A RTS domain definition in PDDL for ORTS

Author: Vidal Alcázar Saiz
Tutors: Daniel Borrajo Millán, Carlos Linares López

Introduction

- Games have a very important role in AI research. In particular, RTS games offer great possibilities.
- ORTS is a programming environment that works with RTS games for studying real-time AI problems.
- Strategies in RTS games as mid-term plans:
  - Automated Planning

RTS Games

- Real Time Strategy games
- One of the most popular genres in the game industry
  - Millions of copies sold
  - International competitions with professional players are frequently held
- Fast-paced real time action
- Multiple features appealing to both players and researchers

Characteristics of RTS Games

- Distinctly not turn based.
- Resource gathering.
- Multiple units and buildings.
- Technology tree.
- Tactical combat.
- Incomplete information.

Example of RTS Game

- Dune II: Foundations of the genre
  - Harvesters gather a resource called spice
  - Mouse controlled cursor
  - Technology tree with prerequisites
  - Fog of war
  - Interface with multiple frames

Challenges of RTS Games

- Real time reasoning
- Complex worlds, generally grid-based
- Uncertainty and incomplete information
- Learning and oponent modelling
- Multi-agent environment
- Resource and technology management
- Collaboration at unit level and between players

ORTS

- Open Real Time Strategy
- Conceived in 2001 by Michael Buro and developed at the University of Alberta
- Released under the GPL
- Provides an environment for AI research in RTS games similar to commercial ones
- A competition with different domains is held every year to encourage its use

Features of ORTS

- Flexible game specification
  - ORTS is not a single RTS game, it is a RTS game engine
  - Games are defined using a scripted language
- Server-client architecture
- Client customization
  - Clients are limited only by the communication protocol
- Low level AI modules

AI in RTS games (1)

- Two contrary concepts: Macromanagement and Micromanagement
  - Macromanagement: High level decisions, long term impact, humans are good at it (taking decisions as attacking an enemy base, going up the technology tree,...)
  - Micromanagement: Individual actions, short term impact, computers are good at it (issuing commands like move and attack to control units, gathering resources,...)

AI in RTS games (2)

- Micromanagement is at unit level and relates to tactical concepts
  - Reactive agents, pathfinding
- Macromanagement is at a higher level and relates to strategic concepts
  - Planning, learning, opponent modelling

Automated Planning

- Decision making about the actions to be taken
- Sequences of actions (called plans) are followed to achieve a given goal
- Requires the definition of the domain and the problems
  - Domain: Types of objects, predicates and actions
  - Problem: Initial state and goal

PDDL 2.1

- Planning Domain Definition Language
- Standard of the International Planning Competition
- Based on STRIPS, Lisp-like syntax
- PDDL 2.1 was the version used in the third IPC held in 2002
  - Numeric values as attributes of objects
  - Time reasoning

Objectives

- Defining a domain so a planner can be used to play RTS games
  - The problem to solve is the third domain of the annual ORTS competition
  - The language to be used in the domain specification is PDDL 2.1

Initial State

- A control center, 6 workers and 600 starting minerals
- A nearby mineral patch
- Start locations not symmetric
- Randomized terrain

Operators of the Domain

- Operators correlate to complex actions in the game.
- Actions that may need an unknown amount of time use up the unit (so it can not be used later in the plan). These actions are: Attack, Defend, Harass, Scout and Mine.
- The other actions take a given amount of time known a priori. They are the actions that produce new units and buildings.

Goal Definition (1)

- There are several issues regarding this:
  - Making the goal match the winning condition is unfeasible.
  - Replanning will happen often to adapt to the dynamic environment.
- Therefore, the goal will be getting a good enough plan that contributes to gain an advantage.

Goal Definition (2)

- Actions that are useful grant rewards; that is, they are soft goals
- Units have a cost associated to them depending on their relative position and the tasks they have performed. This cost is subtracted to the rewards.
- Hence, the goal is maximizing rewards minus costs, or rather minimizing a final cost qith an initial value after *n* actions:

$$totalCost = initialCost - \sum_{i=1}^{n} (reward_i - cost_i)$$

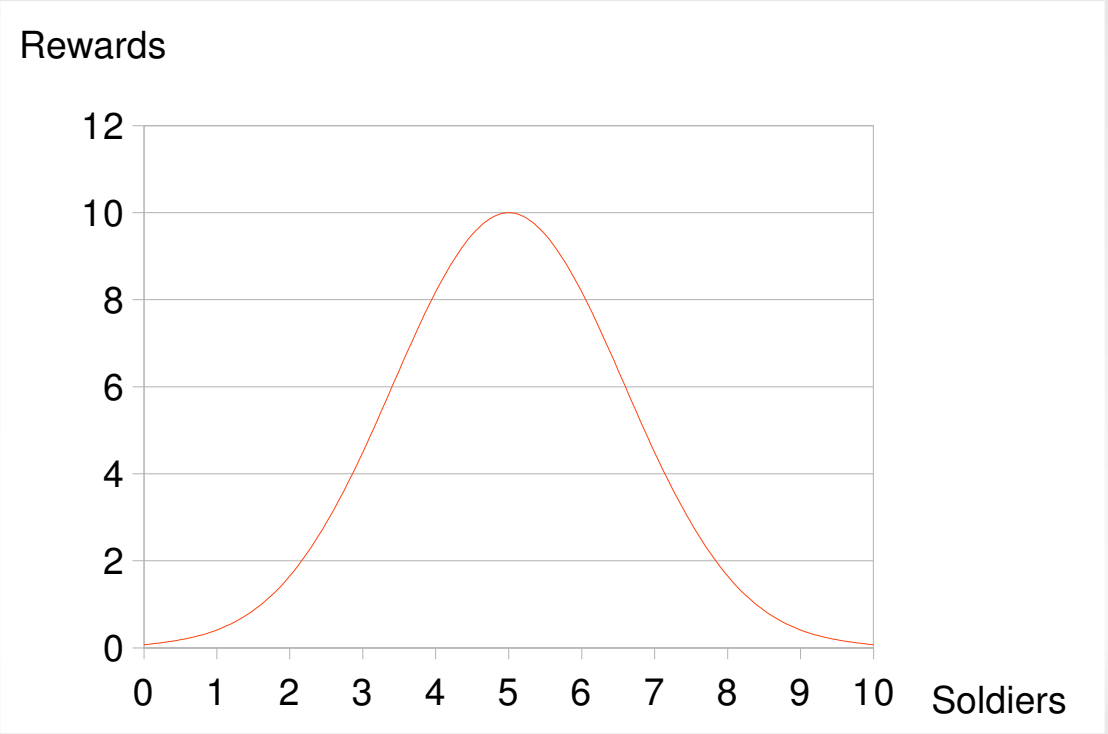
Granting Rewards (1)

- Complicated strategies require a given number of units
  - For example, when attacking an enemy base, the number of attacking units should be the number of defending units plus a little value. Using less units could result in a defeat and using more wastes units or resouces that could be used for other tasks.
- Rewards are granted every time the operator is used
- Rewards for each unit depend on how many units perform the same action
  - The planner does not know a priori how many units will be attacking!

Granting Rewards (2)

- Solution: Using a gaussian function in which a is the maximum reward a unit can get, b is the position of the maximum reward (which should correspond to the number of enemy soldiers in the opposing force plus a little value) and c the width of the Gaussian bell:

$$a\,\mathrm{e}^{-\frac{(x-b)^c}{2\mathrm{c}^2}}$$



Minimizing the Total Cost (1)

- Rewards and costs depend on the state throughout the plan
- State-of-the-art planners can not minimize a non-monotonic value
- Solution: Converting the cost to calls to an auxiliary operator after storing it in a temporal fluent called *pre-total-cost* and using plan length.

```
(:action TO-END
    :precondition  (and  (> (rewards) (rewards-threshold))
                         (> (pre-total-cost) 0) )
    :effect   (and (decrease (pre-total-cost) (cost-increment))
                   (increase (total-cost) (cost-increment)) ) )
```

Minimizing the Total Cost (2)

- Once *pre-total-cost* is lesser than zero the execution is finshed and the plan is an equivalent one with additional calls to the TO-END operator. An example of a plan would be the following one:

0: ATTACK SOLDIER2 ENEMYBASE0 POS3-3 POS1-1

1: ATTACK SOLDIER1 ENEMYBASE0 POS5-1 POS1-1

2: HARASS SOLDIER0 ENEMYBASE0 POS5-1 POS1-1

3: SCOUT WORKER0 POS5-1 POS6-2

4: TO-END

5: TO-END

6: END

Results

- The plans obtained are coherent sequences of actions that can be the base for the strategic aspect of a computer player.
- Plans are generated in a reasonable time, so they are adequate for a real time environment.
- Different planners can be used, and the values of the parameters can be modified, so further improvements may be possible.

Conclusions

- The domain definition has proved to be a good solution to a quite complex problem
- Analysis of the different challenges that the problem poses has lead to the implementation of independent techniques not related between them
- The proposed implementation only requires some PDDL 2.1 features to be supported, so it can be used by many planners and under different constraints

Future Work

- The techniques are implemented in the definition, so it could be interesting to integrate them in planners to see the results
- The problem has many points in common with many other complex problems, so this work can be easily extrapolated to other domains
- Further work with RTS games is encouraged, being it a good environment for AI research as shown by this work
- A client will be implemented for the 2008 ORTS competition based on these ideas