Application of Artificial Intelligence in Real-Time Strategy Games

[1]Nishant Agarwal, [2]Abhishek Kumar https://orcid.org/0000-0002-9925-8137, [3] Achintya Singhal
[1]Scholar, [2]Assistant Professor, [3]Associate Professor
[1]Department of Computer Science,
[1] Banaras Hindu University, Varanasi, India

Abstract : This report presents an overview of the development of bots for competitive real-time strategy (RTS) games using Artificial Intelligence. A comparative study is made between the design of AI bots for strategic board games and RTS games and the challenges involved in the development of an AI bot for RTS games are highlighted. Existing work addressing these challenges are discussed. Important concepts used in designing an RTS game AI such as Imitation Learning, Adaptive AI and Reinforcement Learning (RL) are explained. An implementation of AI bot for the RTS game DOTA 2 (Defense of the Ancients 2) using Reinforcement Learning is presented and the "reward function" used in the implementation is discussed. Finally, the recent developments in board game AI (with respect to Chess and Go) and in RTS game AI (with respect to DOTA 2) are presented and the future prospects are discussed.

IndexTerms **- Artificial intelligence (AI), computer games, real-time strategy (RTS), multiplayer online battle arena (MOBA), defense of the ancients (DOTA), imitation learning, adaptive AI, reinforcement learning (RL), task allocation (TA).**

I. INTRODUCTION

1.1 What is AI?

In computer science AI research is defined as the study of "intelligent agents", i.e., any device that perceives its environment and takes actions that maximize its chance of success at some goal.The term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".Capabilities generally classified as AI include successfully understanding human speech, competing at a high level in strategic game systems (such as chess and Go), autonomous cars, intelligent routing in content delivery networks, military simulations, and interpreting complex data, including images and videos.Important areas of AI application and research include: expert systems, natural language processing, computer vision, robotics, intelligent computer-assisted instruction, automatic programming, planning and decision support.

1.2 What are RTS / MOBA?

The term "real-time" means "without significant delay" or without a time-gap.

- **Real-time strategy (RTS)** is a subgenre of strategy e-sports games where the game does not progress incrementally in turns. Instead, the game time progresses continuously according to the game clock. Players perform actions simultaneously as opposed to in sequential units or turns. Players must perform actions with the consideration that their opponents are actively working against them in real-time, and may act at any moment. This introduces time management considerations and additional challenges such as physical coordination.

- **Multiplayer online battle arena** (**MOBA**), also known as **action real-time strategy** (**ARTS**), is a subgenre of strategy e-sports games that originated as a subgenre of real-time strategy, in which a player controls a single character in a team who compete versus another team of players.

- **Basic objective:** In an RTS, the participants position and manoeuvre units and structures under their control to secure areas of the map and/or destroy their opponents' assets.

- Artificial Intelligence problems related to RTS games deal with the behavior of an artificial player. This consists among others to learn how to play, to have an understanding about the game and its environment, to predict and infer game situations from a context and sparse information.

1.3 DOTA (Defense of the Ancients) 2:

DOTA (Defense of the Ancients) 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve Corporation. DOTA 2 is played in matches between two teams of five players, with each team occupying and defending their own separate base on the map. Each of the ten players independently controls a powerful character, known as a "hero", who all have unique abilities and differing styles of play. During a match, players collect experience points and items for their heroes to successfully defeat the opposing team's heroes in player versus player combat. A team wins by being the first to destroy a large structure located in the opposing team's base, called the "Ancient". DOTA 2 is one of the most popular and complex e-sports games in the world, with creative and motivated professionals who train year-round to earn part of DOTA's annual $40M prize pool (the largest of any e-sports game). The International 2018, which was a $25 million tournament hosted at the Rogers Arena in Vancouver.

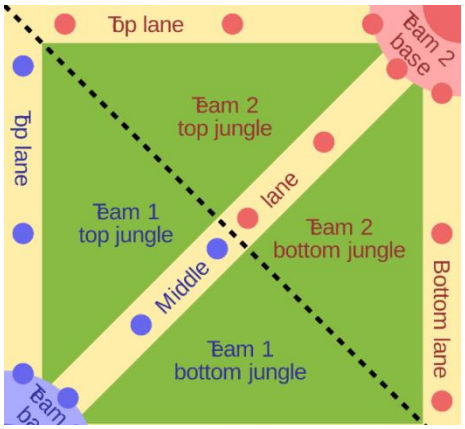

Fig. 1: DOTA 2 Map
Source: Wikimedia Commons

II. Related Work

2.1 Challenges involved in designing an RTS game AI – A comparative study of RTS game AI and Board Game AI

While designing an AI for board games like Chess and Go can be challenging, designing an AI for a real-time strategy game like DOTA 2 involves much more complexities. Standard techniques used for playing classic board games, such as game tree search, cannot be directly applied to solve RTS games without the definition of some level of abstraction, or some other simplification. Researchers S. Ontan˜on, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss have highlighted the challenges related to RTS game AI in their paper *RTS AI: Problems and Techniques*. Some of the differences between board games and RTS games in view of designing an AI bot are discussed below:

- **Long time horizons**: An RTS game involves longer time horizons in terms of average number of moves per game as compared to classic board games. For e.g., DOTA 2 games run at 30 frames per second for an average of 45 minutes, resulting in 80,000 ticks per game. If an AI bot observes every fourth frame, it yields 20,000 moves per game. Chess usually ends before 40 moves, Go before 150 moves.
- **Complex and continuous action space**: In RTS games all units are moving simultaneously and more than one player can issue actions at the same time. Moreover, it involves dozens of units each of which can take a number of possible actions at a given time. For e.g., in a DOTA 2 game a Hero unit can take an average of ~1,000 valid actions each tick. The average number of actions in chess is 35; in Go, 250.
- **Durative actions:** Action in RTS games are durative, i.e. actions are not instantaneous, but take some amount of time to complete. In board games, actions are generally instantaneous.
- **Partially-observed state**: Most RTS games are partially observable: units and buildings can only see the area around them. The rest of the map is covered in a fog hiding enemies and their strategies. This is referred to as the fog-of-war. Strong play by an AI requires making inferences based on incomplete data, as well as modeling what one's opponent might be up to. Both chess and Go are full-information games and players can see the entire action space all the time. –
- **Non-deterministic actions**: Most RTS games are non-deterministic: some actions have a chance of success, and the amount of damage dealt by different units is sometimes stochastic. For e.g., in DOTA 2, it may be difficult for an AI bot to estimate the utility of a ward at a particular spot as the benefit is uncertain. In board games like chess, the chance of success after a particular move can be estimated with a fair degree of certainty.
- **Large and dynamic observation space:** DOTA is played on a large continuous map containing ten heroes, dozens of buildings, dozens of NPC units, and a long tail of game features such as runes, trees, and wards. OpenAI Five model observes the state of a DOTA game via Valve's Bot API as 20,000 (mostly floating-point) numbers representing all information a human is allowed to access. A chess board is naturally represented as about 70 enumeration values (a 8x8 board of 6 piece types and minor historical info); a Go board as about 400 enumeration values (a 19x19 board of 2 piece types plus Ko).

2.2 Addressing the challenges regarding micro and macro decision making

When humans play an RTS game like DOTA 2, they typically divide their decision making as follows:

- **Micro**: It is the ability to control units individually (roughly corresponding to reactive control, and part of tactics). A good micro player can maneuver heroes and controlled units effectively with precision.
- **Macro**: It is the ability to strategize and set short term goals like effectively farming the jungle between creep waves, pushing strategically important towers early and setting up successful ganks (roughly corresponding to everything but reactive control and part of tactics above).
- In comparison, AI bots work by decomposing the problem of playing an RTS game into a collection of smaller problems, to be solved independently. Specifically, a common subdivision is as follows:

- **Strategy**: It corresponds to the high-level decision-making process. This is the highest level of abstraction for the game comprehension. Finding an efficient strategy or counterstrategy against a given opponent is key in RTS games, and concerns the whole set of units a player owns.
- **Tactics**: These are the implementation of the current strategy. It implies unit and ward positioning, movements, timing, and so on. Tactics concerns a group of units. Tactical reasoning involves reasoning about the different abilities of the units in a group and about the environment (terrain) and positions of the different groups of units in order to gain advantage in fights.
- **Reactive control**: This is the implementation of tactics. It consists in moving, targeting, attacking, fleeing, hit-and-run techniques (also known as "kiting") during fights. Reactive control focuses on a specific unit. It aims at maximizing the effectiveness of units, including simultaneous control of units of different types in complex battles on heterogeneous terrain.

Researchers have proposed the following approaches to solve each of the above problem:

- **Strategy** - In the context of RTS games, strategic decision making has been addressed using many AI techniques, such as hard-coded approaches, planning-based approaches, or machine-learning-based approaches. Hard-coded approaches have been extensively used in commercial RTS games. The most common ones use finite-state machines (FSM) in order to let the AI author hard-code the strategy that the AI will employ. The idea behind FSMs is to decompose the AI behavior into easily manageable states, such as "attacking," "gathering resources," or "repairing," and establish the conditions that trigger transitions between them. Commercial approaches also include hierarchical FSMs, in which FSMs are composed hierarchically. These hard-coded approaches have achieved a significant amount of success, and they have also been used in many academic RTS AI research systems. However, these hard-coded approaches struggle to encode dynamic, adaptive behaviors, and are easily exploitable by adaptive opponents.
- **Tactics -** Concerning tactical decision making, many different approaches have been explored, such as machine learning or game-tree search. Additionally, scouting is equally important in tactical decision making as in strategic decision making. All previous approaches, including all game-tree search ones, assume complete information.
- **Reactive Control -** Potential fields and influence maps have been found to be useful techniques for reactive decision making. Some uses of potential fields in RTS games are: avoiding obstacles (navigation), avoiding opponent fire, or staying at maximum attack distance. Potential fields have also been combined with A* pathfinding to avoid local traps.

2.3 Imitation learning

Imitation learning is a type of social learning whereby new behaviors are acquired via imitation. The aim is to perform a task by mimicking human behavior in the given task. An agent (a learning machine) is trained to perform a task from demonstrations by learning a mapping between observations and actions.

The idea behind implementation an RTS game AI using this technique is to collect the game logs of human players in a particular RTS game from repositories and use them to have bots play the games by imitation. Although game artificial intelligence (AI) developers find it attractive, it is not a trivial task to generalize this to new situations effectively. It has been reported that the imitation usually performs poorly because of the mismatch between human perception and the low-level sensory inputs to game AI bots.

Researchers In-Seok Oh, Ho-Chul Cho and Kyung-Joong Kim in their paper *Imitation Learning for Combat System in RTS Games with Application to StarCraft* have proposed the implementation of imitation learning in RTS games with the help of influence maps (IMs). Influence maps analyze the influence of units and the terrains spatially within the RTS game environment. During game play, AI bot produces an influence map of the current game situation. Then it explores the replay repository to search for the best similar moment from an influence map representation by comparing the current IM with all of the IMs stored in the repository. To speed up the comparison, all of the IMs are loaded into memory. In addition, a hashing technique is used to reduce the search time. If the closest IM is found, the AI bot loads the associated raw game events to the IM. The raw data include all of the commands and movement directions of the units. Naturally, the current game scene situation (the number of units, their types, and positions) is not exactly the same as the closest scene from the repository. It is necessary to map and compare the units in the current scene to the ones from the closest scene. Then, the AI bot imitates the behaviours of the closest units if the distance is within a pre-defined threshold and the associated actions are initiated in the game.

Experimental results show that the proposed AI can defeat well-known competition entries a large percentage of the time.

2.4 Adaptive Game AI

Modern video games present a complex and realistic environment in which characters controlled by game AI are expected to behave realistically (human-like). An important feature of human-like behaviour of game AI is the ability to adapt to changing circumstances. Game AI endowed with this ability is called "adaptive game AI," and is typically implemented via machine-learning techniques. Adaptive game AI may be used to improve the quality of game AI significantly by learning effective behaviour while the game is in progress.

A major disadvantage of nonadaptive game AI is that once a weakness is discovered, nothing stops the human player from exploiting the discovery. The disadvantage can be resolved by endowing game AI with adaptive behavior, i.e., the ability to learn from mistakes. Adaptive game AI can be created by using machine-learning techniques, such as artificial neural networks or evolutionary algorithms. In practice, adaptive game AI in video games is seldom implemented because currently it requires numerous trials to learn effective behavior (i.e., game adaptation is not rapid). In addition, game developers are concerned that applying adaptive game AI may result in uncontrollable and unpredictable behavior (i.e., game adaptation is not reliable).

Researchers Sander Bakkes, Pieter Spronck, and Jaap van den Herik, in the paper *Rapid and Reliable Adaptation of Video Game AI*, have presented an approach of case-based adaptive game AI. To allow rapid and reliable adaptation in games, they have described an approach to behavioral adaptation in video games that is inspired by the human capability to solve problems by generalizing over previous observations in a restricted problem domain. In the approach, domain knowledge required to adapt to game circumstances is gathered automatically by the game AI, and is exploited immediately (i.e., without trials and without resource-intensive learning) to evoke effective behaviour in a controlled manner in online play. They performed experiments that test case-based adaptive game AI on three different maps in a commercial real-time strategy (RTS) game. From the results, it can by concluded that case-based adaptive game AI provides a strong basis for effectively adapting game AI in video games.

2.5 RTS game micromanagement as a Task Allocation (TA) Problem

Real-time strategy (RTS) gameplay is a combination of strategy and micromanagement. While strategy is clearly important, the success of a strategy can depend greatly on effective micromanagement. Recent years have seen an increase in work focusing on micromanagement in RTS AI, but the great majority of these works have focused on policies for individual units or very specific situations, while very little work has aimed to address the need for a broadly applicable structure for unit group coordination. Researchers Keith D. Rogers and Andrew A. Skabar in the paper *A Micromanagement Task Allocation System for Real-Time Strategy Games*, have conceptualized RTS group level micromanagement as a multi agent task allocation (TA) problem, and proposed the micromanagement task allocation system (MTAS) as a framework to bridge the gap between strategy and unit level micromanagement.

Micromanagement in RTS games can also be viewed as a TA problem, since at any point in time the many different actions that can be assigned to objects in the game can be viewed as tasks. Based on an object's abilities, these tasks may include attacking enemy units, healing friendly units, constructing buildings, moving to a new location, constructing new units, etc. For example, the option of "attacking enemy unit x" becomes an "attack task," with the target being "unit x.".The challenge is to devise a system that distributes these tasks among the available agents (units and buildings) in such a way that the group achieves the goals passed to it from the strategic level AI.

To illustrate how micromanagement in RTS games compares to other TA problems, and to understand the specific challenges of allocating tasks to a group of agents in an RTS environment, the important conditions that such a TA system must satisfy were presented as follows:

- Dynamic Task Allocation/Re-Allocation
- Centralized Processing
- Agent-Oriented Task Allocation
- Heterogeneity and Cooperative Task Execution
- Nondependent Tasks
- High Throughput
- Fault Tolerance
- CPU Intensiveness

MTAS is presented as a flexible, understandable, and generally applicable model for providing an intelligent allocation of tasks within a diverse group of units in pursuit of goals passed to the group from the strategy-level component. The main advantage that MTAS brings to RTS AI over current techniques is a system that can micromanage groups of units that is flexible enough to be applicable to a wide variety of unit types, can pursue goals set by the strategic component of the RTS AI, provide meaningful feedback in regards to goal completion/failure/invalidation, and behave tolerably in the absence of explicit goals. Groups that can multitask in an intelligent, non predefined manner opens up the possibility of unit groups containing a wider variety of unit types, as members of the group can perform different actions that play to their individual strengths and abilities and unit type assemblages are not restricted to those that the AI programmer has accounted for in advance (as is often the case when using scripted group behavior or other tailored micromanagement solutions).

III. An Implementation of a DOTA 2 AI

3.1 Reinforcement Learning (RL)

Reinforcement Learning is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance.

Like most advances in artificial intelligence, RL is derived from studying the intelligence of humans. The core concepts of RL come from behaviorism, which basically says that everything we do in life is a reflex response to our current environment or a consequence of our past actions.

RL uses trial and error to learn to make the best decisions possible by achieving the best reward possible over a period of time. A good example of this is how you would train a dog. Every time the dog rolls over when you tell them to, they get a treat (reward +1). Every time they pee on the rug they get yelled at (reward -1). Over time they learn to do the things that get them the most positive rewards and avoid the things that get the most negative rewards. Due to the fact that RL uses unsupervised learning, it doesn't need to be told how to achieve something, it just needs to know what to achieve. This means it can find solutions to problems that humans may never have even thought of. It works as follows:

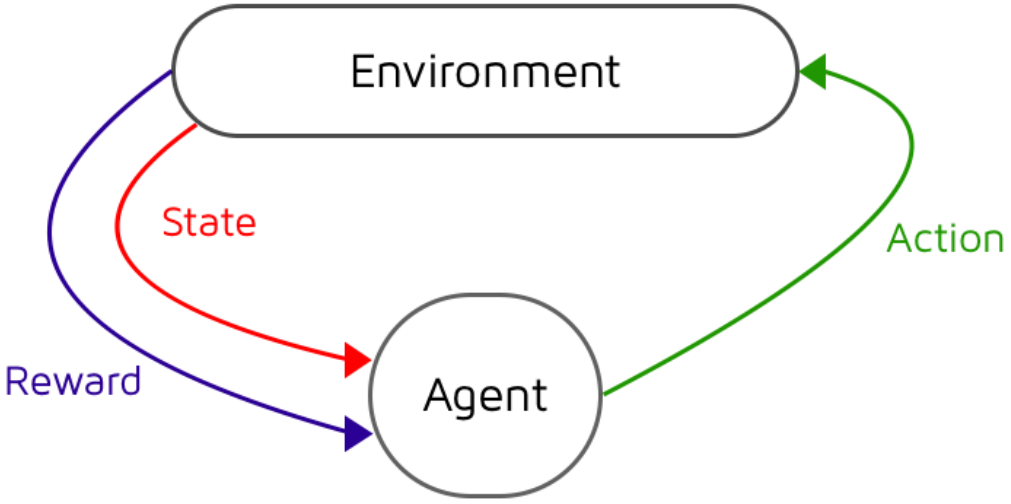

Fig. 2: Reinforcement learning loop
Source: Nervana Deep Reinforcement learning with OpenAI gym.

As the agent makes its way through the environment it goes through a type of learning loop. This is shown in Fig. 2. The agent identifies what state it is currently in and decides the best action to take given its current state. When this action is carried out in the environment the agent observes the new state after taking that action as well as the reward it received for taking an action while in that state. Through this method our agent learns.

The problem of long-term strategic decision making and credit assignments in an RTS game can be addressed with reward shaping, where one breaks down the eventual reward into small pieces, to directly encourage the right behaviors at each step. For example, in OpenAI Five (an AI for the RTS game DOTA 2), rather than learning last-hit is indirectly responsible for the ultimate victory, a last-hit is directly rewarded with a score of 0.16, whereas dying is punished with a negative score of -1. The agent would immediately learn that last-hit is good while dying is bad, irrespective to the ultimate victory of a game.

3.2 DOTA 2 reward function

Reward function is one of the most important aspects of RL. Researchers have presented the following method for optimization of rewards in the RTS game DOTA 2.

The optimization algorithm selects the parameters to maximize the rewards the agent receives. As with many RL tasks, rewards are assigned for behaviors that may lead to the goal (winning the game), without over-crafting the reward to the expectations. Each hero's reward is a linear combination of separate signals from the game. Each of these signals produces a score, and the agent's reward is the increase in score from one tick to the next.

Hero Score Signals

The individual hero's score (before the above averaging and reweighting) is a linear combination of separate signals from the game.

Individual Scores

Most of the scores are "individual" signals related to the individual hero:

Individual	Weight	Awarded for
Experience	0.002	Per unit of experience.

Individual	Weight	Awarded for
Gold	0.006	Per unit of gold gained [1].
Mana	0.75	Mana (fraction of total).
Hero Health	2.0	Gaining (or losing) health [2].
Last Hit	0.16	Last Hitting an enemy creep [3].
Deny	0.2	Last Hitting an allied creep [3].
Kill	-0.6	Killing an enemy hero [3].
Death	-1.0	Dying.

Building Scores

For each important building class, all heroes on the team receive a fixed score if the building is alive, and a bonus score linear in the building's health:

Score for live building = Weight * (1 + 2 * Health Fraction).

Buildings	Weight
Shrine	0.75
Tower (T1)	0.75
Tower (T2)	1.0
Tower (T3)	1.5
Tower (T4)	0.75
Barracks	2.0
Ancient [4]	2.5

The agent receives extra reward upon killing several special buildings near the end of the game:

Extra Team	Weight	Awarded for
Megas	4.0	Killing last enemy barracks.
Win	2.5 [4]	Winning the game.

Lane Assignments

In addition to the above reward signals, the agent receives a special reward to encourage exploration called "lane assignments." During training, each hero is assigned a subset of the three lanes in the game. The model observes this assignment, and receives a negative reward (-0.02) if it leaves the designated lanes early in the game. This forces the model to see a variety of different lane assignments. During evaluation, all heroes' lane assignments are set to allow them to be in all lanes.

Processing

The hero's individual rewards are further processed to account for the competitive and cooperative aspects of the game in three ways:

i. Zero Sum: Each team's mean reward is subtracted from the rewards of the enemy team:

hero_rewards[i] -= mean(enemy_rewards)

This ensures that the sum of all ten rewards is zero, thereby preventing the two teams from finding positive-sum situations. It also ensures that whenever we assign reward for a signal like "killing a tower," the agent automatically receives a comparable signal for the reverse; "defending a tower that would have died."

ii. Team Spirit: At the start of training agents should be rewarded for their own actions, so they can more easily learn the correlations. However later on, their teammates' situations have to be considered by the agent, rather than greedily optimizing for their own reward. For this reason, the reward across the team's heroes is averaged using a hyperparameter τ called "team spirit":

$$hero\_rewards[i] = \tau * mean(hero\_rewards) + (1 - \tau) * hero\_rewards[i]$$

τ is annealed from 0.2 at the start of training to 0.97 at the end of the current experiment.

iii. Time Scaling: The majority of reward mass in the agent's rollout experience comes from the later part of the game. This is due to a variety of factors: the late-game portion is longer, the units have more abilities and thus get more kills and gold, etc. However, the early game can be very important; if the agent plays badly at the start it can be hard to recover. The training scheme should place sufficient importance on the early part of the game. For this reason, all rewards should be scaled up early in the game and scaled down late in the game, by multiplying all rewards by:

*hero_rewards[i] *= 0.6 ** (T/10 min)*

Note:

[1]: The agent receives reward when it *gains* gold but does not lose reward when it loses gold (e.g. by buying an item.)

[2]: Hero health is scored according to a quartic interpolation between 0 (dead) and 1 (full health) to force heavier weight when the agent is near dying.

[3]: This score supplements the score for the gold/experience gained. The explicit "Kill" score is negative to reduce the agents' reward received by a kill, but the total is still positive.

[4]: The goal of DOTA is to destroy the enemy ancient, so when this building dies the game ends. The total reward for winning is thus 10.0 (2.5 for the ancient going from alive to dead, 5.0 for the ancient losing its health, and 2.5 bonus).

IV. Recent Developments

4.1 OpenAI

OpenAI is a non-profit artificial intelligence (AI) research company that aims to promote and develop friendly AI in such a way as to benefit humanity as a whole. Founded in late 2015 (founders being notably Elon Musk and Sam Altman), the San Francisco-based organization aims to “freely collaborate” with other institutions and researchers by making its patents and research open to the public.

OpenAI is the first ever to defeat world's best players in competitive e-sports (1v1 DOTA2 bot match vs Dendi) which are vastly more complex than traditional board games like chess & Go as we have seen. Engineers from the non-profit say the bot learned enough to beat DOTA 2 pros in just two weeks of real-time learning. The bot learned the game from scratch by self-play, and does not use imitation learning or tree search. This is a step towards building AI systems which accomplish well-defined goals in messy, complicated situations involving real humans. According to a statement by OpenAI, the robot team, OpenAI Five, is able to play 180 years’ worth of games each day.

Dennison from OpenAI says that “We are focused on learning DOTA, but we are hoping that this will give us more and more insight about how AI can solve complex problems of any kind”.

For an AI to be able to generalize and learn complex strategies in a game as sophisticated as DOTA 2 is a huge achievement and will push research even closer to achieving artificial general intelligence (AGI). Aside from achieving super human levels of performance in games, RL can be applied to a broad range of fields including financial trading, natural language processing, healthcare, manufacturing and education.

V. Conclusion

RTS games can be seen as a simulation of real complex dynamic environments, in a finite and smaller world, but still complex enough to study a series of key interesting problems. Finding efficient techniques for tackling these problems on RTS games can thus benefit other AI disciplines and application domains, and also have concrete and direct applications in the ever-growing

industry of video games. Thus, research in the field of RTS game AI can help us design an AI bot capable of solving real–world decision tasks quickly and efficiently.

VI. References

1) S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss: "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft" in IEEE Transactions on Computational Intelligence and AI in Games, Vol. 5, No. 4, December 2013, pp. 293-311.
2) S. Bakkes, P. Spronck, and J. Herik: "Rapid and Reliable Adaptation of Video Game AI" in IEEE Transactions on Computational Intelligence and AI in Games, Vol. 1, No. 2, June 2009, pp. 93-104.
3) K. D. Rogers and A. A. Skabar: "A Micromanagement Task Allocation System for Real-Time Strategy Games" in IEEE Transactions on Computational Intelligence and AI in Games, Vol. 6, No. 1, March 2014, pp. 67-77.
4) I.S. Oh, H.C. Cho, K.J. Kim: Imitation Learning for Combat System in RTS Games with Application to StarCraft.
5) S. Ontan˜on, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss: "RTS AI: Problems and Techniques".
6) OpenAI Five: June 28,2018 (https://blog.openai.com/openai-five/).
7) Deep Reinforcement Learning with OpenAI Gym: April 27,2016 (https://ai.intel.com/openai).