Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 54 (2015) 257 – 264

Procedia Computer Science

Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)

Real Time Strategy Games: A Reinforcement Learning Approach

Harshit Sethy*, Amit Patel and Vineet Padmanabhan

School of Computer and Information Sciences, University of Hyderabad, Hyderabad, India

Abstract

In this paper we proposed reinforcement learning algorithms with the generalized reward function. In our proposed method we use Q-learning[1] and SARSA[1] algorithms with generalised reward function to train the reinforcement learning agent. We evaluated the performance of our proposed algorithms on Real Time Strategy (RTS) game called BattleCity. There are two main advantages of having such an approach as compared to other works in RTS. (1) We can ignore the concept of a simulator which is often game specific and is usually hard coded in any type of RTS games (2) our system can learn from interaction with any opponents and quickly change the strategy according to the opponents and do not need any human traces as used in previous works.

Peer-review under responsibility of organizing committee of the Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)

Keywords: Machine learning; Q learning; Reinforcement learning; Real time strategy; Reward functions.

1. Introduction

Existence of a good Artificial Intelligence (AI) technique in the background of a game is one of the major factor for the fun and re-playability in commercial computer games. Although AI has been applied successfully in several games such as chess, backgammon or checkers when it comes to real-time games the pre-defined scripts which is usually used to simulate the artificial intelligence in chess, backgammon etc. does not seem to work. This is because in real-time games decisions has to be made in real-time as well as the search space is huge and as such they do not contain any true AI for learning. Traditional planning approaches are difficult in case of RTS games because they have various factors like huge decision spaces, adversarial domains, partially-observable, non-deterministic and real-time, (real time means while deciding the best actions, the game continues running and states change simultaneously).

1.1 Real time strategy games

Unlike turn based strategy games, where one has the ability to take ones own time, in RTS games, all movement, construction, combat etc., are all occurring in real time. In a typical RTS game, the screen contains a `map` area which consists of the game world with buildings, units and terrain. There are usually several players in an RTS game. Other than the players there are various game entities called *participants, units* and *structures*. These are under the control of the players and the players need to save their assets and/or destroy assets of the opponent players by making use of their control over the entities. We are using RTS games BattleCity for our evaluation. A snapshot of `BattleCity` is given in Fig. 1.

*Corresponding author.

E-mail address: hsethy1@gmail.com

Peer-review under responsibility of organizing committee of the Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)
doi:10.1016/j.procs.2015.06.030

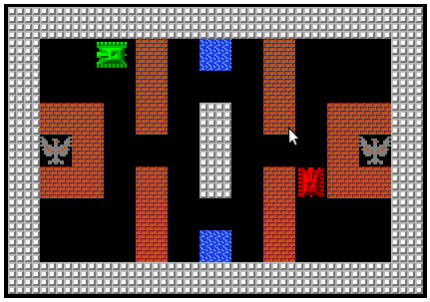

Fig. 1. Map:Bridge-26 × 18.

1.2 BattleCity game

`BattleCity` is a multidirectional shooter video game, which can be played using two basic actions Move and Fire. The player, controlling a tank, must destroy enemy tanks or enemy base and also protect its own base. Player can move tank in four directions (left, right, up and down) and fire bullets in whichever direction the tank last moved, while bases are static. There are three types of obstacle. (1) *Brick wall* tank can destroy it by firing this type wall. (2) *Marble wall* tank can destroy it by firing. (3) *Water bodies* tank can fire through it. Tank can pass through any of above obstacle. Only brick wall can be destroyed by tank so after destroying tank can pass through it.

This paper is structured as follows. Apart from introduction, there are five more sections. In section 2 highlights the review of related works. In section 3 we discuss about reinforcement learning techniques RTS game and outline the various learning algorithms used in reinforcement learning. In section 4 we outline implementation details related to the proposed reinforcement learning algorithms with the generalized reward function for RTS game BattleCity. Section 5 discusses about the experimental result related to our proposed work for BattleCity. We conclude with section 6.

2. Related Work

One of the major works using *Online case-based planning*[5] techniques for Real Time Strategy Games was published in[8]. On-line case-based planning revises `case based planning` for strategic real-time domains involving on-line planning.

In[7] a case-based planning system called Darmok2 is introduced that can play RTS games. They introduced a set of algorithms that can be used to learn plans, represented as `petri-nets`, from one or more human demonstrations. Another work by the same authors which uses Darmok2[9] but addresses the issues of plan acquisition, on-line plan execution, interleaved planning and execution and on-line plan adaptation is[6].

In[2] the authors summarize their work in exploring the use of the first order inductive learning (FOIL) algorithm for learning rules which can be used to represent opponent strategies.

In[10] the authors improve Darmok2 using information related to sensors of the game. The given model is capable of learning how to play RTS games by observing human demonstrations. Using human traces PR-model makes plans to play games. Prioritize the plan according to the feedback of the game and feedback are decided using some rule which depends on the sensors of the game.

Drawbacks of all case based learning[3] approaches as mentioned above are (1) It requires expert demonstrations for making plans (2) after training is done, no further learning takes place (3) to cover large state spaces it would require large number of rules in the plan base (4) no exploration for optimal solution. Only follows human traces. Stefan Wender[4] uses Reinforcement Learning for City Site Selection in the Turn-Based Strategy Game Civilization IV. Civilization IV is the strategy game it is a turn-based game while Battle City is Real time game.

In this paper we aim to do away with the hard coded simulator and propose a learning approach based on Reinforcement Learning[1] (RL) wherein sensor information from the current game-state is used to select the best

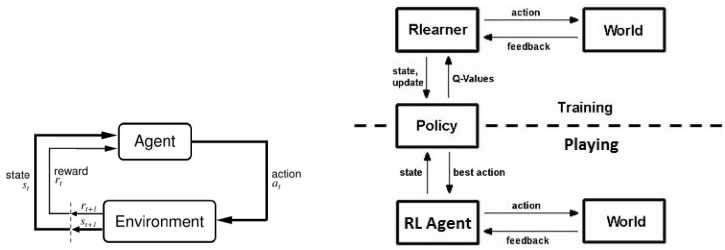

Fig. 2. (a) Reinforcement learning; (b) Architecture for the reinforcement learning.

action. Reinforcement learning is used because of its advantages over previous strategies. Specifically (1) RL cuts out the need to manually specify rules. RL agents learn simply by playing the game against other human players or even other RL agents (2) for large state spaces, RL can be combined with a function approximator such as a neural network, to approximate the evaluation function (3) RL agent always explores for optimal solution to reach the goal (4) RL has been applied widely to many other fields, such as robotics, board games ,turn based games and single agent games with great results, but hardly ever on RTS multi-agent games.

3. Reinforcement Learning

Reinforcement Learning[1] is the field of *Machine Learning* which deals with what to do, how to map situations to actions so as to maximize a numerical reward signal. The learner does not know which actions to take, as in most forms of machine learning, but instead must discover which actions gives the most reward by applying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards.

With comparing reinforcement learning to RTS game environment an AI player learns by interacting with the environment and observing the feed-backs of these interactions. This is same as the fundamental way in which humans (and animals) learn. As a human, we can perform actions and observe the results of these actions on the environment. The same way RL-agent interacts with the environment and observes the result and assign the reward or penalty to state or state-action pair according to the desirability of the resultant state.

3.1 Reinforcement learning architecture

RL Architecture has two main characteristics; one is learning and the other is playing with the learnt experiences. Initially RLearner has no Knowledge about the game. So it does random actions and observe the resultant state using some sensor information of the game and give feedback (in the form of reward which is further used to calculate the Q-Values for the state-action pairs or Q-Table) of that action to the previous state according to the desirability of the current state. After every action policy updates Q-Values for the state action pairs (Q-Table) this policy is used to predict the best action while playing the game. RL agent learns while playing so it again gives feedback and the whole process it going on till the end of the game.

3.2 Basic components of RL

Reinforcement learning contains five basic components which are as listed below.

1. A set of environment states S
2. A set of actions A
3. Rules of transitioning between states
4. Rules that determine the scalar immediate reward of a transition (Reward Functions)
5. Rules that describe what the agent observes (Value Functions)

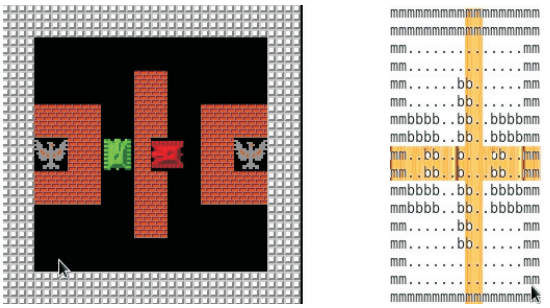

Fig. 3. Snapshot of BattleCity Game and its current 2D maps.

3.2.1 Reward function

The scalar value which represents the degree to which a state or action is desirable is known as *reward*. We are using 2 types of Reward function (1) *Conditional Reward function* (2) *Generalised Reward function*.

3.2.2 Value function

Value Functions are used for mapping from states or state-action pairs to real numbers, where the value of a state represents the long-term reward achieved starting from that state (or state-action), and executing a particular policy. It estimates how good a particular action will be in a given state, or what the return for that action is expected to be. There are two type of value functions.

1. $V^{\pi}(s)$ is the value of a state 's' under policy π. The expected return when starting in s and following π thereafter.
2. $Q^{\pi}(s, a)$ is the value of taking action 'a' in state 's' under a policy π. The expected return when starting from s taking the action a and thereafter following policy π.

There are two methods to define these value functions:

1. *Monte Carlo*[1] *Method*: In this method the agent would need to wait until the final reward was received before any state-action pair values can be updated. Once the final reward is received, the path taken to reach the final state would need to be traced back and each value updated.

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{1}$$

   where s_t is the state visited at time t, R_t is the reward after time t and α is a constant parameter.
2. *Temporal Difference (TD)*[1] *Method*: It is used to estimate the value functions after each step. An estimate of the final reward is calculated at each state and the state-action value updated for every step of the way. This reflects a more realistic assignment of rewards to actions compared to Monte Carlo (MC)[1], which updates all actions at the end directly. TD Learning is nothing but the combination of dynamic programming with the MC method. The formula related to TD learning is given as

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)] \tag{2}$$

   where r_{t+1} is the observed reward at time $t + 1$.

3.3 Sensor representation for BattleCity game

We are using two types of sensor information for assigning reward in battle city game which are explained as follows;

1. *EnemyInline*: If *enemy position* is directly in line with player without any block or wall then sensor is represented by number 2. If there is a wall or block then sensor is represented by number 1. If enemy position is not in line then sensor is 0.

2. *EnemyBaseInline*: If *enemy-base position* is directly in line with player without any block or wall then sensor is represented by number 2. If there is a wall or block then sensor is represented by number 1. If enemy-base position is not in line then sensor is 0.

So far we have outlined our method of obtaining sensor information related to RTS games, BattleCity.

3.4 Action selection policies

We have the following action selections policies which can be used to select desired action according to the behavior of that particular policy

1. $\epsilon-$**greedy**: Most of the time the action with the highest estimated reward is chosen, called the greediest action. But, with a small probability ϵ, an action is selected at random to ensure optimal actions are discovered.
2. $\epsilon-$**soft**: Very similar to $\epsilon-$greedy. The best action is selected with probability $1-\epsilon$.
3. **softmax**: Softmax assigns a rank or weight to each of the actions, according to their action-value estimate. So the worst actions are unlikely to be chosen.

One epoch contains following steps.

1. An action is determined by a decision making function (e.g. ϵ-greedy).
2. The action is performed.
3. The Rlearner receives a scalar reward or reinforcement from the environment according to reward function.
4. Information about the reward given for that state/action pair is recorded.
5. Update the Q-values in Q-table According to Learning Algorithm (e.g. Q-learning or SARSA).

4. Proposed Learning Algorithm

In this section we outline our proposed learning algorithms which we integrated into the RTS games Battlecity. We also provide the implementation details related to selection of parameters and reward functions.

4.1 Parameters

- **Learning Rate α:** The learning rate $0 < \alpha < 1$ determines what fraction of the old estimate will be updated with the new estimate. $\alpha = 0$ will stop the RL-agent from learning anything while $\alpha = 1$ will completely change the previous values with the new one.
- **Discount Factor γ :** The discount factor $0 < \gamma < 1$ determines what fraction of the upcoming reward values will be considered for evaluation. For $\gamma = 0$ all the upcoming rewards are ignored. For $\gamma = 1$ means the RL-Agent will consider the current and upcoming rewards as equal weightage.
- **Exploration Rate ϵ :** In action selection policies there is one policy called as ϵ greedy method which uses the exploration rate $0 < \epsilon < 1$ for determining the ratio between the exploration and exploitation.

4.2 Reward function for BattleCity

Algorithm 1: Reward function is for calculating reward after performing action on current state. According to the result of the action reward or penalty are assigned. In steps 1 to 9 get the positions (x-y co-ordinates) of the player, enemy and enemy base on the map. In steps 10 to 16 if game is over and winner is the RL-Agent (player) then add the reward to the total reward (newReward) else deduct penalty from the total reward. In steps 17 to 18 if enemy is in line with the RL-Agent deduct penalty from total reward so it always tries not to be in line with enemy. In steps 19 to 21 if enemy base is in line with the RL-Agent then calculate the distance between the enemy base and RL-Agent and deduct from 2 times of reward and add to total reward. So it pushes the RL-Agent to come closer to the enemy base. Steps 22 to 24 gives the generalized reward function which makes the RL-Agent quickly attack the enemy base and prevent attack by the enemy.

Input: *state* :- contains positions of entities, reward, penalty
sensorsList :- contains sensors of game domain.
gameState :- contains state of game is running or not
Output: Reward

```
1  Player_x = null, Player_y = null, Enemy_x = null, Enemy_y = null ;
2  EnemyBase_x = null, EnemyBase_y = null, winner = null ;
3  newReward = 0, distance = 0 ;
4  Player_x = getPositionx(state,player) ;
5  Player_y = getPositiony(state,player) ;
6  Enemy_x = getPositionx(state,enemy) ;
7  Enemy_y = getPositiony(state,enemy) ;
8  EnemyBase_x = getPositionx(state,enemybase) ;
9  EnemyBase_y = getPositiony(state,enemybase) ;
10 if gameState == "end" then
11     winner = getWinner() ;
12     if winner == "player" then
13         newReward = newReward + reward ;
14     else
15         newReward = newReward - penalty ;
16 else
17     if sensorList[EnemyInline]==2 then
18         newReward = newReward - penalty ;
19     if sensorList[EnemyBaseInline]==2 then
20         distance = sqrt[2]((EnemyBase_x − Player_x)^2 + (EnemyBase_y − Player_y)^2) ;
21         newReward = newReward + 2 × reward - distance ;
22     newReward = newReward - 4 × distance ;
23     distance = sqrt[2]((Enemy_x − Player_x)^2 + (Enemy_y − Player_y)^2) ;
24     newReward = newReward + 4 × distance ;
25 return newReward ;
```

Algorithm 1. calcReward for BattleCity.

5. Experimental Results

In the previous section we have discussed how we successfully applied reinforcement learning in real-time strategy game called BattleCity. In this section we outline the experimental results related to reinforcement learning in BattleCity.

5.1 BattleCity

We evaluated the performance of RL-Agent with the help of various maps (e.g. *Bridge*-26 × 18, *Bridges*-34 × 26) as well as with two types of opponents called *AI-Random* and *AI-Follower* in each map. We observed that the RL Agent won more than 90% games when played against both opponents (AI-Random and AI-Follower) in simple maps, about 80% to 90% when played against AI-Random in complex maps and 60% to 80% when played against AI-Follower in complex maps. Statistics about the performance of the SARSA, Q-Learning and Darmok2 in the various maps are represented below in the form of graphs. We observed that performance of RL-Agent under SARSA Learning algorithm is better than other techniques and also RL-agent trained by SARSA algorithm takes less time to win the game.

5.1.1 Map: Bridge-26 × 18

This map size is 26 × 18 (refer Fig. 1) so total state space for this map is total combination of the x-y co-ordinates of the player and enemy which is $26^2 \times 18^2$.

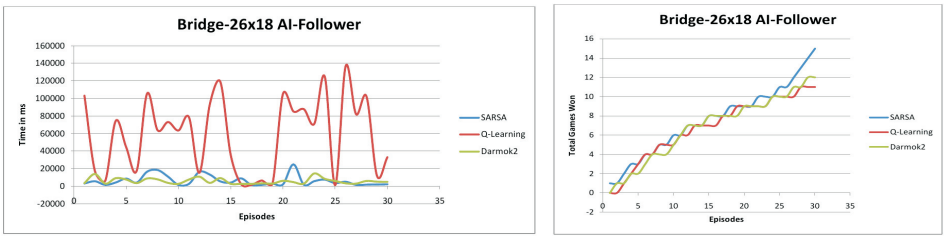

Fig. 4. Map: Bridge-26 × 18 Against AI-Follower.

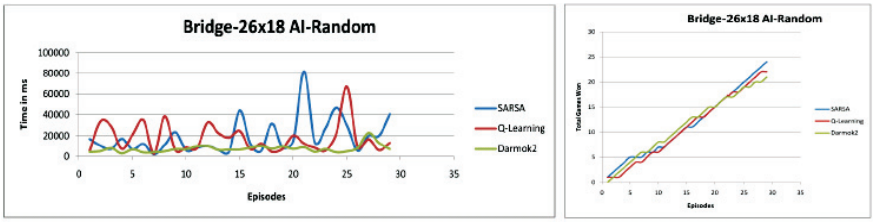

Fig. 5. Map: Bridge-26 × 18 Against AI-Random.

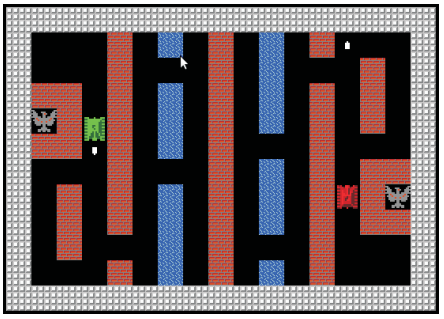

Fig. 6. Map:Bridge-Metal-34 × 24.

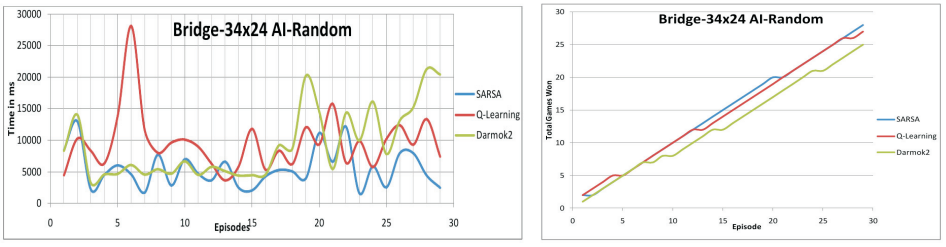

Fig. 7. Map: Bridges-34 × 24 Against AI-Random.

5.1.2 *Map: Bridges*-34 × 24

This is the most complex map (refer Fig. 6) among all on which we have performed our evaluation because of its size and the structure. It is a 34 × 24 map and it has $34^2 \times 24^2$ search spaces.

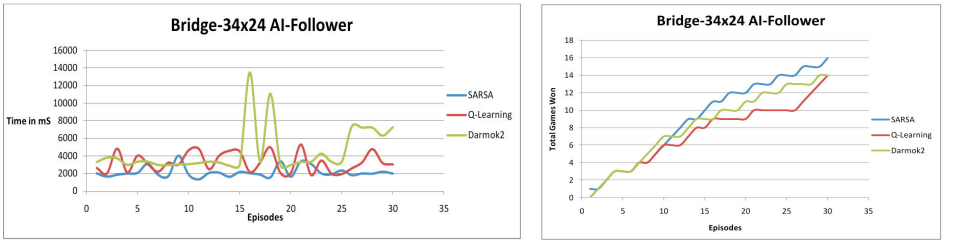

Fig. 8. Map: Bridges-34 × 24 Against AI-Follower.

6. Conclusions

In this paper we proposed a reinforcement learning model for RTS games. The idea is to get the best action using one of the RL algorithms so as to not make use of the *traces* generated by the players. In previous works on RST games using "on line case based learning" human traces form an important component in the learning process. In the proposed method we are not making use of any previous knowledge like traces and therefore we follow an unsupervised approach. Another major contribution of our work is the reward function. Rewards are calculated by two types of reward functions called conditional and generalized reward function. The sensor information related to game is used for calculating the rewards. The reward values are further used by the two RL algorithms

SARSA and Q-Learning. These algorithms make policies according to the reward for the state-action pair. RL agent choose the action using these policies. We evaluated our approach successfully in game BattleCity) and observed that reinforcement learning performs better than previous approaches in terms of learning time and winning ratio. In particular SARSA algorithm takes lesser time to learn and start winning very quickly than Q-Learning.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, A Book Publisher MIT Press; (1998).
[2] Katie Long Genter, Santiago Ontañón and Ashwin Ram, Learning Opponent Strategies Through First Order Induction, In *FLAIRS Conference*, pp. 1–2, (2011).
[3] P. P. Gomez-Martin, D. Llanso, M. A. Gomez-Martin, Santiago Ontañón and Ashwin Ram, Mmpm: A Generic Platform for Case-Based Planning Research, In *ICCBR 2010 Workshop on Case-Based Reasoning for Computer Games*, pp. 45–54, (July 2010).
[4] Stefan Wender and Ian Watson, Using Reinforcement Learning for City Site Selection in the Turn-Based Strategy Game Civilization IV, In *Computational Intelligence and Games (CIG-2008)*, pp. 372–377, (2008).
[5] Janet L. Kolodner, An Introduction to Case-Based Reasoning, In *Artificial Intelligence Review*, pp. 3–34, (1992).
[6] Santi Ontañón, Kinshuk Mishra, Neha Sugandh and Ashwin Ram, On-line Case-Based Planning, In *Computational Intelligence*, pp. 84–119, (2010).
[7] Santiago Ontañón, K. Bonnette, P. Mahindrakar, M. A. Gomez-Martin, Katie Long Genter, J. Radhakrishnan, R. Shah and Ashwin Ram, Learning from Human Demonstrations for Real-Time Case-Based Planning, In *STRUCK-09 Workshop, colocated with IJCAI*, pp. 2–3, (2011).
[8] Neha Sugandh, Santiago Ontañón and Ashwin Ram, On-line Case-Based Plan Adaptation for Real-Time Strategy Games, In *Association for the Advancement of Artificial Intelligence (AAAI-2008)*, pp. 1–2, AAAI Press, (2008).
[9] Santiago Ontañón Villar, D2 Documentation, pp. 1–6, (May 2010), http://heanet.dl.sourceforge.net/project/darmok2/D2-Documentation.pdf.
[10] M. Pranay, Game AI: Simulator Vs Learner in Darmok2, In *University of Hyderabad as M.Tech. Thesis*, (2013).