamic and spatial properties, removing the need to update each member individually. However, as stable groups are defined by their member's dynamic characteristics, the group structure of the entire flock must be validated and maintained. For example, stable groups were merged, and others split, in response to interactions with other groups.

1.2 Our Approach

Like O'Hara [8], the aim of our work is to optimise the simulation of flocking agents through the abstraction of individuals. However, we approach this more from the perspective of player perception and experience (as is the case for LoDing in graphics). Moreover, we consider the problem of maintaining an entire environment rather than a single flock: whilst O'Hara applied his system to a single group of hundreds of agents, our experiments have simulated hundreds of thousands of agents, organised into separate and distinct

groups occupying the same world. Our motivation is that this type of arrangement represents a likely structure for a large-scale game environment; for example, a sweeping plain populated by herds of different types of animals.

The novelty of our method derives from our approach to LoDing a group of agents. Existing approaches (like O'Hara's) use a bottom-up approach, and seek to reduce the representation required for a number of discrete agents acting in a group. Instead, we take a top-down approach in which we consider the group itself to be a single entity, and then add or remove agents as required to achieve an appropriate level of detail.

We have developed a hierarchical model to represent a group of agents engaged in flocking behaviour (using Reynolds steering behaviours [11]). Unlike O'Hara, who maintains adaptive clusters of agents, our abstraction is based on a predefined hierarchical tree structure. Leaf nodes in the tree represent the active group, and each branching node defines a LoD abstraction of its subtree. LoDing is implemented by simply expanding or collapsing individual nodes: in our experiments we limit the depth of the tree, but this is an arbitrary restriction. Our structure also affords the potential to optimise memory usage by deallocating collapsed nodes until required. Whilst we arguably do not maintain the same level of persistence as O'Hara, we are able to retain consistent behaviours for very large numbers of agents, and support seamless transitions through different LoDs by incrementally adjusting the representation of each subtree.

We proceed by presenting our hierarchical group structure in Section 2. In Section 3 we describe our test implementation, where we apply our approach to agents controlled using Reynolds flocking method. Sections 4 and 5 presents the results of our evaluation, and discussion.

2 GROUPED LEVEL OF DETAIL

The main concept behind our approach is that agents of the same type are often instantiated and managed in a distinct group. For example, a single herd of animals may comprise a large number of agents who act together. Within this context, we use LoDing as a means of controlling the representation of the group as a whole (rather than the representation of individuals with the group). We refer to this idea as *grouped level of detail*.

LoDing applied to a 3D graphical models is implemented by varying the number of polygons used to render that model. This may be implemented procedurally, or by using a set of pre-constructed models with the same key geometrical features. Either way, the model itself is considered to be a single entity, and the polygons are the building blocks used to construct it: in principle any number of polygons could be used to construct a particular model.

Our approach to AI grouped level of detail is analogous to the graphical case. We consider that the agents themselves represent the building blocks from which the group is constructed, and vary the complexity of representation of the group by varying the number of agents used to model it. For example, if we want to reduce the group complexity, a set of agents is dynamically removed from the group and replaced by a single agent which retains their spatial and dynamic characteristic. When more detail is required again, these *abstracted* agents are expanded back into the original set.

2.1 Balancing Performance and Representation

Consider a simple case where a group has only two pre-defined LoDs. In one case, the group may be represented with a full and

complete number of agents: this provides the maximum representational complexity, but also requires the highest processing resources to maintain the group's behaviour. In the second LoD, the entire group is represented by just a single entity which abstracts all of its constituent agents. This representation has minimal processing requirements, but only a very simple representation. However these LoDs are managed, transitions between them are likely to present inconsistencies to the player. A key point of our approach is to mitigate this effect by varying the number of agents in the group incrementally.

In order to achieve this we have used a hierarchical tree structure to represent the group, with nodes representing agents at various levels of abstraction. Our tree has a nominal maximum depth, such that leaf nodes at this depth effectively correspond to individual agents in the group. Higher level nodes represent incremental abstractions of their subtrees, and can be independently collapsed or expanded again in response to a requirement for less or more detail. For example, using a proximity-based LoD, subtrees representing agents which are closer to the player are expanded to a greater depth than those which are further away.

An example is shown in figure 1, where a LoD *membrane* passes through a representation of the fully expanded tree to show its current actual expansion. The dark coloured leaf nodes which lie along the surface of the membrane represent the whole group at its current LoD.

Only leaf nodes run full behaviour functions, and they are also used to visualise the entire group during rendering. Nodes below the membrane represent higher LoDs which are not currently instantiated. For example, in figure 1 there are currently 4 leaf nodes. The far left hand side is expanded to its highest level of detail, with 2 nodes at the lowest depth instantiated. These represent individual agents running full behaviour functions. The two adjacent agents have been abstracted to a lower LoD, and are modelled by a single node. The remaining four agents are at an even lower LoD, being represented by a single node on the far right-hand side. If more detail were required, this single node could be expanded back into either 2 or 4 nodes again.

2.2 Node Update and Management

The relationship between tree nodes and LoDs is expressed in the update procedure used for each node. Leaf nodes lying on the membrane surface receive a full update, using Reynold's flocking algorithm. These leaf nodes are the agents used to represent the group at its current LoD. Nodes below the membrane are currently abstracted out of the group, and effectively represent higher LoDs which may be utilised when required. These nodes are not updated.

Nodes above the membrane surface may be used to reduce the current LoD (by simply collapsing them as required). These receive a simplified update procedure which approximates the properties of their subtrees. In our implementation we simply update these nodes by averaging the dynamic attributes (position, velocity, etc.) of their children. This is effected using a recursive *bottom up* tree update procedure.

As mentioned, we use a player proximity test to determine when nodes should be expanded or collapsed. For example, when a node at a higher level of abstraction moves close enough to the player it is expanded into child nodes (where the number is determined by the branching factor), up to the maximum depth of the tree. Conversely, when the two leaf nodes moves far enough from the player to warrant an abstraction to a lower LoD, they are removed and their parent

becomes a new leaf.

We wish to maintain consistency in abstraction, as far as possible, and this raises the issue of how child nodes are best re-initialised when their parent is expanded. We have experimented with two methods: firstly, attributes of abstracted nodes are retained relative to their parent, and used for re-initialisation. This method is preferable in terms of persistence, especially where nodes are quickly expanded and collapsed, or smaller groups are used. However, it does also require that full storage for nodes is maintained down to the lowest depth.

Alternatively, nodes can be re-initialised randomly (within reasonable limits) relative to their parents. We have used this second method in our experiments as it allows for nodes to be deallocated when not in use, and thus utilises memory resources more efficiently. Note that the actual number of individual agents represented by a hierarchical group is given by:

$$Num. Agents = BranchingFactor^{Max. Depth} \quad (1)$$

The depth and branching factor directly affects how many nodes will be given a full update at each cycle. The optimal structure thus depends on several factors, such as the number of agents per group, and how the LoD system is controlled (e.g. whether or not it is based on player proximity).

3 IMPLEMENTATION

Our test implementation was developed in C++, and uses OpenGL to provide a simple graphical representation of the 2-dimensional game world. For each node undergoing a full update procedure we use the three base flocking behaviour components suggested by Reynolds [11], as well as an additional *wandering* behaviour [12]. The random wandering component is important as the three base behaviours utilise relationships between neighbouring nodes in the same group. At the lowest LoD (where the root node represents an entire group) these components are effectively abstracted out of existence. However, the wander component is an underlying behaviour which is retained at all levels.

Reynolds uses a physical model, where behaviours are modelled by applying forces to an agent. To implement this model, we model each node as a point mass with a fixed value of $m_a = 50Kg$. In addition, each node has persistent position and velocity attributes $\mathbf{p}_a$ and $\mathbf{v}_a$ respectively. Each of the four steering components is implemented as a physical force, which is calculated and applied to each leaf node at each update cycle. These are described as follows.

Alignment applies a force to change the heading of the node towards the average heading of its neighbours. The force is applied along the direction of the average heading, which is calculated as:

$$Average Heading = \frac{1}{n} \sum_{i=1}^n \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} \quad (2)$$

where n is the number of other nodes within its immediate neighbourhood. This neighbourhood is defined by a circle of radius N around the node. The velocity $\mathbf{v}_i$ is the velocity of node i in the neighbourhood. The heading represents a normalised target velocity direction.

Cohesion applies a force to steer a node towards the centre of its neighbours. This is implemented by first calculating the centre point, C , of the neighbourhood:

$$C = \frac{1}{n} \sum_{i=1}^n \mathbf{p}_i \quad (3)$$

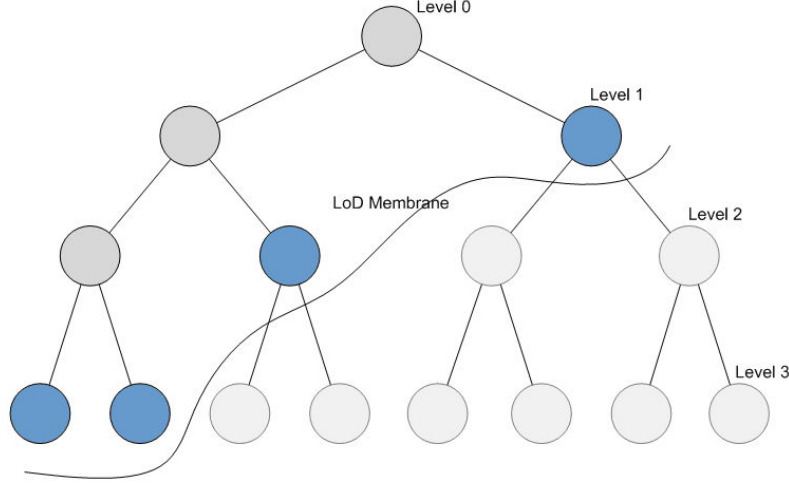


Figure 1. Hierarchical representation of a group of agents

where $\mathbf{p}_i$ is the position of node i . Once $\mathbf{C}$ has been calculated, a *seek* function is used to steer the node towards $\mathbf{C}$. This applies a force given by the following:

$$\text{Seek Force} = \frac{\mathbf{C} - \mathbf{p}_a}{\|\mathbf{C} - \mathbf{p}_a\|} v_{max} - \mathbf{v}_a \quad (4)$$

where v_{max} is the maximum speed.

Separation has an opposing effect, steering nodes away from very close neighbours. This force is given by the following:

$$\text{Separation Force} = \sum_{i=1}^n \frac{\mathbf{p}_a - \mathbf{p}_i}{\|\mathbf{p}_a - \mathbf{p}_i\|^2} \quad (5)$$

where $\mathbf{p}_i$ is the position of neighbour node i .

Wander is the final behaviour used, and provides a pseudo-random component. A force is applied to steer each node to a point $\mathbf{T}$ offset randomly along the perimeter of a circle of radius $\mathbf{w}_r$, projected a distance $\mathbf{w}_s$ ahead of the node's current position and heading (we used values of $\mathbf{w}_r = \mathbf{w}_s = 3$). The force applied is then given by:

$$\text{Wander Force} = \mathbf{T} - \mathbf{p}_a \quad (6)$$

Maximum force and velocity values were also used in order to prevent nodes moving too rapidly. These were set to 500 and 20 respectively. The full update function for a node is thus performed by calculating the steering forces, summing them, and applying them using the following dynamic update equations:

$$\mathbf{s}_a = \mathbf{v}_a t + \frac{1}{2} \frac{\text{TotalForce}}{m_a} t^2 \quad (7)$$

$$\mathbf{v}_a = \mathbf{v}_a + \frac{\text{TotalForce}}{m_a} t \quad (8)$$

where $\mathbf{s}_a$ is the change in position, and t the time since the last update. The magnitude of $\mathbf{s}_a$ is limited to a maximum based on the time since the last update, and $\mathbf{v}_a$ is initially set to the velocity calculated at the end of the previous update. This full 4-component update model is applied to each leaf node (along the membrane surface in

figure 1). Nodes above this level are updated by simply computing the average position and velocity values of their children.

We also made use of *prioritised dithering*, a method presented by Reynolds which uses probabilities to determine whether a particular force is applied on a particular update. Each force component is thus assigned an associated probability, and also a weighting factor, which can be used to fine-tune the behaviour functions. The values we used for these two additional parameters are given in Table 1.

Parameter	Value
Alignment Probability	0.4
Alignment Weight	20
Cohesion Probability	0.2
Cohesion Weight	50
Separation Probability	0.5
Separation Weight	2
Wander Probability	0.3
Wander Weight	20

Table 1. Prioritised Dithering Parameters

As mentioned, Reynold's flocking behaviour is used to update the current leaf nodes (that is, nodes lying along the expanded tree membrane). Nodes at higher levels of detail (below the membrane) receive no update as they are not instantiated. Nodes above the membrane (at lower LoDs) receive a reduced update procedure where their position and velocity attributes are simply calculated by averaging those of their children.

3.1 Controlling the Level of Detail

The LoD is controlled using a proximity based test, where the distance between the player and each node is used to adjust the corresponding level during each game update. We also experimented with a simpler implementation where we used only the distance between the player and the group root node. However, we found that managing each node separately gave a superior representation, with

partial expansions allowing smoother transitions between different group LoDs.

We set distances (from the player) which define the LoD for each subtree. Each of these defines a radial LoD zone around the player, and there is necessarily one zone for each possible depth level in the tree. Thus, if a leaf node moves into a zone which represents a higher LoD, then the node is expanded into child leaf nodes. As mentioned, these nodes are initialised randomly: in our implementation we used a maximum offset radius of 10M, and allowed the magnitude of the randomised velocity to be anything below the maximum defined for an agent.

Conversely, child nodes need to be collapsed to their parent if they move far enough away from the player. We detect this condition by examining the position of the parent of each leaf-pair. If, after the parent has been updated, it lies in the zone corresponding to its own LoD then the children are removed from the tree and the parent becomes a new leaf.

3.2 Running the Simulation

A software timer was used to call the AI update function every 500 milliseconds. This interval was chosen as a minimum which provided smooth behaviour for large numbers of agents. We also used a separate software timer to measure the amount of time actually spent on each AI update cycle: we used this to quantify our evaluation, presented in the section 4. Our implementation also used a single player entity, which could be moved using keyboard input, and we rendered a top down view of the 2-dimensional world inhabited by the agent groups. Note that other than the player and the agents, no other objects existed in the game environment. Figure 2 shows a visualisation of one rendered update frame from our LoD system whilst running.

4 EVALUATION

In order to test the performance of our system we undertook a series of experiments. These were executed on PC running the Windows 7 (32-bit) operating system, with a dual core 2.53GHz processor and 4GB of memory. The neighbourhood distance, N , used by the flocking algorithm was set to 20M, and the distance between successive LoD zone thresholds (from the player) was set at 50M. This means that, for example, groups whose nodes were all closer than 50M would be represented at their highest detail level by a fully expanded tree.

An experiment with a large number of agents and groups was used to compare the performance of our grouped LoD system with an analogous system that did not make use of optimisations. The LoDed system used a tree with a maximum depth of 5, and a branching factor of 2. This provided a simulation of 32 agents per group, and the test was run with 2048 groups giving a total of 65,536 agents. The experiment was run for 500 update cycles, and then the mean update times were taken for each test. Figure 3 shows the comparative results. As can be seen, there is a very significant reduction in update time when grouped hierarchical level of detail is used.

We ran a second experiment to determine how many agents could be reasonably represented while keeping the AI update time below 100 milliseconds. A similar set up to the previous test was used, with a maximum depth of 5 and a branching factor of 2 giving a group size of 32 agents. We started with 2048 groups in the simulation, and repeatedly doubled the number of groups up to 65,536 (2,097,152 agents). However, beyond 32,768 groups (1,048,576 agents) the pro-

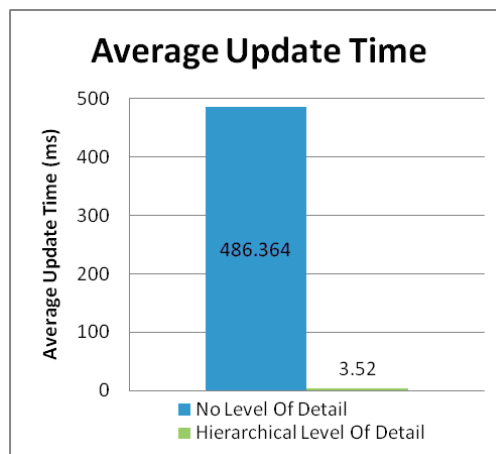


Figure 3. Average update time comparison.

gram became unstable due to lack of memory. The results of this test are shown in figure 4.

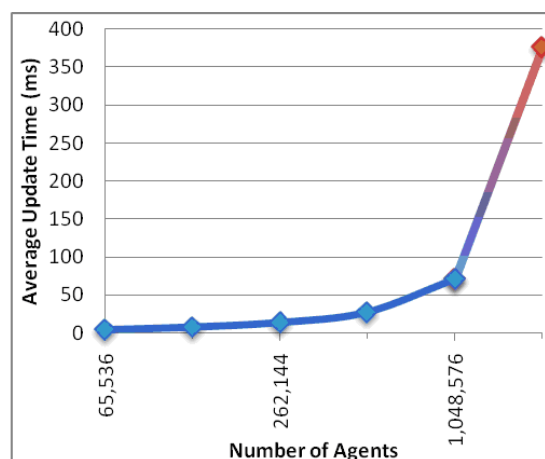


Figure 4. Update times for varying numbers of agents.

Whilst very large numbers of agents could be represented effectively, it is worth emphasising the point that system memory is a limiting factor. In practical applications we expect that the actual number of agents would be significantly lower.

5 CONCLUSION

We have proposed a group level of detail system, with the objective of achieving performance optimisations in systems simulating large numbers of agents organised into groups. Our implementation has integrated our approach with Reynolds' force-based steering behaviours.

Our experiments suggest that large performance gains can be achieved while maintaining a relatively seamless representation for the player. However, it should be noted that while the number of

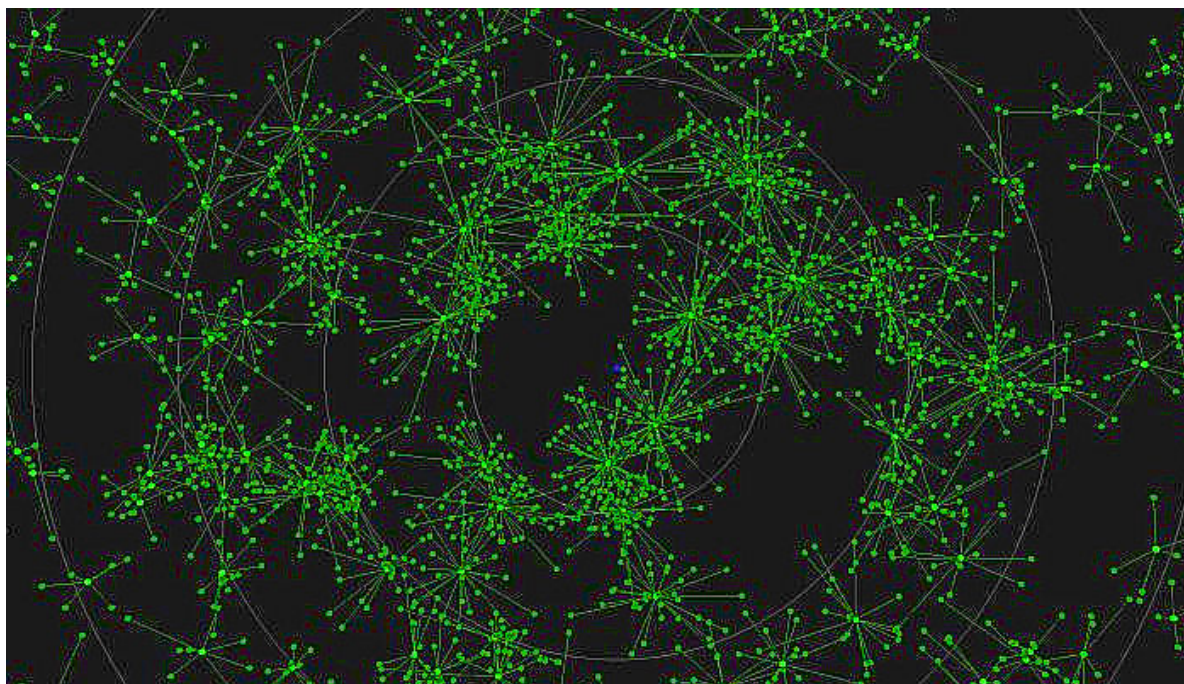


Figure 2. Visualisation of our implementation running. The central point of each cluster represents the root node of a group (its mean position). Each connected point is an instantiated child node running a full update.

agents simulated in our experiments is very high, our agents are using only simple steering behaviours, with minimal additional processing. Nevertheless, most games requiring advanced behaviours will not require anywhere near as many agents as we have used, and additional performance optimisation techniques could also be utilised, such as reducing the amount of update time scheduled for lower LoDs. One aspect of our work which we have not considered is that of the interaction between different groups: for example, how interactions between nodes at different LoDs might be effected. We leave this issue as future work.

REFERENCES

- [1] M. Brockington, "Level-Of-Detail AI for a Large Role-Playing Game," *AI Game Programming Wisdom*, Charles River Media, pp. 419-425, 2002.
- [2] C. Brom, O Sery and T. Poch, "Simulation Level of Detail for Virtual Humans," *Proceedings of the International Conference on Intelligent Virtual Agents*, pp. 1-14, Paris, 2005.
- [3] A. Champandard, "Hierarchical State Machines," *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviours*, New Riders Publishing, pp. 557-571, 2003.
- [4] J. H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, 19(10), pp. 547-544, 1976.
- [5] Crytek, "CryENGINE 2 - Specifications," [Online] 2007. [Cited: 19 December 2009.] <http://www.crytek.com/technology/cryengine-2/specifications>.
- [6] S. Musse and D. Thalmann, "A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis," *Workshop of Computer Animation and Simulation (Eurographics97)*, Budapest, 1997.
- [7] C. Niederberger and M. Gross, "Level-of-Detail for Cognitive real-time Characters," *The Visual Computer: International Journal of Computer Graphics*, 21(3), pp. 188-202, 2005.
- [8] N. O'Hara, "Hierarchical Imposters for the Flocking Algorithm," *Computer Graphics Forum*, 21(4), pp. 723-731, 2003.
- [9] N. Pelechano, J. Allbeck and N. Badler, "Controlling individual agents in high-density crowd simulation," *ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 99108, , 2007.
- [10] J. Pettre, P. de Heras Ciechowski, J. Mam, B. Yersin, J.-P. Laumond and D. Thalmann, "Real-time navigating crowds: Scalable simulation and rendering," *Computer Animation and Virtual Worlds*, 17(3) pp. 445455, 2006.
- [11] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioural Model," *Computer Graphics*, 21(4), pp. 25-34, 1987.
- [12] C. W. Reynolds, "Steering Behaviours for Autonomous Characters," *Games Developers Conference*, San Jose, CA, pp. 763-782, 1999.
- [13] W. van der Sterren, "Squad Tactics: Team AI and Emergent Maneuvers," *AI Game Programming Wisdom*, Charles River Media, pp. 233-246, 2002.
- [14] W. van der Sterren, "Squad Tactics: Planned Maneuvers," *AI Game Programming Wisdom*, Charles River Media, pp. 247-259, 2002.
- [15] M. Sung, M. Gleicher and Stephen Chenney, "Scalable Behaviors for Crowd Simulation," *EUROGRAPHICS* 23(3), 2004.
- [16] L. Williams, "Pyramidal Parametrics," *Computer Graphics, (Proc. Siggraph 83)*, 17(3), pp. 1-11, 1983.
- [17] I. Wright and J. Marshall, "Egocentric AI Processing for Computer Entertainment: A real-time Process Manager for Games," in *Proceeding of International Conference on Intelligent Games and Simulation*, pages 42-46, 2000.

Group Emotion Modelling and the Use of Middleware for Virtual Crowds in Video-Games

Olivier Szymanczyk¹, Grzegorz Cielniak¹

Abstract. In this paper we discuss the use of crowd simulation in video-games to augment their realism. Using previous works on emotion modelling and virtual crowds we define a game world in an urban context. To achieve that, we explore a biologically inspired human emotion model, investigate the formation of groups in crowds, and examine the use of physics middleware for crowds. Furthermore, we assess the realism and computational performance of the proposed approach. Our system runs at interactive frame-rate and can generate large crowds which demonstrate complex behaviour.

1 INTRODUCTION

Large virtual worlds with open-world game-play have been widely used for recent video-games, as they are often graphically impressive and help achieve an immersive experience. However, these genres of games have often been criticised for feeling lifeless. Particular examples of games which suffer from a lack of ‘life’ are “FarCry 2” [1], “Crysis” [2] and “Grand Theft Auto VI” [3]. One way of improving the experience of playing of these types of games could involve the simulation of virtual crowds. Implementing crowds in real and non-real time applications is a difficult and challenging task. One of the greatest challenges is the constant need for efficient variety management at every level, whether it is visualisation, motion control, animation, behaviour or sound [4]. Individuals in a crowd should look, move and react differently [4]. Even though the available computing power is constantly growing, it is still not possible to create real-time simulations filled with thousands of simulated agents necessary to achieve a realistic result. Current research of virtual crowds ranges from the visual diversity of simulated individuals [5], rendering [6], to the crowd behaviour generation [7, 8] and animation [9, 10]. Despite significant progress in these areas there are still several issues which need to be addressed (e.g. performance issues when simulating larger crowds with complex behaviour [7, 8], individuals passing through each other [10]).

In recent years, several solutions to develop virtual crowds to populate video-game worlds have been realised. For example, the video-game “Assassin’s Creed” [11] simulates crowded medieval cities during the third crusade in the Holy Land. The cities are filled with merchants, knights and peasants who fulfil daily routine activities (up to 150 onscreen characters [12]). The player can interact directly with the crowd. In return they generate realistic reactions as part of the game-

play. “MotoGP07” [13] also uses virtual crowds to simulate the crowd excitement of motorsport racing events. Even though the player does not interact with the crowd, the look and feel of the game is improved [13]. Although solutions for virtual crowds in video games exist [10, 12, 13], further research in the use of simulated crowds is required to achieve genuinely believable virtual worlds in video-games. Virtual crowds have already been widely used in other non video-games related applications. For example, crowd safety applications use real-time evacuation simulations to identify possible health and safety hazards. Another area where large virtual crowds have been used are non-real-time productions such as feature films including “Lord of the Rings” and “I, Robot” [14]. Virtual crowd solutions in those domains could be re-implemented in video-games situations. However, due to their high computational requirements [4], non real-time solutions have limited applicability to videogames.

In this work, we propose a method for simulating a real-time crowd in an urban context based on Millington’s goal-oriented algorithm [15] and investigate the feasibility of PhysX for behaviour modelling and animation. Furthermore, we suggest an approach to recreate the behaviour of groups in crowds. We demonstrate that with the proposed method up to 30,375 individuals that show complex behaviour within interactive frame-rate (>30FPS) can be simulated. Both qualitative and quantitative evaluation of the realism and performance of our simulation are presented and discussed. Our system is comparable to other real-time crowd simulations [8, 10]. This paper is structured as follows: we start with a review of related work in Section 2. Section 3 presents our virtual world including a method for crowd behaviour modelling. Section 4 details the implementation of our algorithm. Various metrics used for evaluation of our approach are presented in Section 5. Section 6 presents the results. The paper concludes with discussions about future work in Section 7.

2 RELATED WORK

Several methods for implementing virtual agents have been proposed [9, 10, 16, 17]. Reynolds [7] simulates a flock of birds. They navigate according to their perception of the environment, a set of predefined behaviour rules and the simulated physics that dictates a bird’s motion. The Sony Playstation 3 (PS3) multiprocessor architecture was recently used to further improve his approach [10] and a virtual crowd up to 15,000 agents at 60FPS was simulated. It was concluded that further hardware optimisations to generate larger crowds at interactive frame-rate were required and he proposed improvements in the collision avoidance approach to enhance the quality of his simulation.

¹ Lincoln School of Computer Science, University of Lincoln. LN6 7TS, UK. Email: {oszymanczyk, gcielniak} @lincoln.ac.uk



Figure 1. Our simulated world with 25,000 agents at interactive frame-rate (30FPS). Individuals and goals (e.g. shops) are rendered as textured 2d quads. The icons above each individual represent their current goal, icons above shops symbolises their offered actions and service.

Helbing’s Social Force Model [9] describes pedestrian dynamics as a force model. This model takes into account an individual’s acceleration force, possible repulsive effects, interactions with other individuals, boundaries and other possible attraction effects of pedestrian traffic. This approach has been implemented and tested. However some undesired features (unnatural shaking and vibrating) were reported [18].

A Cellular Automaton (CA) is an effective technique for modelling complex emergent collective behaviour of crowds [17]. The principle of CA is the division of the virtual world in a checkerboard pattern. Each cell can contain up to one individual, and contains localised neighbourhood rules of occupancy. The CA layers can be extended with sets which contain information such as behavioural rules of an individual that is occupying the cell. A crowd simulation using extendable layers CA was also developed [8]. It generates around 40,000 agents within interactive frame-rate on a 1.6 GHz computer with 2GB RAM. Despite CA models being simple and fast to implement, their disadvantage is the disallowance of individuals to come in direct contact with each other. Each agent can only move to a free adjacent cell. CA crowd are considered to have unrealistic animation which can only handle low to medium densities [18].

The use of a biologically inspired emotion model to generate realistic behaviour of virtual individuals has been widely discussed [16, 19, 20]. Emotions influence and dictate the way many decisions are taken, and how plans for the future are made by individuals. A framework for modelling emotionally driven context aware crowd behaviour was implemented in a train station simulation [18]. The conclusion argued that it is reasonable to assume that emotions widely increase the believability of simulated human behaviour. Delgado-Mata [16] also used an emotional model to simulate the behaviour of autonomous creatures. Agents communicated their emotions with help of artificial pheromones. He concluded that the use of emotions offer complex individual behaviour which in return offers greater believability. Emotion modelling has also been used in the popular video-game “The Sims”, in which a small

number of individuals use personality descriptors to drive their actions and goal selection [15]. Millington [15] developed a goal-oriented behaviour algorithm to implement this goal/action selection behaviour. Characters are presented with a range of possible actions and the algorithm chooses the one that best meets their immediate needs.

Recently, in the video-game industry, game developers have started to take advantages of the use of middleware physics engines such as Havok [21] or PhysX [22] to simulate in-game characters and objects. The advantages of middleware physics engines are their high computational performance which utilise multithreading and GPUs, and their accurate physics simulation [23]. For example, the video-game “Assassin’s Creed” uses Autodesk Maya’s Human Inverse Kinematics middleware for collision detection and character interaction with the environment [12].

In our approach we investigate the feasibility of emotion modelling (using Millington’s goal-oriented algorithm [15] presented in Section 3) and the usability of PhysX in a scenario that involves a larger number of in-game characters. Furthermore, we investigate the influence of groups (shared emotions between individuals) on the behaviour realism and computational performance.

3 THE VIRTUAL WORLD

The virtual world W consists of a set of N individuals I , M goals S and L actions A :

$$W = \{I, S, A\}, \quad (1)$$

where $I = \{I_1, \dots, I_N\}$, $S = \{S_1, \dots, S_M\}$ and $A = \{A_1, \dots, A_L\}$. Each Individual I_n is described by its physical state $\mathbf{p}$, the emotional state $\mathbf{e}$ and the current goal g :

$$I_N = \{\mathbf{p}, \mathbf{e}, g\}. \quad (2)$$

The physical state $\mathbf{p} = \{position, velocity, mass, shape\}$ describes the position, velocity, mass and shape of an individual. These quantities are used to model character movement.

The emotional state $\mathbf{e} = \{e_1, \dots, e_K\}$ represents the current values of character's mood (e.g. weights describing the current strength of different emotions). The emotional states are used for goal selection. The goal $g \in S$ is the current goal of an individual.

Each goal S_m is described by its physical state $\mathbf{q}$, its available actions $\mathbf{a}$ and a service limit γ :

$$S_m = \{\mathbf{q}, \mathbf{a} \subseteq A, \gamma\}. \quad (3)$$

The physical state $\mathbf{q} = \{position, influence_range\}$ describes the position and influence range of a goal. There is a limit γ on a number of individuals that a goal can service. Each action A can influence (positively or negatively) any number of available emotions $\mathbf{e}$. Each single action A_o is composed out of K amount of emotional influences μ :

$$A_o = \{\mu_1, \dots, \mu_K\}. \quad (4)$$

A goal is represented by an influence area in which a subset of all available actions can be taken (Eq. 3). For example, consider a kettle as a goal. The actions of a kettle are 'switch on/off, fill/pour water'. If a character's goal is to 'get boiling water', they need to go into the range of the kettle and use its available actions to achieve their goal. In the urban context of our virtual world, a goal is represented as a shop, which offers certain actions e.g. food related actions. Individuals in our virtual world are emotion driven. They eat when they're hungry and drink when they're thirsty. If individuals are hungry, their goal would be to go into the influence range of a shop which offers an action related to food and to perform this action to satisfy their hunger.

Action calculation	For $i = 1$ to L $d_i = \sum_{k=1}^P (\mathbf{e}_k + A_i(\mu_k))^2$ $bestaction = \arg \min(d_i)$
Goal calculation	For $j = 1$ to M If $(S_j \subseteq bestaction) \& \overline{\gamma_j}$ $g = S_j$

Algorithm 1. Discontent/Action/Goal selection algorithm for one individual I_n .

Algorithm 1 explains the goal-oriented algorithm for one individual I_n that result in the emotional driven behaviour. To find which action to take, an individual calculates which action is the best suited for his current set of emotions. The aim is to find the action which would reduce the discontent value as much as possible. An individual looks at all possible actions present in the world and predicts (using Algorithm 1) which one produces the lowest possible discontent value. This action is then considered as optimal. The final step for an individual is to find a goal which has not reached its service limit that offers this action.

When the current goal is selected, an individual turns and moves towards their selected goal. When a collision with other individuals occurs, the collision is resolved, and then the individual pursues their route towards the goal again. An individual takes the chosen action while passing through the goal's influence range. The more crowded the goal is, the longer it takes to complete the action. It simulates the complex behaviour of busy urban areas. When leaving the goal, the action is completed and the person's emotional state is updated. At this point the new goal is calculated using Algorithm 1.

3.1 Group Emotion Modelling

We introduce the concept of groups in our virtual world. In this context, group can be considered to contain either a small number of individuals (e.g. friends, family) or a larger category of people (e.g. participants of a larger event such as a concert, riot or demonstrations). Each member tends to have the same goal as other members of their group. If we split our virtual crowd into groups, we can allow one instance of Algorithm 1 to be applied between a subset of U individuals who are in the same social group. Each individual in the group has their emotions as equally weighted as other group members, thus allowing Algorithm 1 to be used once to calculate the next goal for all individuals in the same group. Individuals in a group do not share the same physical state. While using groups, the world W (Eq. 1) is redefined to contain V groups of individuals G :

$$W = \{G, S, A\}, \quad (5)$$

where $G = \{G_1, \dots, G_V\}$. A group G is composed out of U individuals who share the same goal g and emotional state $\mathbf{e}$:

$$G = \{I, g, \mathbf{e}\}, \quad (6)$$

and $I = \{I_1, \dots, I_U\}$. The presented concept can be supposed to mimic social groups. In addition the complexity of the algorithm is reduced.

4 IMPLEMENTATION

We implemented the virtual world in Visual Studio 2008 using C++ with the help of the GLUT libraries and PhysX (version 2.7.3). No manually coded optimisations, e.g. no multi-processor support, was utilised at this point however we did use the hardware support for PhysX. A simple rendering system for evaluation purposes was implemented. The object-oriented code allows every object of the virtual world W to be parameterised.

The physical state $\mathbf{p}$ of individuals I is handled by PhysX (NxBBoxShapeDesc class). The emotional state $\mathbf{e}$ consists of five different components which symbolise the value of strength of Hunger, Thirst, Health, Time and Bathroom. Their values are initialised randomly at start-up of the application. Upon initialisation, individuals are positioned in a square pattern around the virtual world.

The physical state $\mathbf{q}$ of goals S is also handled by PhysX. (NxBBoxShapeDesc class, flag NX_TRIGGER_ENABLE set). Each goal contains a randomly defined subset of available actions $\mathbf{a}$. Goals are positioned randomly around the virtual world. In our implementation we defined $M = 100$ goals S and $L = 23$ actions. The number of Individuals N was varied between 1

and 65,000. Actions A are related to the emotions e of individuals. They influence positively and negatively the emotional needs of hunger, thirst, health, time and bathroom. For example, the ‘get a large drink’ action greatly satisfies the thirst strength of an individual, but negatively affect the bathroom value. These values were manually generated.

The size of the virtual world was defined and rendered as a 2d surface of 500x500m. There are no physical urban obstacles (e.g. no cars, no buildings, no roads). Goals were rendered as 2d shops and individuals were rendered as 2d characters. Actions and the current goal of individuals were rendered as small 2d icons (see Figure 1)

5 EVALUATION METRICS

To assess the believability of our approach, we applied the following set of evaluation metrics. The first metric is based on the fact that there has been shown to be a linear relationship between the average speed and the flow of the pedestrian density in crowds [24]. The maximum speed of pedestrians is achieved when the density is at its lowest. As the density increases, individuals slow down to finally reach the point of congestion.

Levels of service were used to define the quality of the pedestrian flow. This metric details a person’s freedom to choose their speed, their possibilities to overtake, their ability to walk in the direction opposing the major flow and their capacity to manoeuvre without conflicts [24]. Seven different levels of service were defined: jammed, congested, crowded, constrained, impeded, unimpeded and open (see Figure 2).

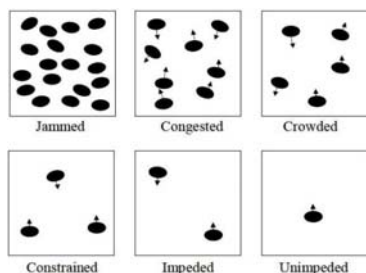


Figure 2. Illustrative depiction of the levels of service [24]. Each blob represents an individual with direction of movement depicted by arrows.

The formation of platoons [24] and the formation of lanes [9] is another observable feature of crowd behaviour. Platoons are temporary clusters of pedestrians behind slower groups e.g. people leaving a train. The formation of lanes occurs in crowds moving in opposite directions. In some circumstances it is possible to observe the formation of some of people moving in the same direction. The reason for lane formation is the related decrease in the frequency of required deceleration and avoidance manoeuvres, which increases the efficiency of the pedestrian flow [9].

6 EXPERIMENTS

Our objective is a qualitative and quantitative evaluation of our virtual world. To achieve this, a series of experiments were conducted involving different conditions while varying and controlling the most important system variables. We evaluate the

realism of the simulation using a set of metrics presented in Section 5 based on informal observation. Furthermore, we compare and evaluate the computational performances by measuring the frames per second (FPS) performance in different scenarios. The scenarios are: varying number of individuals, breakdown performance between different parts of the system and effect of group modelling on the performance. We also describe the effect of group modelling on crowd behaviour. The experiments were carried out on an Intel Core 2 Quad CPU Q6700@2.66GHz, 2048MB Ram, GeForce 8800 GT (256MB).

6.1 Crowd Realism

We observed a relationship between crowd density and average speed. The higher the density, the slower a persons is (see Figure 3). There are considerable differences between the estimated speed/density values from [24] and values obtained in our simulation. This is most probably caused by the crude shape of individuals in our system.

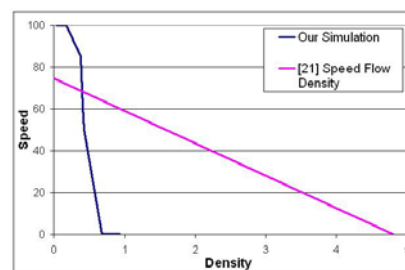


Figure 3. Average speed/number of people in an area (crowd density).

Table 1 compares the different levels of service with values obtained in our application. The overall tendency follows the values reported in [24]. However, results show that the denser an area is, the larger the difference between our system and the values specified in [24].

levels of service	Area/ped	avg speed (our world)	avg speed from [21]
Jammed	0.2 ~ 1.0	<40%	<70% of free flow
Congested	1.0 ~ 1.4	~ 60%	75% of free flow
Crowded	1.4 ~ 2.3	~ 75%	>80% of free flow
Constrained	2.3 ~ 3.7	~ 85%	Approaching free flow
Impeded	3.7 ~ 12	~ 99%	Virtually as chosen
Unimpeded	12 ~ 50	100%	As chosen
Open	>50	100%	As chosen

Table 1. Levels of service of the virtual crowd flow.

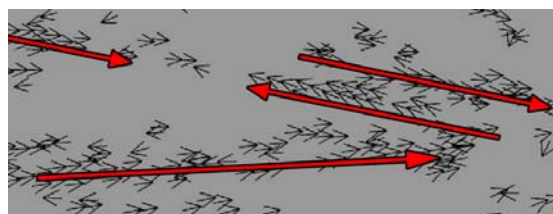


Figure 4. Examples of pedestrian flow (red arrow), each black arrow represents a pedestrian and their walking direction.

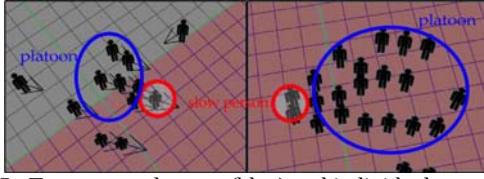


Figure 5. Temporary platoons (blue) and individuals causing it cause (red).

The formations of platoons and lanes can be observed in our application (see Figure 5). Figure 4 shows an example of the occurrence of pedestrian flow in our application

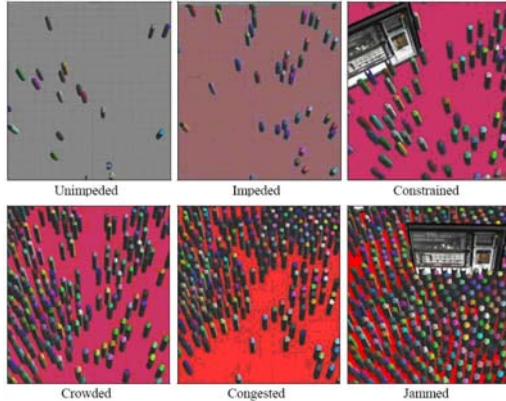


Figure 6. Levels of service in our application.

If we visually compare the levels of service in Figure 4 with the depicted illustration in Figure 2, similarities can be observed.

There is a difference of resulting behaviour between the world W defined by Eq. 1 (no groups) and Eq. 5 (group modelling). In the first case individuals fulfil their goal on their own and the formation of groups is temporary. In the second case, individuals stay in their attributed group. This behaviour is mostly visible with larger group. The distribution of individuals through of the simulated area is also influenced. A more uniform density is obtained when using smaller groups or the ‘no groups’ approach (see Figure 7).

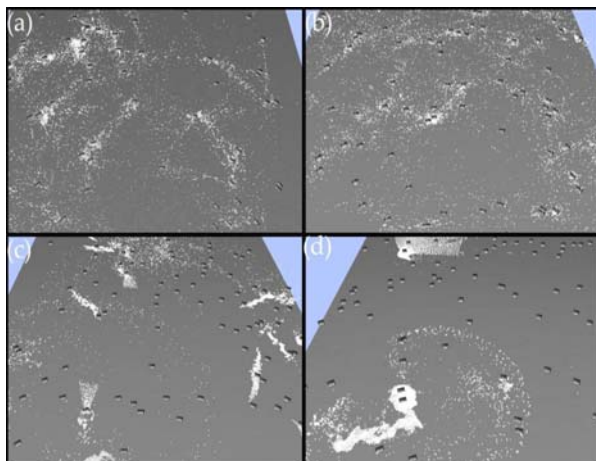


Figure 7. Our simulation after 10,000 iterations. (a) no groups (b) 50 individuals per group. (c) 500 individuals. (d) 15000 individuals.

6.2 Computational Performances

There are different computational costs related to different parts of the system (e.g. PhysX, Rendering, Emotions). For a crowd consisting of 25,000 individuals, the data collected showed that 52.07% of the update time is used to compute the discontent/action/goal algorithm, 18.35% is spent on rendering and 29.58% is used by PhysX.

Figure 8 shows the average performance of the application with respect to the number of simulated individuals (no groups, no rendering). The relationship between the number of individuals N and the performance of the application is exponential and indicates a possible bottleneck of the prototyped approach.

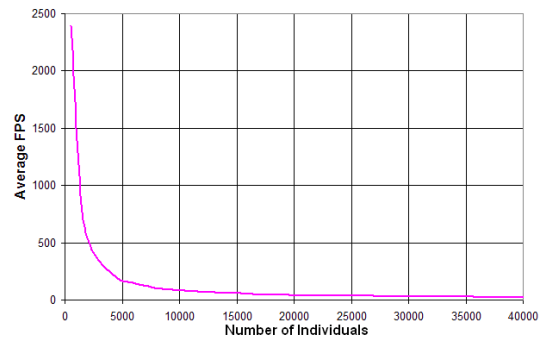


Figure 8. Average FPS performance with respect to the number of individuals N .

Figure 9 shows the computational performance between values of 60 to 30 FPS. There are different data sets representing, the computational effect of differently sized groups on the performance.

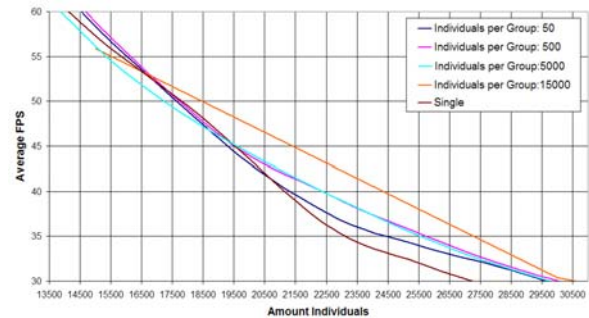


Figure 9. Performance of the application, no groups and groups of different sizes.

The FPS performance of the ‘no groups’ data set is the lowest. The maximum number of individuals the ‘no groups’ approach could handle on our test machine above 30FPS was 27,000. If groups are used, the performance gradually increases. The total number of individuals that can be simulated at interactive frame-rate increases as shown in Table 2. In other words, the larger the group size is, the higher the increase in performance.

Group size	Total individuals	Increase
50	29.500	7.62%
500	29.750	8.40%
5000	30.000	9.16%
15000	30.375	10.28%

Table 2. Performance increase when using different group sizes compared to no groups.

7 CONCLUSION AND FUTURE WORK

We have developed and evaluated a virtual crowd which uses a biologically inspired human emotion model, investigated the formation of groups, and examined the feasibility of PhysX in a large crowd simulation. The results demonstrate that PhysX [22] and Millington's goal-oriented algorithm [15] can be used to simulate a large virtual crowd. The formation of groups in real crowds was implemented by using the concept that emotions are shared between each member of the group. The use of groups improved the computational performance of the virtual crowd while also affecting the behavioural results of each individual. In our simulation, each person in a group tends to stay within close range of their attributed group, thus visibly reducing the spread of the crowd over the simulated virtual world. This behaviour was particularly visible in situations when larger groups (e.g. 15,000) of people were used. For a particular application, consideration of the group sizes should be made as they affect the behaviour. Larger groups of individuals can be used to model participants of a larger event whereas small groups can be used to simulate families or groups of friends.

On our test machine, we were able to generate up to 30,375 agents at interactive frame-rate. Although a precise comparison with other virtual crowd system is difficult (mainly due to hardware differences) our implementation offers similar performance results as virtual crowd applications used in [8, 10]. The qualitative evaluation showed that in a crowd of 25,000 individuals, 52% of the update time is used to compute the discontent/action/goal algorithm, 18% is spent on rendering and 29% is used on PhysX of the computational performances. The quantitative evaluation shows that our developed crowd demonstrated realistic crowd aspects such as the formation of lanes and platoons and the dependency between density and average speed.

In a future work, it would be interesting to investigate more complex models of group emotion. Currently, our application only allows groups of the same size to be used. Therefore, implementing the ability to vary the size of the groups would augment the realism of the application. The computational performance of the discontent/action/goal selection algorithm could be improved with the help of Millington's discussions about the computational performances of his algorithm [15]. Furthermore, it would also be interesting to apply a group model approach using different crowd behaviour and animation model. In addition, the virtual world could also be improved by adding an appropriate rendering system and complex urban elements (e.g. building, roads). Collision detection and response approaches could be improved as currently individuals seem to be 'pushing' other crowd members, giving the feeling that most individuals are busy, late or unmannered. More extensive metrics on crowd behaviour realism could be investigated, and the results used to further improve the system.

8 REFERENCES

- [1] Ubi Soft, FarCry 2, <http://farcry.uk.ubi.com/> (2008)
- [2] EA Games, Crysis, <http://www.ea.com/games/crysis> (2007)
- [3] RockStar Games, Grand Theft Auto IV <http://www.rockstargames.com/IV/> (2008)
- [4] Thalmann D. and Musse S. R., *Crowd Simulation*, 1st ed. Springer, (2007)
- [5] McDonnell, R., Larkin, M., Dobbyn, S., Collins, S., and O'Sullivan, C. Clone attack! Perception of crowd variety. In *ACM SIGGRAPH* pp 1-8 (2008)
- [6] Tecchia, F. and Chrysanthou, Y. Real-Time Rendering of Densely Populated Urban Environments. In *Proceedings of the Eurographics Workshop on Rendering Techniques* pp 83-88 (2000).
- [7] Reynolds, C. Flocks, herds and schools: A distributed behavioral model. In *Proceedings 14th annual conference on Computer graphics and interactive techniques*, pp 25 - 34 (1987)
- [8] Banerjee, B., Abukmail, A., and Kraemer, L. Advancing the Layered Approach to Agent-Based Crowd Simulation. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation* (2008).
- [9] Helbing D, Molnar P., Farkas I. J, and Bolay K. Self-organizing pedestrian movement. *Environment and Planning B*, 28:361-384, (2001)
- [10] Reynolds, C. 2006. Big fast crowds on PS3. In *Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames* (2006)
- [11] Ubi Soft, Assassin's Creed, <http://assassinscreed.uk.ubi.com/> (2006)
- [12] Monret, J. Code Hashashin Scimitar Engine ++ <http://ib0sum.blogspot.com/search?updated-max=2008-12-19T23%3A10%3A00-05%3A00> (2008)
- [13] Jefferies, D. Postmortem: Black Rock Studios' MotoGP'07. http://www.gamasutra.com/view/feature/1971/postmortem_black_rock_studios_.php (2007)
- [14] Kavan, L., Dobbyn, S., Collins, S., Žára, J., and O'Sullivan, C. Polypostors: 2D polygonal impostors for 3D crowds. In *Proceedings of the 2008 Symposium on interactive 3D Graphics and Games* pp 149-155 (2008)
- [15] Millington I., *Artificial Intelligence for Games* (The Morgan Kaufmann Series in Interactive 3D Technology), Morgan Kaufmann Publishers Inc., San Francisco, CA (2006)
- [16] Delgado-Mata C.. Emotion Signalling In Multiple Intelligent Virtual Agents for Believable Artificial Animals. PhD thesis, University of Salford (2004)
- [17] Adler, J. Cellular Automata Microsimulation for Modeling Bi-Directional Pedestrian. In *Transportation Research Journal Part B*, Vol. 35/3 pp 293-312 (2001)
- [18] Pelechano, N., Allbeck, J. M., and Badler, N. I.. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM Siggraph/Eurographics Symposium on Computer Animation* pp 99-108 (2007)
- [19] D. Isla, B. Blumberg. New Challenges for Character-based AI for Games. In *Proceedings of the AAAI Spring Symposium on AI and Interactive Entertainment*, Palo Alto, CA (2002)
- [20] Dey, P. and Roberts, D. A Conceptual Framework for Modelling Crowd Behaviour. In *Proceedings of the 11th IEEE international Symposium on Distributed Simulation and Real-Time Applications* pp 193-200 (2007).
- [21] Havok Inc, Havok Physics engine, <http://www.havok.com/> (2009)
- [22] Nvidia, PhysX, http://www.nvidia.com/object/PhysX_new.html (2009)
- [23] Ballanyntyne, R. Collision Overload: Reducing the Impact in Real-time Physics. <http://www.doc.ic.ac.uk/teaching/projects/Distinguished07/ianballanyntyne.pdf> (2007)
- [24] F. Feurtey, Simulating the collision avoidance Behavior of pedestrians, Master's thesis, University of Tokyo, Department of Electronic Engineering, (2000)

Digital Footprints – recording bot movement and behaviour to improve AI opponents in FPS games

Thomas Welsh¹⁾, David C. Moffat¹⁾ and Alastair Hebson²⁾

Abstract. To improve the experience for games players in FPS games, it is important that AI controlled bots are not only challenging and compelling opponents but also authentically human-like. This paper considers the use of an FPS visualization tool created by the authors called *Digital Footprints* to analyse and evaluate bot play behavior. This visualization tool gathers data and formats it as a series of heat maps which provide easy to understand diagrammatic representations of the events of a game session. By examining this data, the conclusion drawn is that current AI controlled bots exhibit key differences in their play behaviour to human players which reduces their believability and therefore diminishes the play experience for the player.

1 Introduction

Artificial intelligence in games is no longer a luxury; it is an aspect of game design which is integral to an engaging game experience[1]. With increasing visual fidelity available due to hardware advances in home consoles and PCs, more resources are available to developers and designers to create engaging behaviour for their artificial characters in their increasingly convincing virtual worlds [2]. If a game environment sets a level of expectation in the player which the virtual inhabitants of that environment cannot match then the artificial intelligence of the game will be the aspect which limits the experience and enjoyment of the player [3].

This paper concerns itself with the evaluation of the Artificial Intelligence applied in one genre of game, the first person perspective shooting game. First Person perspective Shooting games (FPSs) are amongst the most popular form of game on the PC[4]. In these games, the player does not see their on-screen avatar. The player simply sees the weapon they are carrying and a cross hair to represent where their weapon will shoot. They are able to walk forwards, walk backwards, turn, sidestep (strafe) and jump[1]. A number of different game rules can apply in an FPS game but generally the player's goal is to kill their opponent as many times as possible. Each kill, also known as a frag, adds to the players score. Typically, when a player is killed they will have to wait for a set period of time before being reborn (spawned).

FPSs can be online multiplayer games or narrative based games. Narrative based games often have a complex storyline which progresses as you play (e.g. Half Life 2) or in some cases very simple storylines (e.g. Doom). Narrative based games are usually one-player games where the computer controls the actions of all inhabitants of the game world other than the player. Online multiplayer FPSs can have computer-controlled inhabitants too, which can take the place of human players. These computer-controlled opponents are referred to as bots. At this point it is worth stating that the differentiation between online FPSs and narrative FPSs is much less well defined than in the past [5]. Games like

Halo 3 and Call of Duty 4 have a comprehensive single player narrative campaign as well as a robust multiplayer online component [6].

In an online FPS game, a typical rule set is a deathmatch. The term deathmatch in this instance merely refers to a specific set of rules that govern a battle between groups of players in an FPS game. Any of these players may be controlled by bots or be human players. In a deathmatch, the player's battle until an arbitrary number of frags is reached or a time allocation expires. The player with the most frags wins. The deathmatch rule set therefore provides a simple game structure into which we can place bots to evaluate their performance [7].

This paper looks at the bots available in the popular FPS game *Unreal Tournament 3*. These bots are evaluated using the *Digital Footprints* visualization tool which has been developed by one of the authors of this paper. *Digital Footprints* gathers data and creates visualizations on the major events of an FPS game, focusing on the movements and the actions of the players. The rich data that *Digital Footprints* can generate will be used to evaluate the differences between human and bot play behavior.

The importance of this research is that it will determine if the current state of single player AI in FPS games is sufficient to create an engaging play experience. Should the bots in FPS games exhibit behavior markedly different to human players, then the significance of these differences on play experience must be considered.

The first goal of this research therefore is to measure the current success of AI in FPS games to determine if improvements need to be made. It is assumed that the computer controlled AI will provide a less compelling games playing experience than facing human players and game flow will suffer as a result [8].

2 Methodology

The visualisation tool, *Digital Footprints*, has a number of possible uses, but one of the most important is its ability to record player data for both bots and humans to aid in the understanding of both. The tool records information on the movements and actions of game characters whether they are player avatars or computer controlled bots and provides valuable data which can be used to evaluate and inform the programming of AI controlled Bots.

In a typical FPS game the player views the action from the first person perspective. In this view the player sees what his in game avatar sees and as a result has a limit on the amount of information available. The player would not be able to determine where his opponents were or what actions they were taking unless they were within his or her line of sight.

Digital Footprints shows a plan view of the level that the players inhabits and thus provides an overview of where all players are and what they are doing. This in itself is of use for any research into game play behaviour as it gives the researcher a greater breadth of information than a single point of view of any one player and is also

¹ Glasgow Caledonian University

² Rockstar North, 1 Greenside Row, Edinburgh

easier to follow than watching the action from the viewpoint of multiple players.

For the purposes of this research, *Digital Footprints* was used in the *Unreal Tournament 3* game engine using the *Rising Sun* level. This map takes place “outside” meaning that there is no ceiling above the players and the game takes place on a flat plane with no multi-level buildings. Although *Digital Footprints* can be used on maps with multiple levels, a simple map was chosen in this case to make the heatmaps easier to read. Figure 1 shows the map using the game graphics and figure 2 shows the wireframe model of the map.

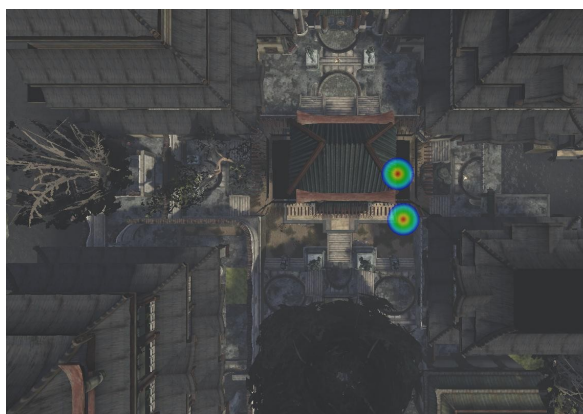


Figure 1. Map of *Rising Sun* level as rendered by game engine

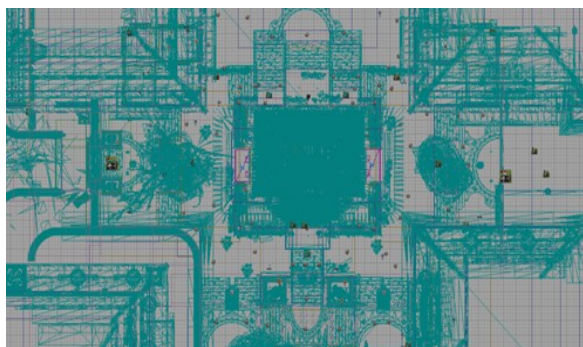


Figure 2. Map of *Rising Sun* level in wireframe

Beyond the relatively simple 2D overview provided by the application, the tool includes a number of additional functions intended to help analyse the data it gathers. It can highlight areas of a displayed map based on the amount of time spent in that area by the players. This heat map allows the user of the tool to determine the busiest parts of the map and the areas where either Bot or human players fail to explore. A simpler version of this type of heat map was used in the development of *Halo 3* [9]. In that case, the heatmaps showed only human player navigation through a level and was used to balance the difficulty and play test the single player campaign. Currently *Digital Footprints* is only compatible with *Unreal Tournament 3* but future versions could be adapted to work with multiple game titles.

If both human and Bot players avoid a particular part of the map then this may or may not be an acceptable outcome. If the level designer was to make use of this tool, they may decide that it is unacceptable to have parts of the map which are not explored often or at all. For the AI programmer however, it makes little difference if the

bots and humans avoid the same areas and is broadly indicative of a success on the part of the Bot.

However, if it is found that the Bots and the human players frequent different areas of the map then the Bot is in most likely failing to exhibit believable human-like AI. At the most basic level the Bots navigation path around the level is different to the humans and this ranks as one of the most obvious giveaways. The human player will quickly be able to determine the bot nature of their opponents merely by observing their geographical placement in the game world.

This visual shading of the overview map provided by the visualisation tool can be seen as a useful resource when developing and testing Bot AI. If sections of the shaded map for the Bot are significantly different from the human player versions then changes in the path finding or high level AI can be done to alleviate one of the most obvious giveaways exhibited by Bots. This represents the most basic functionality offered by the visualisation tool.

The visualisation tool also monitors the weapon use of all players in the game. Number of shots fired, accuracy, ammo used and collected and a number of other statistics are available as log files or can be overlaid directly onto the visualised map. Again, the comparison of these statistics can be done easily. Analysis of this data will provide insight into the human and bots behaviour after the game is completed. Additionally the diagrammatic view will in some cases provide direct and obvious differences in the behaviour of the Bots and humans. One example would be the number of shots fired. By examining the heat map, it will be very clear if the human controlled players are firing their weapons more often than the Bots or vice versa. This is another area of bot behavior which can be easily modified.

The visualisation tool will represent a key technique in gathering quantitative player data. This tool will enable a more complete model of human play behaviour to be built because of the volume of data it captures and the power it gives the researcher to query that data effectively. Gathering player data can be difficult and any method that can do so without explicitly asking the player for feedback is beneficial in games design [10].

The simple setup for the experiment was as follows. Ten human players each played one game session of the FPS game *Unreal Tournament 3* on the *Rising Sun* level for ten minutes against a single computer controlled opponent in a controlled environment. Data was extracted from each of these games using the *Digital Footprints mutator*. This mutator is a small program which modifies the game engine of *unreal tournament* and allows the *Digital Footprints* tool to analyse the data gathered after the game was complete. *Digital Footprints* itself is a separate application written in C# and run after the game has finished which processes the gathered data and generates diagrams and charts including heat maps. More details of the analysis as well as the results will be outlined in the following section.

3 Findings

The findings for this experiment focus on the area of navigation. These findings represent the distribution of time by the human players and their bot counterparts spent in each part of the level. Put simply, this is how much time the players spent in each area of the map. This is represented by *Digital Footprints* as a Heat Map.

Figure 3 is a detailed heat map of a human player's location data in the level.

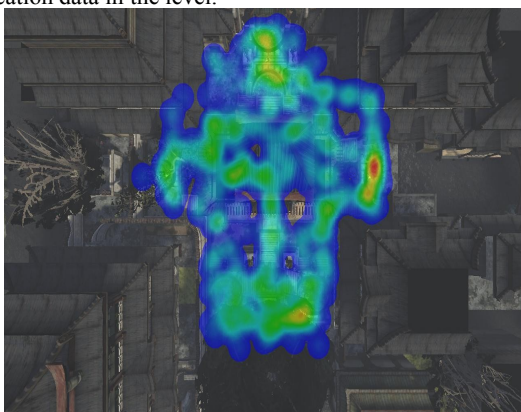


Figure 3. Heat map of human player navigation

This can be compared to a bot heatmap of location data as shown in figure 4.

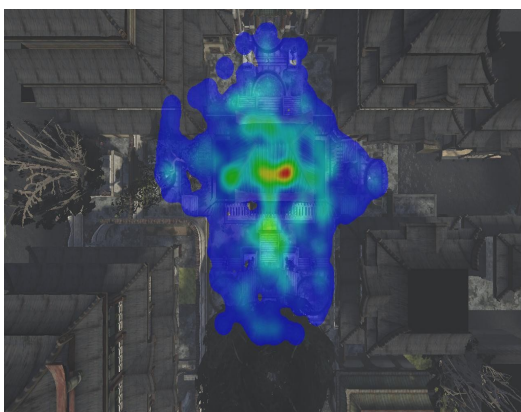


Figure 4. Heat map of bots navigation

Cursory examination of the two heatmaps shows obvious differences. This in itself provides little insight as variations between players would be expected regardless of whether they were human or bot controlled. Despite this, more careful examination of all of the heat maps shows that regardless of their opponent, all of the bot heat maps show significant similarities. The human heat maps meanwhile show more variation, but can similarly be logically grouped together based on appearance. It would be repetitious to print all heat maps in this paper, but further examples below in figure 3 and 4 show the continued similarities.



Figure 5. Additional heat map of human navigation

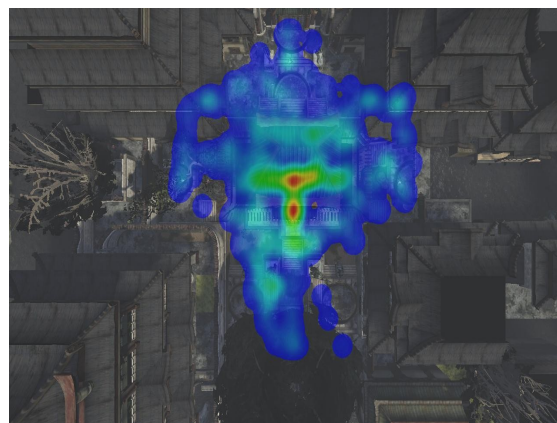


Figure 6. Additional heat map of bots navigation

Merely looking at the heatmaps provides useful information on bot and human play behaviour. A more robust method of differentiating and interpreting the data is necessary however. It is important to determine whether the differences are observable only to the authors of this paper. To eliminate any possible bias, impartial observers were asked to help.

To do this, ten people were asked to group the heatmaps based on their basic structure and appearance into two sets. Each individual would examine the ten heatmaps for human players and the ten for bots from the games that were played and split them into type one or type two. The people asked to help did not know anything about the heatmaps and most were not even games players.

Of the ten people asked to do this, nine of them were able to correctly group all of the heatmaps into the correct categories of bot and human. This is a significant and is provided here as evidence to support the observations made by the authors.

This method of determining whether bot and human heatmaps are noticeably different is quite primitive. Future work, discussed in more detail later, will consider an algorithm which could be used to automate this process.

It can be seen that from examination of the heat map generated from a game alone we could determine whether the data was for a human or bot player. This supports previous work which shows that human players can easily differentiate between human and bot opponents [3]. In this paper by Welsh, navigation was highlighted as one of the most important areas of differentiation between humans and bots with many players surveyed mentioning how disappointing the movements of the bots in the game were.

Looking at the heat maps in more detail, it can be seen that there are areas of the map that the bot never travels through. Additionally, the bot spends more time around the centre of the map and generally moves around less than human players.

4 Exposition and conclusion

This work represents an initial attempt to use the software tool *Digital Footprints* to determine if it can be used to discover meaningful data that can be used to improve bot behavior. From the data gathered we can see that there are noticeable differences between human and bot play behavior and these differences include the way that bots and human navigate through a level. This work focuses

on only one area: navigation. Despite this it nonetheless provides clear evidence showing the large disparity between human and bot behavior.

To consider the usefulness of these findings we must consider how they could be used to improve bot behavior. One possibility is that at the simplest level, the bot paths used by the bots could be modified. These results show that the bots do not move through the level in the same way that human players do. By using the heatmaps for human players as a reference, we could adjust these bot paths.

Bots navigate through a level based upon a series of nodes set out by a level designer. This experiment has shown that these bot paths are not representative of human navigation. It is possible therefore those substantial improvements in bot believability could be achieved by using human generated heat maps to influence bot navigation. This could be done simply by adjusting the navigation points of the bots with the software and development tools included with the game. This could be performed offline as part of the game design process or even online as part of a learning algorithm in a more complex client-side bot as considered in other work by Spronk et al [11].

5. Further Work

This represents only the beginning of the functionality offered by *Digital Footprints* utilising only the simplest of the functions provided by the software. By examining only the navigation of the players we have already gathered useful information on human and bot player behavior. By expanding the data gathered to include weapon choice, kill and death locations and weapon trajectories it will be possible to more fully capture human play behavior and gain a more complete understanding of how to replicate this behavior in bots.

There are a number of different directions this work should be taken in the future. Firstly, this tool has shown that it is effective in capturing human player behavior. As has been argued, human play behavior is currently poorly understood and only by carefully studying that behavior can we attempt to capture it in bot play behaviour [3].

This paper has also shown a key difference between human and bot play behavior: navigation. In this instance the way that human and bot players move through the game environment is very different. This could be one key reason why bots are often perceived as poorer opponents than humans [7]. By examining heat maps generated by human players it should be possible to create bot paths which more closely correspond to human player's movements. This process could be strengthened further by automating it with an algorithm which would analyse heat maps and differentiate bot and human behaviour. A tool which differentiated between humans and bot could be a key tool in testing the performance of bot and a useful interface for tool for developers. Furthermore, by aggregating many human players' heat map data during the design stage of a level it should be possible to test and refine bot behavior. A useful experiment would be to try this and create bot paths based upon such heat maps. The result of this could be more convincing and believable bots.

Digital Footprints is a versatile and powerful tool. With creativity and adaptation it can be used in a number of ways in AI design, game design and testing. In this case it has been used to successfully demonstrate key differences in human and bot play behavior. In

highlighting navigation as a problem area, it has also given an indication of where AI developers and programmers may look to improve the bots that they create in the future.

REFERENCES

- [1] LAIRD, A. 2000, "It knows what Your Going to do: Adding anticipation to a QuakeBot" *Papers from the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Technical Report SS-00-02, AAAI Press.
- [2] MCGLINCHEY, STEPHEN. 2002, "Learning of AI Players from Game Observation Data", *Proceedings of CGAIDE 2004*, University of Wolverhampton, UK
- [3] WELSH, THOMAS. 2005, "Parameterisation of human FPS strategies", *CGAIMS 2005 conference proceedings*, Louisville, Kentucky
- [4] ELSPA, 2007. "Top selling PC games of 2007", *Chart track sales report*, <http://www.elspa.com/?l=charttrack>
- [5] DAVIES, N.P. MEHDI, Q.H. AND GOUGH, N. (2005), "Creating and Visualising an Intelligent NPC using Game Engines and AI Tools", *19th European Conference on Modeling And Simulation conference proceedings*, University of Wolverhampton. UK
- [6] WIXON AND PAGULAYAN. 2008, "Halo 3: the theory and practice of a research-design partnership", *ACM That's Entertainment Journal* Vol 15 No1 p52-55
- [7] LAIRD, J. VAN LENT, MICHAEL. 2001, "Human-Level AI's Killer Application: Interactive Computer Games", *Proceedings of AAAI 2000*, Austin, Texas
- [8] CSIKSZENTMIHALY, MIHALYI. 1991, "Flow: The Psychology of Optimal Experience", ISBN 0060920432
- [9] THOMSON, CLIVE. 2007, "Halo 3: How Microsoft Labs Invented a New Science of Play", online at http://www.wired.com/gaming/virtualworlds/magazine/15-09/ff_halo
- [10] SYKES, J. BROWN, S. 2003, "Affective Gaming: Measuring Emotion Through the Gamepad", *Published at CHI New Horizons 2003*, Fort Lauderdale, Florida
- [11] SPRONK, PIETER, SPRINKHUIZEN-KUYPER, IDA, POSTMA ERIC. 2003, "Online Adaption of Game Opponent AI with Dynamic Scripting", *International Journal of Intelligent Games and Simulation*, Vol 3 No 1, p45-53

Procedural Content Generation and Level Design for Computer Games

Simon Cooper¹, Prof. Abdenmour El Rhalibi¹, Prof. Madjid Merabti¹ and Jon Wetherall²

Abstract. Procedural methods show great potential but are an underused solution to manual content creation. Limitations to these methods include the lack of control of the output due to its random nature and the absence of integrated solutions, although more recent publications increasingly address these issues. This paper describes procedural methods applied to terrain modelling, including the variable realism of their output, performance and control users can exercise over the procedure. In this preliminary paper, we focus on the techniques and processes we used to create a web based tool in Java that quickly creates and exports a variety of stylized 3D computer game levels using procedural content generation with random and user selected constraints.

1 INTRODUCTION

Computer game development has moved on a long way in the last 20-30 years. As gaming machines become more and more powerful the average development time has significantly increased.

A large portion of this time is used to create sprawling levels with highly detailed assets, requiring game development teams to have over a hundred full-time people, including not only dozens of programmers but also equal numbers of artists and level designers. These assets also require large amounts of storage space on optical media, with read breaks in gameplay to stream new content in to the game engine or an installation to a local mass storage drive with greater memory access to counter this delay.

A way to reduce these issues would be to procedurally create assets programmatically or creatively reuse a small number of predefined assets and using recursive algorithms, Artificial Intelligence, rule systems, Iterated Function Systems, noise generation, random or pseudo-random processes and shape transformations in a manner such as to avoid obvious pattern repetition.

Our goal is to create a system that can procedurally generate content for a game level, using either random constraints or those set by a user before the generator is started. The final level should then be exportable to a generic file format, which could be loaded in at a later time, even in another game engine or programming language. The software we develop will be interoperable to be distributed on a variety of platforms including modern web browsers and multiple operating systems and system architectures; including Google Android in the future. Once completed the application will allow a

game designer to set up parameters and initial constraints for procedural world generation and then define agent AI rules. The agent AIs (think of them as robots) will then adjust the game world, place pickups, objects, and setup game play using Rules imposed by the game designers.

The remaining chapters are as follows: In section 2 we discuss related work. In section 3 the techniques to generate the environment in particular terrain, plants and water. In section 4 we discuss possible algorithms to build road networks using L-systems or A* pathfinding. In section 5 we present the techniques based on shape grammars, developed to design city environment with procedurally generated buildings, textures and arranged primitives. In section 6 we briefly introduce the implementation with Homura (the engine we have used to develop the applications), the interface and the performance of the application. In section 7 we conclude the paper and highlight future work to be undertaken.

2 RELATED WORK

Other researchers have tackled procedurally generated game environments with varying degrees of success. The most well known and most significant research has been carried out on the CityEngine [1], a system that is capable of producing realistic and detailed models. The generation algorithms are inspired by the modelling of natural phenomena in string grammars [2] and L-systems are used to construct road networks and buildings [1]. Procedural building generation has focused on the application of grammars to describe structure, like the Shape Grammars original proposed by [3]. These have been applied in various guises to the construction of building geometry [4].

In [5], the authors define a specific grammar to characterise building structure. This system defines rules to operate on shapes and can be used to design a range of detailed buildings in several architectural styles. Recently this approach has been extended to employ imaging techniques to aid in the acquisition of generation rules from existing building façades [6; 7].

Other approaches include the application of intelligent agents [8], real-time frustum filling [9] and template based generation [10]. In [11; 8] the authors propose to simulate the evolution of cities by modelling land use and evolving a city usage map over time that can be used later to create a cityscape. Real-time city generation has also been attempted.

In [9; 11] the authors implement a city generation system that fills the view frustum rapidly with buildings of various shape combinations but their placement is restricted to a road network consisting of a regular grid. In [10] the authors propose the application of templates that encapsulate patterns such as raster radial to generate road networks.

Compared to these techniques our approach introduces several novel aspects including: interoperable

¹ School of Computing & Mathematical Sciences, Liverpool John Moores University, Byrom Street, Liverpool, L3 3AF, UK. Email: s.cooper@2003.ljmu.ac.uk, {a.elrhalibi, m.merabti}@ljmu.ac.uk.

² Onteca, Toxteth TV, 37-45 Windsor Street, Liverpool, L8 1XE, UK. Email: jon@onteca.com.

solution development allowing deployment across many platforms including web browsers, multiple OS and the Google Android platform; instant in-game generation, rendering and recreation of 3D tile based and organic environments; agent AI Bots to terraform the environment after the generation stage and according to other user-set constraints, and structural constraints induced by the game world; and full control of the environment generation by the user, as well as semi random generation options.

3 ENVIRONMENT

To generate a natural environment procedurally we use a variety of techniques, which are broken down into the sections presented below. These sections are the basic parts of a natural world and provide the building methods for land mass/terrain, seas, rivers & streams, and trees/plants.

3.1 Terrain & Midpoint displacement fractals

The terrain is created first, and determines where everything else can be placed, as every other part must be built up on top of it.

To create a random terrain we will use the midpoint displacement algorithm to create fractal height maps. Four corner points are chosen from the map grid and assigned random colour values between 0 and 1. The square is then subdivided equally into four equal squares and the midpoints of each edge are given a colour value equal to the average value of the two adjacent points (Figure 1). The centre point then gets its colour value from all four corners averaged with the addition of a random offset value, which is scaled by the size of the grid and a roughness variable.

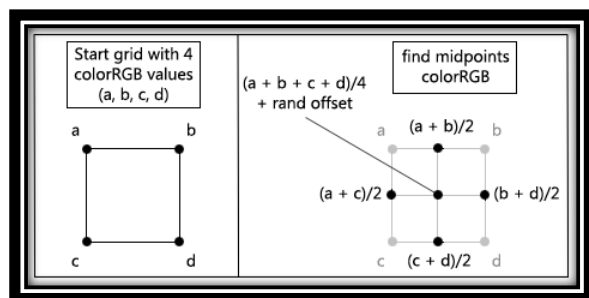


Figure 1 Midpoint Displacement Algorithm

The higher this roughness variable the more dramatic the peaks and troughs will be. This process is repeated to divide recursively until the smallest divisor/unit size is reached, which in this case is 1 pixel. This height map is then output as a black and white image with a grid size of 128x128. These numbers/pixel colours correspond to the terrain height at point [x, y] (or [x, z] in our 3d game world). The colour values are a percentage of the maximum height of land, with black representing 0% height - the lowest points and white representing 100% height - the high points.

3.2 Water

We split water creation up into the following three parts:

- The general flat water level created by the sea.

- The edge detection between sea and land creating the shore line/shallows.
- The generation of rivers and streams, which need to be routed around the map in a natural way.

3.2.1 Sea and Shoreline

For the sea we create a water level plane; the generated height map for the land mass can determine which parts of the map are above water level and which are submerged. The sea level depends on the range of terrain heights and a water level variable between 0 and 5, which divides the range of terrain heights into five with zero being no water, five being flooded and the values in between setting the level at 1-4 fifths of the max height.

Once the land and sea are created we can add a shoreline of lighter water with an animated tide texture. To implement this we use textured quads placed just above the sea quad. An algorithm scans across each tile in the height map grid that is equal or less than sea level, performing a four-way check to see the surrounding tiles are above sea level. If one direction passes the tile is added to the checked list and not tested further. If the unchecked list has remaining tiles they are checked off one by one until it is empty. Next for each tile in the checked list a new quad will be created at the same coordinates with the direction of the shore tiles tide pattern determined by the sides that are touching land.

3.2.2 Rivers & streams

Rivers can be harder to create dynamically as their path heavily depends on the terrain. We can start with the assumptions that:

- Rivers will flow into a larger body of water (in our case the sea or in-land pools created by creators in our terrain) or off the screen if there are none present.
- Rivers will flow downhill due to gravity usually taking the easiest path.

Using these rules we can generate a random start and end point for the river, either on one edge of the map, in the sea or in a lake pool. Next we can set an A* pathfinding algorithm to compute the route between these two points, using a custom 'cost heuristic', similar to a possible technique for road generation (see section 4.2 for more detail). The heuristic will look for the most cost effective path, with steps taken downhill being cheaper.

3.3 Trees & plants

To add more features to our map we add trees and plants. The problem is defining where to place them in a pattern that looks natural. If the trees and plants are scattered using 'n' randomly generated [x, z] positions within our grid the results can look "neither random nor natural".[12] Due to the way trees and plants grow: from seeds falling; to growing and taking in nutrients, water, CO₂, light and competing with other plants, their scattering pattern can look more complicated. A simple model for this pattern would include each plant having an exclusion zone around it that no other plants can occupy. Plants and trees can be arranged in a grid with an equal distance between them before a small random offset is applied to scatter them. This offset must also be small enough to stop two trees overlapping. More variation can be added later by randomly rotating and scaling the trees

(including their exclusion zones), as some trees will be more fully-grown than others.

4 ROADS

Once the land is created it can be populated with an urban city structure. The first main step is to divide up the level into lots (smaller areas that can be used to place buildings and other man made areas such as parks or town squares) by forming a road network. There are two main structural layouts for the roads, which are dependent on the area being in an urban or rural setting.

For a big city the grid plan road structure is most fitting, but in rural areas the streets need to be non-uniform (smaller, more winding and with further separation). As the land and natural features are determined first, the height map from terrain generation can be used to control where the roads can and cannot be built. Features like water and hill gradients can be factored in to exclude roads or define points for bridges and tunnels.

There are two useful techniques we could use to generate the roads: L-systems and A* Pathfinding. These could be used individually to give different results or be used together in passes to create roads more intelligently.

4.1 L-Systems

An L-system or Lindenmayer system is a parallel rewriting system using string rewrites, where each symbol is interoperated as a rule step. They can be used to generate self-similar fractals. An L-system contains: variables, constants, a start string (*start*, *axiom* or *initiator*) and rules. The original L-system was used to model the growth of algae and can be seen in Figure 2 below.

This L-system contains:

Variables: A B
 Constants: none
 Start: A
 Rules: (A → AB)
 (B → A)

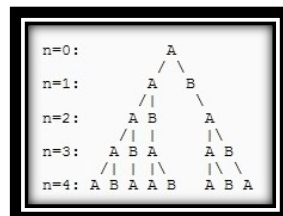


Figure 2 Original L-System for modelling the growth of Algae

We use this system to create our roads starting with a small line drawn in the initial forward direction. As the steps for 'n' increase more lines will be added and for each one a random number check is done with a small chance of returning true. If the test returns true up to three branches will be created in the north, east and west directions from the original point at around 90 degrees to create a block pattern (this number could be changed, or have a random offset to give different road patterns).

To create a rural non-linear road system the distance for each line drawn every 'n' could be shortened and the maximum number of branches reduced from 3 to 2. Also as mentioned previously the chance of the angle changing could be increased and the angle value varied to create a more curved or 'dog-leg' shaped road.

To make the road creation more dynamic we overlay the height map from the terrain whilst building the road [13]. Areas that have water could be impassable or require a bridge and the gradient can be checked to stop roads having too steep an incline and force them to

spiral uphill gradually. We also add a population density map, which can control where main roads are build and the density of the roads. These rules can all be added to the drawing process 'A' mentioned above and alter the return true statement for adding new branches.

Our L-system is structured in the following way (Figure 3):

Variables: X A
 Constants: + -
 Start: X
 Rules: (X → [[-AX] +AX] + AX)
 Angle: 90°
 X = road growth control variable
 '[' = Push
 ']' = Pop
 '-' = Turn Left Angle°
 '+' = Turn Right Angle°
 'A' = if random check true draw line of distance 'D' in forward direction.

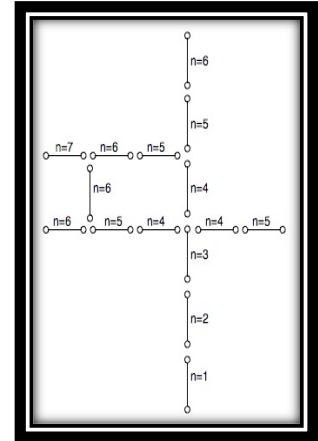


Figure 3 L-system for grid block system roads

4.2 A* algorithm and heuristic pathfinding

Pathfinding is used to navigate from one point to another whilst avoiding any obstacles in the way by finding a suitable route around them. A popular algorithm for this is A*, which can run in real time on a reasonable sized map. As our roads are generated in an editor a slight delay is not as big an issue as it would be in other cases such as an enemy AI in a shooter game. We can scatter points around the map then connect them using pathfinding to create the roads along suitable routes [14; 15].

A heuristic developed by multiple authors for the game 'Open Transport Tycoon' devised a logical cost heuristic for a road-building pathfinder, which also includes provisions for turns, slopes, bridges and tunnels [16].

5 BUILDINGS

When the roads are placed the shape of the lots between them can determine where the buildings can go.

5.1 Shape grammars

Shape grammars are used to provide a computational approach to the generation of designs. In architecture, building structures have many repeating shapes and patterns that can be described using a rule. Shape grammars consist of:

- Shapes – the structures building blocks.
- Spatial Relations – how these shapes are positioned relative to one another.
- Rules – to describe how and where new shapes are added.
- Rule Labels – give a set orientation marker to the rule.
- Transformations – translation, rotation and scale of the shapes.
- Derivation – repeat the rule for n steps to create multiple designs.

Figure 4 shows a labelled rule for a rectangle. For each step a new rectangle is added with a clockwise rotation of 90° and aligned to the original shapes narrow edge (labelled with a dot). The pattern repeats for 3 steps, forming a closed square shape with a gap in the centre.

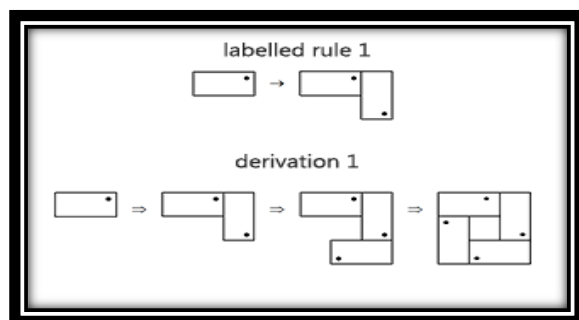


Figure 4 Labeled Rule Example 1 [17]

Figure 5 shows that with exactly the same shapes that changing the label dot, thus changing the rule, can produce a different structure. This time the new rectangle is rotated 90° in an anticlockwise direction and aligned with the short marked edge.

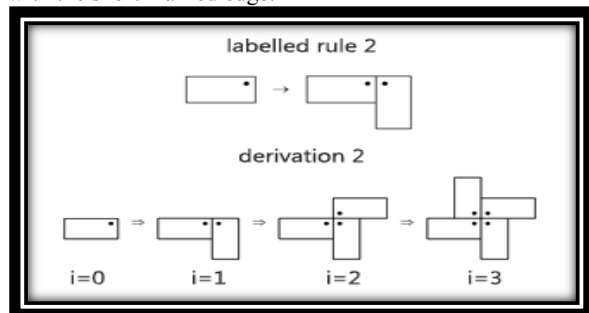


Figure 5 Labeled Rule Example 2 [17]

Using these examples with randomly sized boxes can produce some interesting tower block shapes for the city buildings. The number of steps n can be also set randomly, so just by using the two rules above 8 different structures can be made and when combined with the randomly sized boxes this creates hundreds of different buildings.

To create cylindrical office buildings we consider the technique used in [18]. A circle is generated in 10° slices, with a random chance of skipping ahead 90° , creating a bigger slice and a flat edge. This process continues with up to three 90° skips, until the total rotation is a full 360° circle. The circle can then be extruded upwards with a random height value and capped to create a cylindrical office building.

We experimented with shape grammars to also create smaller houses and other buildings such as a church. Figure 6 below shows two basic house shapes (A & B) created in 3DS Max and loaded into our game engine as a shared mesh which can be cloned.

Using some labelled rules and a combination of these two shapes (along with a thin box for a chimney) many more complicated houses can be built (as depicted in C, D, E, F, G & H). For example the house 'D' is a combination of 'B' and 'A' rotated 90° clockwise. House 'H' is more complicated using translation, rotation and scaling on four instances of house 'A' along with the originally positioned house 'B' and two chimney boxes.

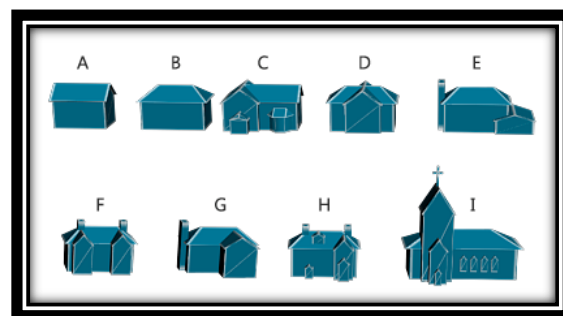


Figure 6 Example House Configurations

5.2 Procedural Textures

The textures used in the scene are generated in two main ways for the land, trees, water and buildings:

- Existing textures produced before hand by an artist in the PNG format are dynamically chosen and scaled according to the object and a set of rules.
- The texture is created procedurally using AWT image graphics and saved as a texture state, which then can be treated as the predesigned textures above.

To create the textures for our buildings we use the latter option to create a series of windows in different sizes, colours and designs. To make the process easier to start off with, we use a procedure similar to the one described in [19], where buildings are dark to mask the missing detail and defined by lit windows, which gives a night time city effect. A 512×512 texture is reserved in memory and each pixel is coloured in several stages:

- First determine the number of windows across and down and what their size is when equally dividing up the texture.
- Set a base colour for the window that is dark (unlit), white or orange (lit). These colours also have a random offset to give a small variation in tone.
- There are two settings for the distribution of the lit windows to simulate either residential tower blocks or offices. The residential buildings have a random scattering; the lit and unlit windows are selected using the math library's next random integer function, with the probability set to roughly 1:3 lit to unlit and an equal chance for white or orange to be chosen for the lit windows. The offices are lit in random blocks consisting of up to a whole floor of windows as companies usually occupy a whole floor or several units and some are open later than others.
- Next for each pixel row the selected base colour is modified by subtracting a small value each time to create a light to dark gradient. This gives the windows more depth and matches the light sources, which are usually at the top of a room.
- To create more variation a per-pixel random colour noise is added.
- Now adding black lines or pixels can create different window shapes/designs and for smaller houses a separate texture can be created with fewer windows spread out over a wider space.

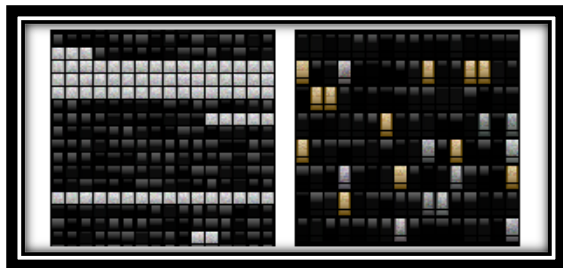


Figure 7 Procedural Window Textures

To create more variety we consider an assortment of window shapes and sizes, especially for the smaller houses textures. Houses have less windows and light, with a more varied array of designs including the bay styled windows.

6 IMPLEMENTATION

6.1 Homura Game engine

We developed our test applications using the Homura Game Engine, another research project we are currently undertaking. The Homura project's development framework provides an Open Source API for the creation of Java and Open GL based hardware-accelerated 3D games applications, which support cross platform, cross-browser deployment using Java Web Start (JWS) and Next-generation Applet technologies. The framework bundles together several example applications and technical demos, which demonstrate how to implement common games functionality in your application; An application template, which acts as a great starting point for development your own Homura related games; The APIs of both Homura and the key open-source projects it builds upon including the Java Monkey Engine scene graph API, jME Physics Library, MD5 Model Importer, GBUI User Interface Libraries and many more; External Tools for the creation of Font Assets, Particle Effects and Levels for the games [20].

We have implemented a user interface using swing. Some controls are provided to the user to constrain elements of the level generation, including the option to turn off some generated features in a level or completely randomise the generation for a two-click level design.

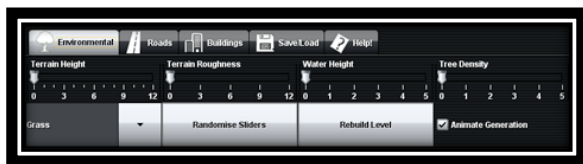


Figure 8 Swing User Interface Layouts

Some example settings that would be useful are:

- Terrain Height – flat/large peaks.
- Terrain Roughness – flat/many peaks & troughs.
- Water Height – no water/some water/flood.
- Tree Density – no trees/ many trees.
- Texture theme – Grass, Desert, Snowy
- Rebuild Level – Rebuild geometry using the above settings.
- Show Tweening Animations – on/off.
- Buildings – on/off
- Population Density - City/Rural
- Roads, Bridges, Tunnels – on/off toggle each.

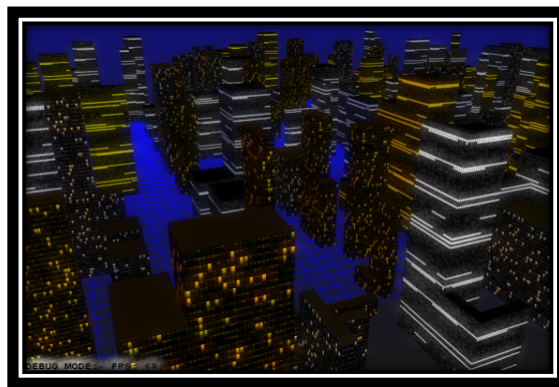


Figure 9 City buildings

6.2 Optimising geometry

To optimise the creation of our geometry we use shared meshes. There are many objects in the scene that are copies of each other and shared mesh allow us to share that data between multiple nodes. "A provided TriMesh is used as the model for this node. This allows the user to place multiple copies of the same object throughout the scene without having to duplicate data." [21] Each copy node has unique translations, rotations, scales and render-states applied to them. Another useful feature to include is object pooling. We can initialise a store of objects in a pool when the game starts for objects like new buildings, trees and textures; so that when one is created it can be quickly taken from this pool.



Figure 10 Tile based map

6.3 Performance evaluation

After completing a test application that runs a tile based map generator we had compelling results. The actual generation of a new level only takes milliseconds, so a tweening animation was added to show how the level is built up in layers, similar to a scan line renderer. The variety of the generated levels is noticeable with different types of water levels, landscapes & tree scattering. At this point in time the buildings are created in a separate application, which will be merged with the environmental one when it is nearer completion. The frame rate runs steadily at around 30fps on a regular specification computer, and much higher through the WebStart version, but deters proportionally with the size of the map being requested. An optimisation for running these maps in a game environment would be to overlay a better scene management system such as a Binary Space

Partition Tree, Octree or Clipmap [22; 23]. The other aspects we have considered are:

- Novelty: contains element of randomness and unpredictability.
- Structure: is not merely random noise, but contains larger structures.
- Interest: has a combination of randomness and structure that players find engaging.
- Speed: can be quickly generated.
- Controllability: can be generated according to a set of natural designer-centric parameters.

The program is still under development and a full usability evaluation will be performed once it is completed, to analyse users' reaction to and satisfaction with the tools.

7 CONCLUSIONS & FUTURE WORK

This paper describes procedural methods applied to terrain modelling, including variable realism of their output, performance and control users can exercise over the procedure.

In this preliminary paper, we focus on the techniques and processes we used to create a web based tool in Java that quickly creates and exports a variety of stylized 3d computer game levels using procedural content generation with random and user selected constraints

In future work we will produce with the game company Onteca, a demonstrator running on a commercial platform. The system will allow a game designer to set up parameters for procedural world generation, initial constraints and then define agent AI rules. The agent AIs (think of them as robots) will then adjust the game world, place pickups, objects, and setup game play using Rules imposed by the game designers. This demonstrator will allow us to explore whether 'fun' can be automated.

8 REFERENCES

1. D. Bekins, D. A. 2005. "Build-by-number: Rearranging the real world to visualize novel architectural spaces". In IEEE Visualization, 143–150.
2. Greuter, S., Parker, J., Stewart, N., AND Leach, G. 2003. "Real-time procedural generation of 'pseudo infinite' cities". In Proceedings of Graphite 2003, ACM Press, New York, USA.
3. Lechner, T., Watson, B., Wilensky, U., AND Felsen, M. 2003. "Procedural city modelling". Tech. rep., Northwestern University, November.
4. Lechner, T., Watson, B., Ren, P., Wilensky, U., Tissue, S., and Felsen, M. 2004. "Procedural modelling of land use in cities". Tech. rep., Northwestern University, August.
5. Muller, P., Wonka, P., Haegler, S., Ulmer, A., and Gool, L. V. 2006. "Procedural modelling of buildings". In Proceedings of ACM SIGGRAPH 2006, ACM Press, New York, NY, USA, 614–623.
6. Muller, P., Zeng, G., Wonka, P., AND Gool, L. V. 2007. "Image-based procedural modelling of facades". In SIGGRAPH '07: ACM SIGGRAPH 2007 papers, ACM Press, New York, NY, USA, 85.
7. Parish, Y. I. H., AND Muller, P. 2001. "Procedural modelling of cities". In Proceedings of ACM SIGGRAPH 2001, ACM Press, New York, NY, USA, 301–308.
8. Prunskiewicz, P., and Lindenmayer, A. 1990. "The Algorithmic Beauty of Plants" (The Virtual Laboratory). Springer-Verlag, New York, New York, NY, USA.
9. Stiny, G. 1980. "Introduction to shape and shape grammars". Environment and Planning B: Planning and Design 7, 343–351.
10. Sun, J., Yu, X., Baci, G., and Green, M. 2002. "Template-based generation of road networks for virtual city modelling". In VRST-02, ACM Press, New York, NY, USA, H. Sun and Q. Peng, Eds., 33–40.
11. Wonka, P., Wimmer, M., Sillion, F., AND Ribarsky, W. 2003. "Instant Architecture". In Proceedings of ACM SIGGRAPH2003, ACM Press, New York, NY, USA, J. Hodgins and J. C. Hart, Eds., vol. 22(3), 669–677.
12. Mick West. (2008, August) Gamasutra. [Online]. http://www.gamasutra.com/view/feature/1648/random_scattering_creating.php
13. Lintfordpickle. (2009, August) Britonia Game Blog. [Online]. <http://britonia-game.com/?p=28>
14. Patrick Lester. (2005, July) gamedev.net. [Online]. <http://www.gamedev.net/reference/programming/features/star/>
15. Kevin Glass. Coke & Code. [Online]. <http://www.cokeandcode.com/pathfinding>
16. Contris. Various. (2009, August) Open TTD Wiki. [Online]. <http://wiki.openttd.org/AI:RoadPathfinder>
17. Terry Knight. (2002, March) MIT Open Courseware. [Online]. http://ocw.mit.edu/NR/rdonlyres/Architecture/4-184Architectural-Design-Workshops--Computational-Design-for-HousingSpring2002/285C5722-AAFC-4DA7-A744-49F31BFE069D/0/fixlecture2slides_final.pdf
18. Shamus Young. (2009, April) Twenty Sided Tale. [Online]. <http://www.shamusyong.com/twentsidedtale/?p=2968>
19. Shamus Young. (2009, April) Twenty Sided. [Online]. <http://www.shamusyong.com/twentsidedtale/?p=2954>
20. LJMU CMP Research Team. (2009, Dec.) Homura Games Framework. [Online]. <http://java.cms.livjm.ac.uk/homura/about.php>
21. Mark Powell. JME Java Docs. [Online]. <http://www.jmonkeyengine.com/doc/com/jme/scene/ShareDMesh.html>
22. Joshua Slack. (2009, November) renanse.com. [Online]. <http://blog.renase.com/2009/11/terrain-example.html>
23. Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones, "The Clipmap: A Virtual Mipmap," in Proceedings of the 25th annual conference on Computer graphics and interactive techniques, NY, 1998, pp. 151–158.

Evolving Pixel Shaders for the Prototype Video Game Subversion

Andrew Howlett¹, Simon Colton¹ and Cameron Browne¹

Abstract. Pixel shaders can be used to create a variety of visual effects in 3D environments, far more efficiently than if produced using the standard graphics pipeline. For such efficiency reasons, pixel shaders are commonly used in video game rendering, to add artistic or other visual effects. We investigate the automated creation of novel shader programs for rendering scenes in the Subversion virtual game world, with a view to providing the player with a visually richer and more diverse 3D environment. We show how shader programs based on the OpenGL shading language may be represented in a hierarchical tree form. This representation admits an evolutionary approach to shader creation, and we show how the application of genetic programming techniques can lead to the evolution of new and interesting shaders. We harness this for an approach where the user supplies details of a fitness function for the overall look of the city environment. We experimented with a number of different fitness function setups in order to produce some preliminary results about this approach. While generally successful in the creation of novel and visually interesting shading effects with little effort, we find some drawbacks to the approach and suggest methods for improvement.

1 INTRODUCTION

Subversion (www.introversion.co.uk/subversion) is a video game prototype being developed by Introversion Software Ltd. that takes place in a virtual 3D world. The environment features procedurally generated landscapes and cityscapes produced by a custom 3D generation engine. The Subversion engine generates terrain and cities on the fly and allows the user to custom-design their own buildings [1] to increase their enjoyment of and engagement with the game, and to make the 3D environment richer and more diverse. The city models created by the software adhere to a single – slightly retro – style by default, as seen in the screenshot of an example city provided in figure 1. Hence, another way in which users may customise the game world is to specify preferred rendering schemes for the procedurally generated world content, in order to enhance the visual presentation of the game world and possibly to make the game experience more immersive. In addition, such rendering schemes may be applied automatically by the game engine in order to reflect certain moods and emotion, for example, forests may be predominantly green in hue, oceans blue, deserts yellow, or action scenes may be tinged with red to foster a sense of menace and tension.

The elements in a virtual Subversion world are defined by 3D polygons, which are broken down into triangles and passed to the machine's graphics card for rendering. This allows vertex shaders and pixel shaders to provide a convenient way to apply custom rendering schemes on the fly [2]. A vertex shader is a program that operates upon the geometry of the triangle vertices passed to the graphics pipeline, while a pixel shader is a program for performing per-pixel rendering of a scene based upon this underlying geometry. Both are typically optimised for and executed on the machine's graphics processing unit (GPU) to maximise performance. As Subversion already handles the vertex geometry, we were interested in customising the pixel shading program to apply custom rendering schemes to the resulting cities. We aimed to determine a viable and convenient method for users to define custom colouring schemes in which the procedurally generated Subversion game content may be rendered. Ideally, users should be able to define shaders easily and with no technical knowledge of the shading language used.

One approach to enabling users to specify aspects of the visual look and feel of the game is to allow them to specify a fitness function, and evolve pixel shaders in order to produce cityscapes rendered to look approximately like the user specified. For instance, if the user specifies a reddish hue with highly saturated (vibrant) colours, this can be expressed as a fitness function for an evolutionary search for pixel shaders. We investigate the plausibility of such an approach here. In order to do this, we specify a representation of pixel shaders in a hierarchical tree format, as described in the next section. In the following section, we describe how an evolutionary search can be set up, by detailing the evolutionary operators we employ and the nature of fitness functions which drive the search for pixel shaders. In the experimentation section, we describe 21 short sessions with the evolutionary tool, ranging over six different fitness function setups. We demonstrate the feasibility of evolving pixel shaders, and we conclude by putting this work into the context of related work and suggesting future directions.

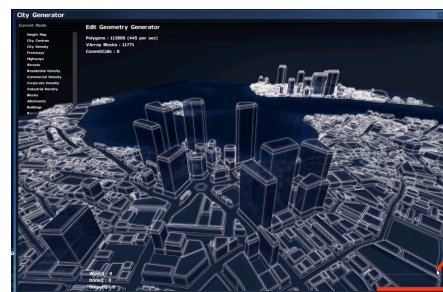


Figure 1. Example city generated by a standard Subversion rendering (taken from www.introversion.co.uk/subversion).

¹ Computational Creativity Group, Department of Computing, Imperial College, London. www.doc.ic.ac.uk/ccg

2 REPRESENTING PIXEL SHADERS

We chose to implement our pixel shaders in the OpenGL Shading Language (GLSL). GLSL is a high-level shading language based on the C programming language, and is probably the most prevalent shading language for these purposes [3]. This, together with the fact that Subversion is optimised for OpenGL, made GLSL an obvious choice. GLSL applications do not operate in isolation but in conjunction with OpenGL, which provides an API extension for pixel shaders. Each custom shader program is represented as a tree of operators, with child nodes acting as operands of their parent nodes. The output from each tree after normalisation is an RGBA colour value which describes how each pixel in the user's viewport is to be rendered. Each leaf node represents an attribute of the pixel to be shaded, or a constant 4-vector. The attributes can either refer to a 3D co-ordinate (in which case the fourth vector value is zero), a normal to a plane or a lighting component of the pixel. Whenever a floating point number (attribute or constant) is required to be a vector, it is converted by duplicating the floating point in all four components (with Booleans similarly handled as duplicated ones or zeros). All possible pixel attributes are described in table 1. Note that TEXTS and TEXT together define the approximate position on a wall, roof etc., that the pixel corresponds to. Note also that the NodeGrid is an internal data structure for representing buildings used by Subversion (see [1] for details).

Leaf node value	Type	Description
ISSTRE	Boolean	Checks whether the pixel is part of a road
ISBUIL	Boolean	Checks whether the pixel is part of a building wall
ISROOF	Boolean	Checks whether the pixel is part of a building roof
ISGROU	Boolean	Checks whether the pixel is part of the ground
ISWATE	Boolean	Checks whether the pixel is part of a river
OBJPOS	Co-ordinate	The 3D world (city) co-ordinates that correspond to the pixel
VIENOR	Direction	The normal of the polygonal face to which this pixel belongs, in user view space
OBJNOR	Direction	The normal of the polygonal face to which this pixel belongs, in world (city) space
SPECOM	Lighting component	The pixel's specular lighting component
DIFCOM	Lighting component	The pixel's diffuse lighting component
TEXTS	Co-ordinate	The s-texture co-ordinate for the pixel
TEXT	Co-ordinate	The t-texture co-ordinate for the pixel
NGRAND	Float vector	Per-NodeGrid random value for the pixel
PROPOS	Co-ordinate	The pixel's on-screen position
VIEPOS	Co-ordinate	The pixel's position in user view space

Table 1. Pixel attributes, each of which is a four-dimensional vector. Boolean vectors are either entirely ones or zeros, and are generated by a conditional acting on the pixel attributes. Co-ordinate vectors describe 3D co-ordinates of the pixel either in the world view, or the user view, and can be relative to building walls, etc. Direction vectors represent normals to planes in the world view. Lighting vectors represent specular or diffuse lighting components for rendering.

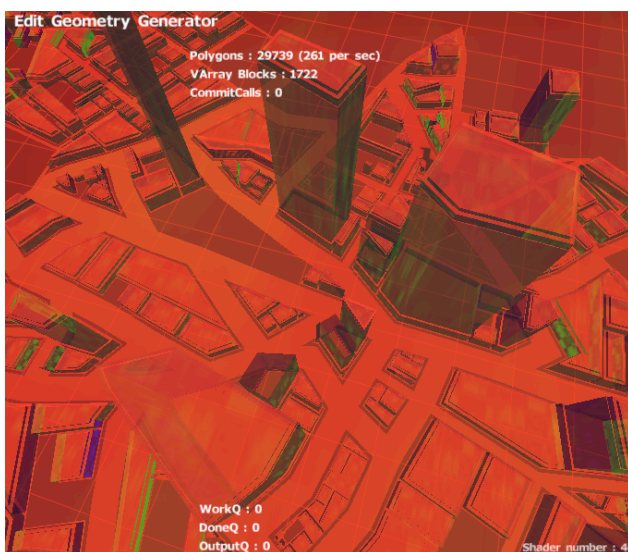
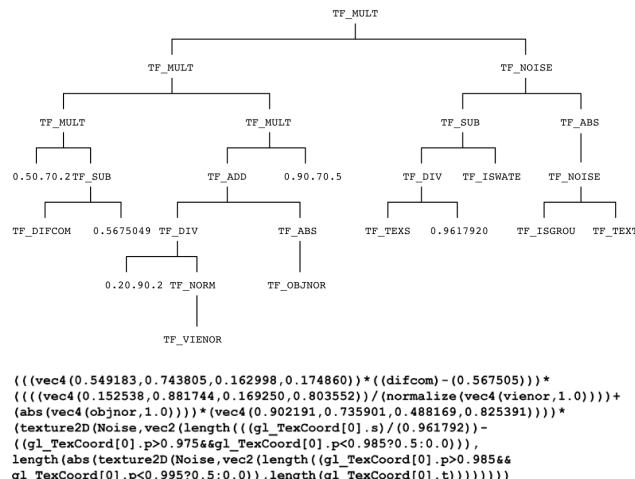


Figure 2. An example shader tree, compiled script and resulting rendering by the pixel shader. Note that some of the numerical details are missing from the tree.

Branch nodes are mostly arithmetical functions, including ABS, MULTIPLY, DIVIDE, ADD and SUBTRACT, and act on the vectors in the obvious ways. There is also a NOISE node, which produces its output using its inputs as coordinates to lookup an area of a texture. The texture was manually created in Photoshop, using the clouds filter. Each tree is compiled to a pixel shader script or program in GLSL format that is wrapped between appropriate header and footer text, then passed to the graphics pipeline to direct the per-pixel aspects of the rendering of the scene at runtime. As an example, the tree shown in figure 1 generates the GLSL script shown below it in the figure. The image at the bottom of the figure shows the final result of rendering a scene from the Subversion world with this shader program.

3 SHADER EVOLUTION

Genetic programming (GP) involves the evolutionary optimisation/generation of computer programs [4] via the simulation of genetic operators such as crossover and mutation, and the simulation of genetic drivers, such as survival of the fittest, tournament selection, etc. We use a standard GP approach for evolving shader programs. The process starts with a pool of N candidate shader trees that are generated randomly in a straightforward way by iteratively growing the trees. Then a particular scene is rendered with each shader, so that the shader can be evaluated according to a user-defined fitness function, as described below. Individuals are then chosen for an intermediate population using roulette selection, hence potentially copied a number of times according to their relative fitness. Pairs of (possibly identical) parents are then randomly selected from this intermediate population and either copied, crossed over or mutated, as described below. Each such mating produces two children that are added to the next generation, until the required number of children are created. The process then repeats. In the experiments described below, the evolutionary process runs for a total of 10 generations by default, and each population comprises 60 individuals by default.

Standard one-point crossover is performed between each pair of parents, in which a subtree is randomly chosen from each parent and the two subtrees swapped. We also implemented an additional form of crossover in which a new tree is created with a single 'if' node at its root and the two parents attached as the true and false branches to create a new child program. This clearly has the potential to lead to program bloat, and we need to perform further analysis to determine the extent of this problem. Crossover is made with a 15% chance per node. We implemented two forms of mutation: subtree and data mutation, each with a 70% chance per node. In subtree mutation, a subtree of the parent is randomly selected and replaced with a randomly generated tree with depth proportional to the depth of the subtree being deleted. This is essentially a crossover with a randomly generated subtree. In data mutation, floats and vectors may be assigned random values, vectors may have their components swapped or zeroed, or randomly selected variable nodes may be replaced with function nodes (for which the original variable node is attached as an operand). It was found that the last step was useful for encouraging shaders to produce noticeably different results in the presence of certain variables. For example, the object normal (OBJNOR) would typically produce results from a very limited palette unless this step was performed at least some of the time. This tree modification process was also applied a number of times to all trees in the initial generation, for the same reason – random tree node generation tended to construct trees that would appear similar if they used a similar set of variables.

Figure 3 shows the screenshot of an evolutionary session in action. The user may stop the evolutionary process at any point, and may view and/or export any tree, corresponding shader program or rendered scene throughout the process. In the figure, the user has chosen to view the rendered output from four candidates of each generation as a way of keeping an eye on progress. Each generation can be saved for later reload.

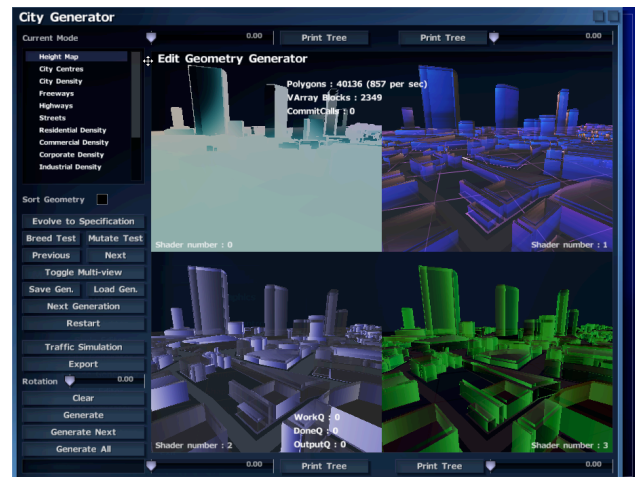


Figure 3: Screenshot of a session with the evolutionary system.

The user specifies a preferred fitness function using the parameters shown in figure 4. The main controls are three sliders indicating the user's preferred hue, saturation and luminance in the HSL colour space [5] and the relative importance of each component, all in the range 0 to 1. The user may also choose to either penalise or reward the presence of noise and textures in the shading algorithm (although we do not use these parameters in the experiments described below).

The fitness function may be applied in two ways: per-pixel or by blocks. In per-pixel mode, one pixel in four of the rendered image is sampled and the average hue, saturation and luminance values found over all sampled pixels is calculated. Then, the shader's fitness is defined as the maximum discrepancy between any of these three average values and the corresponding values specified by the user, modulated by importance. In block mode, a similar measurement is made but instead of measuring individual pixels, the image is segmented into blocks and the hue, saturation and luminance values most characteristic of each block are averaged. The hue sliders may be adjusted at run time throughout the evolutionary process, allowing the user to interact with the system and drive the creative process more directly. If the user does not adjust the sliders, then the process continues automatically to seek shaders that most closely resemble the user's specified hue requirements.



Figure 4. User interface for specifying the fitness function.

4 INITIAL EXPERIMENTAL RESULTS

In order to begin to determine the plausibility of evolving pixel shaders to the user's specification of a fitness function, we have undertaken some preliminary experiments. Bearing in mind that the long-term application of this approach will be to allow players of Subversion to describe the look and feel of the cities that they play the game in, we note that long evolutionary times are not appropriate. Hence, we restricted our sessions with the evolutionary system to produce just 10 evolved generations, in addition to the initial randomly generated one. The first question to ask with evolutionary approaches is whether the evolution actually has an impact, i.e., we wanted to assess whether any improvement over random generation of shaders is possible in such short evolutionary sessions. Moreover, to restrict the experimental conditions, our experiments here concentrate on achieving particular colours, specified by the Hue, Saturation and Luminance of the colour (HSL). The first three fitness function setups were therefore aimed at producing particularly coloured pixel shaders, and were varied by changing the weighting of the H, S and L values, as per table 2. We speculate that the Hue value is less likely to be used in practice than the saturation and luminance values. This is because changes in saturation and brightness of scenes is more likely to affect a change in mood than the hue of the scene. For instance, a dark scene (whether dark blue, red or any other colour) is likely to produce a more sinister mood than a light scene with highly saturated colours, which is more likely to produce a carnival feel to the city. With this speculation in mind, for fitness setups 4, 5 and 6 in table 2, we fixed the hue weighting to zero, so that it did not guide the search.

S	Hue (v/w)		Saturation (v/w)		Luminance (v/w)	
1	30.21	0.50	0.81	0.50	0.40	0.50
2	30.21	0.55	0.81	0.22	0.40	0.22
3	210.0	0.55	0.75	0.22	0.75	0.22
4	0.00	0.00	0.65	0.50	0.82	0.50
5	0.00	0.00	0.85	0.50	0.20	0.50
6	0.00	0.00	0.20	0.50	0.50	0.50

Table 2. Fitness function (S)etups for evolutionary sessions. Both the (v)alue and the (w)eighting are shown.

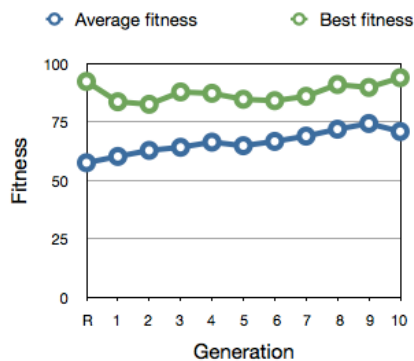


Figure 5. Average and best fitness of individuals in the populations, starting with the randomly generated population (R), and for 10 successively evolved generations. This figure depicts run Q from table 2, with fitness setup 5 from table 1.

In total, we ran 21 sessions to evolve pixel shaders according to the six fitness function prescribed in table 2. For each session, we fixed the evolutionary parameters. In particular, we specified a session of 10 generations, with a population size of 60 individuals. The crossover and mutation rates were as described in the previous section. In figure 5, we portray how the average and the best fitness of individuals in each population changed from generation to generation for a representative session. The results from the 21 sessions are presented in table 3, with summary statistics corresponding to the six fitness functions presented in table 4. We analyse these results in the conclusions section.

Run	S	Rand	Av. last	Av. bests	Best (gen)
A	1	67.0	79.2	90.7	93.5 (9)
B	1	79.1	76.8	92.2	93.7 (6)
C	1	62.8	72.3	87.1	88.8 (6)
D	1	72.3	69.6	85.8	87.5 (0)
E	1	68.4	72.5	87.7	89.3 (5)
F	1	60.7	71.6	84.5	87.5 (0)
G	2	63.8	74.0	88.7	93.6 (1)
H	2	74.1	71.6	85.8	88.6 (6)
I	2	71.4	69.6	81.9	84.2 (0)
J	3	65.7	70.8	85.3	87.5 (4)
K	3	68.3	71.3	88.2	93.6 (7)
L	4	62.6	76.2	90.4	94.7 (7)
M	4	77.9	71.8	92.6	96.2 (3)
N	4	71.3	78.4	91.8	94.5 (9)
O	4	77.6	78.1	89.5	92.9 (0)
P	4	65.1	71.0	87.4	91.6 (0)
Q	5	57.5	70.9	87.5	93.9 (10)
R	5	60.7	72.8	90.9	93.7 (5)
S	6	74.5	80.0	93.1	95.3 (7)
T	6	79.8	82.8	94.8	95.9 (1)
U	6	82.8	78.3	95.1	96.8 (3)
Av.		69.7	74.3	89.1	92.0 (4.2)

Table 3. Average fitness of individuals in the initial random generation (Rand); average fitness of individuals in the final generation (Av. last); average of the fitnesses of the best individuals in each generation (Av. bests); fittest individual in any generation, and the generation where it was seen (Best (gen)) for six different fitness functions (S)etups over 21 runs of the evolutionary system.

S	Rand	Av. last	Av. bests	Best (gen)
1	68.4	73.7	88.0	90.1 (4.3)
2	69.8	71.7	85.5	88.8 (2.3)
3	67.0	71.1	86.8	90.6 (5.5)
4	70.9	75.1	90.3	94.0 (3.8)
5	59.1	71.9	89.2	93.8 (7.5)
6	79.0	80.4	94.3	96.0 (3.7)
1+2+3	68.5	72.7	87.1	89.8 (4)
4+5+6	70.98	76.03	91.31	94.6 (4.5)
All	69.7	74.3	89.1	92.0 (4.2)

Table 4. Summary statistics for each of the fitness function (S)etups (and collections of setups), averaged over all the runs for each setup (or collection).

5 ILLUSTRATIVE PIXEL SHADERS

Figure 6 portrays the rendering of cities afforded by four example pixel shaders that were evolved during the sessions described above. They are provided as an indication of the kinds of outputs achievable by the system.

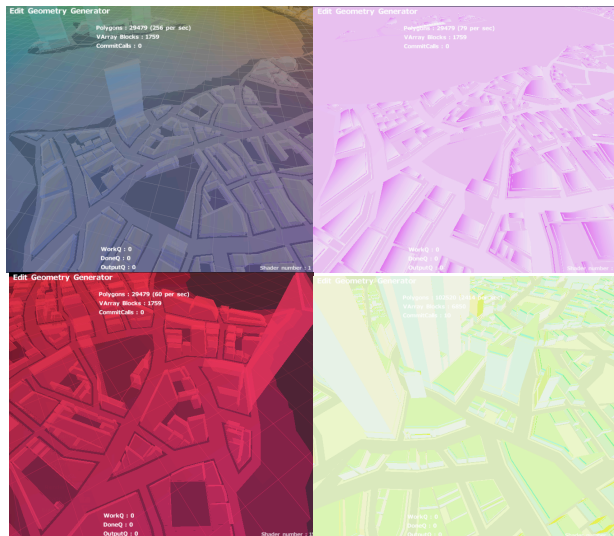


Figure 6. Four examples of evolved pixel shaders.

6 RELATED WORK

With the exception of games based on notions from A-life, notably games in the so-called ‘God-genre’ such as Black and White, evolutionary methods are rarely employed in commercial video games. This may be because of the long time usually required by the evolutionary processes (and a general tendency in the industry to avoid techniques that may be unpredictable). Within the realm of AI research, evolutionary techniques have been experimented with for the following purposes:

(a) Novel content generation. For instance, in [1], the authors describe a user-driven approach to generating buildings (also for the Subversion environment). A genetic algorithms approach to conceptual building design is presented in [6], where the emphasis is on decision support systems to assist designers. Moreover, in [7], the authors evolve weapons for the Galactic Arms Race game, and in [8], entire levels are evolved for the game Mario.

(b) Novel presentation of existing content. These applications are the most relevant to the work presented here. The idea to use genetic programming to evolve shader programs was initially suggested by [9] and has been successfully demonstrated by various authors. For instance, the approach described in [10] is user-driven, i.e., the user acts as a fitness function during the session by choosing the pixel shaders they prefer, rather than by specifying a fitness function in advance. Our approach also differs from their work in terms of the hierarchical tree-based

representation of shader programs that we use. The shader representation they use comprises a (possibly variable length) set of integers which are mapped via a lookup table to commands for NVIDIA’s high level Cg shader language, and arguments for the commands. Earlier work on evolving shaders include [11], which used a fixed length representation of shaders which mapped to the assembly language of the hardware they ran the system on. We are not aware of any other approach where the primary target of GPU-based pixel shader evolution has been a video game world.

(c) Controlling of avatars for 1-player modes. For instance, in [12], the authors evolved behaviour trees to control an avatar that played the nuclear war simulation game DEFCON, and in [13], the authors similarly evolved players for Quake 3.

(d) Controlling non-player characters (NPCs). For instance, in [14], the authors evolve psychosocial behaviours for NPCs, and in [15], the authors use evolutionary learning for adaptive game intelligence.

Given the processing power of Graphical Processing Units, various researchers have attempted to use them for efficiently evolving software. For instance, in [16], the authors evolve a vision system for object recognition on a GPU. There is an extensive bibliography on genetic programming for GPUs at www.gpgpgpu.com, and in [17], the authors give a good overview of the efficient evolution of programs on GPUs. In [18], the authors describe a parallel evolutionary implementation on a consumer graphics card, and show its value in achieving up to five times speed up with tests using vertex shaders.

7 CONCLUSIONS AND FUTURE WORK

The experimentation presented above is quite preliminary in nature, and more work is needed for us to be able to draw any general conclusions about the plausibility of evolving pixel shaders to user-required fitness considerations. The graph in figure 5 is typical of the progress of the evolutionary approach that we observed over the sessions – slow but steady nearly-linear increase in average fitness over the populations. From table 3, we see that the average fitness of the individuals in the tenth and final population was higher than the randomly generated population in 16 of the 21 sessions, with average fitness increase during a session of 4.6. This is encouraging, but it is surprising that in five sessions, ten generations of evolution did not produce a better than random population. These results are indicative of a general trend observed over most runs that the best fitness scores tend to improve slightly as the number of generations increases, which suggests that the approach is basically valid. Having said this, the results were sometimes visually disappointing, although this may be partly due to the limited support for shaders currently implemented in the Subversion rendering engine.

While the most fit individual was often found in the earlier generations (sometimes the first generation), this is not a particular worry, as the user would want to be presented with multiple pixel shaders to choose from at the end of the session, hence it is more important that the overall fitness of individuals

increases, as we see in the majority of the sessions. Moreover, recalling the long term goal of enabling Subversion players to prepare their own cities, one mode of supplying pixel shaders during the (possibly lengthy) evolutionary process would be to present them with the best individual from each generation. As we see from table 3, this would ensure that they see more fit shaders than randomly generated ones, with an average fitness of 89.1, rather than 69.7 for the average random generation. In table 4, we see that for each of the six setups, on average a session would show an increase in average population fitness from start to finish, which is again encouraging. We also note that it was more difficult to maximise the fitness functions from setups 1, 2 and 3 than from setups 4, 5 and 6. This was expected, as the latter fitness functions had no hue requirements to satisfy.

In addition to the automated approach described above, we have performed some initial experimentation with a user-driven approach, where the user chooses their preferred shaders for crossover and mutation in successive generations, thus acting as the fitness function on a local level. While the software had a ludic quality, from initial testing, it was our overriding feeling that the process was more successful in finding visually interesting shaders when automated than when user-driven. In particular, the user feels a tangible lack of control during evolution, possibly because the results of crossover and mutation are not always very intuitive. This suggests that we need to experiment with the settings for crossover and mutation, as they are too destructive from generation to generation.

While successful as a preliminary study, the interface we developed needs significant work if it is to be used commercially. For example, it would be useful to compare two arbitrary shaders from the population (this facility is not currently implemented), allow the user to take a more active role in the evolutionary process and override the fitness function to specifically nominate candidates that they find appealing, and perhaps even allow specific mutation requests such as inserting preferred colour values or requesting different shading algorithms for different parts of the image, e.g. walls as opposed to roof areas. Further, the evaluation method might be improved both in speed and accuracy by counting only those regions of interest deemed to be foreground (city) objects rather than sampling over the entire scene. Also, the tree operations that we define only exercise a small fraction of the functionality available in the OpenGL Shading Language. It would be interesting to implement a more complete set of operations and to determine which operations are more likely to be useful, so that tree growth may be biased accordingly and the software may be directed towards more fruitful areas of the search space.

Working with the Subversion city generation engine, in addition to the work presented here, we have also investigated the evolutionary generation of buildings [1], in addition to user-guided citywide look-and-feel (e.g., ratio of commercial and residential building types, etc.) and fitness function guided evolution of traffic schemes, where the fitness functions encourage city design for fun driving experiences, rather than best traffic flow or other utilitarian purposes. We hope to show that using evolutionary techniques both at game design time and for users to generate content at run time has much potential to increase efficiency of game design, and increase user enjoyment.

ACKNOWLEDGEMENTS

This project was funded by a grant from the UK Technology Strategy Board. We would like to thank the staff of Introversion Software Ltd. for access to their code and for their very valuable input into the project. In particular, we would like to thank Andrew Lim for his help with the Subversion city generation engine. We would also like to thank the anonymous reviewers for their constructive and useful comments.

REFERENCES

- [1] A. Martin, A. Lim, S. Colton and C. Browne. Evolving 3D Buildings for the Prototype Video Game Subversion. In the Proceedings of the EvoGames workshop, 2010.
- [2] W. Engel. Programming Vertex & Pixel Shaders, Charles River Media, 2004.
- [3] R. Rost, B. Licea-Kane, D. Ginsburg, J. Kessenich, B. Lichtenbelt, H. Malan and M. Weiblen. OpenGL Shading Language, 3rd Edition, Addison Wesley, 2009.
- [4] J. Koza. Genetic programming. On the programming of Computers by Means of Natural Selection, MIT Press, 1992.
- [5] J. Foley, A. van Dam, S. Feiner and J. Hugues. Computer Graphics: Principles and Practice, 2nd Edition, Addison Wesley, 1990.
- [6] Y. Rafiq, J. Mathews, and G. Bullock. Conceptual building design-evolutionary approach. Journal of Computing in Civil Engineering, 17(3), 2003.
- [7] E. Hastings, K. Guha, and K. Stanley. Evolving content in the galactic arms race video game. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2009.
- [8] N. Sorenson and P. Pasquire. The Evolution of Fun: Automatic Level Design for 2D Platformers. In Proceedings of the 1st International Conference on Computational Creativity, 2010.
- [9] F. Musgrave. Genetic Textures. In D. Ebert et al. (Eds.), Texture & Modeling, A Procedural Approach, 1998.
- [10] M. Ebner, M. Reinhardt and J. Albert. Evolution of Vertex and Pixel Shaders, M. Keijzer et al. (Eds.), In Proceedings of EuroGP, 2005.
- [11] J. Loviscach and J. Meyer-Spradow. Genetic programming of vertex shaders. In Proceedings of EuroMedia, 2003.
- [12] C. Lim, R. Baumgarten and S. Colton. Evolving Behaviour Trees for the Commercial Game DEFCON. In the Proceedings of the EvoGames workshop, 2010.
- [13] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels. Evolution of human-competitive agents in modern computer games. In Proceedings of the IEEE Congress on Evolutionary Computation, 2006.
- [14] C. Bailey and M. Katchabaw. An emergent framework for realistic psychosocial behaviour in non player characters. In Proceedings of the 2008 Conference on Future Play: Research, Play, Share, 2008.
- [15] Ponsen, M., & Spronck, P. (2004). Improving adaptive game AI with evolutionary learning. In Proceedings of Computer Games: Artificial Intelligence, Design and Education, 2004.
- [16] M. Ebner. An Adaptive On-Line Evolutionary Visual System. In Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 2008.
- [17] S. Harding and W. Banzhaf. Fast Genetic Programming on GPUs. In Proceedings of the EuroGP, 2007.
- [18] K. Fok, T. Wong and M. Wong. Evolutionary Computing on Consumer Graphics Hardware. IEEE Intelligent Systems 22(2), 2007.

AEINS: The Role of Interactive Narrative in Fostering Character Education

Rania Hodhod¹ and Daniel Kudenko and Paul Cairns

Abstract. Interactive narrative allows collaboration of humans and computers in the creation of innovative experiences. Different narrative techniques have been used in computer narrative-centered learning environments to help in creating enriching experiences for its users, sparking problem-solving skills, and transfer of tacit knowledge. Graph structured narrative and dynamically planned generated narrative are two examples that have shown advantages and disadvantages when deployed in these environments. This paper proposes the integration of both techniques in a way that allows: (a) balanced user agency during the game interaction course; (b) the generation of a dynamic continuous story that exhibits a story arc; (c) the presence of virtual evolving agents that provide dramatic and pedagogical roles. The proposed idea has been applied to AEINS, an adaptive educational game that aims to foster character education. Results from AEINS evaluation are presented and discussed.

1 Introduction

“Stories are connections to the past and yet carry us into the future; they speak of relationships, of human connections, and to what gives a quality to our lives.”[29]

Simpson (1998) in her article emphasizes the importance of stories in our lives and their role in tightening human relationships [29]. Stories help children to cope with feelings and develop their imagination. Role playing allow children to put themselves in another one’s shoe and experience how others feel. It helps students to transfer their knowledge and beliefs into actions and see how their decisions affect other people and things [24]. In addition to its feasibility to be used as an appropriate medium to transfer tacit knowledge to the students in the classroom environment [3].

Interactive narrative allows collaboration of humans and computers in the creation of innovative experiences and plays an important role in cognition and culture. Narrative-centered learning can be viewed as rich generated stories that allow a kind of unintentional learning process to happen through an engaging and appealing experience, and the student is seen as an active participant in the construction of his own knowledge. As a result, narrative-centered learning environments have been the subject of increasing attention in the last few decades and has been used as a testbed for many application areas such as games, edutainment systems, and educational games.

Narrative-based educational games aim to offer education through an engaging experience in which narrative is a key instrument in facilitating teaching and learning. Such environments should exhibit certain educational properties in order to be able to achieve the educational targets. They may contain one or more of the following

tutoring components: tutorial planning, student model, learning objectives, domain and pedagogical models. Some environments also provide feedback or hints.

This paper focuses on the application of narrative in the current educational games and the existing limitations, in addition to proposing an approach that addresses those limitations. The main narrative problem encountered is due to the contradicting nature of games and learning. On one hand, games allow high user agency for an engaging entertaining experience and do not expect transfer of knowledge to real world. On the other hand, personalized learning requires restricted user agency to allow tracking and assessing the learning process and expects transfer of knowledge as an outcome. Therefore, there is always trade off between both gaming and learning.

The current educational games either use continuous planning to generate a dynamic story and lose some aspects of the educational process such as keeping track and assessing the learning process [22, 28, 31], or use predefined paths in order to preserve the educational goals [14, 1, 32, 12, 26, 5, 13], which limits the freedom of the student [27] and requires an extensive amount of authoring work [9]. Some efforts have been done to integrate both narrative techniques, such as that of [23, 13]. Nevertheless, these edugames have not take full advantage of storytelling potential of games seen in interactive drama applications, such as the presence of a story arc and/or evolving virtual agents that can achieve more than simply being single virtual guides and virtual tutors. Their role could be extended to interact with each other as a set of characters to present a dramatic storyline, in addition to serve as all sorts of sparring partners for players to interact with, such as representing the bad guys, or companions who ask for help [34]. These properties allow highlighting the relationships among the narrative elements, which is considered as a key point for provoking active thinking and supporting meaning construction [5].

The main idea to present in this paper is the integration of a dynamically generated narrative and a scripted branching narrative approaches in a way that allows: (a) balanced user agency during the game interaction course; (b) the generation of a dynamic continuous story that exhibits a story arc; (c) the presence of virtual evolving agents that provide dramatic and pedagogical roles. This idea has been applied to AEINS, an adaptive educational interactive narrative system that aims to foster character education in the ethics domain [15]. AEINS integrates interactive narrative and intelligent tutoring in order to have strong learning objectives underpinned by effective story telling. For more information about AEINS educational pedagogy, see [17]. The next section motivates the choice of the ethics domain and character education. Sections 3 and 4 present AEINS and the interactive narrative generation techniques used. Finally, evaluation and conclusion are presented in sections 5 and 6.

¹ University of York, England, email: rania.hodhod@cs.york.ac.uk

2 The Ethics Domain

The ethics domain is a challenging domain for its ill-defined nature. It exhibits the following characteristics: (a) ambiguity in classifying a problem's answer as correct or incorrect [12], (b) lack of defined rules that helps in transferring the solution path from the initial step to the final step [25]. The ethics domain can be considered as one of the the important ill-defined domains because it holds in nearly every action in our daily life. Developing child and adolescent awareness of social and moral responsibilities is a prominent aim, so-called character education. Effective character education promotes core ethical values such as caring, honesty, fairness, responsibility, and respect for self and others, along with supportive performance values such as diligence and perseverance that form the basis of good character [20].

According to Kohlberg, quoted in [8], if children get engaged in enough independent thinking they will eventually begin to formulate their own conceptions of rights, values, and principles by which they evaluate existing social arrangements. Watson also mentioned a very important point which is the desire for good [33]:

“The trick lies not solely with knowing what is right and good but also in building a love for the good and the worthwhile.”

Watson points out that by giving the students the chance to see successful people do what's right and good, chances are better that students will be biased to follow suit themselves than they might otherwise. Interactive narrative environments are rich by agents and objects and story worlds that can provide the suitable environment to tackle these issues.

Another generally accepted fact is that learning about moral virtues is different from doing them, as Watson clarifies [33]:

“Getting high scores in an ethical course does not guarantee at all the actual behavior of that student.”

This explicitly indicates that teaching in this domain is not based on acquiring more knowledge but on the development of skills and transition in the children's reasoning process. These properties make it very difficult to use intelligent tutoring, tracing and assessing algorithms, to teach in an ill-defined domain. Therefore, providing an environment that allows situated learning in a pleasant interactive way can be seen as an advantage. Transferring what the children had learned to the real world or indications that this can happen can be seen as a plus.

3 AEINS

To the extent of our knowledge, there is no educational game developed yet to develop core ethical values at young children. AEINS is the first step taken to provide such environment. AEINS is an educational game that aims to foster character education. The architecture composed of two levels; tutoring and narrative, see figure 6. Each module in the system is fully separated from the other parts, however all the modules interact together through sets of production rules. AEINS is intended to assist 8-12 year old children to learn ethics and citizenship. The student is able to act in the environment influencing the path of the unfolding story, and at the same time be monitored and guided by the tutor. AEINS aims to inform the students about various ethical values by giving them the chance to explore and experience different moral actions, gain new insights and allow them to see the consequences in a safe environment. It uses stories and interactive narrative as a source of inspiration and direction for moral

conduct. Following the constructivist approach to learning, AEINS provides a problem based learning approach [11] designated in the teaching moments. In these approaches, learning is anchored around a specific problem and context, and the focus is on the student solving a problem. The student is able to discover what knowledge gaps and deficiencies they may have along with skills they may need to develop.

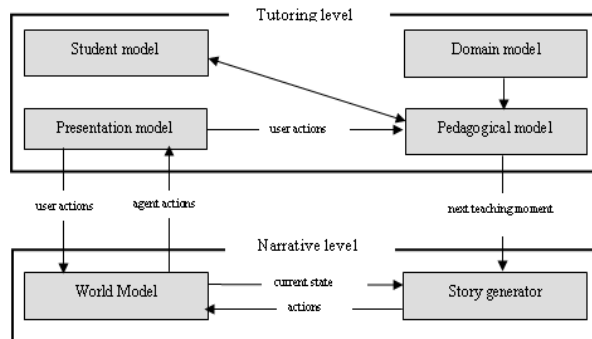


Figure 1. AEINS architecture

AEINS provides adaptation on the tutoring level by incorporating four components: domain model, student model, pedagogical model, and presentation model. The domain model is a main component of tutoring systems. It consists of the educational concepts and their relationships (dependency upon each other). The domain knowledge has been collected from [6] and then arranged in the suitable hierarchical order to represent the learning progression. The domain model in AEINS is divided into two parts. The first part is a hierarchical frame representation of all the moral values the student required to gain skills of. The frame representation provides desirable flexibility by allowing constructing the relationship between the domain concepts in an easy and clear way. The second part of the domain model comprises of teaching moments repertoire. The teaching moments can be thought of as a variety of ethical problems that require tough decisions. The idea behind the current design of the teaching moment is based on analyzing moral dilemmas and transforming them to a story graph structure, then specify the decision points that reflect the specified skills.

The pedagogical model adapts instruction following a model of human tutoring expertise that balances motivational and cognitive goals. The model uses the Socratic Method as it has been shown to be a highly effective approach [7] that helps children develop new ideas and gain new insights. The pedagogical model is able to infer knowledge about the student's skills and expand the student model. The student model is a crucial component that mainly aims to create an individual model for every student. The student model is currently a simple form of the overlay model represented in the form of rules. AEINS builds a model of the student's learning process by observing, recording and assessing the student's actions and choices from the generally accepted ethical views. The student model consists of three parts: the student's personal details, the student knowledge and the student intention. Finally, the presentation model handles the flow of information and monitors the interactions between the user and the system and vice versa. To interact with the story, AEINS offers a GUI, where the student is able to take different actions, such as moving between different places, inviting friends, ask friends round

to play, and so on. More details on the tutoring level and AEINS educational outcomes can be found in [16, 17].

The narrative level consists of two components: a story generator and a world model. The story generator uses continuous planning for the dynamic generation of narrative at run-time and allows a graph structured planning to interfere (whenever required) as part of the story to present the various moral dilemmas (teaching moments). The motivation of learning roots in the uncertainty and curiosity resulted from the nonlinear structure of paths and different ending alternatives of the teaching moments [21]. In this way, the player expects to play the game many times, trying different strategies each time. The teaching moments are selected and presented in a way that allows the student to draw analogies from one problem to another (if needed).

The world model houses the objects, such as places and tools, and the AI character models associated with their properties. Each AI character is implemented as a set of rules that describe the character personality and control the character behavior. All the characters share the same basic knowledge base to support interacting with the world and other characters. By the end of the whole experience, the learner would have experienced some emotional and moral complexities. This kind of experience, especially when theses complexities develop over the course of a game-like environment, can leave the player with a better and deeper understanding [10]. For more information about AEINS working model, see [15]. The following section focus on the narrative side in AEINS and how it addresses the currently existed shortcomings.

4 Interactive Narrative in AEINS

AEINS aims to allow students to move from the *making moral judgments state* to the *taking moral actions state*, from the knowing state to the doing state, which we consider a very important step in moral education. AEINS combines dynamically generated narrative and scripted branching narrative approaches in order to allow: (a) balanced user agency during the game interaction course; (b) the generation of a dynamic continuous story that exhibits a story arc; (c) the presence of virtual evolving agents that provide dramatic and pedagogical roles. The rest of this paper explains how AEINS manages to reach these aims.

Balanced user agency

The hybrid narrative generation can be considered as a further step in addressing the agency problem discussed previously. The problem lies in the trade off that always occur when combining learning and narrative-based games. In other words, the use of dynamic generated narrative allows high agency but overrides the educational side, and the use of graph structured narrative allows tracking the learning process but offers low agency. The blend of both techniques can act as a solution to this problem. It sustains balance between the participant's freedom and keeping track of the educational process. It allows student's high agency and increase his feel of control and in the same time, conserves the educational process and allows tracking and assessing the student's actions.

In AEINS, A STRIPS-like planner is used to generate the story in favor of the learning objectives. The student is able to act freely influencing and controlling the direction of the story. This freedom is allowed at any time outside the teaching moments (i.e., before reaching or after finishing a teaching moment). A more restricted agency is required during the interaction with the teaching moment to preserve the educational targets. This is acceptable because the teaching mo-

ments themselves are relaxed through varying the places and characters that can participate in their worlds; this part is illustrated in [15]. The dynamically generated story can be seen as the 'glue' that sticks the teaching moments together to form a continuous coherent story. This principle is very similar to a game play where the player is exploring the game environment and at certain points he has only one path to go through, in order to force him (implicitly) to perform the required tasks.

The recognized result obtained from the empirical evaluation is that the used techniques are integrated together in a seamless way managing the student's game and learning experiences. Although there are no current problems with the STRIPS planner, scalability issue could be raised when using bigger worlds.

4.1 Story Generation in AEINS

The story in AEINS starts with setting the scene for the user. The user starts by allowing the user to choose his friends from a number of virtual characters. The story unfolds in a logical coherent way based on the student actions. Climax is reached within a teaching moment. The events of a teaching moment story become complicated and through the user interactions the conflict in the story is revealed. The teaching moment has more than one end point that can be reached according to the student's actions. This is very similar to real life situations with a start, a climax and a resolve leading to an end. The whole story in AEINS can be seen a one long story that is interleaved with various situations and the student is an active participant affecting how the story unfolds.

Dynamic continuous story generation

AEINS uses planning for its efficiency and its ability to generate different narratives for different users, and also different narratives for the single user on subsequent play turns [27]. This technique enhances the user's sense of control in the narrative environment [27]. The main story in AEINS is generated using a STRIPS-like representation planning algorithm, similar to the work of Barber and Kudenko [2] that selects a story event to be executed based on a set of authored story actions, see figure2. However, AEINS extended the used planner by implementing a reactive planner that allows the agents reactions to be more relevant to the student's actions and keep the sense of realism.

action name	preconditions	postconditions
move(Y,Z)	place(Y), charat(stud,Y)	place(Z), charat(stud,Z)
invite(X,Y,Z)	char(X), place(Y), charat(X,Y)	place(Z), charat(X,Z)
make_friend(X,Y)	char(X), char(Y), like(X,Y)	friend(X,Y)
play_with(X,Y)	friend(X,Y)	enjoy_play_time(X,Y)

Figure 2. Example of the story world operators

The presence of reactive planner in AEINS offer the agents the opportunity to act according to the latest student's action instead of only basing the action choice on the whole past history of the narrative, as the STRIPS planner usually does. Imagine the following situation, the planner picks 'be_friend.to' and 'move' actions to be executed. Luckily, the student's follows the plan and chooses to be_friend.to one of the agents. In normal planning this will lead to the execution of the move action automatically. Through the use of the reactive planner, the agent responds first to the be_friend.to invitation then the reactive planner hands over the role to the STRIPS planner to

continue the execution of the original plan. We argue that this kind of prioritizing the latest user's action increases the believability of the agents and keeps the sense of realism of the world. It is worth to mention that the reactive planner does not interfere with the original plan. In other words, the reactive actions can not invalidate the original STRIPS planner, see figure3.

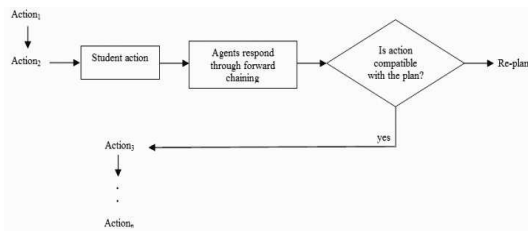


Figure 3. Plan execution flowchart

action name	preconditions	postconditions
reply_friendship (Agent, stud, N ₁)	like(Agent, stud)	agree_to_be_friend & friend(Agent, stud)
respond_to_play (Agent, stud, N ₂)	not(current_TM(TM ₂)) & friend(Agent, student)	accept_play_invitation

Figure 4. Example of the reactive planner operators

The reactive planner is used by the non-playing characters (agents) and uses the same actions representation as the STRIPS planner with an increment of one more parameter denoting the suitability cost. The suitability cost (#N) is an integer that guides the agent to choose the most suitable response action to the current student action. An example of two action operators for the reactive planning is shown in figure4. The N value changes dynamically during run-time through forward chaining rules based on the story events. The advantage of using forward chaining rules within the reactive planner is that it allows the reception of new data and can trigger new inferences, which makes the engine better suited to dynamic situations in which conditions are likely to change. For example, as the relationship between the student and the agents inhabiting the world evolve over time, agents reactions towards the student need to suit the current conditions of the world. The more the action is a logical response to the student actions the more the story is believable.

At the level of story generation, a STRIPS planner is used for the dynamic generation of narrative in the run-time where, currently, every user's action that violates the plan, can be accommodated by re-planning. Extending the STRIPS planner through the use of the reactive planner have had its effect on the user's experience as it provides more realistic and dramatic story. An example that illustrates the narrative generation can be seen in figure6. In the example, the first row represents the current story world and the last row represents the goals to be satisfied. The left column shows the first plan the story generator produces, the actions in italic are assumed learner's actions. If the learner's action does not satisfy the first plan, another plan is developed; the second plan in the middle column. Again if the learner's action violates the plan, a third plan is developed; the plan in the right most column. This continues till the goals (teaching moment narrative preconditions) are satisfied. This representation allows the use of an intelligent tutor to follow

the student's actions and to make an assessment for them in a step by step follow up. Ideally, each teaching moment path describes an inquiry-based narrative, a story in which the protagonist is the user in the role of making moral decisions. These dilemmas allow students to pursue different procedures for solving the problem. These various procedures arise from allowing different perspectives based on students perceptions and interpretations of the nature of the problem. Through role playing and discussions, students can see that their decisions affect other people and things [24]. In addition, the students' understanding gained through this process is situated in their experience and can best be evaluated in terms relevant to their experience.

Scripted narrative

Although dynamic generated narrative was used to generate the stories connecting different teaching moments together in a one continuous coherent story, it was practical not to follow the same technique in designing the teaching moments, where it is very important to conserve the educational targets. Branching graphs were used to develop the teaching moments solution spaces, see figure 5. This data structure allows the presence of a set of reasoning lines that a student may pursue. This helps the system to decide which line of reasoning underlies a student's action [4].

Despite the different branches of every teaching moment are hand coded, each teaching moment exhibits variability through allowing different characters and places to present the teaching moment depending on the story world state. Each teaching moment represents a part of the whole story and focuses on teaching a certain concept (value). Each teaching moment has certain prerequisites that must be fulfilled before the execution of the teaching moment takes place. The teaching moments priority is computed via pre-defined rules as follows:

*Trigger: teaching moment X₁ has not been presented
and teaching moment X₂ has not been presented
and value Y is not held by the user
and value X is held by the user
Action: set priority to teaching moment X₂*

The capital letters in the rules represent variables and the representation denotes that if (a) a specific pattern of teaching moments has not been presented to the student yet; (b) the user holds certain values and does not hold others, the action part of the rule executes (the next teaching moment priority is identified).

Interaction with teaching moments

Within every teaching moment, the learning tasks are tightly coupled with the narrative where specific skills (goals) have to be acquired by the student. The interaction of the student with the teaching moments is monitored and evaluated by the cognitive tutor. Such interaction has the following structure:

- The teaching moment begins with a specific theme to act as a starting point or trigger for learning.
- Generate possible problem solutions (ethically approved solutions).
- Assess the viability of alternative solutions by constructing arguments and articulating beliefs.
- The student is asked about his opinions through series of questions
- Evaluate the students answers and construct arguments that lead

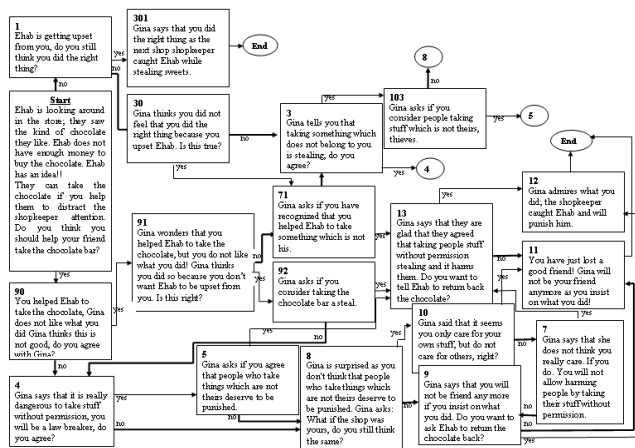


Figure 5. Example of a branched teaching moment graph

the students to examine the validity of his opinion or belief.

- If the student agrees with a desired ethical choice, the teaching moment ends
- Else if the student sticks to an unethical decision, the system raises the ante (“what if” style questions).
- If the student keeps sticking to the unethical choice, choose another teaching moment and repeat the above steps.

The teaching moments have two types of preconditions; tutoring and narrative. The teaching moment is selected according to the current student model (tutoring precondition) and is presented after the system satisfies the story preconditions (narrative prerequisites) through the planner (story generator). Within the teaching moment the learning tasks are tightly coupled with the narrative where specific skills (goals) have to be acquired by the student.

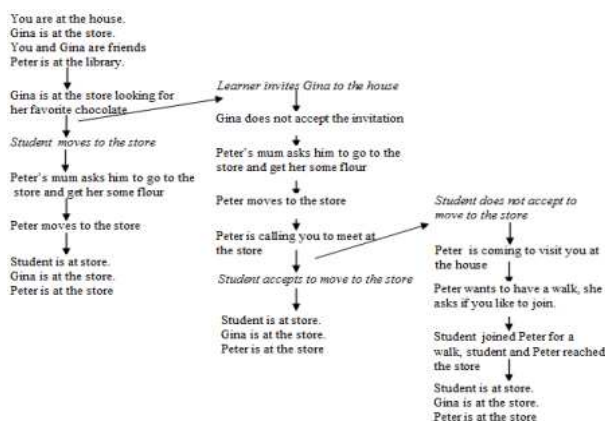


Figure 6. Example of the generated narrative

Dramatic and pedagogical roles of the evolving virtual agents

The game nature of AEINS allows the existence of non-playing characters (pedagogical agents) and objects in AEINS story world.

The purpose of pedagogical agents is not to perform tasks for users or to simplify tasks, but rather to help users learn how to accomplish tasks [30]. They aim to increase problem solving effectiveness by providing students with customized advice [19]. The pedagogical agents in AEINS are semi-autonomous agents, where on one hand they are able to act and react according to their state and the current world state. On the other hand, the story generator can dictate them, when required, what to do in order to preserve the coherence and dramatic tension of the whole story. The presence of a continuous story with characters personalities evolving during the story unfold helps with the mental and emotional engagement of the student.

The AI of the non-playing characters is represented in the form of rules, these rules can be modified during the story course as a result of certain actions. The main advantage of having more than one non-playing character is to have the freedom to portray agents who do not share the student's goals, for example provide negative examples [31]. On the other hand, they can also act according to the moral goals and give positive examples or help the student to stay on the right track. For example, a character who is a friend to the student can become an enemy as a result of a student action or a holder of an unethical moral virtue character can become a holder of a good moral virtue as a result of some interactions with the surrounding world.

The student and the agents are responsible for the story unfold as it is generated based on their actions. When it is time to present a teaching moment, the currently involved agents in the main story will take the corresponding roles (that fits their current personalities and relationship to the student). If there exist a role that is still needed to be occupied or an agent is not capable to take that role, the story world with the assistance of the story generator will allow the inclusion of another agent smoothly through the narrative. Once the scene is set, the teaching moment starts.

As mentioned previously, the predominant teaching pedagogy is the Socratic Method. The holder of the good moral virtue uses the Socratic Voice to provide discussion, hints and feedback to the student. The text dialog produced encourages the student to think critically in order to solve the discrepancies encountered in the moral situation(s) they are facing. In addition, students have opportunities to choose among different options and to reason which criteria lead to the chosen option [18]. When the teaching moment ends, the student along with the agents are free to act again influencing the story. The story world receives the required agents' actions to be executed and displayed to the student by the presentation model.

5 Evaluation

This paper shows how AEINS successfully worked on the drawbacks mentioned in the related work in the area of the development of narrative-based educational games to teach in the ethics domain. The main narrative problem discussed was the implementation of narrative-based educational systems that either uses continuous planning and loses some aspects of the educational process such as keeping track of the learning process and being able to assess the student or uses the graph structured stories and constrains the student's freedom in order to maintain the educational goals. AEINS succeeded in overcoming this by integrating both continuous planning and branching planning approaches. The former was used to generate the outer story that links the teaching moments together and forms a coherent, continuous story. The latter has been used in structuring the teaching moments and allows the use of an intelligent tutor.

A full study has been completed to test AEINS for different criteria such as the technical infrastructure, functioning, ability to support

or enable specific activities, and generation of predicted educational outcomes. We chose qualitative approach rather than quantitative one as it allows the use of semi-structured interviews which provide flexibility in dealing with children and gives them the chance to explain themselves and their thoughts freely and without being constrained to any gorges.

The study was conducted on a voluntary group of 20 children aged 7 to 12 years to test the hypothesis of building an educational game that integrates multiple narrative techniques and an intelligent tutor techniques in order to provide personalized learning process in the ethics domain. The students were of different origins and had different cultural backgrounds, for example Egypt, UK, China, Malaysia and Fiji. In each assignment, the student has been left to explore and interact with the system at their own pace. The student is not merely learning about a process or concept undertaken by an ethics teacher, but he experiences that process himself.

To evaluate AEINS, post interviews were conducted focusing on the system, its interface and the experiences of playing. The first category includes questions related to the technical infrastructure and its functioning. The second category includes questions related to the functions and features inherent in the system and its ability to support or enable a specific activity. The third category includes questions related to the student tasks. The fourth category includes questions related to the capability for specific technology-based activities to generate predicted outcomes. And finally the fifth category includes questions related to re-playability and self reflection. Results from this evaluation are presented below for the first time.

Students did not experience any interruptions in the narrative as the story was generated smoothly and successfully accommodating the teaching moments in a seamless way without affecting the student's experience. Accordingly, we can say that AEINS manages to give the student the appropriate freedom and is able to track and assess the learning process within a single generated narrative.

The evaluation shows that children appreciate the social characteristic in the system, as they were able to recognize the genuine social aspects and the realism represented in the game. The analytical questions confirm this recognition. For example, the presence of some verbal statements during and after the session that show some students' opinions, from a personal perspective:

P1: "The best moment was when my parents and my teacher were proud of me because of what I did."

More evidence that the students' were recognizing the social situation and recognizing the non playing characters as real friends. One of the interviewees said

P2: "I felt as if I am in a real world and these characters are really talking to me, their reactions were very believable."

another student said

P1: "I did not mean to upset my friend, I felt as if it really happened and I had lost my friend who would not talk to me ever again. I think I will be careful next time."

Actually, what was really interesting is the way the students personalize the non-playing characters in the game. They do not only interact with them as their friends in the game but also they gave them lives and pictured how these characters would behave beyond these moments. For example, one interviewee said

P5: "I do not like Gina when she lies, I want to tell her that this is wrong and she has to stop lying"

, the interviewee added

"If she keeps doing this now, no one will believe her in the future."

The students also believe the non-playing characters personalities, they like some and dislike others. One student said

P7: "I like Peter the most, he is funny."

another student said

P8: "I do not like Gina, she is not a real friend. She always ask me to do wrong things."

The realism present in AEINS allows the students to think about the non playing characters as real friends who can feel and expect certain actions from them. For example, one student quoted

P9: "If I choose to be on the side of one friend, the other one could become angry."

Moreover, the students were treating the situations as real ones and responding to them in a realistic way, for example one student said "I found the homework situation very confusing." and when he has been asked why, he replied

P10: "When my mum called me to see the TV, I was *scared* as I still have homework to do and the teacher will figure out the next morning."

These results reflect an important point that the students were able to react to the aspects of the domain and to apply their current and potential capacities in this game. From the students' answers, we have figured out that most of them were not treating the game as just a game, they do respect and appreciate the interactive narrative generated. The game was able to generate an engaging continuous story that is interleaved with difficult situations (graph planned teaching moments) within which the students tried to prove themselves and use their skills in order to solve the discrepancies faced.

What has been also observed that the game is not giving certain skills but it empowers the students to use the skills they have. The students were able to recognize their abilities and skills, they feel proud when they succeeded in solving problems and support their friends.

P1: "It was really nice solving my friend's problems."

P1: "it is good to feel that your friends rely on you, and ask you for help when they needed to."

These results go well beyond the educational theories that state the importance of stories in transferring tacit knowledge and speak about relationships and human connections. In addition, being involved in stories and moral dilemmas help in emphasizing the moral behavior and give the students' the chance to take different roles and experience various situations.

6 Conclusion

There has been significant interest in deploying narrative in education. Narrative provides a good medium to transfer tacit knowledge, create enriching experiences for its users, and sparking problem-solving skills. Many narrative-based educational systems have been discussed highlighting the shortcomings existed. An adaptive educational interactive narrative system, AEINS, has been introduced that

addresses those shortcomings and aims to help teaching in the ethics domain.

AEINS integrates multiple interactive narrative techniques and an intelligent tutoring technique in order to provide an adaptive engaging learning experience. Two types of narrative were used; the first type is graph structured teaching moments with non-interactive story presentations that are interleaved with student's decision points. The teaching moments are selected and presented in a way that allows the student to draw analogies from one problem to another (if needed). The second type is the run-time dynamic generated narrative, which aims to connect the teaching moments together and form one continuous story.

AEINS offers a safe environment that encourages and recruits situated, experiential, and embodied forms of learning and thinking. It offers the required privacy and an environment that the children can explore. It allows the inclusion of different dilemmas that the student can interact with and learn from, and hopefully transfer what he learns to the real world. Most importantly it offers learning at the student's pace and adaptation that provides personalized teaching and feedback.

The evaluation of AEINS shows positive results starting from being engaged with the environment passing through interacting with an adaptive learning process to developing new ideas and thoughts about basic ethical values. Overall, we believe this research provides students with a practical means of exploring abstract issues in concrete settings and allows students to practice making ethical decisions in a realistic context.

REFERENCES

- [1] L. S. Zettlemoyer, S. Y. Lee, J. C. Lester, B. W. Mott, C. B. Callaway and M. Rukeyser, 'Towards narrative-centered learning environments', in *Narrative Intelligence: Papers from the 1999 Fall Symposium, Menlo Park, CA: American Association for Artificial Intelligence*, eds., In M. Mateas and P. Sengers, pp. 78–82. Press, (1999).
- [2] H. Barber and D. Kudenko, 'Adaptive generation of dilemma-based interactive narratives', in *Advanced Intelligent Paradigms in Computer Games Series: Studies in Computational Intelligence*, volume 71. Springer, (2007).
- [3] G. Bolton, *Acting in classroom drama: A critical analysis*, Heinemann, London, 1999.
- [4] C. Conati and K. VanLehn, 'Probabilistic plan recognition for cognitive apprenticeship', in *Proceedings of the 18th annual conference of the Cognitive Science Society*, (1996).
- [5] G. Dettori and A. Paiva, 'Narrative learning in technology-enhanced environments: An introduction to narrative leaning environments', 55–69, (2009).
- [6] D. H. Elkind and F. Sweet. Good character. <http://www.goodcharacter.com/>.
- [7] D. H. Elkind and F. Sweet. The socratic approach to character education, 1997.
- [8] C. A. Willard, J. Woods, F. H. Van Eemeren, D. N. Walton and D. Zarefsky, *Fundamentals of argumentation theory: A handbook of historical backgrounds and contemporary developments*, Mahwah, NJ: Lawrence Erlbaum Associates, 1996.
- [9] R. Figueiredo, A. Brisson, R. Aylett, and A. Paiva, 'Emergent stories facilitated: An architecture to generate stories using intelligent synthetic characters', in *1st Joint International Conference on Interactive Digital Storytelling (ICIDS 2008)*, (2008).
- [10] D. Freeman, *Creating Emotions in Games. The Craft and Art of Engineering*, New Riders Publishing, 2004.
- [11] N. Gordon and M. Brayshaw, 'Inquiry based learning in computer science teaching in higher education', *E-Journal, ITALICS*, 7(1), (2008).
- [12] D. Gomboc, A. Karnavat, H. C. Lane, M. G. Core and M. Rosenberg, 'Intelligent tutoring for interpersonal and intercultural skills', in *Interservice/Industry Training, Simulation and Education Conference (ITSEC)*, pp. 1–11, (2007).
- [13] C. Merchant, H. Vilhjalmsson and P. Samtani, 'Social puppets: Towards modular social animation for agents and avatars', in *Lecture Notes in Computer Science*, volume 4564/2007, pp. 192–201. Springer Berlin / Heidelberg, (2007).
- [14] W. G. Harless, 'An interactive videodisc drama: The case of frank hall', *Journal of Computer-Based Instruction*.
- [15] R. Hodhod and D. Kudenko, 'Interactive narrative and intelligent tutoring for ill defined domains', in *Proceedings of Ill-Structured Domains Focusing on Assessment and Feedback Workshop, held at the 9th Int. Conf. on Intelligent Tutoring Systems (ITS'08)*, (2008).
- [16] R. Hodhod and D. Kudenko, 'Aeins: Adaptive educational interactive narrative system to teach ethics', in *Intelligent Educational Games Workshop held at the 14th Int. Conf. on Artificial Intelligence in Education (AIED09)*, (2009).
- [17] R. Hodhod, D. Kudenko, and P. Cairns, 'Character education using pedagogical agents and socratic voice', in *Proceeding of FLAIRS2010: The 23rd International FLAIRS Conference*, (2010).
- [18] D. Kuhn, 'Science as argument: implications for teaching and learning scientific thinking', *Science Education*, 77(3), 319–337, (1993).
- [19] S. Converse, B. Stone, S. Kahler, Lester, J. and T. Barlow, 'Animated pedagogical agents and problem-solving effectiveness: A large-scale empirical evaluation', in *Proc. of the 8th World Conference on Artificial Intelligence in Education*, (1997).
- [20] T. Lickona, E. Schaps, and C. Lewis. Eleven principles of effective character education, 2007.
- [21] J. Lo, N. Ji, Y. Syu, W. You, and Y. Chen, 'Developing a digital game-based situated learning system for ocean ecology', 51–61, (2008).
- [22] D. Dini, M. O. Riedl, A. Stern and J. Alderman, 'Dynamic experience management in virtual worlds for entertainment, education, and training', 4(2), (2008).
- [23] B. S. Magerko, *Player Modeling in the Interactive Drama Architecture*, Ph.D. dissertation, Department of Computer Science and Engineering, University of Michigan, 2006.
- [24] J. L. McBrien and R. S. Brandt, *The Language of Learning: A Guide to Education Terms*, Association for Supervision and Curriculum Development, Alexandria, 1997.
- [25] T. C. Ormerod, *Planning and ill-defined problems*, London: Psychology Press, 2006.
- [26] M. Riedl and A. Stern, 'Believable agents and intelligent story adaptation for interactive storytelling', in *3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, (2006).
- [27] M. O. Riedl and R. M. Young, 'From linear story generation to branching story graphs', *IEEE Comput. Graph. Appl.*, 26(3), (2006).
- [28] S. Lee, S. McQuiggan, J. Rowe and J. Lester, 'Story-based learning: The impact of narrative on learning experiences and outcomes', in *Proceedings of the 9th Int. Conf. on Intelligent Tutoring Systems (ITS-2008)*, (2008).
- [29] D. E. Simpson, 'Dilemmas in palliative care education', *Palliative Medicine*, 1(2), (1998).
- [30] E. Sklar, 'Agents for education: when too much intelligence is a bad thing', in *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 1118–1119, Melbourne, Australia, (2003). ACM, New York, NY, USA.
- [31] J. M. Thomas and M. Young, 'Becoming scientists: Employing adaptive interactive narrative to guide discovery learning', in *AIED-07 Workshop on Narrative Learning Environments*, (2007).
- [32] A. Waraich, 'Using narrative as a motivating device to teach binary arithmetic and logic gates', *SIGCSE Bull.*, 36(3), 97–101, (2004).
- [33] C. E. Watson, 'Using stories to teach business ethics: developing character through examples of admirable actions', *Journal of Teaching Business Ethics*, 7(2), 93–105, (2003).
- [34] S. A. Weiß and W. Miller, 'The potential of interactive digital storytelling for the creation of educational computer games', in *Book Series Lecture Notes in Computer Science*, pp. 475–486. Springer Berlin / Heidelberg, (2008).

Using a Cognitive Architecture for Opponent Target Prediction

Simon Butler¹ and Yiannis Demiris¹

Abstract. One of the most important aspects of a compelling game AI is that it anticipates the opponent's actions and responds to them in a convincing manner. The first step towards doing this is to understand what the opponent is doing and predict their possible future actions. In this paper we show an approach where the AI system focusses on testing hypotheses made about the opponent's actions using an implementation of a cognitive architecture inspired by the *simulation theory of mind*. The application used in this paper is to predict which target the opponent is heading towards, in an RTS-style game. We improve the prediction accuracy and reduce the number of hypotheses needed by using path planning and path clustering.

1 INTRODUCTION

The use of human-like artificial intelligence (AI) in complex computer and video games has the potential to provide gamers with not only the ideal opponent, but also helpful supporting non-player characters or even just useful predictions of the opposition's strategy in a multi-player game [16]. To achieve this, one avenue to explore is to enable the AI system to recognise and predict the opposing player's actions and movements using observations of their units, obtained from within the game. Once the opponent's objectives can be estimated, the system has a better chance of selecting appropriate moves for the computer-controlled units.

Traditionally, the AI systems in commercial video games were based on scripted events or unwieldy finite state machines, which were time-consuming to construct and error-prone [24, 21]. More recently, games have begun to make use of planners to decide on the computer-controlled player's actions [20], however, research into making predictions about the opposing player is still in its infancy. We propose that taking an approach inspired by neuroscientific and psychological data may provide a more principled way to design parts of the AI decision making system that can predict a player's behaviour. This could lead to an AI that produces more realistic results and, ultimately, a more enjoyable game.

We propose that an interesting way to create an AI that attempts to predict the opponent's actions is to base it on theories of how the human mind operates. The benefits of such cognitive architectures are that they can:

- infer possible mental states of the opponent;
- adapt the performed behaviour, based on such inferences;
- be scalable to the amount of computational resources available;
- have a well-defined architecture, which makes the complexities more manageable.

In this paper we discuss how we adapted a cognitive architecture based on the *simulation theory of mind* to opponent prediction within a gaming scenario. The system is designed for any application that can make use of predictions of multiple units from observations. Here, we have limited the scope to just display the most likely target for each of the opponent's units in a multi-player RTS-style (Real-Time Strategy) game. We detail the techniques we used to improve the accuracy of predictions and reduce some of the computational cost of using this approach.

2 BACKGROUND

It can be seen that recognising and predicting the goals and intentions of the opponent is essentially a problem of matching the state (the situation of each of the players and the game environment at a single point in time) to that produced by a model, where the model represents a possible goal or action. It is also thought that something similar to model-matching happens in the human mind too, through an ability known as *theory of mind*, which attributes *mental states* (beliefs, desires, intentions, etc.) to others and uses them to understand how others will behave [19].

2.1 Theory of Mind

Psychology gives us some clues as to how we, as humans, perform theory of mind, i.e., how we create these models and use them to 'read the mind' of other people. There are two main competing categories of explanations for this cognitive function: *theory theory* and *simulation theory*. Theory theory suggests that the mind builds up a set of rules and laws that are used to give us 'common sense' theories on how others will behave. On the other hand, simulation theory suggests that we do not use explicit theories, rather, we perform a mental simulation (i.e. imagine) of how we would respond if we were put in the situation of the opponent and use that to come to a prediction or explanation [19, 10, 9, 18].

These two possible mechanisms behind theory of mind are broadly analogous to the two main approaches found in the games, agent and robotics domains for the problem of plan recognition and prediction. The first approach matches the game states to generalised templates or rules, which are collectively known as *descriptive* models, and this is somewhat related to theory theory. The second is the *generative* approach, which is similar to simulation theory. It uses the current state to generate possible future states and performs matching against the generated state.

¹ Electrical and Electronic Engineering Dept., Imperial College London, UK.
Email: {simon.butler, y.demiris}@imperial.ac.uk

2.2 Descriptive Models

The descriptive approach takes the currently observed state information and, using various transforms, compares subsets of this transformed state with pre-existing descriptions in another state space. In other words, it uses the extraction of low-level features to match against representations created prior to the start of the scenario. For example, in the multi-agent domain, work has been done to analyse spatio-temporal traces for coordinated motion, whether moving apart or together [8], and, using a similar technique, spatio-temporal models can be created to encode group behaviour, then the traces matched to these models [22]. These approaches have a lower computational cost and may be more robust than generative models, however, more complex behaviours may not be able to be described using these types of models alone.

Other methods include using training data to construct a model. For example, in the games domain, the player's tactical behaviours are shown to be robustly recognised using a naïve Bayes classifier and are applied to directing a non-player character [25]. Also, in the domain of a football (soccer) match, explicit rules are extracted from training data [1] and applied to the current observations. These approaches make it easy to classify new situations, however they could be prone to over-fitting if only a limited amount of training data is available.

2.3 Generative Models

With the generative approach a set of latent (hidden) variables are introduced that encode the causes that *can produce* the observed data. Using these variables for a recognition and prediction task involves modifying the parameters of the generating process until the generated data can be favourably compared against the observed data.

This approach has been shown by using graphical models, such as Hidden Markov Models (HMMs), in the single-agent domain [3], and in the games domain [11]; by using Monte Carlo simulation [2] or by simply using a planner when the the number of choices can be limited [15]. A number of architectures have also been directly inspired by simulation theory by using internal models to perform prediction through simulation [7, 6].

One such approach (dubbed HAMMER—Hierarchical Attentive Multiple Models for Execution and Recognition of actions) uses a hierarchical system of inverse (plan) and forward (predictor) models, with each model pair (collectively known as internal models) being configured to a particular action [7]. The system finds the matching action by using the model that has the lowest error between the predicted state from each forward model and the actual observed state. This architecture has the advantage that it works well in on-line situations because it will always produce the most closely matching model at any point, rather than other approaches which make a selection then backtrack if it fails. However, it assumes a vast number of hypotheses can be evaluated in parallel, which may not always be practical. A review by [6] shows the applicability of the use of these internal models to multi-agent systems, and HAMMER has been adapted to be used in a multi-agent fully-observable predictive system, as described in [4]. Related work by [23] on segmenting spatio-temporal traces of multiple agents paves the way for applying inverse models to distinct groups of agents.

2.4 SYSTEM

Of the two theories proposed by psychology (see Section 2.1), it is simulation theory has a large body of support from neuroscience

[13]. It is this biological-plausibility that has inspired cognitive architectures based on simulation theory, mainly in the robotics domain.

Simulation theory gains credence through research in several fields suggesting that 'thinking' consists of simulated interaction with the environment. The evidence outlined in [13] suggests that: when we think of an action we use the motor structures of the brain but actuation is deactivated; perceptual activity can be internally generated within the brain; and these two areas of the brain can be connected internally. Essentially, simulated behaviour causes perceptual activity that is similar to actually performing the behaviour.

It follows that if we try to recognise behaviour of another person, we assume that we are in their position and use this internal simulation mechanism to process possible actions and predict their response [14]. Not only does this theory explain how we infer how others will respond to certain situations, but it also attempts to explain the experience of an 'inner-world' within our brains where perceptions are not externally triggered. Also, this use of brain structures for both action and recognition is an attractive proposition from an engineering perspective as it allows subsystems to be reused for different purposes.

This theory conceptually maps quite well to a cognitive architecture for action recognition: another agent's intentions can be inferred by creating plausible plans (or hypotheses) that are consistent with a range of possible goals. These plans are, as time progresses, simulated to find the closest match with the observed behaviour. Subsequently, the objective that was used to create the best-matching plan is used as an estimate of the intent of the agent. This can be thought of as recognition through the generation of multiple competing plans [6].

Given the successful implementation of the simulationist approach to action recognition in the robotics domain [7] we decided to apply these principles to the games domain.

2.5 Simulationist-architecture

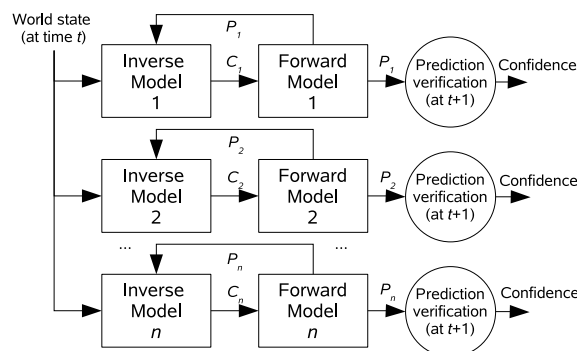


Figure 1. The HAMMER architecture implements the principles of the simulationist approach. Multiple inverse models receive the world state and suggest possible commands (C_1 - C_n), which are formed into predictions of the next world state by the corresponding forward model (P_1 - P_n). These predictions are verified on the next time step, resulting in a set of confidence values.

Starting at the lowest level, taking a simulationist approach to recognition and prediction of plans requires a method to generate and evaluate certain primitive actions that players can perform. The HAMMER architecture provides a starting point for the system (see Figure 1). It is comprised of three main components: the **inverse**

models (plan generators), the **forward models** (predictors) and the **evaluator** [7, 6].

An inverse model (IM) takes the current world state (which includes the location of each of the units the player controls), a set of units to control, and, optionally, target goal(s) or other parameters. It outputs the required waypoints or other control signals (a *plan*) that, according to the model, are necessary for each unit to perform so that they collectively achieve the implicit or explicit target goal(s). Each parallel instance of an inverse model is paired with an instance of the forward model (FM) that provides an estimate of the events that will occur if the generated plan is followed. At each time step this estimate is returned to the inverse model to tune any parameters of the actions to achieve the desired goal(s).

To determine which of these inverse/forward-model pairs most accurately describes the events that are occurring, on each timestep the output of each forward model is compared with the actual world state. These comparisons result in confidence values that behave as an indicator of how closely the observed events match each particular prediction, and they are subsequently accumulated over time until such a point that one model pair achieves a clear separation from the others. This model can then be simulated further into the future to provide a prediction of upcoming events.

2.6 Hypotheses and Internal Models in Games

The inverse and forward models used in this architecture are application-dependent. In the games domain, the inverse models will likely depend on the type of plans and actions available to each of the units—fortunately it is very likely that these models already exist as a library of actions a computer-controlled player can perform. For example, the *go-to* IM could output a series of ideal waypoints to route a unit to a specific goal position, or the *formation* IM could simulate repulsive forces between units that updates the target positions of the units to keep them in a specific formation.

The forward models will depend on the game dynamics, and could be statistical models or simplified physics models. However, these may have fairly low fidelity as, for instance, the actual trajectory that the unit will take will be dependent on the interpolation used between the waypoints generated by the IM, the type and gradient of the terrain, and any use of local obstacle avoidance; or there may be some dynamic team behaviour, such as the unit may have to maintain a position in a formation. The effect of these factors cannot necessarily be easily encoded in a statistical model, therefore, the simplest way to get a high quality forward model is to use multiple instantiations of the game environment itself. This enables each instantiation to be fed different hypothesised actions of the opponent's units from the inverse model, and for them to return a predicted state to be compared with the actual state of the game. Hence, we chose to use the game engine to simulate the outcome of the inverse models for greater prediction accuracy.

Several other aspects of the HAMMER architecture have to be modified for application to the games domain. Firstly, the single timestep interval between predictions needs to be lengthened because the timestep of a typical game is less than 30ms. Secondly the objectives of the opponent may change at any time, so simply accumulating the confidence values of each model could mean a long latency before recognising a change in objective. Therefore, instead, the confidences are averaged over a time-window and the model with the resulting highest confidence is considered to be the best prediction.

In this paper we define a plausible action for a unit to be a *hypothesis*; a hypothesis that has been assigned a unit and parameters

to be an *instantiated hypothesis*; and the execution of a hypothesis instance to generate a predicted state to be an *internal simulation*.

3 IMPLEMENTATION

The architecture is not limited to any particular game genre, however the predictive power of the system is best illustrated with a game that has an inherent strategic element. Hence we used our own RTS-style game as an implementation platform.

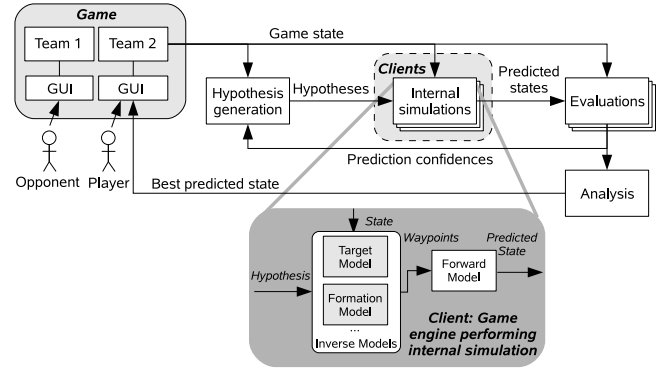


Figure 2. System implementation. The HAMMER architecture implemented into a game framework.

The implementation of the system is divided into three main sections (as shown in Figure 2): the player-controlled game instances that host each team of units (tanks or soldiers in this case); the clients running the game engine in ‘hypothesis’ mode, executing the requested internal models; and the hypothesis manager that is responsible for generating hypotheses based on the game state, sending the state to the client at the required time and evaluating and analysing the predictions to present the results to the player.

The game hosts two teams of units, n_o opposing units (the ‘red’ team) and n_i targets for the opposing units (the ‘blue’ team). Each team is controlled by a single player. Hypothesis instances are created by the *hypothesis generator* and are assigned a specific IM with any optional parameters to achieve the goal, a set of units to apply it to, a duration, and a speedup parameter. These units are then simulated by the FM (in faster than real-time, as controlled by the speedup parameter) using the control signals generated by the IM (the *inverse-forward model pairs* block). The predicted positions of the units and the actual positions are compared (the *evaluations* block) after the specified duration—the result of which is used to calculate the confidence that those units are achieving the goal. The best performing hypotheses are analysed (the *analysis* block) and a future predicted state is sent back to the game to be presented to the player.

The hypotheses instantiated by the hypothesis generator form sets H_s^u where $u = 1..n_o$ and $s = 1..S$ where S is the number of different hypothesis types (i.e., the number of available IMs). H_s^u consists of $h_s^u(\mathbf{b})$ which are instantiated hypotheses of the same type and applied to the same unit, but differ in their parameters $\mathbf{b}$.

3.1 Game Engine (and Forward Model)

The engine of the game is based on Delta3D [5], which is an open-source project to integrate various software libraries, such as Open

Scene Graph (OSG), Open Dynamics Engine (ODE), Character Animation Library 3D (Cal3D), Game Networking Engine (GNE), etc., into a coherent platform for simulation and games.

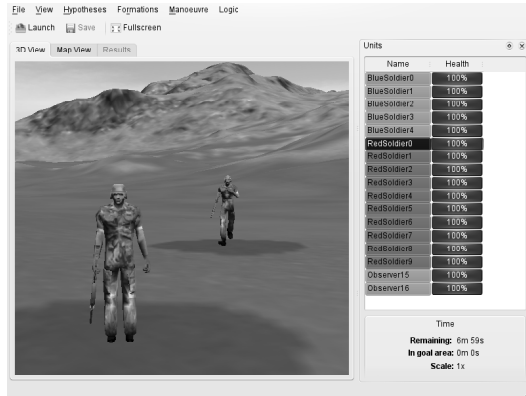


Figure 3. Game screenshot. This shows the graphical interface for the human-controlled teams.

The 3D engine (OSG) was used to model a large outdoor terrain (see the screenshot in Figure 3), with the height and other features (texture, trees, buildings, etc) of the terrain being displayed based on 2D feature maps. The physics engine (ODE) was used to accurately model the movement of the various characters and vehicles, with additional control of their aiming and firing mechanisms. The noise introduced by the physics engine provides a stochastic element to the outcome of scenarios.

When using the game, the players are responsible for choosing goal positions for their units, so they are free to perform manoeuvres and formations as they see fit.

To support the use of the game engine as a forward model, the engine has to support a few features not usually exposed with an external API. It needs to be able to read the complete current state of the environment and transfer that state to a new instance of the engine. This involves extracting the pertinent position, orientation and motion properties of each unit, along with dynamic and static attributes such as health and firing range. This information then needs to be serialised and sent to an independent instance of the engine, where the state information is initialised and executed for the specified duration, after which the resultant unit positions are returned to the main game engine.

3.2 Inverse Models and Parameterisation

Any reasonably complex game will have many different inverse models, and these need to be parameterised to cover the range of behaviour that each unit can perform, whilst reducing the number of models needed to a manageable level.

Most parameters are real numbers (e.g., positions, speeds, distances), but in many cases only a subset of these numbers need to be considered, which greatly reduces the number of IMs to execute. For example, a position parameter that could be used by a *go-to* inverse model could hypothetically be any point on the terrain. However, even if the possible target positions are reduced to a lower resolution, it is not necessary to test every position because certain targets are more likely than others. For example, units are likely to be either heading to: attack a group of opposing units, a good defensive position, or rendez-vous with another group of units. This can be taken

further by exploiting relationships between sets of parameters so only those that are sufficiently different need to be executed.

As an example we shall discuss optimising the target position parameter for the *go-to* IM in our RTS game. The simple approach is to instantiate an IM that sends the unit in a straight line towards the designated target, and to instantiate that IM for each of the targets in the game (we shall refer to this as the straight-line-to-target hypothesis). However, there is no guarantee that by following the straight line the unit is able to reach the target, and additionally, there is likely to be considerable duplication of effort if multiple targets lie on the same heading from the unit.

One way to address these flaws is to generate paths to each of the possible targets and then cluster them to find similar paths (we shall refer to this as the path-to-target hypothesis). Therefore only paths that are in the centre of the clusters need to be simulated. This approach is discussed in detail below.

3.2.1 Path Generation

In games, a popular way to calculate an optimal path between two points is by using the A* search algorithm, which uses heuristics to reduce the graph-based search space [12]. The heuristic is based on the minimal cost path to the target.

The A* algorithm can be summarised by first using the evaluation function $\hat{f}$ in (1) for each of the adjacent nodes of a starting node, then selecting the node with the lowest $\hat{f}$ score. This is repeated for each subsequent selected node until the goal node is reached, whilst maintaining a list of visited nodes to avoid loops. Details of the algorithm can be found in [12], but the main implementation detail is the evaluation function:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \quad (1)$$

where n is any node in the subgraph M_s that is accessible from the start node s , $\hat{g}(n)$ is the cost of the path from the start node s to n with the minimum cost found so far, and $\hat{h}(n)$ is the least-cost estimate from n to the goal node t .

In our implementation, we used the low resolution 512×512 pixel terrain heightmap grid as the graph M to search. For the least-cost estimate $\hat{h}(n)$ we approximate the minimum time it would take for the unit to travel to the goal position. We set the path cost function $\hat{g}(n)$ to depend on the speed of the unit and the type of terrain to traverse. It also depends on the gradient between each node, as each unit type has a limit on its maximum traversable gradient, hence some nodes may be unreachable to certain unit types.

3.2.2 Path distance metric

The A* algorithm ensures that relevant terrain features are considered when appropriate paths to targets are generated, however it still means that there could potentially be very similar paths to targets that are near to each other. These paths could easily be combined into a single hypothesis instance using a clustering algorithm to save simulating all of the possible paths.

Usually clustering algorithms operate on points in a two-dimensional space, using euclidian distance between points as a measure of similarity. However, in this application the paths form a more abstract space, so we have to define our own metric to decide how similar two paths are.

For our purposes the location of the destination is less important than how the paths diverge near the starting point. This is because the confidence values for a hypothesis are generated after a short

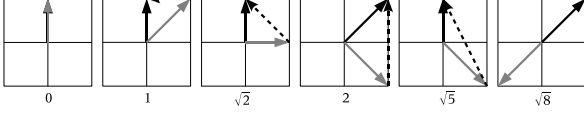


Figure 4. Examples for each of the possible path deviation scores. The vector for one segment of the first path (p_v^i) is shown in black and a segment from the other path (p_w^i) is in grey. The difference between the vectors is shown as a dashed arrow, and the path deviation score ($\|p_v^i - p_w^i\|$) is the magnitude of this vector, the value of which is shown below each example.

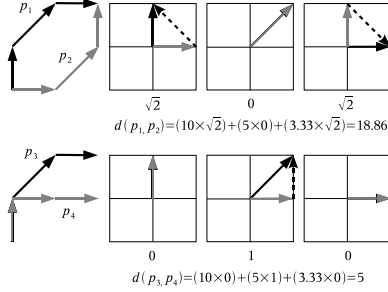


Figure 5. Two examples of the path distance metric being used on pairs of paths. For p_1 and p_2 the paths go between the same two points, but take different routes, which gives a high distance measure. Whereas for p_3 and p_4 the paths share a similar path before diverging and ending at different points, yet the distance score is low. Note that differences in the paths near the start of the path have a higher weighting, and low scores indicate good path similarity.

amount of time, so only paths that have diverged in that time can be differentiated.

The paths are made up of vectors that go to adjacent nodes in the terrain map, so we chose to compare paths using the magnitude of the difference between the two vectors for each segment along the path, i.e. a pointwise measure of path deviation (e.g. see Figure 4), up to the shortest of the two paths. A scale factor was added that decreases along the path, so deviations at the beginning have more influence over the path distance score, see Figure 5. Therefore the distance between two paths p_v and p_w is:

$$d(p_v, p_w) = \sum_{i=1 \dots m_{\min}} \alpha(i) \cdot \|p_v^i - p_w^i\| \quad (2)$$

where $m_{\min} = \min(|p_v|, |p_w|)$ ($|p_v|$ is the number of path segments in path p_v), and $\alpha(i) = \frac{10}{i}$. The constant in the α term was set experimentally.

3.2.3 Path clustering

Once we have a measure of the distance between two paths, we can use it to find the similarity matrix that compares each of the paths with all of the others. After that we can use several different clustering algorithms, however here we shall be using a spectral clustering algorithm that automatically chooses the number of clusters [17, 23].

From the similarity matrix we form an affinity matrix using the equation

$$A_{ij} = \exp\left(\frac{-d^2(p_i, p_j)}{\sigma^2}\right) \quad (3)$$

where $\sigma = 10$. The σ term controls the scaling, hence it affects how ‘close’ the items have to be to be considered to be in the same cluster. This term is set experimentally and only needs to be set once, depending on the order of magnitude of the similarity metric.

Details of the rest of the algorithm can be found in [17], but to summarise: the eigenvectors of the normalised affinity matrix are rotated to get block diagonals using gradient descent. The number of eigenvectors that are used determines the number of clusters, and this is done by finding the number of eigenvectors that produces the best quality rotation. The paths are then assigned to clusters based on the maximum dimension of the rotated eigenvectors.

3.3 Evaluation Process

Now that we have an inverse and forward model pair producing predicted unit positions, we need to evaluate which of these predictions match the observed behaviour.

After a hypothesis instance $h_s^u(\mathbf{b}_i)$ has finished executing it returns the starting position $\mathbf{x}_{\text{start}}$ (recorded at the beginning of the execution of a hypothesis), the actual position $\mathbf{x}_{\text{actual}}$ (recorded after the chosen hypothesis time), and the corresponding predicted position $\mathbf{x}_{\text{predicted}}$ (as calculated by the forward model).

From this information we can calculate the confidence function:

$$c(h_s^u(\mathbf{b}_i)) = \begin{cases} 1 & \text{if } |\vec{a}| < d \text{ and } |\vec{p}| < d, \\ \hat{a} \cdot \hat{p} \cdot \frac{\min(|\vec{a}|, |\vec{p}|)}{\max(|\vec{a}|, |\vec{p}|)} & \text{otherwise.} \end{cases} \quad (4)$$

where $\vec{a}$ is the vector from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{actual}}$ for unit u and $\vec{p}$ is the vector from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{predicted}}$ for unit u in hypothesis h , $\hat{a}$ and $\hat{p}$ are the normalised vectors, $|\vec{a}|$ and $|\vec{p}|$ are the magnitude of the vectors, and d is the deadzone that below which the unit is considered to be stationary (we used a value of $d = 1$ for our experiments).

This confidence function is designed to be able to distinguish whether the unit is moving towards or away the predicted position, or in some other direction. This is accomplished by normalising $\vec{a}$ and $\vec{p}$ (to get $\hat{a}$ and $\hat{p}$) and taking their dot product. This has the desired characteristics that if the unit moves towards or away from the predicted position then the confidence approaches 1 or -1 respectively, or if it moves perpendicular to the predicted position then the error is zero, where 1 means high confidence, 0 means unknown confidence, and -1 means no confidence.

The dot product term tells us what direction the unit travelled, but we would also like to know how close it got to the predicted position. However this needs to be invariant to the different speeds of the units, so that the error of different unit types can be compared. Therefore we scale the result of the dot product by the magnitude of the shortest vector ($\min(|\vec{a}|, |\vec{p}|)$), as a proportion of the magnitude of the longest vector ($\max(|\vec{a}|, |\vec{p}|)$).

An additional condition was added so that if the magnitudes of both vectors are less than d (in our case one metre) then the confidence is 1. This is to reduce results based on the heading when $|\vec{a}|$ and $|\vec{p}|$ are both very small, but $|\vec{a}|$ is non-zero, which occurs where the unit drifts or slides slightly. Therefore this term adds a dead-zone with a radius of 1m, within which the unit is counted as stationary, i.e. if we predict no movement and there is only a small movement, then we are confident of our prediction, regardless of heading.

It follows that the best matching hypothesis for a particular unit u and hypothesis type s is that with the highest confidence over all the tested parameters $\mathbf{b}$:

$$c_s^u = \max_i c(h_s^u(\mathbf{b}_i)) \quad (5)$$

4 EXPERIMENTS & RESULTS

To test the effectiveness of the path-to-target hypothesis that uses the path clustering (as described in Section 3.2), we compared it with the straight-line-to-target hypothesis.

The opponent ('red' team) has one unit, and has the objective to destroy one of the 3 targets (part of the 'blue' team) that are positioned throughout the environment.

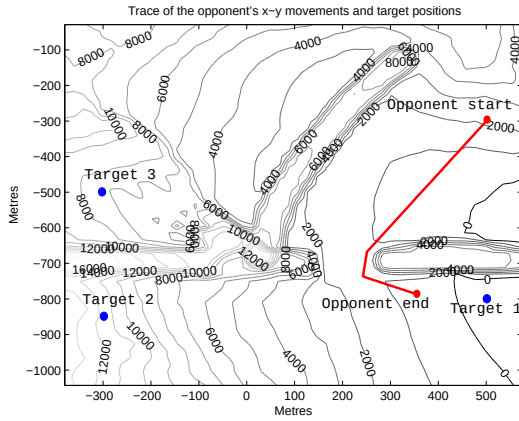


Figure 6. Scenario overview. This shows the movement of the opponent's unit, and the positions of the target units for the duration of the scenario.

The environment and location of the targets is shown in Figure 6. A barrier is shown going east-west across the environment with a small gap in the middle. Targets 1 and 2 lie beyond the barrier, one on either side of the gap. A third target lies behind a barrier that is only accessible by going around it to the north. The scenario is played out so that the opponent heads towards the gap in the barrier which leads to Targets 1 and 2, and then, at around timestep 130, it goes towards Target 1. Target 3 remains behind the barrier so that it requires the opponent to move in the opposite direction in order to reach it. The targets are static throughout the scenario.

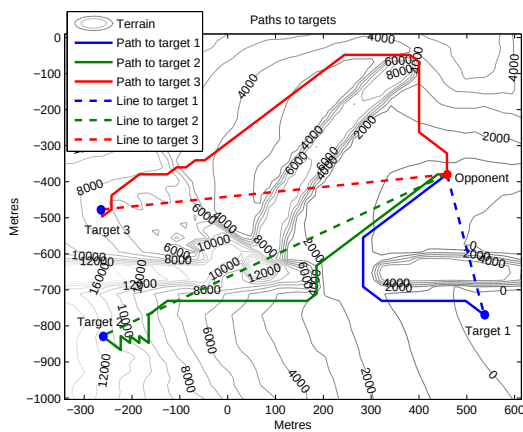


Figure 7. Predicted paths. This shows the paths for both hypotheses at timestep 41. It can be seen how the paths to targets 1 and 2 have a high path similarity, and that the straight-lines are quite dissimilar to the paths for targets 1 and 3.

The two hypotheses (path-to-target and straight-line-to-target) were instantiated for the opponent's unit, with the target positions as parameters. Therefore we have two hypothesis sets H_s^u where $u = 1$ and $s = 1, 2$, and each contains three hypothesis instantiations, one for each of the three targets, $h_s^u(b_i)$ where $i = 1..3$, making a total of six hypothesis instances to evaluate. The interval between observations is set to be 5 seconds, there is one client available for each of the hypothesis sets to execute the internal models, and the forward model contained within it is running at 4 times real-time. This means each each internal simulation runs for approximately 1.25 seconds to generate a prediction, therefore all instances in each set can complete within the interval. Hence, each hypothesis set generates confidence values for each target approximately once every 5 seconds.

The path planning and clustering occurs each time the path-to-target hypothesis set starts a new prediction, and it is used to decide which hypothesis instances (i.e., targets) need to be executed. The time taken to compute this is negligible (approx. 1ms) relative to the savings from any reduction in the number of hypothesis instances executed.

An example of the outputs of the inverse models for both of the straight-line-to-target hypothesis and the path-to-target hypothesis for each target at timestep 41 is shown in Figure 7. From this it can be seen how the paths to targets 1 and 2 have a high path similarity, and that the straight-lines are quite dissimilar to the paths for targets 1 and 3.

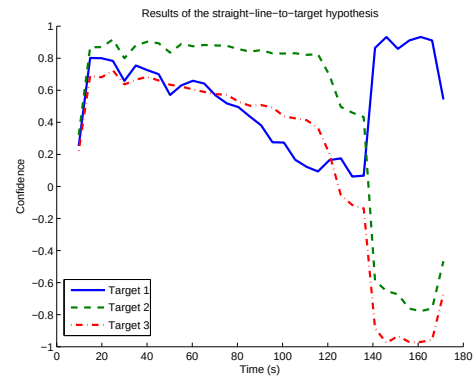


Figure 8. Target confidence using the straight-line-to-target hypothesis. Target 1 is only identified as the most confident target after timestep 140.

The confidence level to each of the targets using the straight-line-to-target hypothesis is shown in Figure 8. The highest confidence target is correct throughout the scenario, however, as expected, the terrain features are not considered so it misses the fact that the opponent also has a high likelihood of additionally heading to Target 2 up until timestep 130, and that Target 3 is an unlikely target as the shortest accessible path is in the opposite direction to the observed movement.

As can be seen from Figure 9 the path-to-target hypothesis correctly identifies and combines Targets 1 and 2 in the first part of the scenario, whilst giving Target 3 a low confidence. After the opponent traverses the gap it correctly splits off Target 1 and gives it a highest confidence, and combines the other two targets and assigns them a low confidence.

It can also be seen that the path-to-target hypothesis manages to out-perform the straight-line-to-target hypothesis whilst also execut-

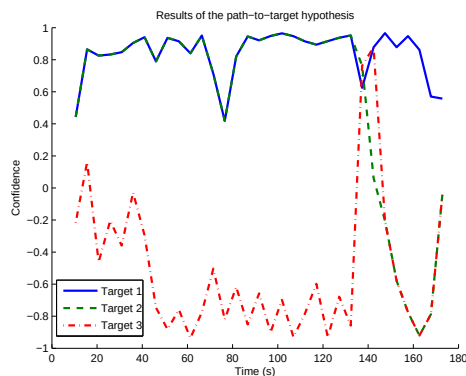


Figure 9. Target confidence using the path-to-target hypothesis. Target 1 has a consistently high confidence throughout the scenario.

ing the hypothesis with the need for fewer computational resources. Obviously it is not a surprise that path-planning improves the accuracy on an uneven terrain, but it does show that it can be done even when executing fewer internal models. This is because throughout the path clustering algorithm combines two of the paths that are similar, hence only paths to two out of the three targets need to be executed and tested.

5 CONCLUSION

In this paper we have discussed a cognitive architecture based on simulation-theory that is capable of providing predictions of an opponent's actions and intentions. We have shown some initial results on how this can be applied to the prediction of a single action (the selected target) of an opposing player in an RTS-style game. Through the use of A* path finding and spectral clustering of the resultant paths, the target predictions out-perform a naïve straight-line target predictor, whilst also successfully combining similar paths to avoid redundant executions.

The platform is attractive because it uses the same models for both perceiving and acting, therefore it has the scope to be useful as a basis for game AI systems that need to anticipate the opponent's behaviours and to use this information to execute its own behaviours. It also has the added benefit that these models are likely to be already available in a typical game, and, although the work shown here uses two simple inverse models, the same architecture can be applied to any action the units can perform.

However, even though we have shown a way to reduce the number of hypothesis instantiations, problems are likely to occur when many different hypothesis sets (i.e., different types of inverse model) need to be executed concurrently, so future work involves investigating other approaches to reducing this computational complexity, such as using hierarchical inverse models. Also, adding the constraint of partial observability to the system would reduce the number of hypotheses as they can only be executed when there are reliable observations. This would also making the game more realistic, as the opponent would avoid getting the impression that the AI is "cheating" by having a global view of the environment. Additionally, there is an assumption that the opponent will behave in a manner that is similar the output of the models—future work includes looking at methods to tune the inverse models to match the observed behaviour of the opponent.

REFERENCES

- [1] M. Beetz and B. Kirchlechner, 'Computerized real-time analysis of football games', *IEEE pervasive computing*, **4**(3), (2005).
- [2] Y. Björnsson and H. Finnsson, 'Cadiaplayer: A simulation-based general game player', *Computational Intelligence and AI in Games, IEEE Transactions on*, **1**(1), 4–15, (March 2009).
- [3] H. Bui, S. Venkatesh, and G. West, 'Policy recognition in the abstract hidden markov models', *Journal of Artificial Intelligence Research*, **17**, 451–499, (2002).
- [4] S. Butler and Y. Demiris, 'Predicting the movements of robot teams using generative models', in *Distributed Autonomous Robotic Systems* **8**, 533–542, Springer, (2009).
- [5] R. Darken, P. McDowell, and E. Johnson, 'The Delta3D open source game engine', *IEEE computer graphics and applications*, **25**(3), (2005).
- [6] Y. Demiris, 'Prediction of intent in robotics and multi-agent systems', *Cognitive Processing*, **8**(3), 151–158, (September 2007).
- [7] Y. Demiris and B. Khadhour, 'Hierarchical attentive multiple models for execution and recognition of actions', *Robotics and autonomous systems*, **54**(5), (2006).
- [8] M. Devaney and A. Ram, 'Needles in a haystack: Plan recognition in large spatial domains involving multiple agents', in *National Conference on Artificial Intelligence*, (1998).
- [9] V. Gallese and A. Goldman, 'Mirror neurons and the simulation theory of mind-reading', *Trends in cognitive sciences*, **2**(12), 493–501, (1998).
- [10] R.M. Gordon, 'Simulation vs theory-theory', *The MIT Encyclopædia of the Cognitive Sciences*, 765–766, (1999).
- [11] P. Gorniak and D. Roy, 'Probabilistic grounding of situated speech using plan recognition and reference resolution', in *Proceedings of the 7th international conference on Multimodal interfaces*, p. 143. ACM, (2005).
- [12] P. E. Hart, N. J. Nilsson, and B. Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *Systems Science and Cybernetics, IEEE Transactions on*, **4**(2), 100–107, (July 1968).
- [13] G. Hesslow, 'Conscious thought as simulation of behaviour and perception', *Trends in Cognitive Sciences*, **6**(6), 242–247, (June 2002).
- [14] M. Johnson and Y. Demiris, 'Perceptual perspective taking and action recognition', *International Journal of Advanced Robotic Systems*, **2**(4), 301–308, (2005).
- [15] J. Laird, 'It knows what you're going to do: adding anticipation to a quakebot', in *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pp. 385–392, New York, NY, USA, (2001). ACM Press.
- [16] J. Laird and M. van Lent, 'Human-level AI's killer application: Interactive computer games', *AI magazine*, **22**(2), 15, (2001).
- [17] Lihi Z. Manor and P. Perona, 'Self-tuning spectral clustering', *Eighteenth Annual Conference on Neural Information Processing Systems, (NIPS)*, (2004).
- [18] M. Michlmayr, *Simulation Theory versus Theory Theory: Theories concerning the Ability to Read Minds*, Master's thesis, University of Innsbruck, Mar 2002.
- [19] S. Nichols and S.P. Stich, *Mindreading: An integrated account of pretence, self-awareness, and understanding other minds*, Oxford University Press, USA, 2003.
- [20] J. Orkin, 'Three states and a plan: The AI of F.E.A.R.', in *Proceedings of the Game Developer's Conference (GDC)*, (2006).
- [21] B. Scott, 'The illusion of intelligence', in *AI Game Programming Wisdom*, ed., Steve Rabin, 19–20, Charles River Media, Inc., (2002).
- [22] G. Sukthankar and K. Sycara, 'Robust recognition of physical team behaviors using spatio-temporal models', in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 638–645, New York, NY, USA, (2006). ACM.
- [23] B. Takács, S. Butler, and Y. Demiris, 'Multi-agent behaviour segmentation via spectral clustering', in *Proceedings of the AAAI-2007, PAIR Workshop*, pp. 74–81. AAAI Press, (July 2007).
- [24] P. Tozour, 'The perils of AI scripting', in *AI Game Programming Wisdom*, ed., Steve Rabin, 541–547, Charles River Media, Inc., (2002).
- [25] W. van der Sterren, 'Being a Better Buddy: Interpreting the Players Behavior', in *AI Game Programming Wisdom 3*, ed., Steve Rabin, Charles River Media, Inc., (2006).

Towards Automatic Player Behaviour Characterisation using Multiclass Linear Discriminant Analysis

Robin Baumgarten¹

Abstract. The automated classification of player behaviour and a subsequent adaptation of game rules in video games is an area of research of interest to both the games industry and to academia, requiring a combination of AI, HCI, psychology, statistics, data mining and machine learning. In this paper, we describe an experiment to answer the question whether it is possible to measure and distinguish behaviour of a player playing a simple game with very limited inputs and a limited variety of possible actions. We use the arcade style action game *PacMan* to record a variety of game-related metrics in an online survey comprised of 245 players, each playing 5 sessions of *PacMan*. Then, the resulting data is analysed using discretization methods and linear discriminant analysis. The results indicate that differentiation between different players and different playing styles is possible, and that the methods used are suitable to automatically identify the most significant distinguishing features of player behaviour.

1 TEST BEDS FOR PLAYER BEHAVIOUR

A suitable game to research player behaviour on has to allow for a reasonable spectrum of different actions. Games with too limited input dimensions might not allow for sufficiently diverse behaviour patterns. This might make it harder or even impossible to separate player groups according to their actions. Examples of games with a low dimension of inputs are games with a very small state space such as *Noughts and Crosses* and *Rock-paper-scissors*, luck based board games such as *Ludo* and *Snakes and Ladders* and card games such as *Uno* and *Blackjack*.

On the other hand, games that are very complex and allow for a vast number of actions are not suitable for this project, because this freedom of choice often results in a high player behaviour complexity which might adversely affect the clustering process. That is, when a player has a lot of actions to choose from, it is harder to generalise behaviour patterns, because actions are repeated less often and game data (such as keyboard strokes or mouse input) is more diverse. This increased complexity in behaviour patterns might make it harder to separate points in the game data space that is to be clustered, thus requiring more dimensions through additional measurement metrics to mitigate this complexity.

Therefore, a game with an input dimensionality as large as required for a reliable differentiation in behaviour patterns but as small as possible to keep the clustering process efficient has to be found. As the development of video games has seen a steady increase in game complexity, recent titles are less likely to fall into this category than older titles. In fact, old arcade games such as *Pac-Man* and *Space Invaders* are both limited in its input dimensions but at the same time

provide sufficient freedom for the player to accommodate for different playing behaviours.

There are additional advantages when using such simple arcade games. In particular, because there is no to little commercial interest in most of these games, it is easier to get hold of games with supplied source code. There are dedicated web pages such as *Galaxy Arcade*², *Flash Game Code*³, and *Geheee*⁴ where source code of games can be downloaded.

Furthermore, using small games such as *Pac-Man* allows for online distribution as a browser-based flash game, which simplifies the dissemination of our survey and broadens the target audience significantly. However, Internet based surveys restrict measurement methods to direct inputs to the game, namely keyboard strokes and mouse movement. While it would be interesting to augment the player actions with external input, such as cameras for facial expression detection, acceleration sensors, brain activity scanners, and specialised input hardware that supplies physiological information, e.g. mouse devices with skin conductivity sensors or keyboards with pressure sensors, this information will not be available when the classification algorithm is run in real-world environments, i.e., when used as a tool by game developers to adapt game rules online in video games.

In *Pac-Man* (compare figure 1), the player can navigate in 2 dimensions (up, down, left, right) to eat all pills scattered over the playing area. He has to avoid touching the four ghosts, which hunt him through the labyrinth. He can temporarily turn invulnerable to ghosts when he eats one of the four *power-pills* in the level, and gets bonus points when eating ghosts during this phase. Eating all four ghosts under the influence of a power-pill results in a significant amount of bonus points that are key to obtaining a good score. Occasionally a fruit appears somewhere in the level that can be eaten for bonus points as well.

2 SURVEY METHODOLOGY

In our survey we asked each participant to play 5 rounds of *PacMan*. This enabled us to optimise the result space under the assumption that these five games of a player convey his playing style consistently without too much variation.

² Galaxy Arcade: http://www.galaxyarcade.co.uk/custom_games/whitelabel_games.php, checked 24/05/2008.

³ Flash Game Code: <http://flashgamecode.net/tetris>, checked 24/05/2008.

⁴ Geheee: <http://www.geheee.com/source.html>, checked 24/05/2008.

¹ Imperial College London, UK, email: Robin.Baumgarten06@doc.ic.ac.uk

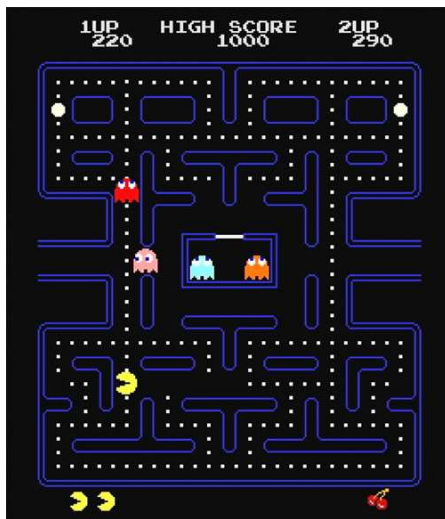


Figure 1. Original PacMan

2.1 Technical Details

A data-logging version of *Pac-Man* has been implemented using Adobe Flash. Using Facebook as survey platform simplified handling user authentication and allows for the collection of simple statistical metrics such as age and gender groups. The game uses a *MySQL* database as back-end and stores all relevant game information, so that games can be fully reconstructed post-hoc. This was then used to identify suitable metrics which are extracted and used in the classification process.

2.2 Recorded Metrics

Apart from the mentioned low level input measurements (keyboard and mouse data), game specific metrics can be recorded. These metrics, including concrete instances for *Pac-Man* are given below.

- Player actions, or type of actions (i.e., aggressive, passive, defensive, mistakes, bonus⁵). In *Pac-Man*, these actions are: eating a pill, eating a power-pill, eating a ghost, getting eaten by a ghost, eating a fruit, finishing a level, and changing the movement direction.
- Rate of player actions, which are often ratios of actions per action or actions per time. In *Pac-Man*, this is pills per minute, ghosts eaten per power pill, ghosts eaten per minute, power pills per minute, key strokes per pill, changes in movement directions per pill⁶
- Reaction time to game events. In *Pac-Man*, the only such events are appearing fruits and the direction changes of ghosts, thus the time-span of fruit appearance to fruit eating can be recorded and possibly ghost-player interaction, which requires a delicate rule to

⁵ A bonus action leads to a higher score but otherwise does not affect the goal of the game, for example eating a fruit in *Pac-Man*

⁶ The distinction of key strokes and changes in movement direction aims to capture the difference of interaction behaviour with the game of players; it is possible to press a direction key several times with only one resulting direction change.

decide when a player input is triggered by ghosts directly. In the current implementation, reaction times are not yet considered.

- Rate of progress through the game. In *Pac-Man*, this is given by the level completion time.

While this information depends on the game used, it is desirable to keep the information sufficiently general, as the classification process is aimed to be game independent⁷.

In this survey, the metrics described above have been extracted from the raw *MySQL* data using a python script, which also prepared the data for input into an *R* algorithm⁸.

2.3 Linear Discriminant Analysis

Discriminant analysis is used in statistics and machine learning to characterise or separate classes of objects based on a set of measurable features and class information of these objects. Linear discriminant analysis (*LDA*) utilises a linear combination of these features to (attempt to) separate the groups of objects. This combination can be used as a linear classifier or for dimensionality reduction. In other words, it maximizes the ratio of between-class variance to the within-class variance in any particular data set, thereby guaranteeing maximal separability.

In the context of our experiment, we use *LDA* to both perform a dimensionality reduction and extract information about player behaviour from the resulting transformed space. The weights of the features in the first dimensions of the *LDA* transformed space (which we will call *LDA space* from now on) indicate the most important features that determine the behaviour of a player and how it differs from other players. A positive side-effect of this method is that unimportant features are eliminated automatically, and it is not harmful if features are correlated, which is bound to happen as we look at rates of player actions such as pills per minute and ghosts per minute.

As we treat each set of 5 sessions of a player as one class, we need multi-class *LDA*. This can be understood as an algorithm that finds axes that best separate the classes. The axes define an optimal space through the *n*-dimensional cloud of data that best separates the session-groups. The rationale of using this approach is that if the sessions of a player are grouped together and the features recorded in the game are used to separate players as much as possible, these distinguishing features become apparent and we have found a space that reflects the playing style and skill of a player. Furthermore, players with similar behaviour and playing style will have a small distance in this *LDA* space, effectively grouping similar playing styles together. In this sense we can use *LDA* as a classifier for player behaviour; not necessarily predicting the correct class (i.e., player) for a given object (playing session), but the local group of classes will give an indication of the playing style this player is exhibiting.

Of course, the quality of this indication of playing style hinges on the quality of the linear discriminant analysis. If the measured features are unsuitable for distinguishing player behaviour, or the game does not provide for enough expressivity, a good class separation will not be possible, thus creating a space that does not represent player behaviour and skill well.

A technique closely related to *LDA* is principal component analysis (*PCA*). While *LDA* is focused on the difference between the classes of data, *PCA* does not take these differences into account

⁷ This can of course not be validated in a survey with a single game and is therefore subject of further research.

⁸ *R* is a programming language specifically developed for mathematical tasks and supports many statistical methods

and only looks at the features of objects. It effectively transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible, thus potentially reducing the number of required dimensions to describe the data. In our experiments, we want to make use of class information as explained above, and thus will not use PCA for our dataset.

3 RELATED WORK

Linear discriminant analysis has been used in areas such as face recognition, bankruptcy prediction[6], marketing and product management[5]. In face recognition research, it is mostly used together with PCA as means of dimensionality reduction of the feature vectors (the resulting vectors are then called *Fisher faces* when LDA is used and *Eigenfaces* in case of PCA)[1, 7].

LDA has also been applied to estimate feature weights for evaluation functions in an Othello game tree[2]. Buro found that choosing quadratic over linear discriminants did not necessarily provide an improvement in estimation accuracy, and the increase in complexity should be carefully considered against the speed and simplicity of LDA. He also applied logistic regression which gave promising results as well.

4 RESULTS

The survey garnered a good response, mainly due to its ease of access and being (indirectly) linked to by several social network sites such as reddit⁹ and facebook¹⁰. This resulted in a dataset of 245 players, each playing at least 5 sessions of *PacMan*. The data was collected over the span of two months. Logging every input, action and event during each session resulted in a dataset of about 4000 entries per session. This was then reduced to the metrics mentioned in section 2.2, and fed into the script that produced a linear discriminant analysis of the data.

4.1 LDA Results

As the axes in a LDA space are ordered from accounting for most between-group variance to least, which is why LDA is commonly used in dimensionality reduction[3, 4]. The first 5 dimensions are enough to account for 98% of between-group variance in our dataset.

A five-dimensional point cloud containing 1225 points and 245 classes is hard to visualise, so we will use projections and subsets of the data here. Figure 2 shows the entire dataset projected onto the first two LDA dimensions. Each number represents a player, and each number will appear five times, its position marking the coordinates of the feature vector defined by one session of that player. Additionally, all sessions of a player are colour-coded. Figure 3 shows a detail of that plot.

To understand what the individual axes of the LDA-transformed space indicate, we look at the base transformation matrix that transforms the result space with (normalised) features as dimensions into the LDA result space. The weights for the first five LDA dimensions are shown in table 1. The biggest absolute values (within 70% of the maximum) in each column have been highlighted.

⁹ <http://www.reddit.com>

¹⁰ <http://www.facebook.com>

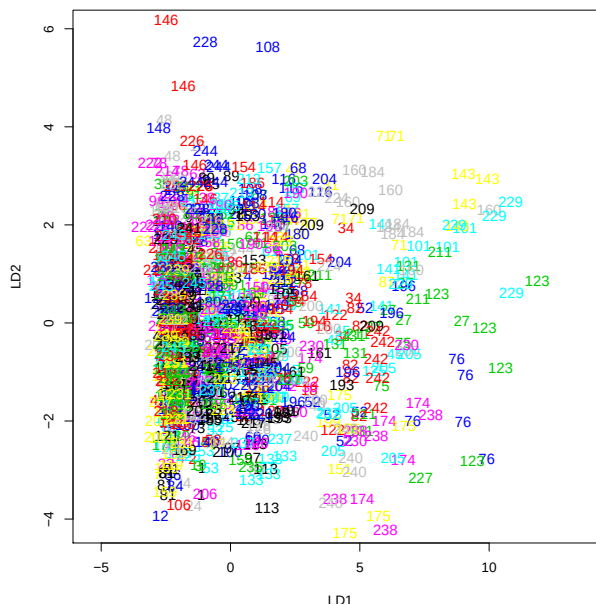


Figure 2. LDA transformed data space, projected onto the first two LDA dimensions. Each number represents a player, and each number will appear five times, its position marking the coordinates of the feature vector defined by one session of that player.

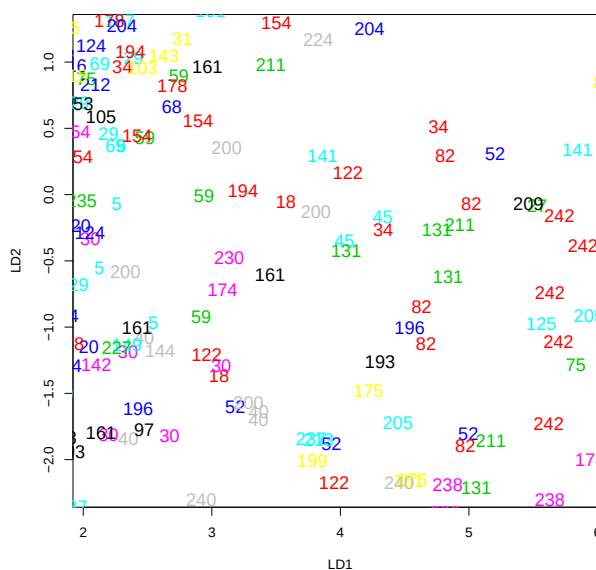


Figure 3. Detail of the LDA transformed data space, projected onto the first two LDA dimensions. Each number represents a player, and each number will appear five times, its position marking the coordinates of the feature vector defined by one session of that player.

	LD1	LD2	LD3	LD4	LD5
totalPills	-0.20	0.43	-1.85	2.22	-1.05
pillsPerLife	0.37	-0.33	1.68	0.38	-1.10
pillsPerMinute	0.21	0.09	0.01	-0.16	0.79
totalGhostsEaten	0.03	0.72	-3.15	-2.00	-0.63
ghostsEatenPerMinute	0.05	0.27	-1.00	0.08	2.09
ghostsEatenPerPowerPill	-0.09	0.45	0.73	0.04	-0.83
totalPowerPills	-0.14	-1.32	2.92	-3.48	3.48
powerPillsPerMinute	-0.25	-0.95	-0.38	-0.19	-1.27
eatenByGhosts	0.10	-0.09	0.46	-0.14	-0.32
totalTimePlayed	-0.29	-0.40	-0.23	-3.17	-0.51
totalFruitsEaten	-0.03	0.07	0.21	0.09	0.02
totalKeystrokes	0.41	-0.06	-1.59	-0.19	-1.33
keyStrokesPerPill	2.42	-0.32	1.16	-0.13	0.86
totalPacTurns	0.25	0.80	-1.03	2.95	3.45
pacTurnsPerPill	-0.15	0.12	-0.10	-0.15	0.24
totalLevelsCompleted	-0.20	0.43	-1.85	2.22	-1.05
score	-0.04	0.03	3.92	1.79	-2.27

Table 1. Weights for the base transformation from feature space into LDA space. Each value indicates the influence of a feature to the linear discriminant dimension.

4.2 Interpretation

The projection of the data onto the first two LDA dimension suggests that the sessions of the players share similarities and the chosen features can be used to classify player behaviour. In the graph we can see groups of the five sessions from the same player clustered together, which indicates a similarity between these sessions, in other words the playing style of the player, has been identified. Additionally, the in-class distances are generally much lower than the average in-between-class distances. Although LDA tries to optimise this relationship, data without features that allow for a distinction of these intra- and extra-class behaviour would not show the mentioned properties. The features used in our dataset are sufficient to reveal these differences and we can therefore interpret the LDA space as an approximation to a space where the distance between sessions is determined by player behaviour and style.

To identify what exactly constitutes to player behaviour and style, we look at the linear combinations of features that define the axes in the LDA space.

The first dimension of the LDA transformed space is dominated by the average number of keystrokes required for each pill (compare table 1). Therefore, the most important feature to distinguish different players (and thus playing styles) is how the player physically interacts with the computer. The amount of keypresses required to navigate a level (normalised by the amount of pills eaten to accommodate for the duration of a session¹¹) represents both the per-turn efficiency of a player (how often do they press a key for PacMan to change the direction once?) and also the skill to navigate the level (how many turns do they need to eat the pills?).

The second dimension is influenced by several related features: There are large negative weights on the number and speed of power pills eaten and positive weights on the number of ghosts eaten and the number of turns. This indicates that this dimension models a game concept of PacMan: Chasing ghosts with a high efficiency, i.e., eating many ghosts while not using up many power pills. Chasing ghosts usually requires a lot of maneuverability, thus increasing the

¹¹ Sessions do not necessarily have the same length; the player might lose all his lifes quickly and die, thus having a smaller number of keystrokes, or managing to complete the level, which obviously requires more keyboard interaction. The number of keys they need to press for a given time frame is more accurately described by giving a pill-dependent ratio.

corners PacMan takes. Or, if the player is on the other end of the spectrum, this means inefficiently eating power-pills without eating many ghosts. One could label this axis as the ghost chaser vs. ignorer dimension.

The third dimension has a high negative weight on ghosts eaten, and a positive weight on score and power-pills. It is less clear how to interpret this dimension, but we can still see that sessions that end up having a high score without the player eating a lot of ghosts will receive a high value in this dimension, i.e., the players that do well but play it safe without taking many risks versus players that end up playing too risky and eating many ghosts but dying earlier.

5 CONCLUSIONS AND FUTURE WORK

The analysis of the survey data presented in this paper indicates successful detection of player behaviour in game data recorded in a relatively simple computer game. We have shown that linear discriminant analysis is a useful tool to process the data, reduce significant dimensions and identify important features, and will continue to use it as a first step in our player behaviour analysis. Of course, more analysis will be required to further quantify the correctness and usefulness of this approach and compare it to other methods such as principal component analysis and bayesian network classification.

In our data, we were able to determine that the most important gameplay features returned by LDA are also what defines a game in general: physical interaction with the device, and secondly the main goals of the game. We discovered that players can easily be aligned along the dimensions of risk averse over risk accepting to risk seeking. In particular, in PacMan that meant that players tried to avoid ghosts, or had an efficient strategy to deal with them, or risked too much and lost lives early.

These findings are valuable for our project of understanding and optimising adaptive games. After having shown the basic feasibility of recognizing player behaviour through basic data logging, we can now proceed to create an adaptive game on the basis of these findings and will use a similar classification system as outlined in this paper to understand a player's playing style.

REFERENCES

- [1] P.N. Belhumeur, J.P. Hespanha, D.J. Kriegman, et al., 'Eigenfaces vs. fisherfaces: Recognition using class specific linear projection', *IEEE Transactions on pattern analysis and machine intelligence*, **19**(7), 711–720, (1997).
- [2] M. Buro, 'Statistical feature combination for the evaluation of game positions', *Journal of Artificial Intelligence Research*, **3**(1), 373–382, (1995).
- [3] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern classification*, Citeseer, 2001.
- [4] M. Loog, R.P.W. Duin, and R. Haeb-Umbach, 'Multiclass linear dimension reduction by weighted pairwise Fisher criteria', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**(7), 762–766, (2001).
- [5] M. Mazzocchi, *Statistics for marketing and consumer research*, Sage, 2008.
- [6] T.K. Sung, N. Chang, and G. Lee, 'Dynamics of modeling in data mining: interpretive approach to bankruptcy prediction', *Journal of Management Information Systems*, **16**(1), 63–85, (1999).
- [7] W. Zhao, R. Chellappa, and P.J. Phillips, 'Subspace linear discriminant analysis for face recognition', *Center for Automation Research, University of Maryland, College Park, Technical Report CAR-TR-914*, **137**, 138–144, (1999).

EDEE: A Flexible Environment for Research

Ningxuan Wang¹, Christopher Cullen¹ and Andrew Nuxoll¹

Abstract. In this paper, we present EDEE, a game environment that we believe is particularly well suited for AI research. EDEE is a complex, highly configurable game that is easy to create agents for due to its simple command structure and recognizable concept. To make this complex environment tenable for agents, we have created an interface design that presents the state of the game to the agent at three levels: an individual unit level, a tactical level and a map level.

1 INTRODUCTION

A major consideration for AI research is selecting an environment that will demonstrate the experimental outcome in a compelling manner but not unnecessarily interfere with or delay the research. We propose that an effective environment should have these properties:

- **Complex:** An environment that is too simple is not compelling.
- **Highly Configurable:** The more flexible an environment is, the easier it is to create a situation that focuses on the agent behavior. Flexible environments are also reusable.
- **Easy to Use:** Despite its complexity, it should be easy to develop an agent for the environment so that the researcher can focus on the AI.

In this paper, we present an environment called EDEE that meets these specifications. We present our interface for this environment and demonstrate how we expect it to expand with the complexity of our agents.

2 EDEE BASICS

The Empire Deluxe Enhanced Edition (EDEE) [1] is a turn-based, tactical strategy world-domination type game. We have found that it embodies each of the three qualities mentioned in the introduction:

- EDEE is complex. The environment supports 13 types of units and 35 types of terrain. The game also supports complex environmental effects like weather and fog of war. Playing the full EDEE game proficiently requires effective resource management and balancing simultaneous goals.
- EDEE is highly configurable. The game gives the user complex control over map generation including map size and ratios of each major type of terrain. A custom map editor is provided for creating a specific environment. The user controls what units are available to an agent. Furthermore, additional unit types can be added to the game without recompiling. The user can set up different rules and victory conditions. The task that an agent performs in the game can be as simple or complex as desired and all of this configuration can be performed without recompiling the environment.

- EDEE is simple to use. For simple tasks the agent need only issue two types of command: *create unit* and *move unit*. As a turn-based game played on a two-dimensional tile map, it is easy to control discrete unit actions and reproduce specific situations.

3 GAME INTERFACE CHALLENGE

To make EDEE convenient for use with AI agents, we wanted to adapt the existing interface to present the game state as a hierarchy of attribute-value pairs regardless of the complexity of the task. To illustrate the scope of this challenge, examples of three levels of complexity are presented below.

Simple Task

To date, we have used EDEE to conduct an AI research project to demonstrate learning with episodic memory [2]. We created a simple 10x10 map for this experiment and only one unit is active at a time. In this task, it is reasonable to present the entire state to the agent at all times.



Figure 1. A Simple Navigation Task with Episodic Memory

Figure 1 displays one path a unit might follow as it searches for a nearby city. Later, it uses its episodic memory to trace its previous paths in order to find the shortest one. Figure 2 shows the results of this experiment. The number of steps the agent takes to find the city decreases exponentially over time.

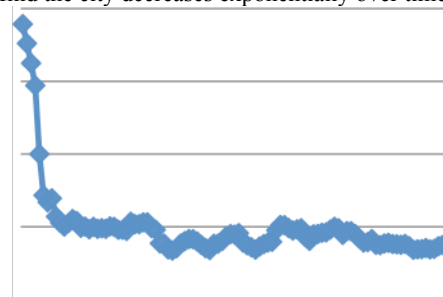


Figure 2. Number of steps to goal over successive iterations.

Medium Task

We've begun testing the feasibility of EDEE for an integrated task. In Figure 3 the agent must coordinate the actions of multiple units to locate its opponent on another island and then transport an assault force to that island. In this task, it is still possible to show the agent the entire state at once but the large amount of extraneous information hinders the agent's ability to act effectively.



Figure 3. An example of a medium difficult task.

Complex Task

In a complex task, an agent must be able to capture and defend multiple islands and manage a supply chain. The large map makes each state far too large to present to the agent in its entirety.

4 EDEE INTERFACE DESIGN

To address the game interface challenge, we designed game interface with two key features:

- **Unit Driven:** Each unit takes a turn receiving a command from the agent during each game turn. So, the agent can focus on one unit at a time rather than the entire game. This design was inspired by the human interface for EDEE, which uses the same approach.
- **Multi-Level:** The agent can zoom in to three different levels within the game: a unit level, a tactical level and a map level. We describe each of these below.

At the unit level, the agent can only examine a single unit and its immediate surroundings. An illustration of this level is shown in Figure 4. This level is sufficient for simple tasks.

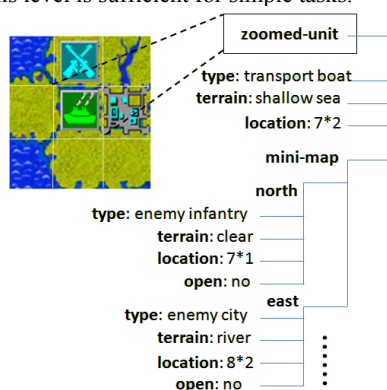


Figure 3. An example of unit level information.

At the tactical level, the agent can zoom out to a specific, larger area (e.g., an island). Information provided to the agent is focused on each player's military power in that area such as the numbers of cities and units. This level may be used to make a tactical plan to conquer or secure an area. The interface is responsible for defining the extent of each island.

At the map level, the agent can view a list of islands that comprise the global map. It will also group related areas into a larger control zone. This level may be used to make a strategic plan to dominate the world.

5 CONCLUSION

EDEE is a complex and highly configurable game that is easy for an agent to play. However, as a complex environment it is difficult to present to an agent in a form that is conducive to learning and overall effective play. We have presented our design for a unit-driven, multi-level interface that allows the agent to separate tactical and strategic decisions from individual unit movements. We expect this interface to allow an AI researcher to use EDEE at all levels of complexity.

REFERENCES

- [1] M. Kinkad. *Empire Deluxe Enhanced Edition information page*. Killer Bee Software, <http://www.killerbeessoftware.com/kbsgames/edee/>.
- [2] Ningxuan Wang, CJ Cullen, Andrew Nuxoll. "Episodic Memory for a Simple Empire Game Environment." *University of Portland Wilson W. Clark Memorial Library*. http://library2.up.edu/ereserves/nuxoll_4xx_01.pdf.

A Design and Implementation of 3D Image Interactive Vision System

SangHeon Han¹, Chang Ok Yun¹, Jung Hoon Kim¹, Jae Ik Jo¹, Tae Soo Yun² and Dong Hoon Lee²

Abstract. Recently, the interactive display is providing various interaction interfaces to users through ubiquitous computing technologies. This means that they can experience the tangible interactive interface environments with various contents. In this paper, we propose a spatial multi interactive system that provides the various touch interactive to multi-user in the large display environment. In other words, it is possible to interact in the 3D space without touching directly on the screen. And then, the users can interact with only natural hand movements through the IR LED Array bar installed in the ceiling. Moreover, by providing the realistic contents through 3D image, the users can enjoy more realistic and tangible contents.

1 INTRODUCTION

Recently, the multi-touch technology in many different fields is providing an intuitive interface. One of them is a Table-top system like Microsoft surface [1], and other is an Interactive Wall Display [2]. These traditional methods are used in a variety of applications, but they seem the difference depending on the formal feature, the installation space and purpose. In other words, research of multi-user on a single system can work in cooperative systems [1,3,4,5,6], the mutual collaboration system [7,8] though the network infrastructure are going to expand. In addition, a form of independent electronic device [1,9,10] through the new user interface is developing.

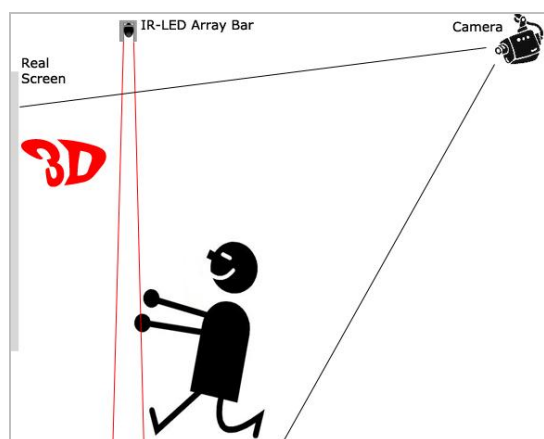


Figure 1. 3D Image Interactive Vision System.

This paper also proposes a user interface system as a new form. There are two features of the system. 1st, our system uses a 3D image system that provides realistic contents to the multi-user. 2nd, we don't touch screen directly as previous systems, but use an interaction layer for spatial interaction. This layer offers a sufficient distance for more user experience, spatial interaction and experience of realistic content.

2 3D Image Interactive Vision System

The system consists of two 3D image projectors, large display for real projection, IR-LED array bar and IR Camera (fig. 1). A system flowchart is as follow (fig. 2). There are two nodes for data processing. One is a client node that processes acquired images from IR camera through a simple image processing method. The other is a server node that generates metaphor information from received data in client node.

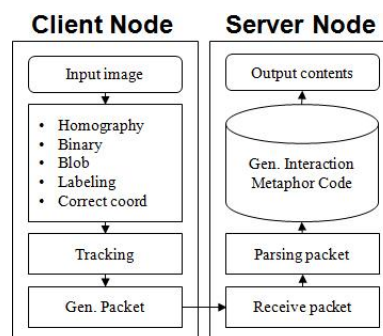


Figure 2. System flowchart.

We must know a hand location and movement to define interaction between user and 3D image contents. For hand tracking, we use IR-LED and IR camera. IR-LED array bar always shoot IR light(we called "Interaction Layer"). If hand touches that, IR light is reflected. Then camera detects it as tracking(or location) data.

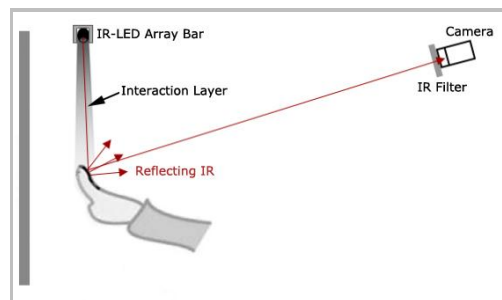


Figure 3. How does it work?

¹ Dept. of Visual Contents, Univ. of Dongseo, Churye-2-Dong, Busan, S. Korea. Email: {alpha815, melc81}@gmail.com, jjikr@nate.com, coyun@hanmail.com.

² Dept. of Game Contents, Univ. of Dongseo, Churye-2-Dong, Busan, S. Korea. Email: {tsyun, dh1}@dongseo.ac.kr.

Through these process, we can interact 3D image contents (fig. 4). Figure 4 is the content of Upo Wetland Ecosystem Pavilion.



figure 4. 3D Image Contents

- [9] P. Dietz and D. Leigh, "DiamondTouch: A Multi-User Touch Technology," In ACM UIST '01, pp.219-226, 2001.
- [10] Loenen, E. Van, Bergman, T., Buil, V., Gelder, K. Van, Groten, M., Hollemans, G., Hoonhout, J., Lashina, T. and Wijdeven, S. "Entertaible: A Solution for Social Gaming Experiences, in Proc., of Intelligent User Interfaces '07, pp.16-18, 2007.

3 CONCLUSIONS

In this paper, we propose the 3D image interactive vision system that provides an interaction UI for 3D experience and reality. It can interact with multi-user in spatial, and they just need simple hand gestures to interact our system. Our system has some limit that can only recognize simple actions and more closed user with IR camera. So we will improve our system for more accuracy, image experience and performance, and add more gestures, screens. We are expecting our system that can be useful to various exhibition areas for 3D experience.

Acknowledgements

This research was financially supported by the Ministry of Education, Science Technology (MEST) and Korea Institute for Advancement of Technology (KIAT) through the Human Resource Training Project for Regional Innovation.

REFERENCES

- [1] Microsoft Surface, <http://www.microsoft.com/surface>.
- [2] iO, Italy, <http://www.naturalinteraction.org>.
- [3] Y. Kakehi, M. Iida, T. Naemura, Y. Shirai, M. Matsushita, and T. Ohguro, "Lumisight Table: Interactive View-Dependent Tabletop Display," In Proceedings of IEEE Computer Graphics & Applications, Vol.25, No.1, pp.48-53, 2005.
- [4] A. Mazalek, M. Reynolds, G. Davenport, "TView Table in the Home," In IEEE Tabletop '07, pp.52-59, 2007.
- [5] C. Ganser, A. Steinemann, R. Hofer, A. Kunz, "InfrActables: Supporting Collected Group Work by Combining Pen-Based and Tangible Interaction," In IEEE Tabletop '07, 2007.
- [6] Scott, S.D., Grant, K.D., Mandryk, R.L., "System Guidelines for Co-located, Collaborative Work on a Tabletop Display," Proceedings of ECSCW '03, pp.159-178, 2003.
- [7] S. Izadi, A. Agarwal, A. Criminisi, "C-Clate: A Multi-Touch and Object Recognition System for Remote Collaboration using Horizontal Surfaces," In IEEE Tabletop '07, pp.3-10, 2007.
- [8] P. Tuddenham, P. Robinson, "Distributed Tabletops: Supporting Remote and Mixed-Presence Tabletop Collaboration," In IEEE Tabletop '07, pp.19-26, 2007.

Distributed drama management: integrating levels of narrative responsibility

Allan Weallans¹ and Sandy Louchart¹

Abstract. Work on drama management may be broadly divided into two categories of approach: namely, the top-down plot-based approach and the bottom-up character-based approach [1]. The relative strengths and weaknesses of these approaches are well-established: the top-down approach, in general, is strong in the areas of structure and plot, but has weaknesses in character consistence and contextualisation; the bottom-up approach, conversely, strong in consistence and contextualisation, but has weaknesses in structure and plot. It is widely agreed that Interactive Storytelling (IS) would benefit from an integration of the different levels of approach. We herein introduce the concept of Distributed Drama Management (DDM), by which autonomous character agents would negotiate between them action selections that would contribute to dramatic narrative structure, with the aim of working towards such an integration.

The DDM concept differs from previous work in that the actions that cause the story to unfold can consist of various types of intervention at several levels of story representation. Through assessment of the current user experience and the needs of the story (in accordance with both established dramatic rules and authorial preferences), a DDM system could make a decision on whether the experience would at that point best be served by plot-level or character-level interventions, or by an emergent approach. Fundamentally, the DDM concept proposes research into the causal relationships between levels of story representation, as this could determine the way in which a DDM system would prioritise a particular type of intervention over another.

The primary research domains currently considered for work on DDM are Interactive Storytelling Systems (ISSs) and Synthetic Actors (SAs). One area of interest to DDM is research that offers insight into how the relative efficiency of given interventions may be quantified, and of how the user experience may be assessed. To this end, work on DDM takes place within the context of several different works on user monitoring. One such work is David Thue's work on monitoring user behaviours in response to narrative stimuli in game and/or story environments [2]. Another is Tychsen et al's work on modelling game masters' assessment of the players' experience in role playing games [3]. Another still is the body of work on physiological monitoring [4] (such as electroencephalography (EEG), electrocardiography (ECG) and galvanic skin response (GSR)) with respect to user experience of games and stories.

Aside from quantification of narrative impact, DDM research must also extend to the technological boundaries of Intelligent Agent (IAs) planning activities. In a synthetic character-based environment, a large proportion of changes directly occur as a result of character actions and decisions. The character therefore acts as the dynamic factor with which story agency must comply. Irrespective of their mode of control (e.g. autonomous or otherwise), it is essential to identify a mechanism that would

permit to dynamically shape narrative agency with respect to characters' decisions.

As the DDM concept is an integrative approach to drama management, encompassing all levels of approach, it is necessary that a DDM system be able to appropriately distribute narrative responsibilities according to current needs. Work on DDM will therefore involve identifying different models of distribution (e.g. models that emphasise stricter central control or models that allow more emergence) and which strengths they emphasise, and determining under what conditions these models would be most suitable.

In order to successfully implement a DDM system, we believe it is necessary to a) Identify the different levels at which narrative intervention can be administrated. b) Conceptualise the various modes of interactions for each level and determine their potentials for managing a user narrative experience. c) Identify factors other than narrative-based ones influencing the selection of narrative elements (e.g. thematic concerns, training requirements) d) Understand the relationships between all levels and how they actually do affect each other on both a theoretical and practical basis.

In summary, the aims of our work on the DDM concept are to perform research into the causal relationships between different levels of story representation. We aim to do this by assessing the efficiency and impact of interventions at different levels, and by identifying a planning mechanism that will allow the maintenance of a story structure through these interventions.

REFERENCES

- [1] D. Sobral, I. Machado, A. Paiva. Managing Authorship in Plot Conduction. In: *Procs. 2nd Int'l Conf. on Virtual Storytelling (ICVS'03)*, Toulouse, France. (2003)
- [2] D. Thue, V. Bulitko, M. Spetch. PaSSAGE: A Demonstration of Player Modeling in Interactive Storytelling. In: *Procs. 4th Conf. on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'08)*, Palo Alto, CA, USA. AAAI Press (2008)
- [3] A. Tychsen, M. Hitchens, R. Aylett, S. Louchart. Modeling Game Master-Based Story Facilitation in Multi-Player Role-Playing Games. In: *Papers from the AAAI Spring Symposium Intelligent Narrative Technologies II*, Stanford, USA. (2009)
- [4] G. N. Yannakakis, J. Hallam. Entertainment Modeling in Physical Play through Physiology beyond Heart Rate. In: *Procs. Int'l Conf. on Affective Computing and Intelligent Interaction, Lecture Notes in Computer Science*, vol. 4783, pp. 256-267 Lisbon, Portugal. Springer-Verlag (2007)

¹ Heriot-Watt University, School of Mathematical and Computer Sciences, Edinburgh, EH14 4AS. Email: aw119@hw.ac.uk, sandy@macs.hw.ac.uk.

Towards the use of Trait Inheritance for Flocks of Navigating Agents

Piotr Piasecki and Christopher Peters¹

Abstract. This preliminary work is focusing on investigating how flocks of entities can use trait inheritance to both endow them with a sense of individuality and to create better solutions for navigation problems over multiple generations using an evolutionary algorithm, with intended application to space environments. Traits are implemented so as to influence agents' actions and movements. They are designed so as to evolve over multiple generations of agents with the goal of further optimising their behaviour. The trait system is implemented so as to provide each agent with their own preferences for performing tasks, and the evolution algorithm is used to promote better set of traits in the agents.

1 OVERVIEW

The creation of intelligent agents with individualised behaviours is an important aspect of modern game design. Yet in many cases, agent credibility vanishes when a player discovers underlying patterns or rules that the agents are using. Furthermore, such agents usually have difficulty adapting to changing environments, circumstances and strategies. Here, navigation behaviours, such as steering [1] and A-Star preferences are parameterised; these values, set for individuals, are refined over multiple generations of agents in order to attempt increase their performance or adapt to changing conditions.

Traits are implemented so as to influence agents' actions and movements. They define, for example, differences in steering behaviours, whether an agent prefers a small but nearer reward, or to go further to obtain a larger reward, and so on. Traits are designed so as to evolve over multiple generations of agents, with the goal of further optimising their behaviour. Every agent is graded based on their actions performed and their success in performing the task, to determine the chance of passing their traits to new agents. The trait system is implemented so as to provide each agent with their own preferences for performing tasks. The evolutionary algorithm is used to promote better sets of traits in the agents over time. This is implemented in a space scenario, where agents visit asteroids to collect resources (see Figure 1).

2 DISCUSSION

One of the main questions explored in this research is whether the agents' traits will converge to a single optimal solution and the consequences for individualised behaviours. The process might consist of speciation of several groups of agents, with each 'family' pursuing a different local extrema. Since the model is based on reproduction

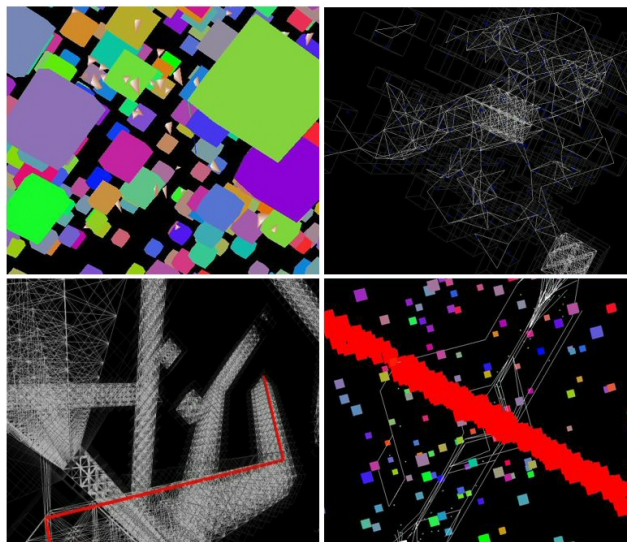


Figure 1. Example of the trait inheritance scenario involving multiple agents in a 3D space environment. Here, agents cooperate and compete by attempting to harvest resources from nearby asteroids.

by self-replication with error, rather than replication between more than a single agent, one solution will most likely perform better than the other in the end, and prevail as the more optimal one. In this case, even though there might exist a best set of traits, each agent will vary, due to reproduction with error, thus retaining unique quirks. A different scenario might be if the environment is dynamic, and the optimal solution changes over time [2].

A disadvantage of this approach is that it requires the system to be parametrised, so it needs to be designed taking that into consideration. It might be difficult to implement this approach for synthesising specific behaviour types, since the evolution process may provide optimal but unrealistic outputs if unconstrained [3]. This suggests a number of possible extensions to the current model and is an interesting domain of future research.

REFERENCES

- [1] C.W. Reynolds: Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4):25-34, 1987.
- [2] M.A. Nowak and K. Sigmund: *Evolutionary Dynamics of Biological Games*. Science, Vol. 303, Feb. 2004.
- [3] L.F. Rodrigues, G.R. de Carvalho, R. de Barros Paes, C.J.P. de Lucena: Towards an Integration Test Architecture for Open MAS. In *Proc. of the 1st Workshop on Software Engineering for Agent-oriented Systems / SBES*, 2005.

¹ Interactive Worlds Applied Research Group, Coventry University, UK, email: Piotr.Piasecki@uni.coventry.ac.uk — Christopher.Peters@coventry.ac.uk

Towards a Goal Oriented Action Planning Approach for Predator-Prey Simulation

Justyna Pultowicz and Christopher Peters¹

Abstract.

Here we summarise a work-in-progress towards implementing a goal oriented action planning approach, GOAP, which formulates a plan from a list of possible actions in order to achieve a goal. One of the potential benefits of such an approach, apart from requiring less specification from the programmer, is that entities may derive unexpected solutions. The GOAP implementation is being used for application to a simple predator-prey simulation.

1 GOAL ORIENTED ACTION PLANNING

Planning is a vital capability for game entities. An important question is how an AI may generate its own goals and actions based on high-level specifications, a storyline for example [1], without the necessity for the programmer to provide details for every possibility that may occur. Adaptive planning systems help in this respect, requiring only the specification of goals, actions and relationships: actions can then be undertaken in an automatic manner in order to achieve these goals. Based on STRIPS, the Stanford Research Institute Problem Solver [2], goal oriented action planning, or GOAP [3], plans a path of actions for fulfilling a goal and has been used for the AI in the computer game, FEAR, by Monolith Productions. GOAP is flexible, and useful for dynamically finding solutions for fulfilling a goal when a previously available action can no longer be executed. Our use of GOAP is intended to replace the decision-making system of a predator-prey simulation, currently implemented using decision trees, in order to provide more flexible planning capabilities for the entities in the simulation and investigate extensions to planning.

2 IMPLEMENTATION

The predator-prey simulation features entities navigating around a simple 2D grid environment, who must forage for food or hunt. Entities currently return to a den depending on foraging/hunting success, presence of predators, and the time of day. In addition to the standard GOAP implementation described in [3], we are investigating a number of extensions: these include varying the types of precondition values used in the algorithm, associating timings with actions in order to be able to formulate efficient plans, and an investigation of different forms of energy metrics [4], for example, taking actions with immediate effects on the goal and other needs into greater account. Thus far, a basic GOAP prototype has been created suitable for scenarios relevant to the predator-prey scenario (see Figure 1). Entities navigate the environment, searching for sources of nutrition

and attempting to avoid predators. Currently, decision-making in this simulation is conducted using decision trees.

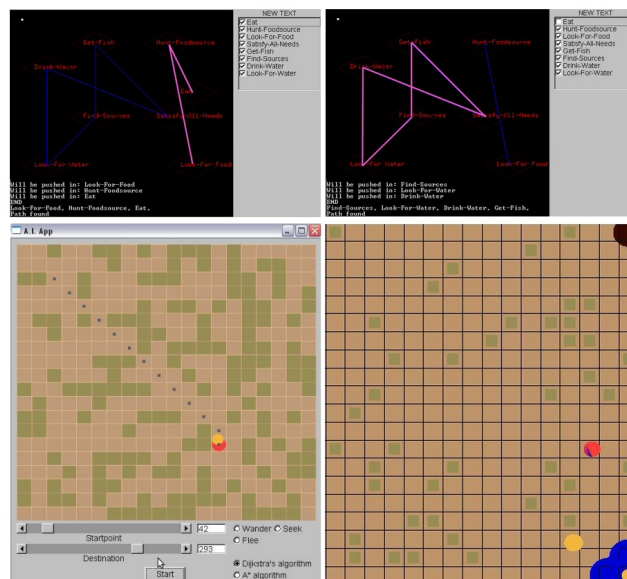


Figure 1. Top row: example scenarios within our GOAP test interface. Bottom row: example of the predator-prey scenario.

Our next step is to integrate the GOAP prototype into the predator-prey simulation, investigating extensions specific to it. An interesting possibility is the consideration by entities of the theorised goals/locations of competitors in the planning of future activities.

ACKNOWLEDGEMENTS

We wish to thank the referees for their in-depth feedback. This work was partially funded by a Nuffield Foundation Bursary, URB/36777.

REFERENCES

- [1] H. Barber and D. Kudenko: Generation of Dilemma-based Interactive Narratives with a Changeable Story Goal, Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment, 2008.
- [2] R.E. Fikes and N.J. Nilsson, STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence, Vol. 2, No. 3-4, pp. 189-208, 1971.
- [3] J. Orkin: Three States and a Plan: The AI of F.E.A.R. Proceedings of the Game Developer's Conference (GDC), 2006.
- [4] I. Millington: Artificial Intelligence for Games. Morgan Kaufmann Publishers Inc., 2006.

¹ Interactive Worlds Applied Research Group, Coventry University, UK, email: pultowij@uni.coventry.ac.uk — Christopher.Peters@coventry.ac.uk

Digital Footprints – A tool for generating heat maps and capturing data from FPS games

Alastair Hebson¹⁾, Thomas Welsh²⁾ and David C. Moffat²⁾

Abstract. This demo paper introduces the *Digital Footprints* software tool which can analyse data from the FPS game *Unreal Tournament 3* and generate heat maps based on game data.

Digital Footprints is a software tool with a number of potential uses. It interoperates data gathered from an FPS game session and formats that data as a heat map. This map can provide insight into player behavior, the actions and behavior of AI controlled bots and the structure and navigation of the game environment. In this way, *Digital Footprints* has applications in game design, testing, difficulty and team balancing and analysis of player behavior.

Digital Footprints is designed to work with the third version of the *Unreal Engine* which is the technology behind *Unreal Tournament 3*. The tool is coded in C# and requires the .net architecture to run.

Digital Footprints shows a plan view of the level that the players inhabit and thus provides an overview of where all players are and what they are doing. Queries can be made which will return the locations of specific, defined events in the game. For example, a query to show where certain weapons were used could be made. If a weapon was used twice at specific locations on the map, the result would look something like figure 1 below.



Figure 1. Plan view of game environment

A number of steps are involved to prepare *Digital Footprints* to display this kind of data. The image of the plan view of the map must first be captured. To do this, *Unreal Tournament 3* is run and when the game begins a mutator that we have created is loaded. This mutator sets the game camera to a fixed position and viewpoint and takes a screenshot of the level. A mutator is a small modification of a game which can be created by anyone with knowledge of UScript; *Unreal Tournament's* scripting language. Once the image has been captured with the help of the mutator it is saved to a specified file and the game can be

stopped. By capturing the image in this way, *Digital Footprints* is able to work with any map available for the game.

Once this has been done, the player begins a game and uses another, different mutator to gather data in the game session. This mutator simply gathers data from the events of the game and outputs them to a log file. The log file is a tabulated, time stamped set of figures describing every event taking place during the game session. Once this log file has been generated the main executable can be run and the user will be able to generate heat maps based on the data captured from their game session.

Once the main *Digital Footprints* program is run the image of the game level is loaded and over this heat maps are drawn. At this point the user can select the data to be included in the heat map.

One of the most simple heat maps the user might generate would be one which shows the navigation of a player throughout the course of a game session. *Digital Footprints* would highlight areas of a displayed map based on the amount of time spent in that area by the players (figure 2). This heat map allows the user of the tool to determine the busiest parts of the map and the areas where either bot or human players fail to explore. A simpler version of this type of heat map was used in the development of *Halo 3*. In that case, the heat maps showed only human player navigation through a level and were used to balance the difficulty and play test the single player campaign.

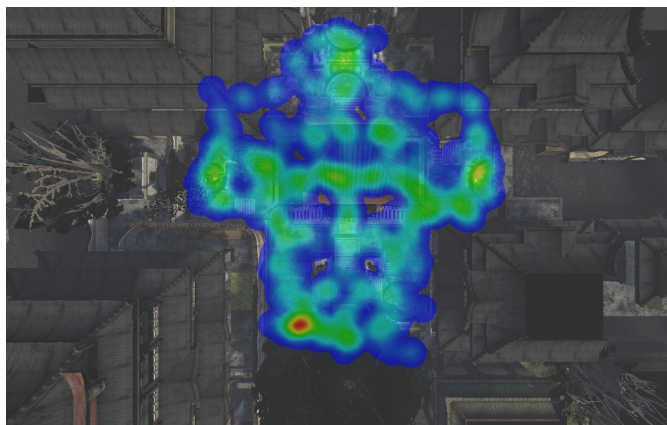


Figure 2. Heat map showing player navigation

The heat maps generated by this program offer unique opportunities for many areas of game research. The heat maps are simple to generate and can offer clear representations of the events of the game which can be easily understood. The uses for such rich and accessible data are limited only by the imaginations of the researchers who use this tool.

1) Rockstar North, 1 Greenside Row, Edinburgh

2) Glasgow Caledonian University

