Single Agent Search Applied to a Simplified Realtime Strategy Game

Yavar Naddaf

Abstract

This paper introduces an AI agent that can play a simplified Real-Time Strategy game by using a single-agent search algorithm. In the simplified RTS game, the AI agent has to find an optimal plan that builds the required buildings and saves enough money to beat the opponent. We treat the game space as a search graph, and apply the Dijkstra algorithm to find the optimal path to a goal state. While our experiments show some promising results, there are some limitations and shortcomings in the current approach that need to be improved.

1 Introduction

Strategy games are a category of computer games in which the player has to manage resources, build infrastructure, create armies and deploy them to defeat the opponent. Currently, commercial strategy games use scripting to create AI opponents. Scripts, similar to screenplays of movies, are sets of instructions that determine how units should act in certain situations. There are two main problems with using scripting for AI opponents. First, the time and labor required to develop and fine-tune scripts grows exceedingly as the games become more complicated [2]. The second drawback is that scripts, by nature, are predictable and most players get bored after beating them a couple of times. An AI algorithm that can learn to play strategy games would allow both cheaper game development and more satisfying game play.

Creating AI agents that can learn to play strategy games is a very ambitious goal. Strategy games are often stochastic and the game states are not fully observable. Also, the player often controls tens of buildings and hundreds of troops, which results in a game tree with a very large branching factor [1]. This makes even solving the deterministic and fully observable version of the game infeasible.

In order to make the problem tractable, we focus on solving a simplified abstraction of strategy games. Our most important simplification is that we assume the individual units in the game are able to act reasonably intelligently. Therefore, instead of micromanaging the troops and trying to solve army deployment problems, the players will only concentrate on the long-term strategy of resource management and choosing what infrastructures to invest on[1]. We also assume that we are playing against an opponent that has already chosen its infrastructure and will not change it any further. This will reduce a two-player game to a single agent search problem, in which the search agent has to find a plan to create an infrastructure that will defeat the opponent's current infrastructure. A more detailed description of the game is included in the Appendix section.

By applying the Dijkstra algorithm on the search graph of the game, we are able to find a plan to goal state within a few seconds. However, our current approach has a number of limitations that are addressed in the Results section.

2 Background

Since solving the complete RTS games is a very hard problem, most research in this domain focus on solving sub-problems in the game. For instance, Kovarsky and Buro have developed several adversarial heuristic search algorithms for abstract combat games [3]. In abstract combat games, each player controls a number of units, and the goal is to destroy the opponent units. In other words, the large problem space of RTS games is reduced to the sub-problem of giving orders to a number of units to attack the opponent's units. Similarly, Sailer *et al.* focus on army deployment sub-problem of RTS games,

[1]This abstraction is inspired by a strategy game called "Sea of Fire" by Armor Games, in which players have no control over individual units and can only decide what infrastructure to build.

and apply strategy simulation in conjunction with Nash-equilibrium strategy approximation to determine how the armies that players build should act in the game. [2].

Similar to the approaches above, this paper also attempts to solve a sub-problem of RTS games. However, instead of focusing on unit micro-management and army deployment, the focus of this paper is on solving the more abstract problem of resource management and choosing what infrastructure to build.

3 Approach

In order to find a plan to create an infrastructure that defeats the opponent, we convert the game to a weighted search graph, where the states of the game are nodes and the actions of the AI agent are edges. The weight of the edge between two nodes n_1 and n_2 is the time it required to move from n_1 to n_2. The following sections define the search problem in more details.

3.1 State Space

Since we are assuming that the opponent will not change its buildings, a search state includes the AI player buildings, plus a discrete bucketing of both players' money. If we divide the amount of money of each player into 10 discrete buckets, with 5 lots and 4 possible buildings, the number of states is:

$$(\text{Number of Lots})^{(\text{Number of Buildings}+\text{Empty Lot})}.(\text{Player1 Money Buckets}).(\text{Player2 Money Buckets}) = 312500^2.$$

3.2 Start State

The start state can be any arbitrary combination of buildings and money for either player. Generally, we are interested in start states that have a possibility of winning.

3.3 Goal State

This search problem has no single explicit goal state. Any state in which the search agent will win the game without further changing its buildings is a goal state.

The naive approach to detect a goal state is to simulate the game for t_{sim} seconds, and see if the AI agent wins the game or not. The problem with this approach is that with a small t_{sim} we will miss some goal states, and with a large t_sim the search will take a long time.

We can improve our goal detection technique, by realizing that to win the game, the AI agent needs to have a higher attacking power than its opponent. We can calculate the attacking power of each player based on its number of Factories and Barracks, the time it takes for a Factory or a Barrack to build an attack unit, and the hit-points of each type of attack unit. Any state in which the AI player has less attacking power than its opponent is certainly not a goal state and can be ignored.

A better improvement can be achieved by simulating the game for a short t_{sim}, and then extrapolate the results for the long future and determine whether the AI player will win the game over the long time. Since players with enough money can instantly rebuild destroyed buildings, the player who first runs out of money will lose the game. To determine which player will run out of money first, we can divide each player's amount of money by the rate at which it is losing money. To do this, we count the number of Refineries each player has, and calculate the income of each player. Similarly, for each player, we can calculate the money it spends on attacking units based on its number of Barracks and Factories and their rate of production and the cost of each attacking unit. Unfortunately, because of the complicated dynamics of how units attack each other, and how Turrets defend the base, it is not easy to calculate how much hitting point each building will receive, and consequently, how much money each player will spend on re-buying buildings. To get the repair cost, we can simulate the game for t_{sim} seconds, look at how each building in the game is hit, and how much is the repair cost based on the price of the building. Once we have all the required data, we can calculate the rate at which each player is losing money, and given its current amount of money, how long it will take until it runs out of money. A state in which the opponent will run out of money sooner than the AI player is a goal state.

[2] In the actual implementation of the game, we made a further assumption that, except in the goal states, the opponent generally has "a lot" of money, and its money does not change during state transitions. This assumption reduces the number of search states to 31250

3.3.1 Actions and edge costs

In each state, the AI player can build any building type on any slot. These actions are only available, when the AI player has enough money and can afford each building. On the search graph, the edge cost of these build actions is the minimum time between AI player actions. In other words, if there is a minimum t_{act} seconds between the AI actions, then the edge cost of each build action will be t_{act}.

The other available action is Pass. This action basically means that the AI player do not want to build anything on this turn, and would like to "pass". It is useful when the AI player is waiting to save money. Therefore, this action is only available in states where the AI player is making positive money. To determine whether the AI player is making positive money in a state, we can run a short simulation and extrapolate the results, similar to the approach taken in section 3.3. The edge cost of a Pass action is the waiting time required for the AI player to move to the next discrete money bucket. This time can also be extrapolated from the results of the simulation.

3.3.2 Search algorithm and heuristics

To find an optimal plan to a goal state, a simple Dijkstra algorithm was implemented on the game search graph. For the current number of states in the game, Dijkstra can always find a plan within a couple of seconds. However, in order to be able to extend the game to a larger number of states, some type of heuristic search will be required. Unfortunately, we have not yet been unable to come up with an admissible heuristic definition for this game.

4 Results

In our experiments, Dijkstra is always able to find a plan to a goal state, or determine that no such plan exists, within a couple of seconds. However, after running a set of experiments, we discovered a number of limitations and shortcomings in the current approach. This section discusses these limitations, and suggests possible ways to improve them.

4.1 Goal states are not always detected

After running some experiments, we discovered that our current method to detect goal states, as discussed in section 3.3, can sometimes miss a goal state. In particular, there are states that our method labels as "not goal", but running a longer simulation reveals that the AI player eventually wins the game. One factor that contributes to this problem is that when both players have low levels of money, the game dynamics become very complicated. For instance, with some low amount of money, a player can rebuild some buildings, for instance a Refinery, but not others, *e.g.*, a Factory. This can change the results of the game, depending on what set of buildings each player has, and what buildings are currently under attack. To detect goal states more accurately, a more complicated model is required that can predict the order in which each player loses its buildings, and how this order determines the outcome of the game.

4.2 Goal states can take a long time to win the game

Another limitation of our current approach is that we treat any state that will eventually win the game as a goal state, and do not take into account the actual time required to win the game in such state. Therefore, the search algorithm finds plans that take a long time to win the game, but could be easily improved to win much faster.

Consider an example in which the opponent's buildings are [Barrack, Factory, Refinery, Factory, Refinery], and in the start state the AI player does not have any buildings. The following is the plan found by Dijkstra:

- Build Refinery in slot #3
- Build Turret in slot #4
- Build Turret in slot #0
- Build Turret in slot #1
- PASS
- Build Refinery in slot #2

- PASS
- PASS
- Build Factory in slot #3

Simulating the game and having the AI player perform these actions show us that, indeed, the AI player eventually wins the game [3]. Basically, the opponent out-spends itself and becomes unable to build enough units to defend its base. The AI player then slowly destroy the opponent's buildings. Note that after the Factory is built, the AI player has enough money to build a second Factory and drastically decrease the time required to win the game. However, our current implementation is unable to find such plan, because as soon as the state with one factory is expanded, it is detected as a goal state, and the search stops.

To solve this problem, we can change the definition of a goal node, from a state that will eventually win the game, to a new abstract node that represents a game already won. When we expand a node, we check if the game can be eventually won in this state. If the answer is yes, a goal node is added to the children of the state, and its edge cost is set to the expected time required for the state to win the game. This modification will allow us to find plans that will win the game in the shortest possible time.

References

[1] Michael Buro. Call for ai research in rts games. *AAAI Workshop on AI in Games*, 2004.

[2] Alexander Kovarsky and Michael Buro. A first look at build order optimization in real-time strategy games. *GameOn Conference*, 2004.

[3] Alexander Kovarsky and Michael Buro. *Advances in Artificial Intelligence*, pages 66–78. Springer-Verlag Berlin, 2005.

Appendix: Game Description

This section contains a short description of the simplified two-player strategy game:

- The game is deterministic and fully observable.
- Each player has 5 available lots in which buildings can be constructed.
- There are 4 buildings available:
  - Barrack: Periodically generates a foot soldier
  - Factory: Periodically generates a Tank
  - Refinery: Periodically generates money
  - Turret: Attacks any enemy unit within a range
- Buildings cost some amount of money to construct. If the lot where the building is created is not empty, there is an extra charge to destroy the previous building.
- The generation of units is automatic. For instance, once a Factory is constructed, it will automatically generate a Tank every certain period.
- It cost a certain amount of money to generate units. Barracks and factories will only generate units if the player can afford them.
- Once built, the units are autonomous: they will pursue and attack any enemy unit on the map. If there is no enemy unit left, they will attack the closest enemy building.
- Once a building is destroyed, if the player has enough money, it will be instantly rebuilt.

[3] A video capture of this simulation can be viewed here: http://www.youtube.com/watch?v=w55Jqa8lDkk

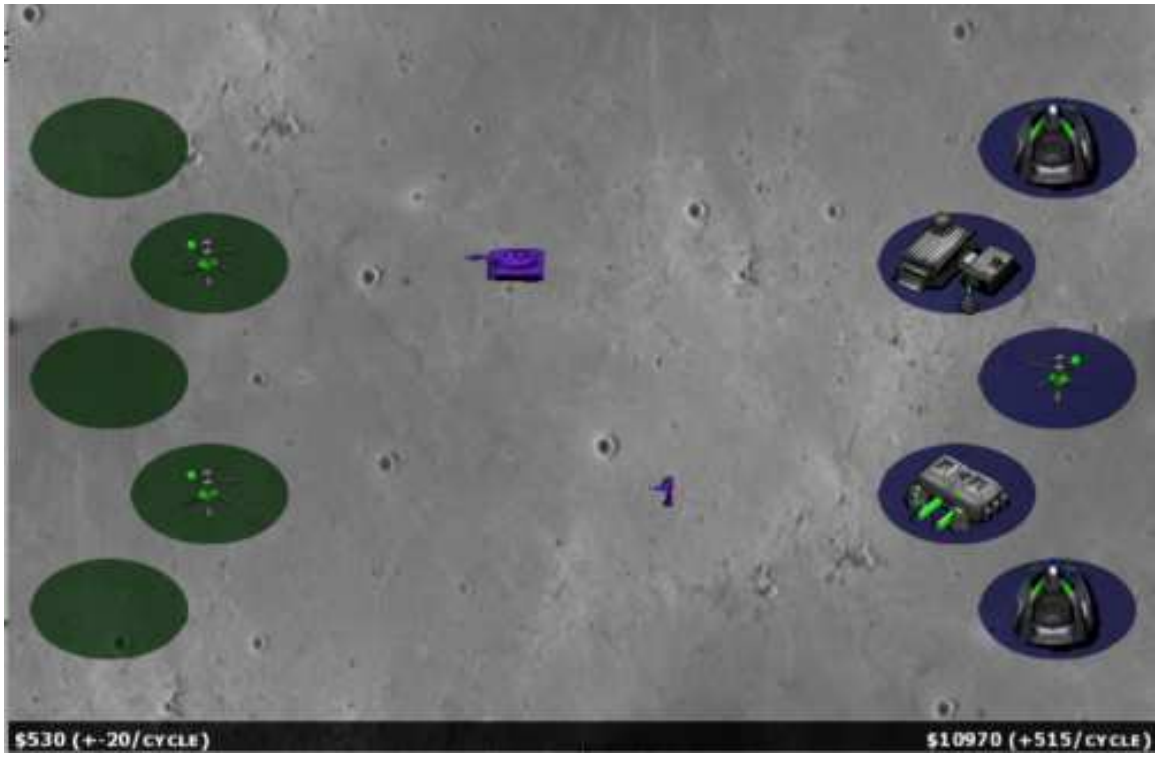

Figure 1. A sample state of the simplified RTS game

- Players can perform one action every t_{act} seconds. A player action is choosing to construct a building in a lot. A Players can also decide to "pass", *i.e.,* do nothing in a round.
- The game ends when one player destroys all buildings of the other player.

Figure 1 demonstrates a game state, where the right player has two Refineries on the corner lots, a Turret in the centre lot, and a Barrack and a Factory. The left player has two Turrets.