

Computer Games Development SE607

Technical Design Document

Year IV

[Leo Chen]

[C00250593]

[26/04/2023]

**DECLARATION**

﻿

**Work submitted for assessment which does not include this declaration will not be assessed.**

* I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
* I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
* I have provided a complete bibliography of all works and sources used in the preparation of this submission.
* I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Leo Chen

Student Number(s): C00250593

Signature(s): Leo Chen

Date: 26/04/2023

--------------------------------------------------------------------------------------------------------

**Please note:**

1. \*Individual declaration is required by each student for joint projects.
2. Where projects are submitted electronically, students are required to submit their student number.
3. The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.

**Contents**

[Acknowledgements 2](#_Toc133346067)

[Project Abstract 2](#_Toc133346068)

[Project Introduction 2](#_Toc133346069)

[Literature Review 3](#_Toc133346070)

[Artificial Intelligence and Learning Algorithms in video games 3](#_Toc133346071)

[Real Time Strategy Games 3](#_Toc133346072)

[StarCraft 2’s learning environment 3](#_Toc133346073)

[Mirco-RTS-Py 4](#_Toc133346074)

[Project Overview 4](#_Toc133346075)

[A new game environment 4](#_Toc133346076)

[Neural Networks and Unity’s Machine Learning Agent Toolkit 4](#_Toc133346077)

[Project Evaluation and Discussion 6](#_Toc133346078)

[Major Technical Achievements 6](#_Toc133346079)

[Project Milestones 6](#_Toc133346080)

[Conclusions 8](#_Toc133346081)

[Future Work 8](#_Toc133346082)

[References 9](#_Toc133346083)

# Acknowledgements

I would like to thank the people throughout my time here who helped me to gain the skills and knowledge for this project.

Lei Shi, my supervisor for giving me advice and insights on the project as a whole. My colleagues for their ideas, feedback and input as well as believing in me and my project. All the lecturers who aided me in reaching to the end of my studies at SETU.

# Project Abstract

The Real Time Strategy Game Genre has provided a concrete and effective method to allow researchers to introduce and study complex algorithms that adapt to its evolving landscape. The popular RTS game StarCraft (SC) and its sequel StarCraft II (SC2) has been utilised as a basis for developing Machine Learning algorithms to further the research on Artificial Intelligence. Previous researchers have constructed a framework to build from that allows others to create and expand their own algorithms as they see fit with great success. The attempt for this project focuses on creating an entirely new, smaller scale RTS game that uses a machine learning algorithm as a core concept for an opponent. The idea is to create a basic environment that allows the algorithm to achieve simple goals that cumulate to the advancement of a much larger and complex objective through the mechanics of the game. In the end the machine learning algorithm would be able to adapt to a human opponent more readily.

# Project Introduction

With the advancement of artificial intelligence, most efforts have been placed on training neural networks to recognise images, understanding and reinterpreting data to provide an output. The purpose of this project was to create a simple but expandable environment with an entirely new game from the ground up. This will allow me to experiment with the algorithm it is present in and create new challenges for it to overcome.

Developed using the Unity engine for the foundation of this game, the platform will be the test bed for Unity’s Machine Learning Agents (MLA) (Unity ML-Agents Toolkit 2023), other neural networks or non-machine learning AIs to work as the opponent. The environment will consist of two main objectives, Control points and Resource points. One will generate score while the other generates funding for their respective player when captured. These areas will be the main points of contention the players will fight over.

The simplistic environment at first will allow for the MLA to get familiar with the controls and actions they will be utilising. Upon performing such actions to an acceptable level, the environment will expand to include more and more complex game mechanics for the MLA to familiarise and act on.

This project will also explore the use of other AIs as a potential opponent for the MLA. Its use as the MLA’s opponent will be used to gauge the effectiveness of what the MLA has learned and compare it to traditionally programmed AIs. Such traditionally made AI’s for comparison would be Finite State Machines, behaviour trees or decision trees.

The aim of this project is a means to demonstrate the potential use of Unity’s ML-Agent’s toolkit for creating adaptive AI opponents in RTS video games. By developing the MLA to evolve into an engaging and challenging opponent, this example could provide a good foundation for the expansion of utilising MLA for their own games in the industry.

# Literature Review

## Artificial Intelligence and Learning Algorithms in video games

The implementation of learning algorithms for expanding the research of artificial intelligence has woven its way into video games. With the cooperation between then Blizzard entertainment and Deepmind, they have developed the SC2LE (StarCraft 2 Learning Environment)(Vinyals et al. 2017), a program that introduces a Reinforced Learning Algorithm into the complex and chaotic game. Though initially created in controlled “mini-game” environments, it has allowed other researchers to expand the scope of the algorithm to encompass the entire game. Different researchers have created AI Agents to compete against StarCraft 2’s built in cheat level built in AIs with great success.

### Real Time Strategy Games

Real time strategy games(What Are Real-Time Strategy Games? 2022) are a genre of video games focused on the commanding of large armies and managing resources to fund said army. Dating back to 1992, the grandfather of all real time strategy games Dune 2 (*Making of Dune II* 2023) paved way for all other RTS games to evolve from. Most if not all features from Dune 2 such as building construction, resource harvesting and unit production. It wasn’t until 1995 where the launch of Command and Conquer (Command and Conquer History n.d.) launched the RTS genre into the limelight in the gaming scene.

For nearly over a decade, its popularity had generated a plethora of games in its field. Many of the games have their roots (Staff 2017) in Westwood’s Command and Conquer game. And thus, many innovations in artificial intelligence sprung from video games (Zhen et al. 2018) as it is a unique environment allowing researchers and innovators to create various kinds of artificial intelligence not limited to just the real time strategy game genre.

The genre itself usually revolves around managing the uses of tens to thousands of units to defeat their opponent in a battlefield and to achieve a goal. Whether it is to hold key locations or total annihilation of their foes, these types of underlying element are present in a multitude of RTS games.

With the advent of machine learning, video games and especially real time strategy games were uniquely suited as test beds for training machine learning algorithms (Skinner and Walmsley 2019). Due to the nature of video games themselves being complex applications of software that require users to manipulate a set of outlined controls to affect it, this field has quite the potential to train artificial intelligences.

### StarCraft 2’s learning environment

Though most have mainly focused on SC2LE(Vinyals et al. 2017), a different project was developed to create a smaller scale RTS that featured a machine learning algorithm. Micro RTS (Huang et al. 2021), developed by Santiago Ontañón, is a small implementation of a real time strategy game which is used to observe and perform AI research. It involves two players controlling and managing small armies of units in which their goal is to destroy their opponent’s base. The implementation of the game is simple enough for algorithms to run out but the capacity for more advanced strategy with more complex unit types are not currently present.

StarCraft 2’s environment and gameplay revolves around much higher and intensive logical computation. The SC2LE was developed by Deepmind (DeepMind n.d.) in partnership with (at the time) Blizzard Entertainment (Blizzard Entertainment n.d.). The platform provides a complex and challenging arena for the AI agents to learn and adapt to its ever-shifting environment. The game itself involves several key aspects of that players need to manage. Players and by extension, the AI will need to manage resources, unit production, micro[[1]](#footnote-1) and macro[[2]](#footnote-2) gameplay, on top of a rapidly evolving environment. Both tactics and strategy are paramount to success in the game.

In the environment itself, it contains several mini-scenarios for the agent to learn from (Vinyals et al. 2017, p.7). Using units from the base game, the development team presented the AI with simple challenges that players would conduct naturally. From selecting and moving units to having them harvest resources or preforming the construction buildings. Though rather simple in conception, this paved the way forward for more advanced forms of machine learning as the environment became a test bed for new researchers.

#### TStarBots

A study by Tencent AI Labs, University of Rochester USA and Northwest University USA developed the TStarBots(Sun et al. 2018) which are two machine learning agents designed to defeat the built-in Cheating AI in StarCraft 2. The first AI known as TStarBot1 is a Macro Action Based Reinforcement Learning Agent (Sun et al. 2018, p.6)developed to act as a single global controller with hard coded macro actions available to it. This multi-layer perception neural network architecture used for the study allows TStarBot1 to learn and act on situations present within an active game by utilising the macros available to it. The second AI known as TStarBot2 is a Hierarchical Macro-Micro Action Based(Sun et al. 2018, p.19) Agent designed for both macro and micro actions. This version can, as the description implies, the ability to command both the Macro level of thinking as well as the Micro section of per unit commands. Both of which has seen success in completing a full match/level/game of StarCraft2 as well as being able to defeat a high ranking player(Sun et al. 2018, p.19).

### Mirco-RTS-Py

Micro-RTS-Py (Huang et al. 2021) by the Farama-Foundation(The Farama Foundation n.d.) is another example for machine learning algorithms in the RTS Genre. It is a platform specifically designed for testing machine learning algorithms in an RTS inspired game environment which is the inspiration for the premises of this project. Unlike StarCraft 2’s more heavily complex game with several times the magnitude of computational need for training neural networks, Micro-RTS has strived for a highly efficient and affordable test bed for machine learning algorithms. Its environment can be used to train agents more quickly relative to the time it would take the likes of TStarBot (Sun et al. 2018) or other machine learning algorithms in more complex environments to train. This test bed allows for researchers to test their machine learning algorithms to compete against other AIs in its field.

In the environment Micro-RTS provides, its 16x16 grid with two corners in which the players’ base is located. The unit and building roster consist of basic entities such as workers and different kinds of soldiers that make up the RTS environment as well as simple production and resource nodes. The simulation of the inputs for the algorithm consists of two main components that allow for the neural networks to act on. The observation and action space utilised features the observation space to generate the data seen on the grid while the action space as the name implies, provides the actions for the algorithm to utilise(Huang et al. 2021, p.3).

Concluding the studies found for machine learning algorithms in real time strategy games provide the potential for producing highly sophisticated game AIs. By utilising various methods of machine learning, an AI agent can create a challenging and engaging opponent for even the most skilled players of the genre. As artificial intelligence evolves over time, it is possible that machine learning algorithms as opponents in video games be a potential avenue for exploration.

# Project Overview

## A new game environment

The reason I wanted a new environment for the AI to train in is so that the I can expand upon the potential actions and expand the unit roster. Starting off small with simple infantry units for the MLA get familiar with. The environment planned was, as detailed as before in the introduction, with 2 main areas of contention for the MLA to fight over. Control Points and Resource Points. One generating score while the other generated funding for both sides. Additionally, for both players, there would be spawn points located on the opposite corners of each other where their respective units will be spawned.

There were three planned units for the players in the game to command, all with weaknesses and strengths to them. Normal infantry armed with firearms, grenadiers with grenades and lastly melee units with shields. Each unit would vary in utility on what the player/MLA could use. Formations are meant to be available as well, with the player/MLA being able to select which formation to use to gain a tactical advantage.

## Neural Networks and Unity’s Machine Learning Agent Toolkit

With the focus of artificial intelligence for the game, Unity’s ML-Agent toolkit was tested briefly on its effectiveness for use in the project. In addition to the toolkit, basic neural network architecture was also looked into for its viability in Unity’s scripting scene. For the toolkit and neural network, it would need to manage controls tailored from how a player would use them. Unit selection, camera movement and unit creation would need to be broken down into their base components to feed into the ML-Agent or neural network.

For the use of a neural network architecture, there needs to have a multitude of inputs for it to be able to process data necessary for it to make use of the actions.

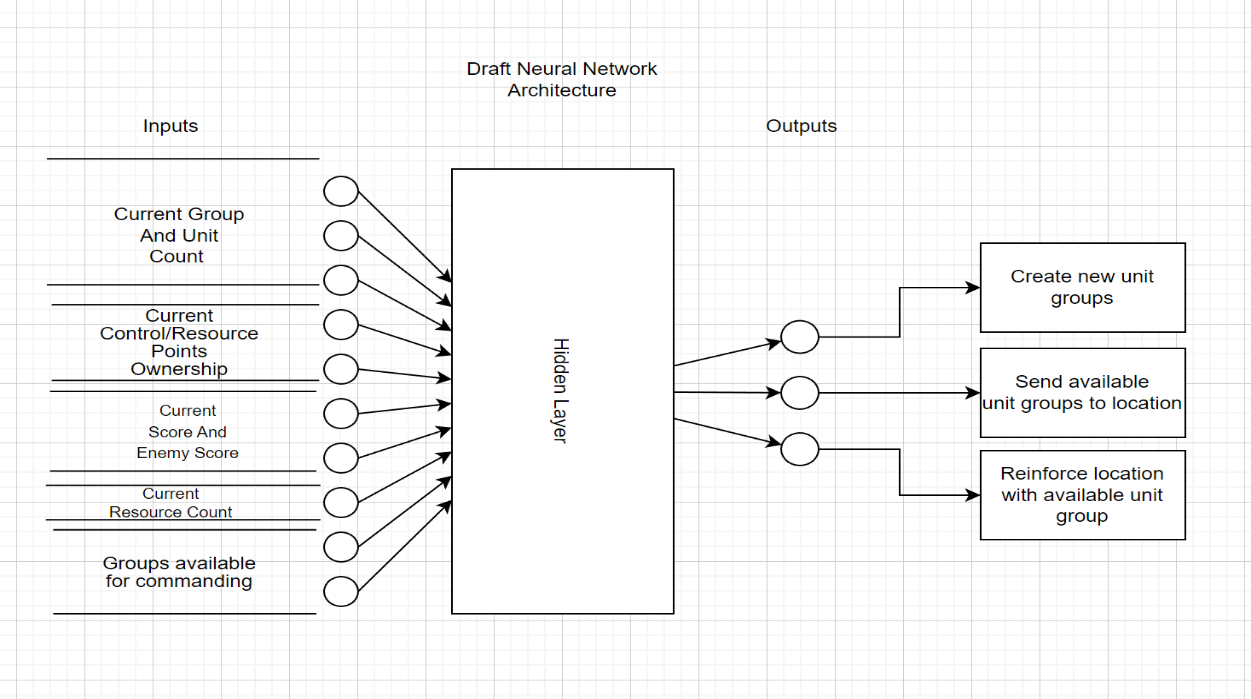


Fig.1 Proposed neural network inputs and outputs

Through the use of several key inputs, the neural network is meant to take what is outlined in figure 1 above to decide on one of three predefined actions to take. The reward for completing the action requires it to be evaluated either after the test runs or during gameplay. Unlike most neural networks’ reward functions, in an RTS, the true reward isn’t properly accounted for until said action affects the environment in the game. Say that the machine learning agent takes the “create new unit groups” output. That in it of itself is one step to take in the grand scheme of the game which will be affecting the next decision. The next logical step would be to send said group of units to a location. Whether that would be an existing capture point that requires reinforcing or to take a neutral/opposing point is up to the network. The true reward is the outcome of the action issued. Has the group of units sent to said location succeed? If it did succeed then a reward would be granted to it. If not, then a punishment will be issued.

Furthermore, the reward and punishments will need to be evaluated by the neural network on why it failed or succeeded. Potentially having a predefined list of reasons for success and failure would alter the next output. This mechanic would have the neural network modify the actions it would take in order to gain a successful reward. Potentially, the algorithm could rally more units to a group or send more than one group to a location.

Unity’s Machine Learning Agents toolkit is a similar process to how the neural network architecture would function. In the ML-Agent toolkit, it requires observations similarly outlined in figure 1. For the machine learning agent’s actions, it required the observations to be broken into several small pieces of information for the agent to process in its observation function. Outlined in the technical design document, the observations would have included the current team affiliation for control points, current score/resource count, unit group positions etc.

Training the agent would require an imitation of the player’s actions in order for it to mimic them. But in order to mimic their actions, the player’s controls would need to be modified in a way that would allow the agent to function. One supposed solution would be to have the agent use a circular collider that would seek for units in the spawn point and order them to move towards a capture point.

## Project Evaluation and Discussion

For the time I had developing the project, there was no tutorial for the player to use in order to playtest the game. A notice I had taken into consideration and noted down but due the more pressing matters at hand, it was neglected in favour of more pressing features.

The controls themselves need a bit more polish in my own opinion. For the purposes of testing the game, it served well but there was always a slight feeling of inadequacy related to them. It required just a bit more polish in order for it to be more responsive.

There was only one of the three planned units in the game but unfortunately even its implementation wasn’t up to par. Though they preformed their role as outlined in the design document, they required more polish in order to increase performance and general feel to them.

For the machine learning aspect of the project, the two largest hurdles for the game was finding the correct parameters for it to utilize and mapping the controls for the agent. As previously mentioned, I never got to the point of being able to implement the network properly. Even during my research, finding the correct parameters for an entirely new game is difficult. Though reading through the documentation on StarCraft 2’s learning environment (Vinyals et al. 2017) gave some hint at what I needed, the rest of the work required me to experiment with what was present in the game if I had the time.

## Major Technical Achievements

Throughout the time I had for implementing the parts of the game. Most of the effort was placed on the core aspects of an RTS itself. Unit selection and pathfinding was arguably the most trouble some of tasks. As Unity’s built in nav-mesh system wasn’t made for 2D games. An external package called nav-mesh plus was needed in order to make it work on a 2D plane but the pathfinding had to be modified to work the way I wanted it to. The nav-mesh agent had to be rewritten following what I had learned in our course based on the book Artificial Intelligence for Games (Millington n.d.).

Unit selection and grouping was another major aspect of an RTS that was required for the use of AI. Following a tutorial on it yielded some of the results I needed but further modification was required in order for it to work with what I had already in game. Though some work was put into the Units themselves, more pressing matters for other features were needed leaving this area to be one of the weakest parts of this project.

Unfortunately, due to time constraints, implementing the machine learning algorithm for the game required more time then was given. Upon further research of the topic, most neural network programming was in python which I was unfamiliar with. Finding the necessary documentation online yielded mostly outdated information due to the nature of artificial intelligence research outpacing the guides and documentation from previous years. This led me to resort to studying neural networks more thoroughly on my own with the attempt at trying to implement one from outdated guides as a reference.

As for an RTS, most documentation and research were centered around the game StarCraft which already had researched into the application of machine learning algorithms for playing video games. Much of the documentation that were used became more of a source of information on how they concluded the application of the neural network. As such, my work had to be focused on creating it entirely from scratch while referring to their utilization as a reference. In addition to the already major task this involved, the game itself required another pass on optimization to allow for training the ML-Agent or neural network.

## Project Milestones

There were a few concrete milestones for this project through its development. The main four were the RTS controls, unit behaviours, objectives for the players to fight over and lastly the AI.

For the RTS controls, most work was focused on the selection of units and their groupings. Achieving this milestone was the first to be completed, although taking longer than I had hoped. As proper development began after the Christmas break, time was of the essence and basic controls were needed to be created before any other feature could be worked on.

Unit behaviours and overall how they felt was the biggest hurdle that needed to be looked at next. With regard to how Unity’s nav-mesh doesn’t work in 2D, an external package was needed to be added in order for development on how units worked in the game. Formations and grouping were required for the use of the AI to limit its need to command every individual unit. Regrettably the implementation of such wasn’t up to a standard I was satisfactory with in addition due to the time constraints I had.

The two main objectives for players to contest over were implemented to a satisfactory degree. The ownership of the control points can be challenged by either opponent. Upon capture the control points would generate their specified resource.

The last thing that was never implemented was the machine learning AI. As previously stated, there were time constraints when trying to implement the AI. Much of the code that currently remains in the repository is test code used to see how one would implement the machine learning element into the game.

# Conclusions

Through the 7 months of working on this project, there were many mountains that were necessary to overcome before a solid threshold had been reached. That threshold was never reached as it led to many problems both in planning and during the development of the ML-RTS. The scope of the project was too vague in the beginning to create a concrete foundation for a suitable test bed for the Machine Learning environment. It wasn’t until after the Christmas break where a more cohesive idea was formed.

Though at times, some goals were reached, I wouldn’t have considered them to be at a satisfactory level and to a level acceptable to my own standards. Much of the game is still left unfinished, with no finished gameplay loop or even the machine learning agent in the game. The Unity game engine itself was sufficient for this project, heavy optimisation was needed to increase performance and potentially reduce training time.

# Future Work

Future work on this project will need to address the underlying issues still present within the project. A refactor of the unit’s code for processing pathfinding as well as a look into mapping the actions the ML-Agent could take. Input for what the ML-Agent or neural network sees requires breaking down the controls and mappings of what a human player could see and use.

If I was to remake this game again from scratch, I would have made a more detailed plan of action in terms of what needs to be added and when. In addition to this, more feedback from peers, colleagues and my supervisor would have helped more as well as getting them to playtest it.

In addition to a complete refactor of the codebase, I would like to have implemented the other unit types outlined in the Game Design Document. Having a larger variety of units to work with allows for both the player and AI to experiment with a combination of tactics more easily rather than solely relying on the one-unit type currently available to the players.

Based on more traditional RTS games, I would like to have included resource collection, base building and other key aspects from RTS games in the past.

Until a time is available for taking up this project once more, I will need to leave it as is.

# References

*Blizzard Entertainment*. Available from: https://blizzard.com [accessed 26 April 2023].

*Command and Conquer History*. Available from: https://www.britannica.com/topic/Command-and-Conquer [accessed 25 April 2023].

*DeepMind*. Available from: https://www.deepmind.com/ [accessed 26 April 2023].

Huang, S., Ontañón, S., Bamford, C. and Grela, L. (2021). Gym-$\mu$RTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning. , 28 July 2021. Available from: http://arxiv.org/abs/2105.13807 [accessed 24 April 2023].

*Making of Dune II*. (2023). *Wikipedia* [online], 13 February 2023. Available from: https://en.wikipedia.org/w/index.php?title=Dune\_II&oldid=1139061495 [accessed 25 April 2023].

Millington, I. *AI for Games, Third Edition, 3rd Edition [Book]* [online]. Available from: https://www.oreilly.com/library/view/ai-for-games/9781351053280/ [accessed 25 April 2023].

Skinner, G. and Walmsley, T. (2019). Artificial Intelligence and Deep Learning in Video Games A Brief Review. In: *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*. 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS). pp.404–408.

Staff, A. (2017). *Build, gather, brawl, repeat: The history of real-time strategy games* [online]. *Ars Technica* [online]. Available from: https://arstechnica.com/gaming/2017/09/build-gather-brawl-repeat-the-history-of-real-time-strategy-games/ [accessed 25 April 2023].

Sun, P., Sun, X., Han, L., Xiong, J., Wang, Q., Li, B., Zheng, Y., Liu, J., Liu, Y., Liu, H. and Zhang, T. (2018). TStarBots: Defeating the Cheating Level Builtin AI in StarCraft II in the Full Game. , 27 December 2018. Available from: http://arxiv.org/abs/1809.07193 [accessed 24 April 2023].

*The Farama Foundation*. *The Farama Foundation* [online]. Available from: https://farama.org/ [accessed 26 April 2023].

*Unity ML-Agents Toolkit*. (2023). , 25 April 2023. Available from: https://github.com/Unity-Technologies/ml-agents [accessed 25 April 2023].

Vinyals, O. et al. (2017). StarCraft II: A New Challenge for Reinforcement Learning. , 16 August 2017. Available from: http://arxiv.org/abs/1708.04782 [accessed 24 April 2023].

*What Are Real-Time Strategy Games?*. (2022). *MUO* [online]. Available from: https://www.makeuseof.com/what-are-real-time-strategy-games-rts-games/ [accessed 25 April 2023].

Zhen, Y., Wanpeng, Z. and Hongfu, L. (2018). Artificial Intelligence Techniques on Real-time Strategy Games. In: *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*. CSAI ’18. New York, NY, USA: Association for Computing Machinery, pp.11–21. Available from: https://doi.org/10.1145/3297156.3297188 [accessed 25 April 2023].

1. Micro: Controlling of units in skirmishing actions and individual battles. [↑](#footnote-ref-1)
2. Macro: The controlling of the strategic layer or “big picture” of the real time strategy game that involves managing resources, global troop movement/placement and general stratagem. [↑](#footnote-ref-2)