# RTS Games and Real–Time AI Research

**Michael Buro & Timothy M. Furtak**
Department of Computing Science, University of Alberta, Edmonton, AB, T6J 2E8, Canada
email: (mburo|furtak)@cs.ualberta.ca

## Abstract

*This article[1] motivates AI research in the area of real–time strategy (RTS) games and describes the current status of the ORTS project whose goals are to implement an RTS game programming environment and to build AI systems that eventually can outperform human experts in this popular and challenging domain.*

Keywords: Real–time AI, simulation, multi–player games

## 1 Introduction

Commercial computer games are a growing part of the entertainment industry and simulations are a critical aspect of modern military training. These two fields have much in common, cross–fertilize, and are driving real–time AI research [11]. With the advent of fast personal computers, simulation–based games have become very popular. Today, these games constitute a multi–billion dollar enterprise. Examples are sports games — in which players control entire hockey, soccer, basketball teams, etc. — and real–time strategy (RTS) games where players command armies which clash in real–time. The common elements of simulation games are severe time constraints on player actions and a strong demand of real–time AI which must be capable of solving real–world decision tasks quickly and satisfactorily. Popular simulation games are therefore ideal test applications for real–time AI research. RTS games — such as the million-sellers Starcraft and Warcraft by Blizzard Entertainment and Age of Empires by Ensemble Studios — can be viewed as simplified military simulations. In these games several players struggle over resources scattered over a 2D terrain by setting up economies, building armies, and guiding them into battle in real–time. The current AI performance in commercial RTS games is poor by human standards. This may come as a surprise because RTS games have been around for more than ten years already and today's low–end computers can execute more than a billion operations per second. The main reasons why the AI performance in RTS games is lagging behind developments in related areas such as classic board games are the following:

• **RTS game worlds feature many objects, imperfect information, micro actions, and fast-paced action.** By contrast, World–class AI players mostly exist for slow–paced, turn–based, perfect information games in which the majority of moves have global consequences and human planning abilities therefore can be outsmarted by mere enumeration.

• **Market dictated AI resource limitations.** Up to now popular RTS games have been released solely by game companies who naturally are interested in maximizing their profit. Because graphics is driving games sales and companies strive for large market penetration only about 15% of the CPU time and memory is currently allocated for AI tasks. On the positive side, as graphics hardware is getting faster and memory getting cheaper, this percentage is likely to increase – provided game designers stop making RTS game worlds more realistic.

• **Lack of AI competition.** In classic two–player games tough competition among programmers has driven AI research to unmatched heights. Currently, however, there is no such competition among real–time AI researchers in games other than computer soccer. The considerable man–power needed for designing and implementing RTS games and the reluctance of game companies to incorporate AI APIs in their products are big obstacles to AI competition in RTS games.

In what follows, we first take a closer look at AI challenges in RTS games to motivate this domain as being well-suited for real–time AI research. Then we will describe the ORTS project in some detail whose goals are the implementation of an RTS game programming environment and to build AI systems that can defeat human RTS game players. Finally, we discuss related work and close with an outlook.

## 2 RTS Games and AI Research

RTS games offer a large variety of fundamental AI research problems, unlike other game genres studied by the AI community so far:

• **Adversarial real–time planning.** In fine–grained realistic simulations, agents cannot afford to think in terms of micro actions such as "move one step North". Instead, abstractions of the world state have to be found that allow AI programs to conduct forward searches in a manageable abstract space and to translate found solutions back into action sequences in the original state space. Because the environment is also dynamic, hostile, and smart — adversarial real–time planning approaches need to be investigated.

A typical strategic problem that illustrates the necessity for adversarial planning is shown in Fig. 1. All corner regions are sealed off by strips of trees. In this Warcraft–2 map lumber and gold are the resources and no air transports are available. It does not take long for human players to realize that it is necessary to cut through the trees

[1] A preliminary and condensed version of this paper has been presented in the IJCAI–2003 poster session.

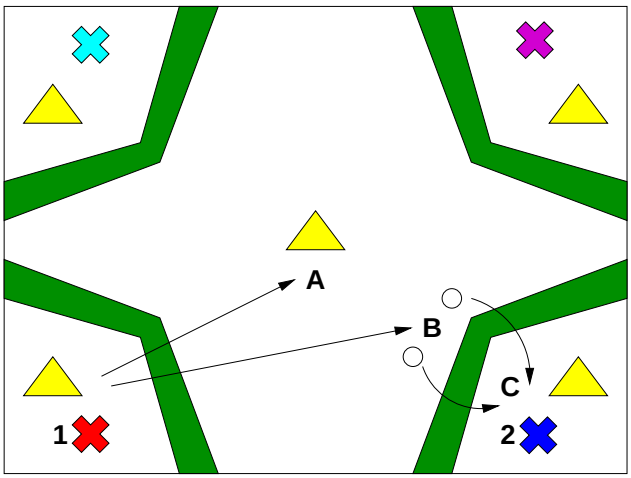


Figure 1: RTS game scenario. The start locations of up to four players (marked with crosses) are sealed off by strips of trees. Triangles represent gold mines. An acceptable AI system needs to figure out that chopping trees right away to reach the mine in the center quickly is necessary to win the game.

right away to claim and defend the gold mine in the center [A]. The computer player (2), however, is clueless and only starts chopping trees after running out of gold. It lost after being sieged by the human player (1) [B,C].

• **Decision making under uncertainty.** Initially, players are not aware of the enemies' base locations and intentions. It is necessary to gather intelligence by sending out scouts and to draw conclusions to adapt. If no data about opponent locations and actions is available yet, plausible hypotheses have to be formed and acted upon.

• **Opponent modeling, learning.** One of the biggest shortcomings of current (RTS) game AI systems is their inability to learn quickly. Human players only need a couple of games to spot opponents' weaknesses and to exploit them in future games. New efficient machine learning techniques have to be developed to tackle these important problems.

• **Spatial and temporal reasoning.** Static and dynamic terrain analysis as well as understanding temporal relations of actions is of utmost importance in RTS games — and yet, current game AI programs largely ignore these issues and fall victim to simple common–sense reasoning [8].

• **Resource management.** Players start the game by gathering local resources to build up defenses and attack forces, to upgrade weaponry, and to climb up the technology tree. At any given time the players have to balance the resources they spend in each category. For instance, a player who chooses to invest too many resources into upgrades, will become prone to attacks because of an insufficient number of units. Proper resource management is therefore a vital part of any successful strategy.

• **Collaboration.** In RTS games groups of players can join forces and intelligence. How to coordinate actions effectively by communication among the parties is a challenging research problem. For instance, in case of mixed human/AI teams, the AI player often behaves awkwardly because it does not monitor the human's actions, cannot infer the human's intentions, and fails to synchronize attacks.

• **Pathfinding.** Finding high–quality paths quickly in 2D terrains is of great importance in RTS games. In the past, only a small fraction of the CPU time could be devoted to AI tasks, of which finding shortest paths was the most time consuming. Hardware graphics accelerators are now allowing programs to spend more time on AI tasks. Still, the presence of hundreds of moving objects and the urge for more realistic simulations in RTS games make it necessary to improve and generalize pathfinding algorithms. Keeping unit formations and taking terrain properties, minimal turn radii, inertia, enemy influence, and fuel consumption into account greatly complicates the once simple problem of finding shortest paths.

Playing RTS games is challenging. Even more challenging is the creation of autonomous real–time systems capable of outperforming human experts in this domain. The range of applications of RTS real–time AI modules is by no means limited to creating smart opponents to entertain human players. High–performance simulators are in high demand for training military personnel today and will become the core of automated combat and battlefield decision–support systems of tomorrow. In [22] it is predicted that 20% of the U.S. armed forces will be robotic by 2015. The current state of real–time command and control ($C^2$) AI, in particular in the RTS games domain, is less than satisfactory: computer opponents do not: smartly adapt to adversaries, learn from their own mistakes, look–ahead in abstracted search spaces, reason about spatial and temporal object relations, nor do they collaborate and communicate well. Human experts, on the other hand, excel in all those areas. This is true not only for the chosen domain. However, by concentrating on a concrete and bounded real–time decision task that features a number of challenging but manageable sub–problems of general interest, we think the chance of accomplishing the goal of creating a strong autonomous $C^2$ AI system within a couple of years is good. The results of RTS game research will increase our understanding of fundamental AI problems — such as opponent modeling and adversarial real–time planning — and will have considerable impact on the real–time control domain in general and the computer games industry in particular which is in need of creating credible computer controlled agents.

It is important to note that our and the games industry's AI objectives differ: while we are interested in creating strong AI systems capable of defeating the best humans, game developers want to maximize the replay value of their titles which precludes average customers losing all the time against computer players. We acknowledge this fact but point out that strong AI components can be toned down to adjust to weaker players, whereas creating

games in which people enjoy interacting with computer allies and opponents is virtually impossible when using weak AI modules. A good example is the widely used technique of scripting RTS game AI based on rule sets. While those systems may be challenging when played by beginners for the first time, humans quickly learn how to counter canned computer strategies and to exploit holes in the fixed rule base.

## 3 The ORTS Project

RTS games have become quite popular in recent years. Tournaments with considerable prize money are being held regularly all over the world. The development of high–performance AI systems can therefore benefit a lot from human expertise. Thus, it is natural to tap into available RTS game resources to learn from human experts and to measure AI performance in tournaments. Unfortunately, game companies are not inclined to release communication protocols or to add AI interfaces to their products. Both of these features are needed to let AI researchers integrate AI modules with a game to aid human players or to play games autonomously. Moreover, current RTS game simulations rely on client–side simulations in which all client machines run the entire game simulation and just hide information from the respective players. While this approach saves bandwidth in case the command frequency is small, it is prone to map–revealing hacks which spoil the game experience just as badly as revealing opponents' cards in poker.

To overcome these problems, the Open–Real–Time–Strategy (ORTS) project was conceived in 2001 [4]. The short term project goal is to set up a programming environment for conducting real–time AI experiments. The long term goal of the ORTS project is the creation of AI systems whose performance surpasses that of human experts in real–time command and control domains. Central to this is the implementation of an RTS game server which allows researchers to connect their AI systems to measure AI performance in real–time domains with the following properties:

- Objects navigate and interact in initially uncharted worlds in real–time. Terrain features include deep seas, rivers, plateaus, and ramps. Objects have radar–like vision which is only obstructed by elevation. They can be airborne, land–based, or naval.
- Players compute actions for a set of objects at their command. The computational model can range from local to global with respect to the objects in order to reflect given command hierarchies and different levels of physical restrictions imposed by the world.
- Object actions include straight–line motion[2], attacking objects, gathering resources, trading goods, building, upgrading, and repairing objects.
- Team players can communicate and can share views of the world and object control.
- Top–level goals of the players include: destroying all opponent objects, reaching a designated location first, or gathering as much resources as possible in a given time period.

Compared with commercial RTS titles the ORTS system has the following advantages:

• **Free Software.** ORTS is released under the GNU Public License (GPL) which means that anyone can download the source code at no cost in order to learn how the system works and to contribute to the project by submitting bug fixes and adding new features. It also means that projects that incorporate ORTS code need to release their source code as well. It is important to point out that this does not prevent users of GPL'ed code from selling their software. The benefit of GPL'ed software releases to the community is huge as witnessed by the success of the Linux, KDE, and Mono projects. We invite game companies to release source code of their popular but no longer sold or maintained games to the community so that we all can learn.

• **Flexible Game Specification.** ORTS is a generic RTS game programming environment. It provides the infrastructure for RTS games including a server and a graphics client. However, the actual game played when using the ORTS system is not fixed — as in commercial RTS games — but scripted. A script is used to define unit properties — such as size, sight range and maximum speed — and unit actions. The freedom to adjust RTS games to the needs of AI researchers is important. However, we also acknowledge the importance of providing standard setups in order to attract human players and to spark competition in form of tournaments. The community can help here because ORTS is free software.

• **A hack–free server–side simulation.** The ORTS game server maintains the entire world state and sends only visible information to players which connect from remote machines. Map–revealing client hacks that are common in commercial client–side simulations are therefore impossible. The additional bandwidth requirement is mitigated by compressed incremental updates.

• **Total object control.** Today's commercial RTS games confine users to single view graphical user interfaces, fix low–level unit behavior, and sometimes use veiled communication. The ORTS system, on the other hand, employs an open message protocol that allows AI researchers and players to connect *whatever* client software they like — ranging from split–screen GUIs, over hybrid systems in which AI components aid the human player, to fully autonomous AI systems. ORTS clients have complete knowledge of all their units and visible terrain at all times. Therefore, there is no need for switching focus back and forth in case of multiple simultaneous battles. Another big advantage is that in ORTS there is no prescribed low–level unit behavior, which in commercial RTS games often is too simplistic and awkward. Instead, clients send each and every micro action — including breaking down paths into straight line segments — to the server. Furthermore, in each simulation cycle actions can be generated for *all* units

[2] This seems to be a severe restriction. However, curved trajectories can be approximated arbitrarily well by line segments.

unlike in client–side simulations where the command frequency is very limited. ORTS clients are therefore in total control of their units.

• **Remote AI.** In commercial client–side simulations the AI code for all players runs on all peer nodes to save bandwidth and to keep the simulations synchronized. This creates unwanted CPU load. Moreover, user configurable AI behavior is limited to simple scripts because there are no APIs to directly connect AI systems that run on remote machines. By contrast, conducting AI experiments in the ORTS environment is easy. Its open message protocol allows users to even connect super computers to either play RTS games autonomously or to aid human players.

Popular games in which human players still have the upper hand are ideal test–domains for AI research. By providing a free software RTS game environment we hope to spark interest in the $C^2$ domain among real–time AI researchers. The open design allows the construction of hybrid AI systems in which human players are aided by AI modules of growing capabilities. Competitive game playing on an ORTS Internet game server is therefore likely to improve AI performance and ergonomic GUI design.

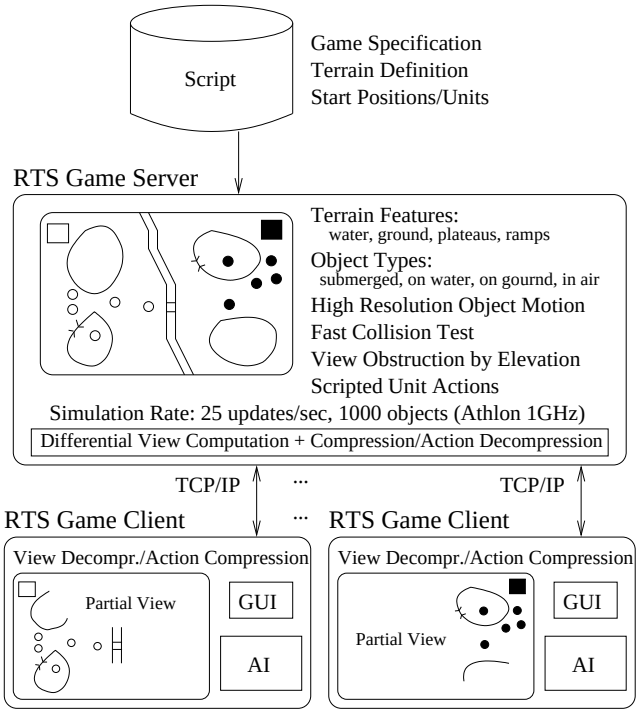


Figure 2: ORTS Component Overview

## 4 ORTS Components and Performance

In this section we will give a high–level overview of the currently implemented ORTS components (Fig. 2), judge their performance with respect to the outlined project goals, and discuss future enhancements.

ORTS games are defined by blueprints that describe objects, their properties — such as sight range and armor — and their possible actions in the ORTS world. When starting the server it reads this information together with the terrain description and initial unit locations from a text file. The server then waits for all ORTS clients to connect, sends the game description to them, and enters the simulation loop. In this loop the server first sends out the individual world views to all clients, it then waits a specific time for the clients' action messages, updates the world state accordingly, and repeats this process until the game is over.

The critical components with respect to server/client performance are object vision, motion, and collision computation, as well as data transmission between server and clients. Efficient algorithms for object motion, collision detection, and incremental client view updates were already presented in [4]. Since then we increased the performance of these parts and added game scripting, view obstruction by elevation, a new robust network protocol, and a 3D graphics client to the ORTS system. In what follows we will sketch the new developments.

### 4.1 Game Scripting

Scripting languages are often used in modern commercial games to simplify game development. By providing a simplified interface to the game engine — often in the form of interpreted high–level languages — developers are spared from dealing with intricate source code. Instead, adjustments to scripts that concisely describe game attributes can be tested rapidly because no recompilation is needed. In RTS games scripting is useful in several places. We started by scripting the game definition and plan to extend scripting to unit control and GUI customization.

ORTS objects are represented as containers for integer variables, actions, and sub–objects. From these containers a set of blueprints is defined which describes the names and initial values of attributes, the available actions, and the structure of an object. All objects in the game are instances of one of the blueprints which are read from a file at the start of a game. With the exception of certain attribute names such as "hitpoints" and "sight", the server does not infer meaning from these elements. Rather, it is the actions attached to each object, all of which may be scripted, that define the interactions within the world. This allows for a wide range of games to be described and simulated in the ORTS environment without the need to add special extensions. The action scripts refer to attributes, components, and actions by name and so do not depend on any knowledge about the object's type. The scripts themselves are able to perform complex actions including:

- **creating new objects** – permanent and temporary objects can be created from a blueprint and assigned an owner.
- **queuing actions** – either for immediate evaluation or for a later cycle.
- **interfacing with the game state** – accessing and

modifying internal data structures via registered functions.

- **iterating over objects in a region** – allowing the scripting of area effects.

The scripting language is defined recursively in terms of integer values (either constants, variables, or object attributes), basic arithmetic and logical connectives, conditional expressions, and pre–defined functions. These functions are inserted as compiled expressions within the script and are used as helpers to provide access to low–level data structures and commonly used internal functions. It is often the case that when the scripting language is unable or is too awkward to use for a specific task, a helper function may be quickly created by defining a keyword and associating a function within the script parser.

In addition to scripted actions defined within the blueprints, actions may be compiled functions within the server that are explicitly added to a list of available actions. These actions can then be referenced by name within the object description and a link to the compiled function added to the internal blueprint.

## 4.2 Object Motion and Vision

The most time consuming server tasks are object motion and visibility computations. In each game cycle moving objects have to be advanced and checked for collisions. In the ORTS implementation sector based computations reduce the naive implementation's $\Theta(n^2)$–time for checking $\binom{n}{2}$ potential object interactions to $O(n)$ [4].

A recent addition to the server is view obstruction by terrain elevation which adds more realism and tactical options to ORTS games. Quickly computing what is visible in RTS games is nontrivial because potentially hundreds of objects are constantly moving and thereby changing their view. The tile–based nature of the ORTS world naturally lends itself to describing an object's visual field in terms of the regions visible from the tiles it is occupying. Regardless of the criteria used to determine visibility, the highly static nature of terrain obstructions allow the results of those visibility computations to be reused for any object looking out from that tile afterward.

ORTS generates the entire view independent of the last cycle. As a result, it is unaffected by changes in unit positions or by large portions of the view being quickly hidden and revealed as would be the case when moving troops through highly obstructed terrain. Because the number of interactions (collisions, enemy encounters, etc.) increases with the number of moving objects, computations have been optimized to reduce the resources needed in this worst case. The execution time of the vision computation is linear in the number of objects ($N$), the number of tiles ($T$), and the number of players ($P$), for a time complexity of $O\left(N + P{\cdot}T\right)$ per simulation cycle (Fig. 3). The performance is initially greater for small map sizes due to a greater number of objects being on the same tile. This reduces the number of tile views used relative to the number of objects. The space complexity is also $O\left(N + P{\cdot}T\right)$ if one assumes a fixed maximum sight range.

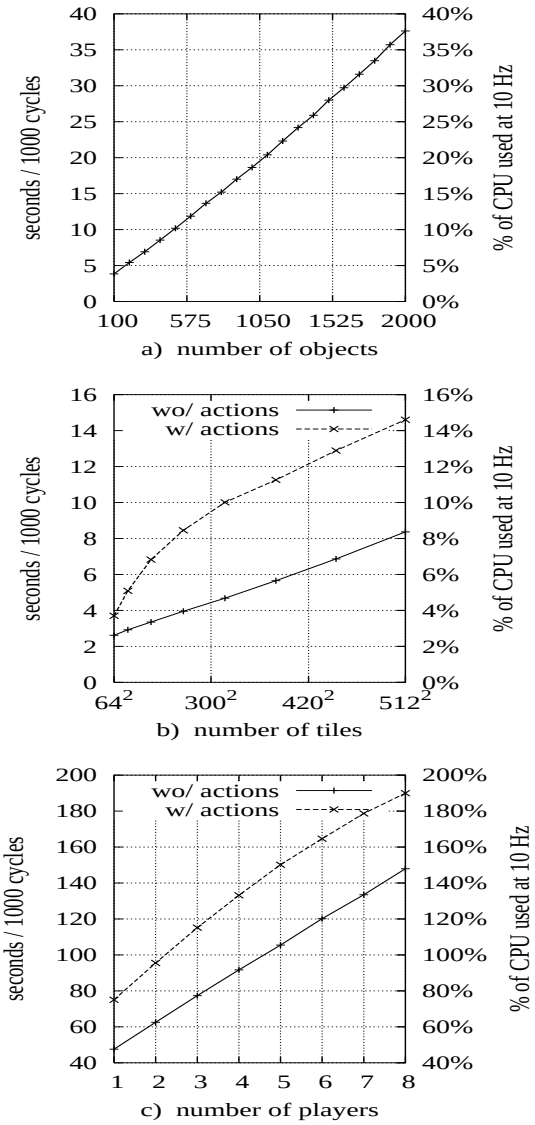


Figure 3: Wallclock benchmarks on an Athlon XP 2400+ with respect to changes in a single variable. Constant values for each graph are: a) one player, 32768 tiles; b) one player, 200 objects; c) $362^2$ tiles, 1680 objects. The running time flattens out in b) around $300^2$ tiles as the number of objects per occupied tile decreases to one.

## 4.3 Communication and Networking

In each cycle the server sends the state of the world to each client as they perceive it and the client responds by sending a list of actions for the objects it has control of. As the world is explored and portions of the map are revealed, the server sends a list of the newly visible tiles along with a description of their topography (height, ground type, whether or not the tile slopes in a particular direction). Objects are entirely described by the index of the blueprint used to create the object, and a vector of their current attribute values. Subsequent viewings of the same object (if the object has

not been lost from sight) are given in terms of the attribute changes from the previous frame.

With the increasing speed of network communication, such data rates are within the limits of current high–speed Internet connections. By applying a moderate amount of compression to the client's view prior to sending it, the necessary throughput is greatly reduced to acceptable levels for even large multi–player games (Fig. 4).

The experimental results reported in [4] suggest that the main communication bottleneck when using server–side simulation and high–speed connections such as cable–modems or DSL is lag rather than data throughput. Lag is induced by data transfer over networks and by associated computational overhead such as message compression/inflation and updating data structures on both communication ends. For the sake of simplicity the first ORTS implementation ignored network lag and indeed used blocking TCP I/O on both the server and client side. As we

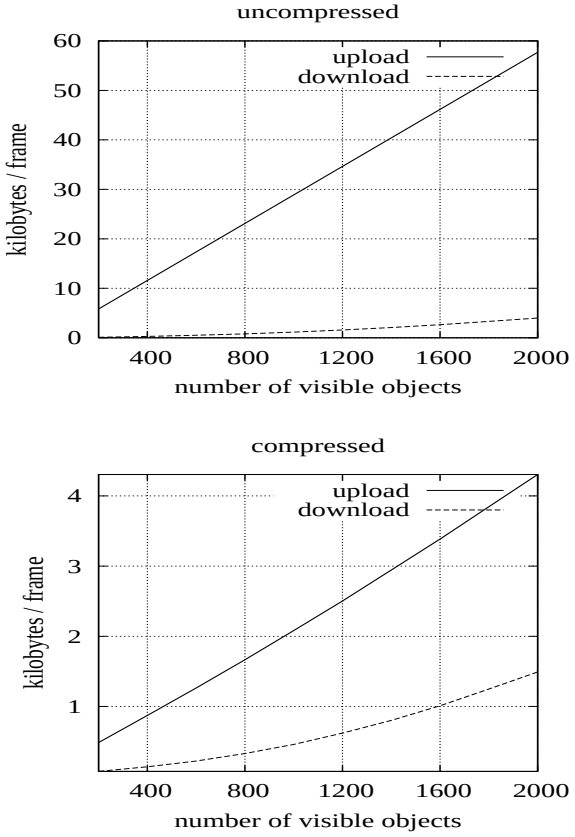


Figure 4: Server–side data rates — with and without compression — averaged over 1000 frames with client generated actions. Each object has on average 26 attributes, and requires approximately 1 byte per attribute to encode the change from the previous cycle. The sending of tile information is included in these measurements, although their contribution to the average approaches zero since each tile is only sent once.

are now moving towards a fully functional RTS game environment latency issues need to be addressed. The most pressing question is what should happen when during a game one or more players experience unusually high network lag. In current commercial RTS games clients detect communication lag and adjust to it by first slowing down the game up to the certain point when players are disconnected.

For ORTS we have developed an alternative scheme that avoids the decision on whether or when to kick out players based on network lag altogether. The idea is that rather than disconnecting clients when they do not answer in time they just forfeit the opportunity to issue actions to their units in that particular simulation frame. Two problems have to be addressed: what should happen when the server blocks while sending out the current view to a client or when the server receives multiple action messages from a client? Both problems can be solved by using buffered I/O: in the first case messages accumulate in the server's send buffer. In order to quickly recover from heavy network lag, several consecutive differential view messages in the send buffer can be replaced by single messages that encode the current view without referring to earlier messages. For resolving the second problem the server needs to be able to remove multiple messages from the receive buffer in a single simulation frame because otherwise clients that experienced lag can never catch up. A simple and effective strategy is to merge action messages from several frames. The outlined asynchronous communication scheme has another big advantage: combined with proactive action messages sent by lagging clients it solves the lag issue in RTS games — provided that a good response to the next world view(s) is highly predictable. In this regard the start/stop nature of actions in ORTS is helpful because even if action messages are delayed, units will — for instance — continue moving or attacking targets once those actions have been initiated.

### 4.4 ORTS Client Software

Ergonomic graphical user interfaces are essential for human RTS game players. Because fast actions and reactions are of utmost importance in this game genre players welcome keyboard shortcuts for common actions, short mouse move distances, action queues, rally points, associating keys with unit groups etc. While most commercial RTS games now provide these functions, their human–machine interface is still limited compared to what is possible in an open software environment like ORTS. For instance, commercial RTS games still only provide a single sector view of the playing field. This forces players to quickly switch views back and forth in case different parts of the playing field need attention. Other problems are the limited ability to tailor GUIs and insufficient AI support in the client.

We recently added a 3D graphics client to the ORTS system on top of the client software which communicates with the server and maintains the game state. The screen shot in Fig. 5 shows a part of the playing area on the left hand side featuring hilly terrain, some ground and air units, and a building. On the right hand side command buttons for the selected aircraft is displayed as well as the minimap


Figure 5: A screen shot of the ORTS OpenGL graphics client.

which gives the player a global view of the currently visible regions and units. It is important to note that ORTS's open architecture does not confine users to a particular interface. Everybody is free to connect their own client. Knowledge of C++ and the willingness to study parts of the ORTS code makes it possible to add new functionality to the ORTS GUI with direct implications to game playing performance. Consider for instance scripts attached to units for handling low–level behavior such as fleeing in the face of strong opposition or patrolling between two locations. Because the entire behavior of all units is controlled by client software it is possible to plug in smart AI helper modules for many tasks that require multitasking and quick reactions. Prominent examples are automatic attacks when opponent units come into sight or handling close combat situations efficiently. Currently, players have to micro–manage many activities such as concentrating fire. This is unacceptable in large RTS game scenarios where often hundreds of units are engaged at different locations simultaneously.

We have just begun to add scripting support to the graphics client which eventually will greatly simplify its customization. Once scripting has matured and a standard ORTS game has been designed we envision interested people to start contributing to the project in form of AI modules and further GUI enhancements.

## 5 Related Work

Research on computer soccer pursues goals similar to those outlined in this paper and has become quite popular [19]. This domain can be regarded as a simplified RTS game with only a few objects, no economy, unremarkable terrain features, and more or less complete information. Another big difference is that agents in computer soccer are required to compute their actions locally. While this decision makes perfect sense when studying collaboration in autonomous multi–agent settings, there is no way of strictly enforcing it in games played on wide–area networks. In RTS games the player's role is a manager who by definition has a global view and needs to think globally. Given its limited view, no single object in RTS games can even approximate global plans because the playing field is big and possibly many local battles in different regions are fought simultaneously. Thus, the AI focus in RTS games has to be on planning on the global scale and the AI system does not need to be restricted to computation local to the units. It is therefore possible to connect any client software running on remote machines to the server without worrying about cheating.

ORTS is not the first and only free software RTS game project. Most of these projects are poorly maintained or are still in the design phase with limited working code available for use by AI researchers. The notable exception is Stratagus (www.nongnu.org/stratagus) — formerly known as FreeCraft. Stratagus uses client–side simulation and is therefore prone to client hacks and not suited for real–time internet AI competitions. Nevertheless, it has recently been used as test–vehicle for MDP related planning research [10].

Many articles on robot motion, planning, temporal and spatial reasoning, and learning are relevant to constructing AI systems for RTS games. The SOAR architecture — for instance — and its application to first–person shooter games [14; 15] as well as M. Atkin's work on the GRASP system that is applied to a capture–the–flag war game [1; 2] are highly significant. Both projects have created high–performance game programs and represent the state–of–the–art in planning research applied to games.

The other large body of literature relevant to this work is on military analyses and applications. Research in this area spans from mathematical combat models [12] over computer generated forces — which are used in simulation and training — to decision–support systems that aid commanders and troops on the battle–field or even control entire weapon systems autonomously. This project brings both research communities together.

## 6 Conclusion and Outlook

In this paper we have motivated AI research in the domain of RTS games. We also described the current state of the ORTS project whose goal it is to implement a programming infrastructure for RTS game AI research and to build AI systems that eventually outperform human players. The rich set of research problems that have to be tackled in order to reach human performance in these games span from pathfinding over temporal reasoning to adversarial real–time planning. Most of these problems have applications outside the game domain. Examples include autonomous robot navigation in hostile environments and simulators for training military personnel. We encourage AI researchers to consider RTS games as test–domain and invite programmers to join our efforts to make our free software RTS

game system attractive to both human players and AI researchers. The resulting competition will then hopefully drive real–time AI performance to new heights as it did in many classic game domains.

## Acknowledgments

We thank Harry Wentland and Keith Yerex for contributing to the ORTS client software. This research is partly funded by the first author's NSERC discovery grant and the second author's NSERC scholarship.

## References

[1] M.S. Atkin. AFS and HAC: Domain-general agent simulation and control. In *AAAI Workshop on Software Tools for Developing Agents*, 1998.

[2] M.S. Atkin and P.R. Cohen. Physical planning and dynamics. In *Working notes of the AAAI Fall Symposium on Distributed Continual Planning*. 1998.

[3] M. Buro. From simple features to sophisticated evaluation functions. In *LNCS volume 1558 – Proceedings of the First International Conference on Computers and Games*, pages 126–145. Springer–Verlag, 1998.

[4] M. Buro. ORTS: A hack-free RTS game environment. In *Proceedings of the Third International Conference on Computers and Games*, pages 156–161, 2002. Software: http://www.cs.ualberta.ca/˜mburo/orts/orts.html.

[5] C. Dawson. Formations. In Steve Rabin, editor, *AI Game Programming Wisdom*, pages 260–271, 2002.

[6] M.E. des Jardins, E.H. Durfee, C.L. Ortiz, and M.J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

[7] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, 2001. 2nd edition.

[8] K.D. Forbus, J.V. Mahoney, and K. Dill. How qualitative spatial reasoning can improve strategy game AIs. *IEEE Intelligent Systems*, 17(4):25–30, July 2002.

[9] J. Gratch and R. Hill. Continuous planning and collaboration for command and control in joint synthetic battlespaces. In *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*, 1998.

[10] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003. Software available at http://dags.stanford.edu/Freecraft/.

[11] J.C. Herz and M.R. Macedonia. Computer games and the military: Two views. *Defense Horizons, Center for Technology and National Security Policy, National Defense University*, 11, April 2002.

[12] A. Ilachinski. Land warfare and complexity. CRM 96-68, Center for Naval Analysis Research, 1996.

[13] A. Ilachinski. Irreducible semi-autonomous adaptive combat: An artifi cial-life approach to land warfare. Memorandum 97-61.10, Center for Naval Analysis Research, 1997.

[14] J. Laird. Using a computer game to develop advanced AI. *Computer*, 34(7):70–75, July 2001.

[15] J. Laird, A. Newell, and P.S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence Journal*, 33(3):1–64, 1987.

[16] D. Pottinger. Terrain analysis in real–time strategy games. In *Proceedings of Computer Game Developer Conference*. 2000.

[17] D. Reece, M. Kraus, and P. Dumanoir. Tactical movement planning for individual combatants. In *Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation*. 2000.

[18] P. Stone. *Layered Learning in Multi–Agent Systems*. Computer Science Department, Carnegie Mellon University, 1998. Ph.D. Thesis CMU–CS–98–187.

[19] P. Stone. Multiagent competitions and research: Lessons from RoboCup and TAC. In *RoboCup International Symposium*, Fukuoka, 2002.

[20] P. Tozour. Strategic assessment techniques. In M. DeLoura, editor, *Game Programming Gems 2*, pages 298–306, 2001.

[21] W. van der Sterren. Terrain reasoning for 3D action games. In M. DeLoura, editor, *Game Programming Gems 2*, pages 307–323, 2001.

[22] S. von der Lippe, R.W. Franceschini, and M. Kalphat. A robotic army: The future is CGF. In *Proceedings of the 8th Conference on Computer Generated Forces and Behavioral Representation*, Florida, USA, 1999.

## Author Biographies

**MICHAEL BURO** is an associate professor for computer science at the University of Alberta. After receiving his Ph.D. in Germany he worked as a scientist at the NEC Research Institute in Princeton for seven years before he moved to Edmonton in 2002. His main research interests are heuristic search in AI and machine learning applied to games. He is the author of LOGISTELLO – a learning Othello program that defeated the human World–champion 6–0 in 1997.

**TIMOTHY M. FURTAK** is a third year honors mathematics major at the University of Alberta. He has spent the last nine months working on the ORTS game engine and is interested in applying machine learning to games.