Planning for ORTS, an open-source real time strategy game

Stewart Armstrong and **Liam Kelly** and **Derek Long** and **Alex Coddington** and **John Levine**
Department of Computer and Information Sciences
University of Strathclyde
Derek.Long, Alex.Coddington, John.Levine@cis.strath.ac.uk

Abstract

We describe an exploratory project in which a planner was used within a real-time strategy game to create plans for behaviours which were then executed under the control of a computer player. We describe the problems we encountered, the solutions we adopted and the questions that remain unanswered in this investigation. One of our key objectives was to explore the viability of integrating existing generic planning technology into the control management of a computer-controlled player and we report on the extent to which we were able to achieve this.

1 Introduction

A team of researchers at the University of Alberta have developed an open source real-time strategy (RTS) game (Buro 2002), called ORTS (open RTS). The game follows the pattern of other RTS games in offering players the challenge of starting from a very limited initial position and building up resources, infrastructure, military power and, ultimately, of defeating all opponents to achieve complete domination of the game world. In this game the setting is a futuristic world in which robotic drones collect resources and support the construction of barracks in which space marines can be trained, factories in which tanks can be constructed and, ultimately, wars waged against threatening opponents.

The game has been designed specifically to support the easy integration of AI controllers for computer players, using a server-client architecture, with the intention of comparing different approaches to the construction of artificial players. The challenge has been codified in a series of specific game scenarios, ranging from the relatively simple task of collecting as much resource as possible within a fixed time period, through waging battles against intelligent opponent forces to the ultimate challenge of constructing an entire infrastructure and achieving complete game world domination. Open competitions to meet these challenges have been held since 2006 (Buro 2003).

This paper describes work carried out by two undergraduate computer science students over a 10-week period in the summer of 2008, sponsored by the EPSRC and Nuffield Foundation through their student vacation-bursary schemes. The objective of the work was to investigate the problems in integrating a planner into the control of an artificial player for ORTS. We describe the problems that were faced, the solutions that were implemented, the strengths and weaknesses of those solutions and the questions that remain open now that the project is complete.

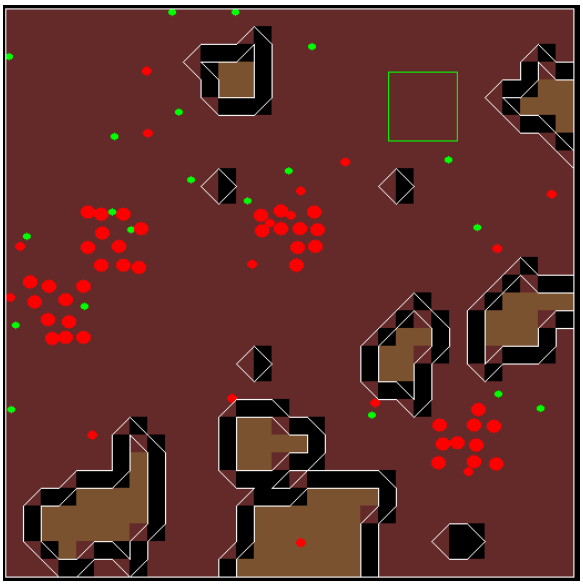

Figure 1: The ORTS world view in a simple 2D viewer, showing the hills and various agents

2 The ORTS Game

The game is played by one or more players who share a limited terrain. The terrain is constructed from square tiles, each of which contain a small 16-by-16 grid of internal locations. This means that objects in the world can partially occupy grid cells and the effective discretisation of the terrain is very fine-grained with respect to the world agents. Some terrain is blocking (representing impassable hills) and this terrain fills entire tiles or half tiles split diagonally (see figure 1). An attractive rendering of one version of the world can be seen in figure 2.

The game engine is designed to make integration of computer players easy: the architecture is a classic client-server structure, in which the server manages the game world simulation and clients, each responsible for the resources of a

Figure 2: The ORTS world view rendered for human viewers, showing agents, buildings and terrain

single player, interact with the server by sending instructions for their agents and receiving information about the state of the world as far as they can perceive it. In the more advanced scenarios the fog-of-war effect typical in RTS games is in effect: the terrain is known to each player only as far as it is observed by the player's forces. This means that scouting, reconnaissance and, potentially, the management of uncertainty, are all part of the challenge the players must face.

The architecture is rather impractical for a commercial RTS, since it introduces overhead into the time-critical frame generation cycle, but it is a very significant design achievement that different player controllers can easily be plugged into the game engine and compared. This supports, in particular, a direct competitive comparison of players and there has been a series of competitions since 2006, with the most recent taking place in August 2008. The competition involves four specific game scenarios, offering different challenges:

- *Game 1* is a resource-gathering game where the objective of the game is to gather as many resources as possible within 10 minutes. It is a single player game with perfect information that involves one control center and 20 nearby workers. The terrain contains a number of irregular static obstacles and several resource patches. A maximum of four workers are allowed to work on a single resource patch. There are a small number of randomly moving small obstacles (or "sheep"). The challenges include cooperative pathfinding and both static and mobile obstacle avoidance. The key to success in this game is to achieve clever traffic management so that workers do not waste time competing for entry into the same space (collisions are resolved randomly in favour of one or other agent and the loser does not move).
- *Game 2* is a two-player game, with perfect information, where the objective for each player is to destroy as many of the opponent's buildings as possible within 15 minutes. Each player has five randomly located control centres with ten tanks each nearby. The environment contains irregular static obstacles as well as a number of randomly-moving small mobile obstacles. The challenges involve being able to engage in small-scale combat, managing units and groups, and adversarial and cooperative pathfinding. This game offers different routes to victory, including direct confrontation with the enemy troops in order to deprive the opponent of the means to inflict damage, direct assault on the enemy buildings to attempt to inflict more damage than the opponent by efficient management of forces, or some combination of the use of screening or diversionary forces and assault forces.
- *Game 3* is a two-player real RTS game. In this game the information is imperfect (there are simultaneous actions and fog of war) and the objective of the game is to destroy all opponent buildings within 20 minutes. Each player has a randomly chosen starting location with one control centre and a resource cluster with six workers nearby. There are four further resource locations placed randomly in the terrain which contains a number of small, irregular static obstacles as well as mobile, randomly moving "sheep". The workers may be used to build control centres, barracks and factories. Building actions tie up one worker which will become immobile and located out of bounds. Workers can be "trained" (produced) in a control centre, marines are trained in barracks and tanks are built in factories. A control centre is required in order to build a barracks, whereas building a factory requires a control centre as well as a barracks. This game has a number of challenges which include, pathfinding, combat, grouping forces, scouting and resource allocation. The hierarchy of buildings is typical of RTS games, although this version uses a very small hierarchy (yet still presents a very difficult challenge to the players who have entered in the competitions so far).
- *Game 4* is a two-player game with perfect information (apart from simultaneous actions). The terrain in unobstructed and contains a number of small randomly moving "sheep". Each player has 20 siege tanks and 50 randomly positioned marines occupying the left or rightmost quarter of the map. Units are diagonally symmetric to the opposing player. The objective of the game is to destroy as many opponent units as possible within 5 minutes. Challenges involve small-scale combat, unit group management, handling heterogeneous groups and adversarial/cooperative pathfinding.

In this project we concentrated on a simplified version of scenario 3, in which we did not have opponents, but were concerned with the construction of a hierarchy of buildings and managing the infrastructure and resources required to achieve that.

2.1 Computer Players in ORTS

The role of a computer player in ORTS is to generate instructions for game agents under the player's control in real time. Game time advances in real time, but is discretised into small simulation steps corresponding to successive frame updates. When instructions take multiple frames to execute the agents enacting the instructions do not need to be given

new instructions, but can be left to continue in their tasks. Thus, the computer player must issue instructions to agents at appropriate times, observing the agents executing their tasks and responding as the tasks are completed. If no instructions are issued then the server will not wait, but simply instruct the agents to perform default activities, which will usually be to stay still and await further instructions.

To make the design of computer players easier, the University of Alberta team have supplied a rich hierarchy of basic structures for the command of agents. The structures form a hierarchy of commanders, with a supreme commander taking control of all the resources and acting as the player's proxy in the game. Subsidiary commanders are allocated resources by the supreme commander and take responsibility for specific subtasks. There are commanders implemented for such things as basic resource gathering using workers, defence and simple attack. There are also simple tools implemented to aid in path planning and navigation of agents.

3 The Role of Planning

When playing an RTS game, human players certainly have the intuition that they are planning. There is a strategic framework for their actions and they construct sequences of basic actions that lead to very specific planned outcomes. It is clear, however, that there are many aspects of playing these games that conflict with the assumptions made in classical planners: there is significant uncertainty both in the initial state, caused by the fog of war, and in the subsequent execution of actions due to the interaction with other players and randomly mobile obstacles; there are opponents who are deliberately attempting to subvert the player's plans; there are events that occur without the intervention of the player. These qualities have been observed in many other domains in which planning appears to play an important role and there are various approaches to attempting to manage them. One is to simply ignore the problems and treat the situation as though it were fully observable, with no opponents and entirely controllable. This approach must be supplemented with a recovery or repair strategy when plans fail due to a conflict between the assumptions and reality. This approach is very much the one adopted in FF-replan (Yoon, Fern, & Givan 2007) which performed surprisingly well in recent competitions for probabilistic planners. A more recent refinement of this approach (Yoon *et al.* 2008) shows that the idea can be extended to deal with more subtle probabilistic domains than those used in the original competitions.

Despite the uncertainty involved in the problem space, a significant part of the activity involved in building infrastructure and managing resources must be conducted without close interleaving with interactions with opponents or handling uncertain aspects of the environment. In fact, a good way to manage game play is to decompose the problems of handling enemies from those of managing one's own development, seeing the latter as a tool to construct resources for the second, but otherwise treating them independently. This makes it possible to plan the resource and infrastructure enterprises and to identify the flow of resources into the military campaign in order to plan the strategic and tactical elements of combat appropriately.

At a suitable level of abstraction, the actions in a plan will be relatively robust to the uncertainties of execution, with those uncertainties being handled by execution monitoring and reactive control at a lower level of detail. Finding this level of abstraction is a challenge, as we discuss below. Furthermore, the process of supervising execution and identifying plan failure is sensitive to this uncertainty and the choice of level of abstraction in planning. This is also discussed further below.

Other researchers have also observed the potential role of planning in guiding computer-controlled players. Kelly *et al.* (Kelly, Botea, & Koenig 2007) consider the role of a hierarchical task network planner, in order to reduce the search problem involved in plan construction, while Fayard (Fayard 2007) considers the use of simulated annealing local search for plan construction. In practice, the winners of previous competitions (for example (Hagelback & Johansson 2008)) have not used general planners, but have focussed on lower level, purpose-built, control architectures.

4 Harnessing Planning in ORTS

The ORTS system includes some infrastructure for creating computer players. This is organised as a hierarchy of commanders, with resources being passed down though the chain of command. The obvious way to harness this structure is to use lower level commanders to implement controllers for specific tasks, such as the collection of resources, construction of particular buildings, management of combat with a specific enemy force, scouting and so on. We explored the instantiation of the architecture shown in figure 3. This architecture is based on our earlier work on the MADbot motivated planning, execution and recovery architecture (Coddington *et al.* 2005), although we reimplemented the architecture within the ORTS framework. As can be seen, the planning element of the player is implemented by making external calls to a planning system (we used Metric-FF (Hoffmann 2003), although we also experimented with LPG (Gerevini & Serina 2002)). To interact with the planner, a component of our ORTS player generates PDDL problem files that can be coupled with a predefined domain file. The plan is captured and then dispatched by a high level commander, through interaction with lower level commanders. The whole process of execution is monitored by a supervisory process. We discuss recognition of plan failure and response to it below.

A key design challenge was to identify an appropriate level of abstraction at which to encapsulate the controllers. The reason this is difficult is that it seems natural to work hard at building strongly encapsulated commanders responsible for relatively high level tasks, such as using the worker force to gather as much resource as possible, but this makes it hard to see what role a planner can play in the exploitation of these commanders. If we have a commander for the military force, another for construction units and a third for resource gathering, then a planner is hardly necessary to see that the plan to win the game is "collect resources, build in-

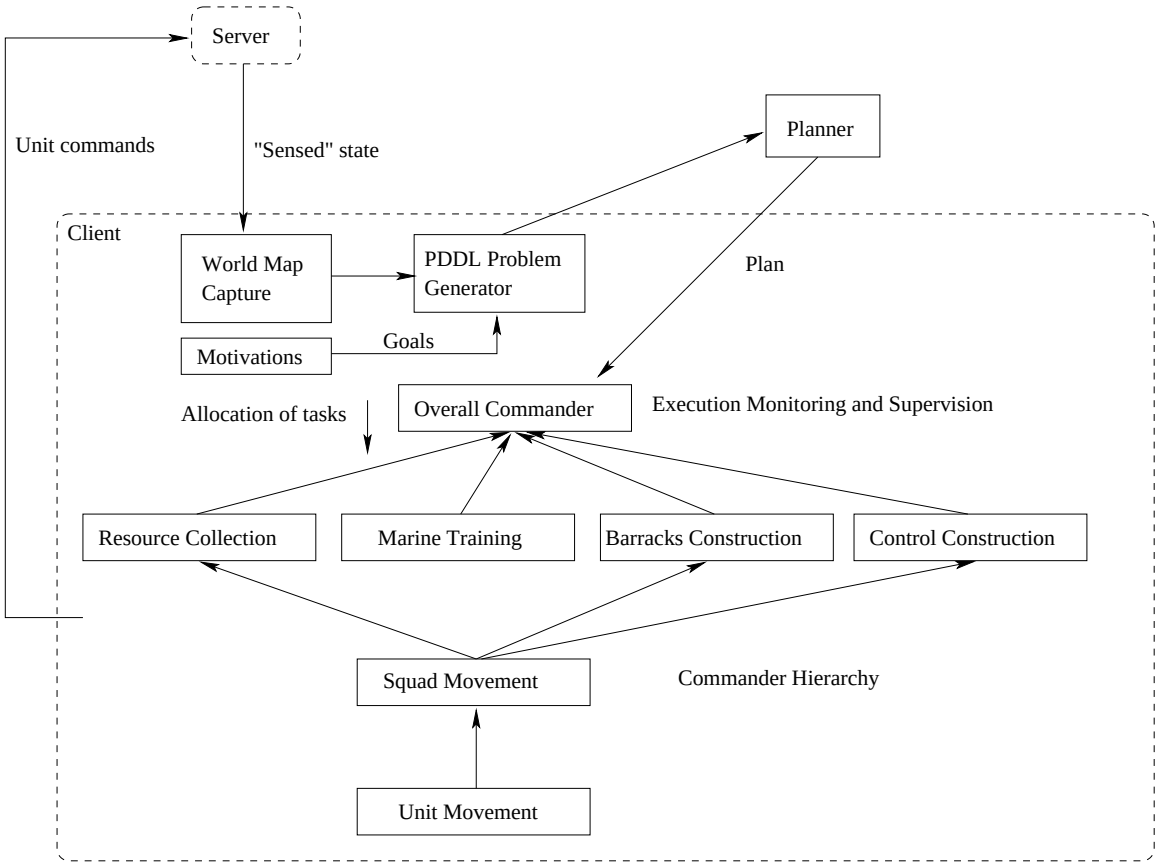

Figure 3: The Architecture of an ORTS Computer Player with Integrated Planning

frastructure and then defeat the enemy in a mighty military conquest". There is no real scope for subtlety in the planning in combining such coarse-grained high-level actions. In order to make a planner a useful tool it is apparent that it must have access to more primitive building blocks — actions that are more primitive than these. We found that as the level of actions is made more fine grained, the planner must confront more and more of the details of the precise spatial organisation of resources and the metric quantities of resources located at specific locations. This reasoning is both harder to model and harder for a planner to reason about well, but it is apparent that shielding a planner from this level of description also makes it impossible for the planner to produce plans that are sensitive to the efficient management of resources and their positioning.

We adopted a level of description in which individual workers and marines are identified, with their locations, allowing us to specify plans to construct, move and employ these units individually. Modelling at this level makes it hard to plan to manage forces of more than a score of units — the existence of large numbers of very similar units makes for a very large branching factor at most choice points and the search space becomes too large for domain-independent planners to explore. We also experimented with team structures, allowing individual units to be absorbed into teams to allow a more efficient reasoning. Actions exist to allow units to enter and exit teams, so that commanders can be used to move entire team structures from place to place or carry out larger tasks such as resource gathering.

4.1 Defining Game Actions

Within the framework of the ORTS system, we constructed a hierarchy of commanders to act as controllers of various game actions (see examples in figure 3). These commanders are responsible for managing limited collections of resources (such as a few workers) to achieve various local and constrained goals (such as the collection of a specific amount of building material). When executing the plan, a general (or commander in control) is responsible for organising the other commanders lower in the hierarchy to carry out their designated tasks. For example, one of the commanders lower in the hierarchy is responsible for constructing buildings. Commanders are responsible for activating the following tasks.

- A Commander is capable of creating a new squad.
- A worker is able to join and leave a squad.
- A commander is able to move a squad - this involves moving each member of the squad to the same destination.
- Agents are able to move from one location to another.
- It is possible to train workers, train marines and build tanks.
- There are a number of "building" actions — it is possible to build barracks, control centres, factories and mechanisation bays.

An agent is defined as being either a worker, a marine or a tank. Once an agent is a member of a squad, that agent can no longer be independently controlled unless it leaves the squad (when it is a member of a squad, the agent's location is determined by the location of the squad).

An important part of the function of commanders is to manage the lower level path planning for the agents for which they are responsible. Although the ORTS system contains some limited path planning machinery, it is not effective at managing path planning for complex obstacle-ridden paths across large parts of the terrain. Furthermore, as mentioned previously, the problem of managing agents in the planning process can only be usefully exposed to the planner if it is given spatial information. We solved both the problem of large-scale path planning and the problem of exposing spatial structure to the planner by the use of a graph-based abstraction of the physical terrain structure, giving waypoints and accessibility relations between them. The process by which the abstraction is constructed is discussed in more detail below.

4.2 Building a PDDL Problem Description and Plan Execution

Once the set of game actions had been developed, a PDDL2.1 domain description was developed containing a one-to-one mapping between each game action and each PDDL2.1 action description. Two PDDL2.1 domain descriptions were developed, a non-metric as well as a metric version. The metric version was more useful as it enabled specific quantities of resources to be modelled. The domain description contains the following action models.

Create squad creates a new squad at a pre-determined location.

Join squad allows an agent to join a squad at a pre-determined location – once the agent has joined the squad it has the same location as that of the squad.

Leave squad allows an agent to leave a squad at a pre-determined location – leaving a squad means the agent is now free.

Move squad and **Move agent** move a squad/agent from one location to another provided the two locations are connected (or joined).

Train marine and **Train worker** create a new marine/worker using a barracks/control centre at a pre-determined location and increases the total number of marines/workers by 1.

Train toaster creates a new toaster (which requires both a barracks and an academy) at a pre-determined location and increases the number of toasters by 1.

Build barracks and **Build control centre** use a worker at a pre-determined location to build barracks/control centre – this action increases the total number of barracks/control centres by 1.

Build mechanisation bay and **Build academy** use a worker to build a mechanisation bay/academy at a pre-determined location. These actions can only be executed if a barracks has already been built and increase the total number of mechanisation bays/academies by 1.

```
(:action TRAIN_TOASTER
 :parameters (?head - toaster
              ?barracksx - barracks
              ?academyx - academy
              ?locationx - location)
 :precondition (and (potentially_exists ?head)
                    (next_exists ?head ?newHead)
                    (currently_exists ?barracksx)
                    (currently_exists ?academyx)
                    (at ?barracksx ?locationx)
                )
 :effect (and (not (potentially_exists ?head))
              (currently_exists ?head)
              (potentially_exists ?newHead)
              (at ?head ?locationx)
              (free_agent ?head)
              (increase (number_of_toaster) 1)
  ))
```

Figure 4: The PDDL2.1 model of *Train_Toaster*.

One difficulty we encountered when modelling the ORTS game actions is that each game requires buildings (such as barracks, academies and control centres) or agents (such as workers or marines) to be built or generated dynamically. This means that it is impossible to know the total number of buildings or agents which are required prior to playing the game. PDDL2.1 domain and problem descriptions must include a definition of all of the objects that are required (it is not possible to dynamically create new objects) so the modelling approach we used was to pre-define lists containing objects of various types (such as barracks, academies, control centres, marines) and model them as "potentially existing" using the predicates *(potentially_exists ?head)*, *(next_exists ?head ?newHead)*, and *(currently_exists ?head)*. If a new object is required as part of the planning process, it no longer "potentially exists", but now "currently exists" and the next object in the list becomes the "new head" of that list. Actions which require new objects have the requirement that such objects "currently exist" as a precondition. Figure 4 shows the approach we used to model the creation of toasters.

Having developed a PDDL model of the actions capable of being executed within the ORTS game, one of the major focusses of the work was to generate a PDDL2.1 problem description to model the dynamically changing ORTS environment. The idea was to construct a simple, abstract representation of the 2D spatial properties of the game field to enable path planning to occur between various locations. Two algorithms were used: the Graham Scan algorithm (Graham 1972) and a customised version of the Voronoi Tessellation algorithm (Watson 1981). The Graham Scan algorithm was used to identify and map the obstacles which exist in the ORTS environment. Any immovable obstacle is a point with a size and shape – the Graham Scan algorithm is used to find the convex hull (the convex boundary of a set of points) of each obstacle – the convex hulls of each obstacle are grown until they collide with the convex hull of other obstacles. The border between two obstacles becomes an edge in the Voronoi tessellation. Parts of the hull which have not collided will continue to expand until all edges have been formed. Any point within one of the newly grown hulls is closer to its obstacle than any other, and, more importantly, the edges are the paths with "greatest clearance" from the obstacle. The Voronoi Tesselation algorithm (Watson 1981) was used to find paths around the obstacles and to define paths connecting various locations within the environment. A node is defined as any intersection of three or more voronoi edges and every location in the ORTS game is a node. Applying the Voronoi Tesselation algorithm initially resulted in a graph representation of a very large number of connected locations. A number of techniques were used to prune the number of nodes to more manageable levels, for example, nodes which were positioned very close to each other were grouped together as a single node, although this sometimes causes problems by over-simplifying the paths between closely-positioned obstacles.

Once the PDDL problem description was generated, the general is able to invoke the Planner (Metric-FF) in order to generate a plan. The resulting plan is then passed by the general to the relevant commanders lower which are then able to ensure the plans are executed. One issue that arose with Metric-FF was that when it was given a "metric goal" (such as a goal to create 9 marines which is specified as *(= (total_marines) 9)*) it sometimes produced poor quality plans with a large number of redundant actions. In order to create 9 marines it is first necessary to construct a single barracks in order to train each of the marines. Metric-FF produced a plan which involved a number of redundant actions — it moved a worker from its current location to some other location for no reason, it then constructed two barracks, trained some of the marines in each of those barracks, then constructed more barracks and trained the remaining marines. The reason for the poor quality plan is that the relaxed plan heuristic over-estimates the amount of work required to achieve the metric goal. The consequence is that Metric-FF can construct very poor quality plans as the overestimated distances give significant slack for redundant steps to be included in the plan. Further examination of alternative models is ongoing and relevant related work includes (Alczar, Borrajo, & Lpez 2008) where models of the ORTS domain in PDDL are also considered.

5 Supervision of Plan Execution and Dealing with Plan Failure

In our current implementation, the plan is constructed by Metric-FF. This means that it is not temporally structured. Therefore, the dispatch of the plan is faced with a further challenge, which is to decide how to interpret the instructions that the plan conveys. There are two extreme choices: one is to simply execute the actions in sequence, as they are recorded in the plan. The other is to use a scheduler to find an organisation of the plan that minimises its makespan. We use a solution that falls between these extremes: we execute steps as their preconditions become satisfied. This is easy to achieve, by initiating the actions in the plan and requiring each commander that is invoked by this process to wait until its preconditions are satisfied before it begins any actions. This approach assumes that causal dependencies are the only important reason for the timing of actions, while there might be other interactions that are present in the plan that are resolved by the plan step ordering and are not correctly han-

dled by simple causal dependency analysis. Nevertheless, this solution is a cheap and effective way to schedule the plan and fails very rarely.

Whenever a planner is used to direct execution in an uncertain environment, there is a risk that the plan will fail. This is particularly likely when the environment contains opponents that are actively attempting to cause plan failure. Although we chose to ignore the problem of opponents in the current implementation, we did consider the structure of the necessary architecture to manage the problems of plan failure. The problems are, firstly, to identify that a plan has failed and, secondly, to construct an appropriate response.

It might seem obvious that plan failure can be recognised when a commander (or whatever execution control component is responsible for executing an action) determines that it is in a state from which it cannot fulfil its responsibility. However, we do not want a commander to give up too quickly — the management of uncertainty in the domain requires that commanders are robust to some local challenges in discharging their responsibilities. This makes the problem of identifying plan failure more subtle: a plan fails when a commander cannot fulfil its local task within the constraints of the resources supplied to it, but a commander should not continue to reactively strive to achieve its goal when it is consuming resources needlessly.

Consider this example: a plan is constructed in which a worker team is required to visit a resource site on the other side of a hill from its current location. The planner arbitrarily chooses to route the team to the right of the hill. During execution, the team encounters an enemy force. The team commander could attempt to control its team to achieve the task of navigating the assigned path, perhaps by sending off decoy workers to distract the enemy, perhaps by rushing the enemy force or perhaps by retreating and waiting for the enemy to move on. However, all of these attempts fail to exploit the fact that the planner selected the route arbitrarily and would have been equally satisfied with the team being routed by the alternative path. This example illustrates that devolving responsibility for execution of an action can lead to poor behaviour because the local execution control does not consider the broader context (the planning dimension of the problem).

Our current implementation contains the infrastructure to support a supervisory level that monitors the execution of the plan above the individual commanders, in order to recognise failures that threaten the plan integrity. This machinery is incomplete and remains an important area of future work.

6 Motivation and the Creation of Goals

One of the limitations with AI planning algorithms is that they require the goals to be specified by some external agent. Such goals are then presented (along with a description of the initial state and set of domain actions) to the AI Planner which generates a plan to achieve those goals. We have already described how a PDDL2.1 domain was constructed which contains a one-to-one mapping of the PDDL2.1 actions to each of the ORTS game actions. Currently, our goals are hard-coded (e.g. a goal to build 9 marines) but the final aim of this work is to explore how we might be able to generate goals both in response to the dynamically changing game state, and to further the aims of the game player.

To do this we have been exploring how modelling the motivations or low-level drives of the agent might enable goals to be generated. Motivations have been defined by Kunda (Kunda 1990) as “any wish, desire, or preference that concerns the outcome of a given reasoning task”. The role of a motivational system is to direct an agent towards carrying out a particular activity. We have a partially implemented motivation system which contains a series of numeric values representing the strength of various “desires” (e.g. the desire to expand, the desire to defend, the desire to build, etc.) as well as the current circumstances. The idea is that these numeric values will change in response to changes that occur within the game environment, and that once these values exceed or fall below some threshold value, a goal will be triggered to enable the player to further their aims. For example, if the numeric value representing the desire to build exceeds some threshold, a goal to build a barracks (or control-centre, etc.) will be generated and passed (along with a PDDL2.1 representation of the current game state and domain actions) to the AI Planner. When fully implemented, the idea is that this system will generated high-level goals for the AI planner. The motivation system is also intended to deal with plan failure — for example, the motivation system could make a commander request extra resources (e.g. a military commander may require more units) to enable the plan to succeed. Desires are related to, but not the same as, rewards associated with goals in an oversubscription problem. The difference is that desires change over time, partly as a consequence of the passage of time itself and partly as a consequence of observed state changes. As a result, the goals themselves change over time and the planner will be invoked to solve quite different problems during the evolving game play.

7 Conclusions and Further Work

The work we have described represents the first stages in exploring the exploitation of planning in a real time strategy game. There are clearly many issues that remain unresolved, but the key findings so far lie in the identification of several parameters that govern the framework of the exploitation. The most important of these is the degree of abstraction in the planning model: what actions should be exposed to the planner and how much responsibility should be devolved to the elements of the executive responsible for realising the actions? A second question is the planning horizon: what goals are appropriate to pose for a planner? The answer to this is partly linked to the performance characteristics of the planner and the speed of events in the game. A third question concerns the recognition of plan failure: when has an action failed to complete and can that recognition be devolved to the executive elements or must it be managed centrally? Finally, where do the goals come from and how do they reflect the overall motivation of the player to win the game?

Our work shows that it is possible to link a simple planner to a RTS effectively, producing plans that can be use-

fully interpreted for execution. However, much more work is needed to find fuller and more focussed answers to the questions outlined above.

References

Alczar, V.; Borrajo, D.; and Lpez, C. L. 2008. Modelling a rts planning domain with cost conversion and rewards. In *Proceedings of Workshop on Artificial Intelligence in Games, at 18th European Conference on Artificial Intelligence*.

Buro, M. 2002. ORTS: A hack-free RTS game environment. In *Proceedings of the International Computer and Games Conference*.

Buro, M. 2003. Real-Time Strategy Games: A new AI Research Challenge. In *Proceedings of the International Joint Conference on AI*.

Coddington, A.; Fox, M.; Gough, J.; Long, D.; and Serina, I. 2005. MADbot: A Motivated And Goal Directed robot. In *Proceedings of AAAI-05: Twentieth National Conference on Artificial Intelligence*.

Fayard, T. 2007. The use of planner to balance real time strategy video game. In *Proceedings of ICAPS'07 Workshop on Planning in Games*.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs. In *Proc. of 6th Int. Conf. on AI Planning Systems (AIPS'02)*. AAAI Press.

Graham, R. L. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1* 132–133.

Hagelback, J., and Johansson, S. J. 2008. Using multi-agent potential fields in real-time strategy games. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numerical state variables. *Journal of Artificial Intelligence Research, Special Issue on the 3rd International Planning Competition* 20:291–341.

Kelly, J.-P.; Botea, A.; and Koenig, S. 2007. Planning with hierarchical task networks in video games. In *Proceedings of ICAPS'07 Workshop on Planning in Games*.

Kunda, Z. 1990. The case for motivated reasoning. *Psychological Bulletin* 108(3):480–498.

Watson, D. F. 1981. Computing the n-dimensional tessellation with application to voronoi polytopes. *The Computer Journal* 2(24):167–172.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 1010–1016.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In Boddy, M. S.; Fox, M.; and Thibaux, S., eds., *Proceeding of ICAPS*, 352–360. AAAI.