AI System Designs for the First RTS-Game AI Competition

Michael Buro[†], James Bergsma[†], David Deutscher[‡], Timothy Furtak[†], Frantisek Sailer[†], David Tom[†], Nick Wiebe[†]
[†]Department of Computing Science
University of Alberta, Edmonton, Alberta, Canada
[‡]Tel Aviv University, Israel
email: mburo@cs.ualberta.ca

KEYWORDS
Real-time strategy games, ORTS, real-time AI systems

ABSTRACT

Real-time strategy (RTS) games are complex decision domains which require quick reactions as well as strategic planning. In this paper we describe the first RTS game AI tournament, which was held in June 2006, and the programs that participated.

Introduction

Creating smart computer adversaries and teammates for human players in modern video games is challenging. AI programmers for such games are faced with limited computational resources (because most CPU cycles are still devoted to graphics), real-time constraints, huge state and action spaces, and imperfect information. In addition, the tight release schedule for video games does not leave much room for conducting AI research in games companies. Therefore, a common approach to practically solving these problems is to create an illusion of intelligence (Livingstone, 2006) by scripting actions for nonplayer characters (NPCs) and providing them with more resources including information that is not available to human players. This way it is relatively easy to create NPCs that by having more knowledge of the game state — or bigger virtual muscles — can reach the playing level of human players or even outperform them. There are however problems with this methodology. Scripted action sequences are brittle — they often cannot deal with new situations and are easily defeatable once known. More advanced variations exist (Spronck et al., 2006), whereby script parts are executed probabilistically and probabilities are updated dependent on past performance. But even with such modifications, opponent AI systems still cannot compete with strong players unless they are given unfair advantages.

To overcome this problem, several AI researchers have started to use video games as test applications for their work in recent years. Conferences are now devoted to progress in computer entertainment AI, and the interaction between computer game companies and academia has increased. Another particularly effective way of spuring research in AI is holding competitions. Great examples are the machine-machine and man-machine competitions in the 1980s and 1990s which produced stronger and stronger programs which eventually played on-par or better than the best human players in chess, backgammon, checkers, and Othello. Other examples which have helped to increase the performance of AI systems considerably include the annual planning competition, SAT competitions, and RoboCup. The goal of competitions like the one which we are going to describe here is to repeat the success of classic game AI systems in the area of more complex video games.

In the remainder of the article we first describe the game genre we are interested in — Real-Time Strategy (RTS) games — and the programming framework ORTS we have developed for it. Then, after presenting the tournament game categories, we describe the programs that participated in the first AIIDE RTS game competition, present their tournament results, and conclude the paper with ideas on future RTS game AI competitions.

RTS Games and ORTS

Real-time strategy games are typically tactical simulations engaged in by two or more players. These games are fast-paced and pose several challenging problems such as incomplete information, the need for long-range planning, and a continually changing world with limited time to plan (Buro and Furtak, 2004).

A player can be in control of potentially hundreds of units, each with several possible actions that may be taken several times a second. A naive search of the available action space is clearly intractable. This necessitates potentially several levels of abstraction, for controlling individual units and larger armies.

Games typically involve simplified economies consisting of gathering resources which may be used to construct buildings, research new abilities, and train offensive and support units. Resource usage must be balanced to construct an army capable of effectively exploit opponents' weaknesses while being able to defend against potential threats.

Determining an effective strategy often relies on accurate opponent models. Specifically, determining the types of enemy units that an opponent will likely produce, and how they will be used to attack, at what time, and at which location.

ORTS

The Open Real-Time Strategy (ORTS) game engine, available from `www.cs.ualberta.ca/~mburo/orts`, provides a flexible framework for studying AI problems in the context of RTS games. The ORTS engine is scriptable, which allows for game parameters to be easily changed, and new types of games, or subsets of existing games, to be defined.

Unlike most RTS games, ORTS uses a server-client framework. Instead of each client maintaining a local copy of the entire game state, each frame the ORTS server only sends a client the information actually available to it. This effectively eliminates the ability of clients to cheat by applying simple map-revealing hacks.

Units in ORTS are simple geometric primitive (circles, rectangles, and line segments) located on a fine grid. Objects may travel at an arbitrary heading, with collisions accurately computed by the server.

Map terrain is specified by a grid of tiles, with each tile capable of having arbitrary corner heights and being one of several terrain types. Boundary objects with various collision masks are automatically created along discontinuities between tiles.

Unit vision is tile-based, with different units having a sight range that determines how many tiles away they can see. When the "fog of war" is enabled, a player only has up-to-date information about tiles that are currently seen by an allied unit. The vision model also supports "cloaked" units which can only be seen by "detectors".

All ORTS components are open-source. Along with the server-client framework, this allows users to create their own AI components capable of acting autonomously or to augment a human player.

The AIIDE RTS Game Competition

The RTS Game Competition presented at AIIDE '06 consisted of three separate game categories, arranged in increasing order of complexity. These categories addressed the tasks of multi-unit pathfinding, local combat, and dealing with imperfect information, in that order. Effective solutions in one category relied on implementations from the previous game types.

Game 1: Cooperative Pathfinding

The first game is stated as the task of gathering as many resources as possible within a given amount of time. The player begins the game with one base surrounded by workers. These workers must travel to resource patches randomly positioned on the game field, spend a short amount of time to collect those resources, and finally bring them back to the base.

At the start of the game the entire map and the locations of all resources are known to the player. To complicate the task, the map contains both impassible terrain obstacles, and indestructible mobile "sheep", which randomly travel a short distance, stop, then continue. The entire scenario is perfect information, except for simultaneous actions on the part of the workers and the sheep.

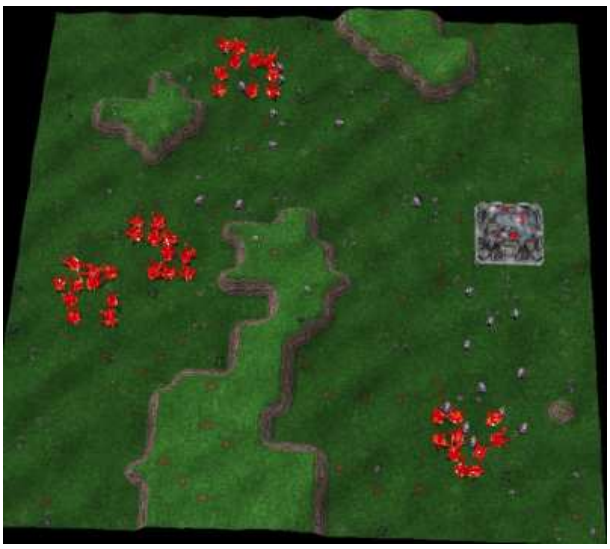

Figure 1: Game 1 client display.

Practically, the task is then to effectively coordinate the motion of the workers to minimize total travel time between the base and the resource patches. Spending a long time to compute near-optimal routes may result in the world having changed to the point where the computed solution is no longer valid.

Game 2: Local Combat

The second game is two-player tank combat, where the objective is to destroy as many of the opposing player's bases as possible within 10 minutes. Each player begins with 5 bases randomly distributed within the playfield, and 10 tank surrounding each base. A game ends immediately if all of one player's bases are destroyed.

As with the first game, each player has full visibility of the entire map. Plateaus, which are impassable and block line-of-sight tank attacks, are randomly placed on the map. Neutral, indestructible sheep also wander randomly.

The focus of this scenario is to effectively engage and destroy enemy squads. Formations which allow one side to concentrate fire on a small number of tanks while exposing themselves to few attackers are preferable. An agent must therefore coordinate the motion of the tanks to bring about these positions while avoiding collisions with other tanks (both allied and enemy) and unpredictable sheep.

Game 3: Mini RTS

The third game is a stripped-down version of a "real" RTS game. Two players begin with one base and several workers located next to a resource patch. The rest of

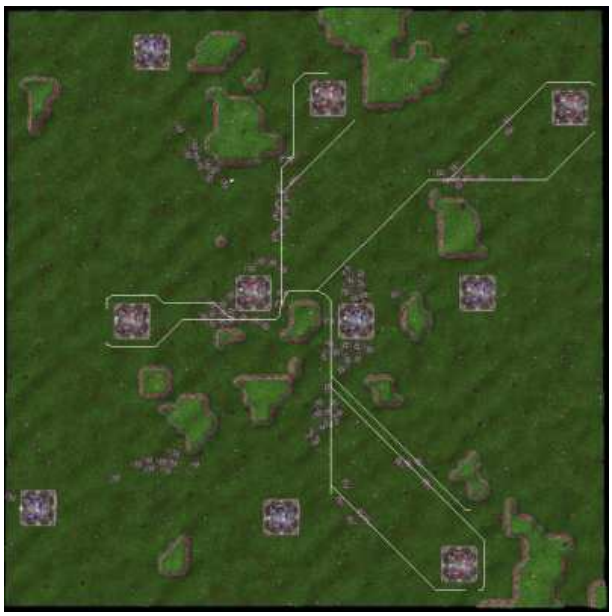

Figure 2: Game 2 client display.

the map and the location of the enemy base is initially unknown. A fog-of-war limits the currently observable parts of the map to those regions that can be seen by allied units.

A player is able to spend minerals and use a worker to construct a barracks and then a factory. Barracks and factories can then be used to train marines and tanks respectively. Tanks have more hitpoints, attack power, and range, but cost more than marines.

The objective of this game is to obtain more points than the opponent before time runs out. Points are awarded for gathering resources, constructing buildings, training units, and for destroying enemy buildings and units. The game ends early if all of one player's buildings are destroyed.

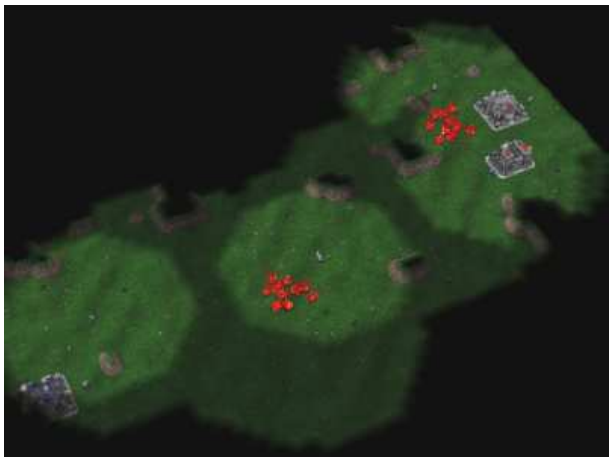

Figure 3: Game 3 client display.

Tournament Setup

All tournament games were played between June 16 and 18, 2006 on 31 undergraduate lab computers in the computing science department at the University of Alberta. Each machine was equipped with a single Athlon XP 1.5 GHz CPU and 512 MB RAM running Linux 2.4.31 and gcc 4.1.1. Shortly prior to the competition a multi-threaded ORTS tournament manager was completed by Krysta Mirzayans. This software greatly simplified running the tournaments and allowed us to play a large number of games.

Authors had access to the tournament computers on which they could upload their programs to test them in individual protected accounts which were frozen just before the tournament commenced. Each participant was asked to send a magic integer to a member of the independent systems group which also set up the tournament accounts. These numbers were then exclusive-or combined to form the seed of the random number generators used for creating all starting positions. This way, no participant was able to know beforehand what games would be played. In order to reduce dependency of game results on starting positions

In what follows we describe all tournament entries in turn and present the results of the tournaments.

Game 1 Entries

brzo1

Author: Michal Brzozowski, University of Warsaw, Poland

Michal's entry used a discrete graph-based terrain representation where neighboring vertices are connected if their connecting edge is traversable. Workers are guided by a finite state machine (FSM) with the following states: move-to, mine, go-back, drop-resources, avoid (entered when hitting a moving obstacle. Avoids obstacles by moving to the left. When it hits a static obstacle, it moves to a random direction), and emergency-path (when hitting a number of obstacles in the avoid state, tries to get back to original path). A coordinator assigns workers to resources based on shortest paths. Each worker picks the closest mineral from its starting point with less than 2 workers assigned already.

creed1

Author: Michal Szostakiewicz (University of Warsaw, Poland)

A search graph is built from nodes representing tile centers. Edge weights depend on mobile objects close by to prevent collisions. Each worker is assigned a random mineral patch and is sent to it. Shortest paths are computed by Djikstra's algorithm. When colliding, workers move to a random location nearby.

umich1

Authors: Joseph Xu and Sam Wintermute (University of Michigan, U.S.A.)

This entry was implemented in the SOAR architecture using a modified version of the standard ORTS pathfinding with an added local obstacle avoidance system. Workers are guided by a mining manager and a FSM. If a worker exceeds its estimated travel time, it requests a new route from the mining manager. The mining manager learns which routes are bad.

uofa1

Authors: David Deutscher (Tel Aviv University, Israel) and Nick Wiebe (University of Alberta, Canada)

This entry is based on three modules:

1. A single-unit path planning algorithm using a simple grid based A* algorithm, which uses a multiple-resolution world representation, pluggable goal definitions (including "touch a target objejct") which can handle variable-sized and shaped objects.

2. A path execution system which calculates the necessary motion at each simulation tick to move a unit along a predetermined path. To do this force fields are used to attract moving units to a point on the path in front of them and to repel them from other objects, buildings, and walls. For each unit traveling along a path, every object, building, and wall whose distance is below a minimum threshold exerts a force on the unit inversely proportional to the square of the distance between them. The movement vector for the unit is the sum of the forces acting on it. This approach solved the problem of path obstruction by sheep and enemy units, as units would just roll off the obstruction. It also is used to give priority to moving units. By calculating and applying these forces to units that were not moving along a path, idle units can be pushed out of the way of moving units. A small randomized vector is added to this pushing force in order to limit the distance that units are pushed in a single direction. Pushing proved to be important for game 2, where it speeds up large group attacks.

3. Dynamic allocation of minerals to workers, based on minimizing a weighted (1:1) combination of the Euclidean distance from the worker's current position and the static path's length between the mineral and the control center (where static means the shortest path found while considering only static obstacles — boundaries, other minerals and the control center itself). Statically-blocked minerals are not assigned and a single worker per mineral is preferred, unless no other option is available. Failures to plan a path or to reach a mineral raises a limited-time flag preventing its use for a couple dozen turns.

Game 1 Results

Initially it was planned to play 300 games per entry lasting 10 minutes each on June 16. But twenty hours into the tournament the tournament manager exceeded its disk quota which was set too low. So, to stay on track, the number of games had to be reduced to 225 per program. The final results were as follows:

rank	name	score	games	ratio
1.	umich1	1458455	225	6482.0
2.	brzo1	1136690	225	(*) 5051.9
3.	uofa1	1136790	225	(*) 5052.4
4.	creed1	559380	225	2486.1

Program brzo1 was leading over uofa1 almost all the time. So, team uofa happily conceded 2AD place to it (*) due to shortening the tournament. Entry creed1 made the server crash several times by referring to fully mined mineral patches which had vanished.

Game 2 Entries

umaas2

Authors: P. Kerbusch, N. Lemmens, M. Urlings, V. Vorsteveld (University of Maastricht, The Netherlands)

This entry creates 5-tank squads in single-file formation. The squad leader plans a path to the nearest base and others follow. When enemy tanks are encountered, a wedge formation is formed and the weakest of all tanks within range is attacked. Tanks move towards the weakest target while firing at the weakest target within range. When no more enemy tanks are in sight, the squad resumes its path in file formation. When a base is destroyed, a new base is located and the squad starts moving towards it. All objects excluding opposing tanks are considered obstacles and each tank reserves one tile.

umich2

Authors: Joseph Xu and Sam Wintermute (University of Michigan, U.S.A.)

This entry is a SOAR agent that attacks tanks before bases. Tanks are grouped by spatial distance, and groups of tanks will try to attack enemy tank groups that are smaller than them. If no such enemy groups exist, smaller groups will try to regroup into larger groups and go for their target then. Unfortunately, a bug was introduced just before the deadline, and most of this behaviour was not realized in the competition.

uofa2

Authors: K. Anderson, J. Bergsma, D. Demyen, T. Furtak, D. Tom, F. Sailer, N. Wiebe (University of Alberta, Canada), D. Deutscher (Tel Aviv University, Israel)

The program first finds a suitable meeting location for all tanks close to the average tank position. Then all

tanks are sent there after joining locally first. When the join operation is finished, the entire group starts hunting and attacking the closest enemy tank. When all tanks are destroyed, bases are attacked. The weakest targets are attacked first while minimizing overkill.

The task architecture utilized in this tournament entry (and also in uofa3) was designed to be simple, yet powerful. Each task is composed of a list of units assigned to it, as well as a list of child tasks, and a current line number. Each task also has an execute function which defines the task's behaviour. A task is defined as a series of statements to execute, and the navigation between these statements done by having a variable pointing to the current line number. The statements can range from giving individual units precise orders to creating subtasks for subsets of units. Finally, each task has an identical update function, which is executed whenever an object belonging to that task has completed an order or has been killed. If all the units of a task have completed their orders, the task executes its next statement as defined in the execute function. If the task has reached the end of its execute function, the task itself completes and notifies its parent and also gives control of the units back the the parent. The parent then executes its next statement, and so on. This framework allows complex strategies to be formulated by creating a series of subtasks, and combining them into more complex tasks.

Game 2 Results

Four hundred two-game matches were played for each player pair on June 17. Each game lasted at most 15 minutes. Here are the obtained results:

rank	name	score	matches	ratio
1.	uofa2	390.0	400	0.975
2.	umaas2	210.0	400	0.525
3.	umich2	0.0	400	0

Entry uofa2 won almost all of its games. It crashed in 20 games, but only lost 10 matches in total. The strategy of all tanks meeting near the center first and then hunting tanks with a big group tanks was quite successful. It is also hard to beat in the absence of area effect weapons. Therefore, in subsequent competitions control centers will likely be made weaker to make leaving bases undefended more risky.

Game 3 Entries

umich3

Authors: Joseph Xu and Sam Wintermute (University of Michigan, U.S.A.)

After gathering enough minerals and having built enough marines this SOAR agent sends marines to explore and attack. Defensive behaviour takes precedence, and all units are pulled into battle if the base is under attack.

uofa3

Authors: K. Anderson, J. Bergsma, D. Tom, T. Furtak, F. Sailer, N. Wiebe (University of Alberta, Canada), D. Deutscher (Tel Aviv University, Israel)

Using the task framework described earlier, this entry implemented a so-called turtling strategy which creates a barracks and enough workers such that each visible mineral patch is mined. It then produces as many marines as it can which wait for the opponent to arrive. The squad combat AI described earlier also controls all combat actions in this game and an older version of the mining AI controls gathering minerals.

Game 3 Results

Two hundred two-game matches were played on June 18 — each one lasting for at most 20 minutes. The results were as follows:

rank	name	score	matches	ratio
1.	umich3	124.0	200	0.62
2.	uofa3	74.0	200	0.37

When watching some replays it becomes apparent that there is much headroom in terms of increasing playing strength in game 3. Neither program expanded to other resource locations, nor did they create tanks in later game stages.

Conclusion and Outlook

In this paper we have presented the results of the first RTS game AI competition which was held in June 2006 and described the algorithms used in the tournament programs. Many areas of improvement have been identified, including ORTS documentation, program and server stability, group pathfinding, and high-level AI. We regard this as a promising beginning of a series of many future RTS game AI competitions which hopefully will help elevating the level of real-time AI to new heights.

Acknowledgments

Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

Buro M. and Furtak T., 2004. *RTS Games and Real-Time AI Research.* In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS).* 63–70.

Livingstone D., 2006. *Turing's test and believable AI in games. Computers in Entertainment (CIE)*, Vol. 4(1).

Spronck P.; Ponsen M.; Sprinkhuizen-Kuyper I.; and Postma E., 2006. *Adaptive Game AI with Dynamic Scripting. Machine Learning*, Vol. 63(3), 217–248.