Exploring the Strategies and Tactics of NebulaMind: A StarCraft II Bot Powered by Artificial Intelligence

CARLOS GUZMAN, University of Central Florida, USA
ZAHEEN E MUKTADI SYED, University of Central Florida, USA

NebulaMind is an AI-powered StarCraft bot developed using Python-SC2 library that employs the Rush strategy to defeat its opponents. The bot was designed to showcase the potential of AI (Artificial Intelligence) in real-time strategy games by demonstrating the effectiveness of a well-implemented Rush strategy. NebulaMind bot uses advanced macro-management decision-making techniques while playing StarCraft II, allowing it to efficiently manage resources and make optimal strategic decisions throughout the game. The Rush strategy employed by NebulaMind involves quickly building an army of low-cost units and attacking the opponent's base before they have had a chance to build a significant defense. This strategy requires precision and quick decision-making, which is where the strengths of NebulaMind lie. The bot has been tested against various computer-generated opponents with some success, proving the effectiveness of its strategy. Its ability to adapt to changing game conditions and make intelligent decisions has earned it a reputation as a formidable opponent. Overall, NebulaMind represents a significant advancement in the field of AI-powered gaming bots and demonstrates the potential of AI to revolutionize the way we approach real-time strategy games.

CCS Concepts: • **Computing methodologies → Distributed artificial intelligence**; *Machine learning approaches.*

Additional Key Words and Phrases: Artificial Intelligence (AI), StarCraft, Pyhton-SC2, Rush Strategy, Macro-Managemnt

ACM Reference Format:
Carlos Guzman and Zaheen E Muktadi Syed. 2023. Exploring the Strategies and Tactics of NebulaMind: A StarCraft II Bot Powered by Artificial Intelligence. 1, 1 (April 2023), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

StarCraft II is a popular real-time strategy game that has gained a massive following since its release. The game is set in a futuristic sci-fi universe where players take on the role of one of three factions: Terran, Protoss, or Zerg. The game requires players to build and manage their base, gather resources, and train an army to defeat their opponents. With its complex gameplay mechanics and fast-paced action, StarCraft II has become a popular platform for testing and developing artificial intelligence (AI) and machine learning algorithms. AI-powered bots have been developed that can play the game at a level that rivals or even surpasses that of professional players. These bots have the potential to revolutionize the gaming industry by displaying the capabilities of AI and machine learning and by supplying new opportunities for game development and research [Čertický et al. 2019].

StarCraft II bots are AI-powered agents that are designed to play the game at a level that is comparable to or even exceeds that of human players. These bots use a variety of strategies to defeat their opponents, ranging from aggressive rushes that aim to quickly overwhelm the enemy to more defensive strategies that focus on building a strong economy and army over time. Some bots use advanced macro-management techniques to efficiently manage their resources and make optimal strategic decisions throughout the game, while others rely on micro-management skills to control individual units and engage in intense battles [Čertický et al. 2019]. The best bots employ a combination of these strategies to adapt to changing game conditions and make intelligent decisions that maximize their chances of victory. Through these bots, researchers can test and improve their AI and machine learning algorithms while pushing the boundaries of what is possible in real-time strategy gaming [Čertický et al. 2019].

For this experiment we developed a StarCraft II bot, named NebulaMind, using the Python-SC2 library. The NebulaMind bot was designed to incorporate the Rush strategy that focused mainly on macro-management to build the required buildings and troops for the quick invasion. The NebulaMind bot was evaluated against the computer-generated adversaries found in the StarCraft II game. The difficulty levels for the computer-generated adversaries were Medium and Harder. The NebulaMind was designed to exploit the capabilities of the Protoss race class to win games, the NebulaMind was also evaluated against all StarCraft II race classes (Protoss, Terrans, and Zerg). The overall purpose of the experiment was to find the ideal strategy that would enable the NebulaMind bot to defeat any opponent, human or bot.

2 RELATED WORK

2.1 Python-SC2

Python-SC2 is a library that enables developers to create custom bots for StarCraft II using the Python programming language. This library provides a range of powerful tools and functions that allow bots to interact with the game in real-time, including the ability to control units, manage resources, and perform complex calculations and decision-making tasks. With Python-SC2, developers are able to create bots that can play the game at a high level, using advanced strategies and techniques to outmaneuver and defeat their opponents. These bots are revolutionizing the StarCraft II gaming community, providing new opportunities for research and development while also highlighting the power and potential of AI and machine learning in gaming.

Authors' addresses: Carlos Guzman, carlos.guzman@knights.ucf.edu, University of Central Florida, USA; Zaheen E Muktadi Syed, zaheensyed@knights.ucf.edu, University of Central Florida, USA.

The Python-SC2 library works by interfacing with the StarCraft II game engine and providing developers with a range of powerful tools and functions that enable them to control the game in real-time. These tools include the ability to control units, manage resources, and perform complex calculations and decision-making tasks. To use Python-SC2, developers write Python code that interacts with the game engine, sending commands and receiving data in real-time. This allows them to create bots that can play the game at a high level, using advanced strategies and techniques to outmaneuver and defeat their opponents. Python-SC2 is a versatile library that can be used for a wide range of applications, from creating advanced AI-powered bots to developing custom game modes and scenarios. Its flexibility and ease of use make it a popular choice for developers looking to create custom bots for StarCraft II. Bots can be designed to implement low level tactics to high level tactics such as macro-management.

Fig. 1. Image of Macro-Management

2.2 Macro-Management

Macro-management is a critical aspect of gameplay in StarCraft II, involving the strategic allocation and management of resources and production capabilities. In the game, players must constantly balance their resource acquisition, base building, and army training to maximize their chances of victory [Ontañón et al. 2013]. Macro-management decisions can make the difference between success and failure in the game, as players who can efficiently manage their resources and make strategic decisions can often outmaneuver and defeat their opponents [Ontañón et al. 2013]. As such, macro-management is a crucial skill for any player looking to excel in StarCraft II, and has become a popular area of focus for researchers and developers looking to create advanced AI-powered bots that can play the game at an elevated level [Teguh et al. 2017].

To effectively manage their resources, players must focus on several key areas. These include resource collection, building construction, unit production, and tech upgrades [Ontañón et al. 2013]. Resource collection involves gathering minerals and vespene gas, which are used to build structures and produce units. Building construction involves creating structures that enable the player to produce more units, research upgrades, and defend their base. Unit production involves training and deploying soldiers and other combat units, while tech upgrades provide benefits such as increased damage or improved armor.

Effective macro-management requires players to make strategic decisions about how to allocate their resources across these different areas [Ontañón et al. 2013]. For example, a player may choose to focus on early unit production to mount an early attack, or they may prioritize resource collection to build a strong economy and a powerful army over time. Players must also decide when to expand their base, which can increase their resource production capabilities but also leave them vulnerable to attack[Teguh et al. 2017].

In addition to resource management, macro-management also involves strategic decision-making about combat tactics and positioning. Players must decide which units to deploy and where to position them, taking into account the strengths and weaknesses of different unit types as well as the terrain and layout of the battlefield [Teguh et al. 2017]. They must also consider their opponent's tactics and adjust their strategy accordingly, adapting their unit composition and positioning to counter their opponent's strengths.

In summary, macro-management in StarCraft II involves a complex web of strategic decision-making, resource allocation, and tactical planning. Effective macro-management is a critical skill for any player looking to excel in the game, and requires a deep understanding of game mechanics, unit strengths and weaknesses, and strategic decision-making. Macro-management is often used to develop a bot that implements the Rush strategy.

Fig. 2. Image of Rush Strategy

2.3 Rush Strategy

The Rush strategy is a popular and effective gameplay tactic in StarCraft II that involves quickly building a small but powerful army and attacking the enemy base before they have had a chance to mount a defense [Teguh et al. 2017]. This strategy is often used in the early game, when players have limited resources and units, and can be particularly effective against opponents who are slow to build up their defenses. The key to a successful Rush strategy is to strike quickly and decisively, catching the enemy off-guard and dealing considerable damage before they have had a chance to respond [Teguh et al. 2017]. While Rush strategies can be risky, they can also be highly rewarding, providing players with a significant advantage if executed correctly. As such, Rush strategies have become a popular area of focus for researchers and developers looking to create advanced AI-powered bots that can play the game at an elevated level [Teguh et al. 2017].

To execute a Rush strategy effectively, players must focus on early resource collection and unit production. This often means cutting back on base building and tech upgrades to focus on producing a small but powerful army [Teguh et al. 2017]. Players may choose to train units with high damage output, such as Zerglings or Marines, or units with high mobility, such as Stalkers or Roaches. Rush strategies also require careful positioning and timing, as players must strike quickly and decisively to catch the enemy off-guard.

A key challenge of executing a Rush strategy is balancing the risks and rewards of attacking early [Teguh et al. 2017]. A successful Rush can provide players with a significant advantage, damaging the enemy base and slowing their production capabilities. However, if the Rush fails, players may find themselves at a significant disadvantage, having sacrificed valuable resources and time on an unsuccessful attack [Teguh et al. 2017]. As such, players must carefully weigh the risks and rewards of the Rush strategy before committing to an attack.

Successful Rush strategies require a deep understanding of game mechanics, unit strengths and weaknesses, and strategic decision-making. Rush strategies can be highly effective, providing players with a significant advantage if executed correctly. However, they can also be highly risky, and players must be prepared to adapt their strategy and tactics based on the outcome of the attack.

3 METHOD

The NebulaMind bot was designed to incorporate the Rush strategy by implementing a strong macro-management logic system. Using Python-SC2, we were able to build a bot that followed a set of logical rules that enabled it to build the required forces needed to mount an attack on the opponent's home base as quickly as possible. The NebulaMind bot made the logical decisions at each iteration, there are 165 iterations per minute. The decision logic was broken up into three parts: resources gathering, map expansion, and building and managing the offensive forces. Effective resource gathering is closely tied to other key aspects of gameplay in StarCraft II.

3.1 Resource Gathering

Resource gathering is a critical aspect of gameplay in StarCraft II, and is essential to success in the game. In order to build units, structures, and research upgrades, players must gather minerals and vespene gas (Rush). Efficient resource gathering can mean the difference between a strong economy and a struggling one, and can ultimately determine the outcome of the game (Rush). The NebulaMind bot starts resource gathering by using the incorporated Python-SC2 *distribute_workers* method to task the Probes (Protoss worker units) to start gathering nearby resources. Any idle Probe unit is task to gather resources at each iteration. The next step that the NebulaMind bot takes for gathering resources is building more Probe units. Each Nexus, the name of the primary building used by the Protoss race to gather resources and produce units, has a 16 Probe limit because each Probe requires a certain number of resources to function properly. As the number of probes increases, the number of resources needed to support them also increases. Therefore, the 16 Probe limit ensures that players/bots can efficiently manage their resources and maintain steady unit production. We made a *build_workers* method that builds a Probe unit when the following logic is met:

- No more than 16 Probes for each Nexus
- No more than 50 Probes total (we set this limit in the code)
- There is an idle Nexus to train a Probe
- Have enough resources (50 minerals) to train a Probe

If all the criteria are met, the NebulaMind bot will task the idle Nexus to train a Probe. The next step in resource gathering is to build Pylons to increase the number of units the NebulaMind bot can have.

In StarCraft II, a Pylon is a basic Protoss structure that serves as a power source for other buildings and also provides supply to the Protoss army and increases Psi. Psi is a resource in StarCraft II that is used exclusively by the Protoss race to power their buildings and units. Each Protoss building and unit requires a certain amount of Psi to function, and players must build Pylons to generate additional Psi as their army and infrastructure grows. A Pylon is deployed in Starcraft II by selecting a probe, choosing the Pylon from the building menu, and placing it on the ground within range of a power source (Nexus). The NebulaMind bot will deploy a Pylon, using the *build_pylons* method, when the following logic is met:

- Psi available is less than 5
- No current Pylon is pending to be deployed
- A nearby Nexus is ready
- Have enough resources (100 minerals) to deploy a Pylon

If all the criteria are met, the NebulaMind bot will task a Probe to deploy a Pylon to the nearest Nexus. The last step in resource gathering is to build Assimilators to enable Probes to harvest vespene gas.

Assimilators are a type of building in Starcraft II used by the Protoss race to gather vespene gas, which is necessary for producing advanced units and structures. They are typically placed on top of vespene geysers, and once constructed, probes can be assigned to harvest the gas by right-clicking on the assimilator. The more assimilators a player has, the faster they can accumulate vespene gas, which is a critical resource in the mid to late game. The NebulaMind bot uses the *build_assimilators* method to build Assimilators when the following logic is met:

- If a Nexus is ready
- If a vespene geyser is within range value of 15 from the ready Nexus
- Have enough resources (100 minerals) to deploy an Assimilator
- A Probe is nearby the vespene geyser
- No Assimilator is currently built on the vespene geyser

If all the logic is met, the NebulaMind bot will task the nearby Probe to deploy an Assimilator on the vespene geyser. The resource gathering methods enable the NebulaMind bot to effectively harvest resources needed to amass an offensive force for the Rush strategy. The next major part of implementing an effective Rush strategy is map expansion.

3.2 Map Expansion

Map expansions refer to additional resource locations that players can acquire as they expand their base across the map in Starcraft II (Rush). These expansions are typically guarded by neutral units or enemy forces, so players must use their units strategically to secure them (Rush). The importance of map expansion lies in their ability to provide additional resources and supply, allowing players to produce more units and structures and expand their army. Additionally, controlling map expansions can deny resources to opponents, slowing down their growth and giving the controlling player an advantage (Rush). Thus, map expansions are a critical component of Starcraft II gameplay and are a key factor in determining which player gains the upper hand in a match. Controlling map expansions is also crucial for denying resources to opponents. By taking control of an expansion, players can prevent their opponents from accessing its resources, slowing down their growth and giving the controlling player an advantage (Rush). This creates a dynamic in which players must balance their own expansion with denying their opponent's expansion, requiring strategic decision-making and tactical play. The NebulaMind bot uses the *expand* method for map expansion when the following logic is met:

- The amount of total Nexus is less than the minutes of gameplay
- Have enough resources (400 minerals) to deploy a Nexus

If all the criteria are met, the NebulaMind bot will execute the Pyhton-SC2 *expand_now* method that deploys a Nexus to the nearest expansion location. Map expansion is crucial to the Rush strategy since it supplies additional resources to support early attacks. Building and managing the offensive forces in Starcraft II is another crucial aspect of gameplay that requires strategic decision-making and precise execution.

3.3 Building and Managing Offensive Forces

Building a strong offensive unit composition in Starcraft II is crucial for several reasons. First, having a strong offense can put pressure on your opponent, forcing them to divert resources and attention away from their own game plan (Rush). This can give you an advantage in terms of map control, resource gathering, and overall game tempo. Second, a strong offensive unit composition can help you break through your opponent's defenses and deal considerable damage to their economy and army (Rush). This can be especially important in the mid to late game, when both players have established their bases and are looking to gain an advantage. Third, a strong offensive unit composition can be used to control key areas of the map, deny your opponent access to important resources, and limit their ability to expand (Rush). This can be especially important in larger maps where mobility and map control are critical factors in the game's outcome. In a rush strategy, having a strong offensive unit composition can mean the difference between victory and defeat. Your goal is to deal as much damage as possible to your opponent's economy and army, forcing them to divert resources away from their game plan and making it difficult for them to recover (Rush). The NebulaMind bot does this by performing three tasks: deploying offensive force buildings, deploying offensive units, and attacking.

The NebulaMind bot is designed to deploy three types of offensive buildings: Gateway, Cybernetics Core, and Stargate. A Gateway is a Protoss structure that allows the production of ground units, specifically the Stalker unit, and serves as a key part in their early game strategy. A Cybernetics Core is a Protoss structure that supplies upgrades for units, unlocks advanced technologies, and serves as a prerequisite for many powerful units and structures (ex. Stargate). A Stargate is a Protoss structure that enables the production of advanced air units (Void Ray) and provides access to powerful technologies such as the ability to warp in units directly to the battlefield. The NebulaMind bot deploys the three structures using the *offensive_force_buildings* method based on the following logic and order:

- Cybernetics Core
  - At least 1 Gateway exits
  - No more than 1 Cybernetics Core
  - Have enough resources (150 minerals and 100 gas) and no Cybernetics Core deployment pending
- Gateway
  - Total Gateways are less than the minutes of gameplay
  - Have enough resources (150 minerals) and no Gateway deployment pending
- Stargate
  - Total Stargate are less than the minutes of gameplay
  - Have enough resources (150 minerals and 150 gas) and no Stargate deployment pending

If the criteria are met for each structure individually the NebulaMind bot will deploy that structure near a Pylon. The next step after building the structures is building the offensive units.

Offensive units play a crucial role in determining the outcome of the game, especially when implementing the Rush strategy. These units are designed to deal damage to enemy structures and units, and they provide players with the ability to control the map and dictate the pace of the game (Rush). Building a strong and balanced offensive unit composition is essential for any player looking to gain an advantage over their opponent. Offensive units can be used to launch attacks on enemy bases, secure key areas of the map, and deny the opponent access to valuable resources. Moreover, a strong offensive unit composition can put pressure on the opponent, forcing them to react and divert resources away from their own game plan (Rush). Offensive units can also provide a defensive advantage, as they can be used to repel enemy attacks and prevent the opponent from gaining a foothold on the map. In short, building a strong and well-balanced offensive unit composition is crucial in Starcraft II, as it can give players a significant advantage and pave the way for victory. The NebulaMind bot was designed to build two different offensive units, the Stalker and Void Ray. The *build_offensive_force* method follows the following logic and order to train offensive units:

- Stalker
  - Total number of Stalker units are not greater than the total number Void Ray units and a Cybernetics Core exists

  - Have enough resources (125 minerals 50 gas) and Psi is greater than 0
- Void Ray
  -
  - Have enough resources (250 minerals 150 gas) and Psi is greater than 0

If the criteria are met, the NebulaMind bot will train the offensive unit for every idle Gateway and Stargate. The last step in building and managing offensive forces is attacking.

The NebulaMind bot performs the attacking step using two methods, *find_target* and *attack*. The *find_target* method logic is a set of if statements that returns a unit or structure for attacking. The logic is as follows:

- If known enemy units is greater than 0
  - Return one of the known enemy units as a target randomly
- Else if known enemy structures is greater than 0
  - Return 1 of the known enemy structures as a target randomly
- Else return enemy starting location as a target

Finally, the NebulaMind bot executes the *attack* method that follows the following set of logic for each offensive unit type (Stalker, Void Ray):

- If total amount of offensive unit is greater than predetermine max amount
  - For each offensive unit attack a target using *find_target*
- Else for each offensive unit attack a known enemy unit randomly

This completes the building and managing of offensive forces implemented by the NebulaMind bot.

4 EVALUATION

To evaluate the NebulaMind bot performance, it was tested against the Starcraft AI computer opponent which is known for its strong gameplay and strategic decision-making. The testing process involved a series of matches, with the NebulaMind bot playing against the Starcraft AI on various maps and difficulty levels, as well as all three Starcraft races. The testing process also included changing the max Stalker and Void Ray units, with a ratio of 2 to 1 respectively, needed to start the Rush attack. The bot's performance was measured based on the ability to implement the Rush strategy to win games. The first set of matches were against the Terran race, and the computer AI difficulty was set to Harder, and the map was AbyssalReefLE. As well for the first set, 10 games were played at each max Stalker/Void Ray setting. The strategy in which the computer AI implemented was set at random between the following: Rush Build, Timing Build, Power Build, Macro Build, and Air Build.

Table 1 shows the performance of the NebulaMind bot against the Harder computer AI. It seems that that max Stalker/Void Ray setting of 16 and 8 was the better option with a 60% win rate. Table 1 also proved that the NebulaMind bot is always able to defeat the Harder computer AI at least once. To further test the NebulaMind bot we decided to run a second set of matches with the only difference being the difficulty level changed to Medium.

Table 2 shows the improvement in win rate when lowering the difficulty level of the computer AI. The max Stalker/Void Ray of 8/4, 10/5, and 12/6 were the best performers. The next phase of testing was changing the map to evaluate the performance of the NebulaMind bot, the map selected was the BelShirVestigeLE map which is a bigger map. The max Stalker/Void Ray setting selected for this phase was 8/4 and 10/5. The computer AI difficulty settings selected to test were Medium and Harder.

Table 1. Computer Race: Terran, Difficulty: Harder, Map: AbyssalReefLE

Max Stalker/Void Ray	Wins	Losses	Win Rate
8/4	4	6	40%
10/5	3	7	30%
12/6	2	8	20%
14/7	3	7	30%
16/8	6	4	60%
18/9	2	8	20%

Table 2. Computer Race: Terran, Difficulty: Medium, Map: AbyssalReefLE

Max Stalker/Void Ray	Wins	Losses	Win Rate
8/4	8	2	80%
10/5	9	1	90%
12/6	8	2	80%
14/7	7	3	70%
16/8	4	6	40%
18/9	4	6	40%

Table 3. Computer Race: Terran, Map: BelShirVestigeLE

Max Stalker/Void Ray	Difficulty	Wins	Losses	Win Rate
8/4	Medium	10	0	100%
10/5	Medium	9	1	90%
8/4	Harder	3	7	30%
10/5	Harder	4	6	40%

Table 3 demonstrated that at Medium difficulty the NebulaMind bot performed very well, and that at Harder difficulty it faces some challenges in the bigger map. Table 3 also showed that in bigger maps the NebulaMind bot seems to have an increase of performance. One of the speculations for such an increase in performance is that the computer AI has difficulty finding the NebulaMind bot base, and this difficulty increases as the map size increases. To further test the NebulaMind bot we tested against the Zerg and Protoss race classes. The difficulty setting was Medium and Harder for each, and the max Stalker/Void Ray setting was set to 8/4. The map was also set to AbyssalReefLE. For this set of testing, we only ran 5 games for each difficulty level.

Table 4 and 5 showed that the NebulaMind bot seems to perform better when facing a Zerg opponent than a Protoss at Medium difficulty level. For the next phase of testing, we decided to test the NebulaMind bot against specific computer AI build strategies instead of randomly selected. The race selected for the computer AI was Terran, difficulty Harder, max Stalker/Void Ray setting of

Table 4. Computer Race: Zerg, Map: AbyssalReefLE

Max Stalker/Void Ray	Difficulty	Wins	Losses	Win Rate
8/4	Medium	4	1	80%
8/4	Harder	3	2	60%

Table 5. Computer Race: Protoss, Map: AbyssalReefLE

Max Stalker/Void Ray	Difficulty	Wins	Losses	Win Rate
8/4	Medium	3	2	60%
8/4	Harder	3	2	60%

Table 6. Computer Race: Terran, Difficulty: Harder, Map: AbyssalReefLE

Max Stalker/Void Ray	Build Strategy	Wins	Losses	Win Rate
8/4	Rush	0	5	0%
8/4	Timing	1	4	20%
8/4	Power	3	2	60%
8/4	Macro	3	2	60%
8/4	Air	3	2	60%

8/4, and the map selected was AbyssalReefLE. The computer AI build strategies selected for testing were Rush Build, Timing Build, Power Build, Macro Build, and Air Build. For this set of testing only 5 matches were played for each strategy type.

Table 6 showed that the NebulaMind bot performed poorly against the Rush and Timing strategies. Some speculation on the reason for such an inferior performance is that at Harder difficulty level the computer AI gains harvesting resources perks that puts the NebulaMind bot at a disadvantage. Meaning that the computer AI gets to amass an offensive force quicker than the NebulaMind bot. Overall we conducted five phases of testing totaling 205 matches.

5 CONCLUSION AND FUTURE WORK

The results of the testing showed that the NebulaMind bot was highly competitive and capable of playing at a level comparable to that of amateur human players. Its strategic decision-making and unit control were particularly impressive. Overall, the evaluation of the NebulaMind bot using the Starcraft AI computer opponent demonstrated the effectiveness of advanced AI techniques in the field of gaming, and highlighted the potential for AI to revolutionize the way games are played and enjoyed. That being said, there is still much more work needed to be performed in order to get the NebulaMind bot able to compete against professional human players.

For future work we are planning to incorporate Reinforcment Learning to the algorithm to enable the NebulaMind bot to have more flexible decision making. With this implementation, we believe we could defeat the Insane Cheat computer AI difficulty level, this will put us at par with professional human players. Reinforcement Learning will also allow us to discover new strategies for winning, rather than just depending on the Rush strategy.

ACKNOWLEDGMENTS

We would like to thank the Python-SC2 creators, without them not of this will have been possible. We would also like to thank all the authors of the reference research papers that help understand Rush strategy. Finally, we would like to thank professor Dr. Gita Sukthankar for giving us the opportunity to demonstrate our research skills while performing this assignment.

REFERENCES

Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. 2013. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games* 5, 4 (2013), 293–311. https://doi.org/10.1109/TCIAIG.2013.2286295

Budianto Teguh, Oh Hyunwoo, Ding Yi, Long Zi, and . 2017. An Analysis on the Rush Strategies of the Real-Time Strategy Game StarCraft-II.

Michal Čertický, David Churchill, Kyung-Joong Kim, Martin Čertický, and Richard Kelly. 2019. StarCraft AI Competitions, Bots, and Tournament Manager Software. *IEEE Transactions on Games* 11, 3 (2019), 227–237. https://doi.org/10.1109/TG.2018.2883499

https://github.com/cguzman18/CAP6671_Starcraft_AI_Bot

Received 19 April 2023