Playing Card-Based RTS Games with Deep Reinforcement Learning

Tianyu Liu[1] , **Zijie Zheng**[1] , **Hongchang Li**[2] , **Kaigui Bian**[1] and **Lingyang Song**[1]
[1]School of EECS, Peking University
[2]Babeltime Technology Co.
{liuty, zijie.zheng, bkg, lingyang.song}@pku.edu.cn lihongchang@babeltime.com

Abstract

Game AI is of great importance as games are simulations of reality. Recent research on game AI has shown much progress in various kinds of games, such as console games, board games and MOBA games. However, the exploration in RTS games remains a challenge for their huge state space, imperfect information, sparse rewards and various strategies. Besides, the typical card-based RTS games have complex card features and are still lacking solutions. We present a deep model SEAT (selection-attention) to play card-based RTS games. The SEAT model includes two parts, a selection part for card choice and an attention part for card usage, and it learns from scratch via deep reinforcement learning. Comprehensive experiments are performed on Clash Royale[1], a popular mobile card-based RTS game. Empirical results show that the SEAT model agent makes it to reach a high winning rate against rule-based agents and decision-tree-based agent.

1 Introduction

Real-time strategy (RTS) games have the characteristics of huge state space, imperfect information, sparse rewards and various strategies, which make it difficult to design intelligent agents for playing this kind of games [Buro and Churchill, 2012]. A number of works have tried to design intelligent agents for general RTS games using heuristic methods [Churchill *et al.*, 2012; Barriga *et al.*, 2018], hierarchical methods [Ontanón and Buro, 2015] and genetic search methods [Clark and Fleshner, 2017]. Some researchers have focused on the StarCraft, a well known PC RTS game, and tried to solve micromanagement tasks or make macro decisions. Methods for the StarCraft include terrain analysis [Uriarte and Ontanón, 2016], bayesian model [Synnaeve and Bessiere, 2011], reinforcement learning [Usunier *et al.*, 2016] and deep learning [Synnaeve *et al.*, 2018].

Card-based RTS game is a special kind of RTS game, where two players fight against each other by real-time decisions using hand cards. Besides characteristics of general RTS games, card-based RTS games are also related to complex card features, so the decisions of card choice and card usage are hard to make. Cards can be used at different positions on the battle field which produces huge state space. A player cannot see the hand cards of the other player, which gives imperfect information. The rewards during playing are hard to define because the game results can only be known after battles end. Strategies like offense, defense, rush and flexibility make the games with great uncertainty. Under these conditions, the heuristic methods, reinforcement learning methods and deep learning methods cannot work well alone in designing intelligent agents for card-based RTS games.

Deep reinforcement learning (DRL) attracted many researchers after its successful attempt in Atari games [Mnih *et al.*, 2013; Mnih *et al.*, 2015]. Since that, DRL has proved to be effective in solving various kinds of games, e.g. board games (Go) [Silver *et al.*, 2016; Silver *et al.*, 2017], FPS games (Doom) [Lample and Chaplot, 2017] and MOBA games (King of Glory) [Jiang *et al.*, 2018]. DRL also helps to promote the development of autonomous actions of robots [Lillicrap *et al.*, 2016; Levine *et al.*, 2016; Gu *et al.*, 2017], question answering tasks [Wang *et al.*, 2017], video captioning tasks [Pasunuru and Bansal, 2017] and physical learning tasks [Watters *et al.*, 2017]. With DRL methods, we can reduce the order of game states (including battle field states and card features), make full use of known information, accumulate rewards and adapt to various strategies flexibly. Thus, the DRL methods are promising to design intelligent agents for card-based RTS games.

In this paper, we focus on a popular card-based RTS game, Clash Royale [2] (CR). Specifically, we design a DRL model to play CR, named as SEAT (selection-attention model), which includes two parts, a selection part for card choice and an attention part for card usage. The selection part transforms battle field states into state images, and makes decisions on choosing which hand card to use. The attention part processes enemy states by trained convolutional layers, and makes decisions on where to use the selected card.

Our main contributions are summarized as following. To the best of our knowledge, this is the first work to study game AI in card-based RTS games. We design and realize the DRL model SEAT, and train intelligent agents for playing CR. Em-

[1]Clash Royale is a trademark of Supercell in Finland and/or other countries. Nothing in this paper should be construed as approval, endorsement, or sponsorship by Supercell.

[2]https://clashroyale.com/

pirical results show that our SEAT model agent achieves a high winning rate against rule-based agents and decision-tree-based agent.

2 Card-based RTS Games

Clash Royale is a popular mobile card-based RTS game among gamers all over the world. A typical battle includes two players. The objective for each player is to destroy the most amount of enemy towers using the hand cards, with the destruction of the enemy major tower being an instantaneous win. A simulation environment of CR on Windows platform is produced for experiments of our model.

2.1 Game Scene

We first briefly introduce the basic information of CR. The game scene of CR is shown in Figure 1, with four main parts (battle field, hand cards, game time and total elixir) and three sub parts (major tower, minor tower, battle troops) in the battle field.

The battle field, where battles between players happen, contains towers, troops and buildings. Each player has one major tower and two minor towers. Only after a minor tower is destroyed can the major tower be attacked. Below the battle field are the hand cards and the total elixir. A hand card is composed of the character image and elixir cost to deploy the card. Hand cards have three categories, troop, spell and building. A troop, member of battle troops, is a warrior that can move and fight on the battle field. A spell is an area effect (usually used to kill a group of troops). A building is fixed somewhere once used, to attack troops it can reach. On righttop of the scene is game time indicating the remaining time of the current time stage. Total elixir affords for using cards and will recover by time.

Players should decide to choose and use cards to fight against each other according to the game states within the time range of a battle, in order to destroy more towers of the enemy and finally win the battle. In the following subsections, we will provide the mathematical model of the game states (information that can be perceived from the game scene and card features) and game process (the operating mechanism of CR) respectively, in details.

2.2 Game States

In this subsection, we provide the introduction to the game states. The game states include the states of the main parts of game scene (the battle field, hand cards, game time and total elixir) as we have mentioned in last subsection.

For the battle field, the states of its units (towers, troops, buildings) are composed of four attributes, *position*, *health*, *class* and *faction*. Similarly, for the hand cards, the states of its units (cards) are composed of three attributes, *health*, *class* and *faction*.

- **Position** The position of unit i, denoted by P_i, is the 2-dimensional coordinate of the unit in the $32,000 \times 18,000$-pixel battle field. The ranges of the two dimensions of P_i (x_i and y_i), are $1 \leq x_i \leq 18,000$ and $1 \leq y_i \leq 32,000$ respectively.

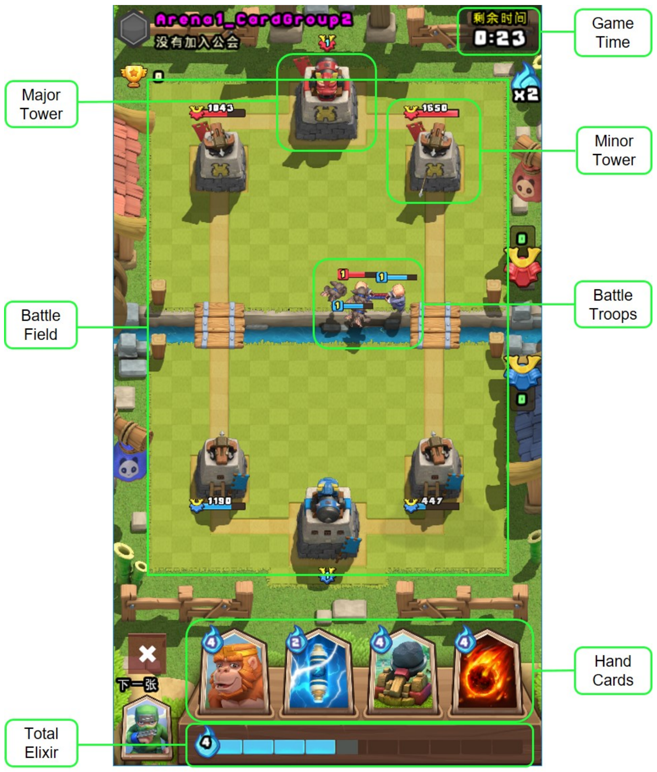

Figure 1: Clash Royale Game Scene.

- **Health** The health of unit i, denoted by H_i, is the amount of attack the unit can bear totally. Different units have different discrete health values. H_i is normalized between 0 and 10, e.g., the full health of a unit is 10.
- **Class** The Class of unit i, denoted by C_i, is the type of the unit, characterized by unit category and unit features (high health, high damage, area effect, tower targeted, etc.). It is also discrete type values of 1 to 10.
- **Faction** The faction of unit i, denoted by F_i, is the ownership of the unit. Faction has two possible values, 0 and 1, representing friend faction and enemy faction respectively.

The game time of a battle, represented by t as some certain time in the battle and T as the total time of the battle, is accurate to a round. A battle is at most 240 seconds, and there are 50 rounds per second. The update episode of SEAT model is 15 rounds. Under such setting, the range of the game time is $1 \leq t \leq 800$ and $T = 800$. The total elixir, represented by m_t as elixir at time t and M as the maximum elixir, has the range of $0 \leq m_t \leq 10$ and $M = 10$.

2.3 Game Process

In this subsection, we provide the introduction to the game process. The game process includes game update, winning condition and player action. Game update is the process of game states changing between rounds. Winning condition is the conditions under which a player is judged as win, tie or lose when a battle is going to end. Player action is to select

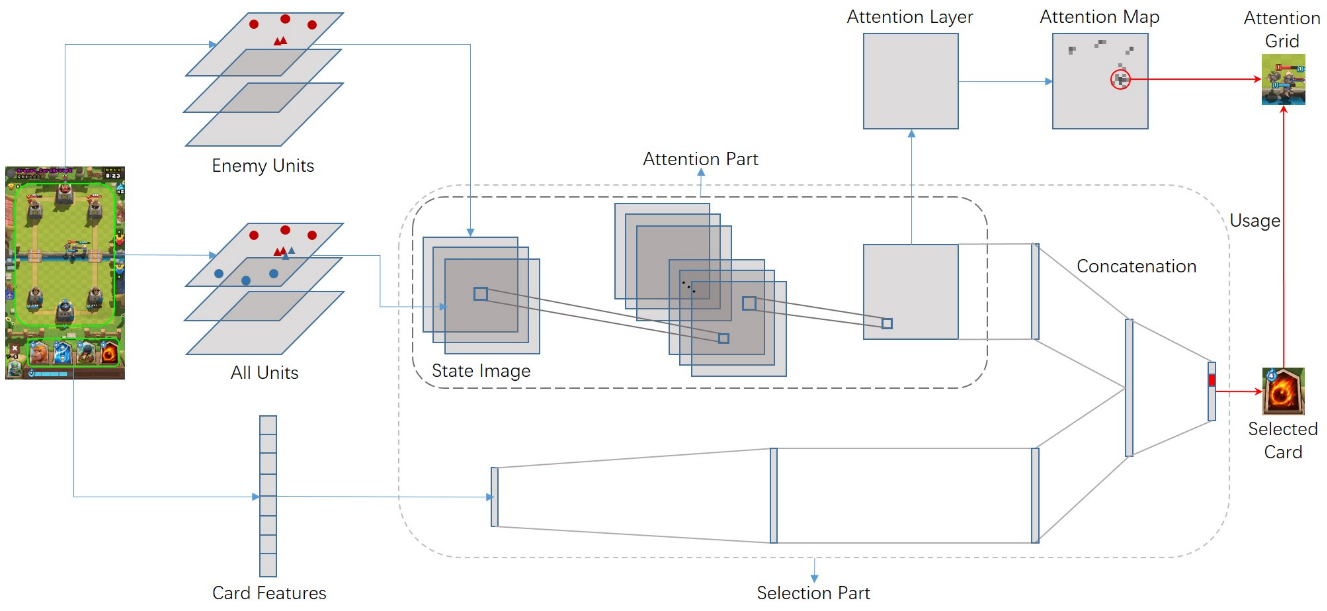

Figure 2: Selection-Attention Model.

and use cards with the help of game states to reach winning condition.

Game Update If a player has enough elixir, a card can be used, and the corresponding unit will follow fixed patterns of action. A troop finds its way to the nearest enemy unit automatically and attacks the enemy unit once within its attack range. A spell takes effect immediately to attack the effected area. A building stays still and attacks enemy units that move into its attack range.

Winning Condition A battle of CR has two time stages, with the first time stage called regular time and the second time stage called over time. During the 3-minites-long regular time, the target is to destroy more towers of the enemy. Once regular time comes to an end, the player with more towers will win. If a tie (the number of towers are equal) appears here, the battle will go on to the 1-minute-long over time. In this time stage, the player who firstly destroys one more enemy tower will win. If no one makes it to destroy a tower during over time, the final result is a tie. Therefore, the battle result for player i, denoted by w_i, has three possibilities, namely 2 for win, 1 for tie, and 0 for lose, as (1) shows.

$$w_i = \begin{cases} 2, & \text{win} \\ 1, & \text{tie} \\ 0, & \text{lose} \end{cases} \qquad (i = 1, 2) \tag{1}$$

Player Action The real-time actions of CR players are to make decisions on when and where to use a card, and which card to use. There are totally eight cards in a card group. The card group is different for two players in the original CR mobile game, while the setting here is the same, to keep fairness. The cards will appear in hand in a random order (controlled by a random seed) with four as the volume of the hand cards, continuously until the end of a battle. Once a card is used, a next one will replace it. We mark our hand cards with index i $(1 \leq i \leq 4)$, and the elixir consumption of card i is denoted as m_i. If a player wants to use hand card i at t, the elixir consumption of the card must be lower than or equal to m_t. After the card is used, the total elixir changes to $m_t - m_i$. Meanwhile, the total elixir recovers 1 point every 2.8 seconds (140 rounds). The actions can be formalized by the probability $U_{i,t}$ of choosing card i and the position $P_{i,t}$ to use it at time t, as (2) and (3) shows.

$$U_{i,t} \in [0, 1], \tag{2}$$

$$P_{i,t} = (x, y). \quad (i = \{1, 2, 3, 4\}) \tag{3}$$

3 SEAT Model

SEAT model includes a selection part and an attention part. In this way, the model can solve card-based RTS games by divide and conquer. Card choice by selection part is based on both battle field states and complex card features, while card usage by attention part is mainly related to enemy states. SEAT model takes a specially designed state representation manner which helps to refine complex states. Besides the total reward of a battle, the reward during each update episode is defined for model training process. A detailed model sketch is as Figure 2. In the following subsections, we will introduce state representation, the selection part, the attention part and reward definition respectively.

3.1 State Representation

Different from Atari games whose agents can learn directly from raw pixelwise images, CR has much larger battle field size and more complex game states. Another difference is that CR contains much information (card features) out of the visually seen battle field. So we propose an effective method

Algorithm 1 SEAT Model

```
Input: State $s_t$ (battle field states and card features)
Init: Initialize model $Q$ parameters $\theta$, target model $Q'$ parameters $\theta'$ and experience $D$
repeat
 1: Start a new battle of CR
 2: while CR battle ongoing do
 3:     if battle ends then
 4:         Go to 1
 5:     else
 6:         Transform battle field states input into state images
 7:         if random value $\leq \epsilon$ then
 8:             $a_t = 0$
 9:         else
10:             Select card by $a_t = \max_a Q(s_t, a; \theta)$
11:         end if
12:         Get attention grid $g$ by processing state images
13:         Use card at $g$ and get next state $s_{t+1}$
14:         Get reward $r_t$ by $r_t = r_t^{troop} + r_t^{card} + r_t^{tower}$
15:         Store $< s_t, a_t, r_t, s_{t+1} >$ tuple into $D$
16:         Get a ramdom batch of tuples from $D$
17:         Calculate gradients of $r + \gamma Q'(s', a')$
18:         Update parameters $\theta$ by gradient descent
19:         Set $\theta'$ with $\theta$ every N steps
20:     end if
21: end while
until done
```

to embed the game states, which can represent the complex states in a straightforward way.

Convolutional neural networks (CNN) can extract spatial correlation features from raw images. Here we leverage the characteristic of CNN and build up a special "image" as the input, which is called state image. A state image has three layers, namely health layer of health attribute values, class layer of class attribute values and faction layer of faction attribute values. Each layer has 32×18 grids, with one grid representing 1000×1000 pixels in the original battle field. Once a unit appears in a grid, the value of its attributes will occupy that grid of the corresponding layers. These state images act as an important part of the input for training. The other part of the input is the features of hand cards. They are represented directly as their attributes at time t. The range of the attributes are as Formula (4) shows.

$$h_{i,t} \in [1, 10], c_{i,t} \in [1, 10], f_{i,t} \in \{0, 1\}. \tag{4}$$

3.2 Selection Part

The selection part is for card choice, meaning to choose which hand card to use. In a battle, there are always four cards in hand as mentioned before. We choose to use some card or do not use a card during a round.

$$\begin{aligned} Q(s_t, a_t) \leftarrow & (1 - \alpha) Q(s_t, a_t) + \\ & \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \right] \end{aligned} \tag{5}$$

The Q-values, representing the values of state-action pairs, are normally updated during each update episode of the training process according to Formula (5). α is the learning rate and γ is the discount factor of reward.

However, under conditions when possible state-action pairs are too many (e.g. states are images), it is needed to make approximation of the Q-value function. The initialization of the approximated Q-value function for the selection part is by building up deep networks to map high order states to Q-values. We also initialize replay experience D that records past experience to enable experience replay mechanism, which makes the training process more stable.

The input of the selection part includes state images of all units on the battle field as well as card features. The state images are firstly processed by two convolutional layers, then transformed into a feature vector. The card features are in the form of a vector with eight values and are processed by two fully connected layers. After that, the features extracted from both inputs are concatenated and mapped to five Q-values representing hand cards 1 to 4 and one empty selection.

The training process of SEAT model is as Algorithm 1 shows. For SEAT model, there is an approximate probability of 80% to use some card during one update episode. While in real games, the frequency of using cards is much lower. Thus, we set up the hyperparameter of abandoning-rate ϵ for better training of the model, which means the choice of cards is given up under a possibility of ϵ.

During each update episode, the selection part either takes an empty action with probability ϵ or selects an action (card choice) by $a_t = \max_a Q(s_t, a; \theta)$. Then the transition (s_t, a_t, r_t, s_{t+1}) is stored in D. The Q-value $Q^*(s', a')$ are related to all possible actions a'. The optimal Q-value is got by selecting the action that maximises the expected value of $r + \gamma Q^*(s', a')$, as Formula (6) shows.

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \,|\, s, a \right] \tag{6}$$

The update of network parameters is carried out by mini batch of transitions randomly sampled from D. Firstly set y_j in the form of (7) and then perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$.

$$y_j = \begin{cases} r_j & \text{terminal } s \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{otherwise} \end{cases} \tag{7}$$

The differential of $(y_j - Q(s_j, a_j; \theta))^2$ is in form of (8). Stochastic gradient descent (SGD) method is used to optimise the loss function.

$$\begin{aligned} \nabla_{\theta_i} L_i(\theta_i) = & \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \\ & [(y_j - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)] \end{aligned} \tag{8}$$

3.3 Attention Part

The attention part is for card usage, meaning to use the selected card on the battle field. Noting that the battle field is compressed to 32×18 grids, we should find the grid that gets most attention, where most attacks are happening or the

Name	Class	Feature
Ninja	Troop	multi-unit
DartNinja	Troop	multi-unit
BatSwarm	Troop	multi-unit
ApeWarior	Troop	tower-targeted
KungFuMaster	Troop	AOE
FireBall	Spell	AOE
Zap	Spell	AOE
Catapult	Building	AOE

Table 1: Card Group Description.

enemy tower with least health is located. Then we use the selected card at that grid to achieve higher reward.

The attention method orginates from the object detection task in computer vision area [Liu *et al.*, 2016; Redmon and Farhadi, 2017]. When checking whether the object box acquired by the detection model is right, some researchers usually stack all the boxes together to see which area in the picture gets the most attetion visually. Here we are inspired by this checking way and put forward the attention part.

The input of the attention part only includes state images of enemy units. Different from the selection part, the attention part is not based on training. During each round, after the update of model parameters finishes, the state images of enemy units are processed by convolutional layers and the output are result values of the last convolutional layer of the selection part. The output, called attention map, is exactly of 32×18 grids, the same as input state images. The grid with maximum value is called the attention grid. For some cards, the attention grid is for their final usage. However, The usage area of troop cards is fixed to the own half battle field before any enemy tower is destroyed. Meanwhile, some special cards have fixed usages in accordance with their features, which will be introduced in the next section.

3.4 Reward Definition

The item reward is used to represent the gain a player gets in a certain round. The defined reward includes three parts, troop reward, card reward and tower reward.

Troop reward, represented by r_t^{troop} as Formula (9), denotes the change in the number of enemy units on the battle field during the update episode from $t-1$ to t. If an enemy unit gets defeated, it brings $+2$ reward. If the number of enemy units does not decrease, r_t^{troop} will be set as 0.

$$r_t^{troop} = (+2) * max(N_t^{etr} - N_{t-1}^{etr}, 0) \tag{9}$$

Card reward, represented by r_t^{card} as Formula (10), denotes the usage of hand cards during the update episode from $t-1$ to t. Once a hand card is used, it brings -3 reward. Here N_t^{card} means the number of cards used between $t-1$ and t.

$$r_t^{card} = (-3) * N_t^{card} \tag{10}$$

Tower reward, represented by r_t^{tower} as Formula (11), denotes the change in towers of both factions during the update episode from $t-1$ to t. If an enemy tower is destroyed, it brings $+20$ reward. However, if an friend tower is destroyed, it brings -30 reward.

$$\begin{aligned} r_t^{tower} =& (+20) * (N_{t-1}^{eto} - N_t^{eto}) + \\ & (-30) * (N_{t-1}^{fto} - N_t^{fto}) \end{aligned} \tag{11}$$

The total reward at time t and the accumulated reward from time t' on are in the form of (12) and (13) respectively. Here the parameter γ is the discount factor of rewards by time.

$$r_t = r_t^{troop} + r_t^{card} + r_t^{tower} \tag{12}$$

$$R_{t'} = \sum_{t=t'}^{T} \gamma^{t-t'} \cdot r_t \tag{13}$$

4 Experiments

The performance of SEAT model are evaluated under the simulation environment of Clash Royale implemented in lua on Windows platform. We firstly introduce the experiment settings including hyperparameters and card group. Then we statistically analysis the performance of SEAT model agent against rule-based agents and decision-tree-based agent.

4.1 Experiment Settings

The SEAT model is implemented in PyTorch. Hyperparameters not specified in last section include discount factor γ (set as 1), the update period N of $\theta' = \theta$ (set as 10) and the abandoning-rate ϵ (set as 0.0 and 0.5 for contrast experiments). As mentioned before, the update episode of SEAT model is every 15 rounds, which is the same as human's fastest reaction time (0.3s) when playing RTS games. In this way, the agent learnt has most similar behaviours as human players which guarantees fairness.

A card group contains eight cards. In our experiment, we use five troop cards, two spell cards and one building card. This is a typical card group choice which ensures the variety of cards, game balance and moderate difficulty. Troop cards include three multi-unit card, one tower-targeted card and one AOE (area of effect) card. Spell cards are both AOE cards. One can attack towers and the other cannot. The building card also makes AOE attack. A detailed description of the eight cards is shown in Table 1.

Catapult is a building card which is normally used in front of one's own major tower. Card ApeWarior is tower-targeted, so it is for attacking the enemy tower with least health and used along river side (in the middle of the battle field). For other troop cards, if the attention grid is in the enemy half battle field, it is then used along the river side also, due to the restriction of card usage area. The spell cards are not limited by usage area and are used directly at the attention grid.

4.2 Experiment Results

Experiments are carried out among rule-based agents (RB-agent), decision-tree-based agent (DT-agent) and the SEAT model agent (SEAT-agent). Rule-based agent has two different strategies, aggresive strategy (ARB-agent) and defensive

battle	ARB-DRB		ARB-DT		DRB-DT	
agent	ARB-agent	DRB-agent	ARB-agent	DT-agent	DRB-agent	DT-agent
winning-rate	40%	60%	20%	80%	30%	70%
battle	ARB-SEAT		DRB-SEAT		DT-SEAT	
agent	ARB-agent	SEAT-agent	DRB-agent	SEAT-agent	DT-agent	SEAT-agent
winning-rate	10%	**90%**	10%	**90%**	30%	**70%**

Table 2: Experiment Results of Agent Battles.

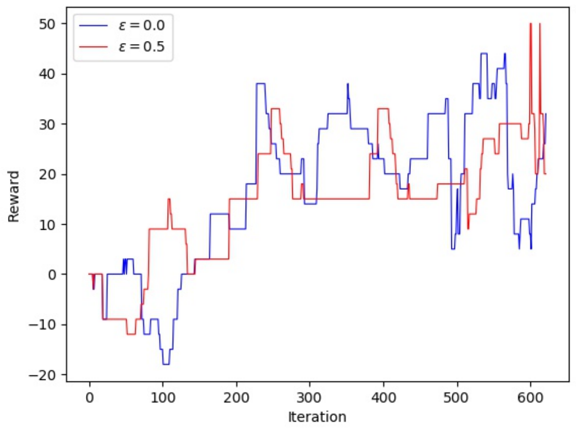

Figure 3: Reward Contrast ($\epsilon = 0.0$ and $\epsilon = 0.5$).

strategy (DRB-agent). The aggresive strategy focuses on continuous attack by using cards whenever elixir is enough, while the defensive strategy focuses on continuous defense by the restraint relationship between different cards. The decision-tree-based agent has more complex strategies implemented by making decisions on what to do under different compound conditions. In other words, it is a combination of aggresive and defensive strategies.

We conduct six groups of experiments among the above mentioned agents. The battles occur between the following agent pairs, ARB-DRB, ARB-DT, DRB-DT, ARB-SEAT, DRB-SEAT and DT-SEAT, with each pair tested 10 times. Results of winning rate are shown in Table 2. As we can see, DRB-agent is better than ARB-agent, illustrating that defensive strategy performs better than aggresive strategy. Both ARB-agent and DRB-agent are weaker than DT-agent, indicating that single strategy can not beat complex strategies. SEAT-agent acquires higher winning rate against all the other three agents, which proves the effectiveness of DRL method for playing card-based RTS games.

From observations of battle experiments, the SEAT-agent has better judgement on battle field states and more accurate card usage position. For example, the ApeWarior is a tower-targeted card which has fast speed and high damage against towers. Once an enemy ApeWarior appears, the SEAT-agent uses multi-unit cards at the river side to form a defense area and destroys the ApeWarior in half way before it starts to attack friend towers. KungFuMaster is the only troop card which makes AOE attack, but it moves slowly. When a KungFuMaster is marching forward, the defensive card used by SEAT model is usually the multi-unit card BatSwarm that makes remote attack and can avoid counterattack by the characteristic of air units.

A contrast experiment is conducted under different settings of the hyperparameter abandoning-rate ϵ. It is set as 0.0 and 0.5 respectively. Under our designed reward form, a reward of more than 10 in a battle represents a large possibility of destroying more towers, which ends up in winning. Reward curve as Figure 3 shows that the model with a relatively higher ϵ (0.5) has a faster learning rate in the early training stage (before iteration 150) and more stable learning process in the later training stage (after iteration 450).

5 Conclusion and Future Work

In this paper, we have proposed a deep reinforcement learning model, SEAT, for playing card-based RTS game Clash Royale. The SEAT model includes two parts, the selection part for card choice and the attention part for card usage. By transforming the battle field state information into specially designed state images, the selection part can map the huge state space to final card choice. By processing enemy states with trained convolutional layers, the attention part can find the grid on the battle field that should get most attention, namely the grid to use the selected card at. Empirical results show that our SEAT model agent outperforms aggresive rule-based agent, defensive rule-based agent and decision-tree-based agent substantially.

For future work, the SEAT model needs to be retrained or restructured to fit new situations. To ensure fairness, our experiment setting here is the same card group for both players. In fact, the card groups are very likely to be different in real battles, so how to deal with different card groups will be the next problem to solve for card-based RTS games. How to form up a good card group from a large amount of various cards can be another problem. Experiments till now have been conducted only among artificial agents. Battles between artificial agents and high-level or professional human players are also challenging problems.

Acknowledgements

This work was supported by the National Nature Science Foundation of China under grant number 61625101.

References

[Barriga *et al.*, 2018] Nicolas A Barriga, Marius Stanescu, and Michael Buro. Game tree search based on nondeterministic action scripts in real-time strategy games. *IEEE Transactions on Games*, 10(1):69–77, 2018.

[Buro and Churchill, 2012] Michael Buro and David Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106, 2012.

[Churchill *et al.*, 2012] David Churchill, Abdallah Saffidine, and Michael Buro. Fast heuristic search for rts game combat scenarios. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[Clark and Fleshner, 2017] Corey Clark and Anthony Fleshner. Fast random genetic search for large-scale rts combat scenarios. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[Gu *et al.*, 2017] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[Jiang *et al.*, 2018] Daniel Jiang, Emmanuel Ekwedike, and Han Liu. Feedback-based tree search for reinforcement learning. In *International Conference on Machine Learning*, pages 2289–2298, 2018.

[Lample and Chaplot, 2017] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[Levine *et al.*, 2016] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[Lillicrap *et al.*, 2016] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

[Liu *et al.*, 2016] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Ontanón and Buro, 2015] Santiago Ontanón and Michael Buro. Adversarial hierarchical-task network planning for complex real-time games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[Pasunuru and Bansal, 2017] Ramakanth Pasunuru and Mohit Bansal. Reinforced video captioning with entailment rewards. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 979–985, 2017.

[Redmon and Farhadi, 2017] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Synnaeve and Bessiere, 2011] Gabriel Synnaeve and Pierre Bessiere. A bayesian model for rts units control applied to starcraft. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 190–196. IEEE, 2011.

[Synnaeve *et al.*, 2018] Gabriel Synnaeve, Zeming Lin, Jonas Gehring, Dan Gant, Vegard Mella, Vasil Khalidov, Nicolas Carion, and Nicolas Usunier. Forward modeling for partial observation strategy games-a starcraft defogger. In *Advances in Neural Information Processing Systems*, pages 10761–10771, 2018.

[Uriarte and Ontanón, 2016] Alberto Uriarte and Santiago Ontanón. Improving terrain analysis and applications to rts game ai. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[Usunier *et al.*, 2016] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

[Wang *et al.*, 2017] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198, 2017.

[Watters *et al.*, 2017] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.