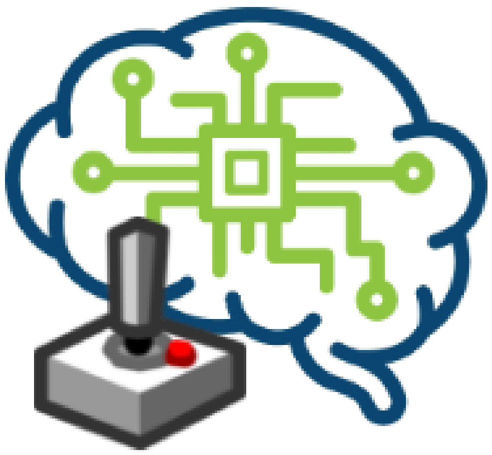
Controlling Playable Characters with AI

Final Deliverable



Name : Saarah Huda Wasif Naheel

ID : H00232155

Supervisor : Dr. Hani Ragab

Program : BSc. Computer Science (Hons) - 4th Year

Date : April 23, 2019

Declaration

I, Saarah Huda Wasif Naheel confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: *Saarah Huda Wasif Naheel*

Date: April 23, 2019

Abstract

In the field of Artificial Intelligence, research has been conducted on a wide range of applications in various fields. One common implementation of Artificial Intelligence is in the world of gaming. Artificial Intelligence is known to bring commercial value to games, and boost user experience. Our project aims to develop an application which provides aid to AI researchers in the execution of machine learning algorithms to run on platform based games by automating the process. This application will have GNU GPL licence. According to research conducted during the timeline of this project, Neuroevolution of Augmenting Topologies (NEAT) algorithm with modified version of tanh as its activation function proves to have a higher learning rate than the existing activation function. There is a possibility of extending the application to input machine learning models implemented by AI researchers and execute them against user chosen games.

Key Words: Artificial Intelligence, Games, Emulators, Machine Learning Models, NEAT, DRL

List of Figures

Contents

Chapter 1

Introduction

Research on artificial intelligence with games has been conducted for a long period of time. AI researchers aim to efficiently control playable and non-playable characters to perform in an intelligent manner to solve a variety of tasks. In the event that the researchers are able to achieve this goal, it can be witnessed how artificial intelligence is closer to achieving true artificial intelligence. True artificial intelligence is when an AI agent showcases human-like intelligence and characteristics. Games provide a platform for AI researchers to execute algorithms due to the player interaction with complex environments which are similar to human interactions with its surroundings.

Our application helps to provide support to the AI researches to run and evaluate performance of machine learning algorithms on games acting as testing platforms for these algorithms. The support is offered by making the algorithm as generic as possible and automate the process of running it on the game.The application is licensed under GNU General Public License because of all the implemented technologies contain open source licenses. This license allows to execute, analyse and alter the software.

1.1 Aim

The aim of our project is to **develop and design an application/tool which aids in the implementation of machine learning models on emulated games and analyse their results**. Therefore the machine learning models are generic in order to accommodate platform based games. The machine learning algorithms that were focused in our project are Neuroevolution of Augmenting Topologies (NEAT) and Deep Reinforcement Learning (DRL). These machine learning algorithms were chosen for this project because of their popularity with game based AI research. The games picked for our project are platform based games, primarily on the Mario series. The Mario series was chosen based on the high-dimensional action space and would require a specific set of actions in sequence for execution. We examined the possibility of proposing improvements on NEAT.

1.2 Objectives

- Develop an application which automates the process of executing a user selected machine learning model and its settings on a game chosen by the user.
- Explore possible improvements to NEAT. We will use Super Mario series for evaluation and testing purposes.
- Explore the ease of adding a platform based game to an existing machine learning algorithm.

1.3 Manuscript Overview

The manuscript consists of 6 chapters. The first chapter provides a brief introduction to the aims and objectives of the project. The second chapter outlines and critically compares the related works carried out in the gaming industry with AI. The third chapter outlines the system architecture, the graphical user interface design, and presents two UML diagrams. The fourth chapter describes the steps taken to implement project execution, along with the technologies used, and the entire implementation process of the software application and research tasks. The fifth chapter evaluates the system requirements, along with a research and usability evaluation. The final chapter is the conclusion in which the achievements, limitations and possibly future works are specified.

Chapter 2

State of the Art

Artificial Intelligence (AI) and gaming share a deep past. It started out with creating AI agents to play games and it did not matter if they had the ability to learn or not. Tic Tac Toe was the first game mastered by a program. Within a few years, more research was carried out in regard to AI learning and traditional board games and this went on for decades. Finally, the Chinook program and IBM's Deep Blue algorithm solved the roadblock of AI on checkers and chess respectively [1]. The next AI benchmark was to learn the board game called Go, because of its greater complexity and a broader search space than Chess or Checkers, with a branching factor of around 250. The Google DeepMind's AlphaGo recently became successful in learning to play Go and therefore overcame this above-mentioned benchmark. AlphaGo is based on a deep reinforcement learning approach [2]. Atari 2600 video games were also learnt using Google DeepMind with raw pixel screenshots as inputs. Most of the AI research on this field concentrates on making the AI play like a human expert or at the very least, learn to play more efficiently. AI can also be used for content or level generation and to model players in games. Most of the recent research on AI and games is based on believable agents. Believable agents are those agents that pass the Turing test for games. [3, pp. 8–10]

To achieve true artificial intelligence, the program or software should be able to think and make decisions on its own given a diverse range of scenarios. Machine learning algorithms perform extremely well on tasks related to finding patterns from data. These algorithms learn through patterns on large amounts of data, and outputs conclusions and observations learnt from the data. Human senses work in a similar manner, but this is not intelligence. Humans do not watch things all day. Humans move around, interact with the environment, make decisions and learn to adapt to varying non-predictable environment [4]. Character interacting with objects in the game is the exactly the same as humans interacting with the environment. The game character learns from the environment and carries out a sequence of decisions to maximize rewards and learn from its failures similar to the way humans do.

AI is also beneficial to the game market. Including some component of AI in the game increases its value and therefore boosts user experience. AI component is included in the game, based on a simple algorithm to find the next move or a complex algorithm to learn character behaviour unless it achieves the above-mentioned purpose. AI can play the game with two different motives. The first motive is that a player plays to learn and optimize its performance. Game testing automation is an example of a significant field where such an AI is beneficial. The second motive is that a non-player that learns to play human like and believable leading to enhancement of player experience and increase in difficulty level of the game. [3, Ch. 1, pp. 23–24]

2.1 Background

This section covers the most common games used for artificial intelligence testing, and their corresponding emulators, along with the games, the emulator and the tools used in the project. Machine learning models and activation functions which are most famous in controlling player characters to show intelligent behaviour are also mentioned.

2.1.1 Games

Platform games were chosen because they contain a single player and a sequence of player actions are required to pass a level in the game successfully. The non player characters and the environment are the only obstacles faced by the player in platform based games. These type of games are interesting in the field of Artificial Intelligence as they provide high dimensional action, observation and state spaces. [5]

Nintendo Entertainment System (NES) is one of the best rated gaming consoles and contains the classic popular games like Super Mario Bros and its variations, Bionic Commando, Bubble Bobble among many other games. Games belonging to this console are programmed to use memory resourcefully and primarily use 2048 bytes of RAM. Most of the game information such as the player's health, score and dimensions of the screen are usually present in static locations in the memory [6]. Manipulating these memory locations provide us with the necessary positions of the character, features (health, score, etc.) and dimensions of the game to run different AI algorithms effortlessly. In this project, the Mario series platform games such as Super Mario Bros, Super Mario World and Super Mario Bros 3 were used.

The Atari 2600 games are most commonly used in AI research because of their balance between classic board games and current complex graphical interface games. AI in Atari games use planning strategies and interpret the game dynamics, which is identical to strategies used for board games. Atari games provide a platform for AI to learn from the actions occurring on the screen relative to recent video games. Atari 2600 encapsulates games of different genres and contains a standard interface design and controls [7]. Atari 2600 games are not being implemented in this project, but most of the relevant AI research are based on these games, thus proving importance.

2.1.2 Emulator

Arcade Learning Environment (ALE) is a well-recognized and high-quality emulator to run AI algorithms on Atari 2600 games [7]. ALE is an open-source platform which focuses generally on planning and reinforcement learning. A variety of interfaces exists to interact with ALE on a wide range of programming languages. ALE provides extra functions like restoring the state and saving the contents stored in the registers, address counters, memory content of the game and so on [8].

NES games are supported by several emulators, but the most renowned emulators are FCEUX, BizHawk etc. Most of the NES emulators provide support to multiple platforms and languages. These emulators provide the same mechanism of saving the state and restoring it as ALE [6]. This project utilizes the the open-source BizHawk emulator as it provides an interface to run machine learning algorithms in LUA programming language. BizHawk also contains features for saving and loading particular state of the game, for searching in-game memory addresses and controller to keyboard key maps.

2.1.3 Machine Learning Algorithms

Basic common algorithms applied in the game domain are supervised learning, unsupervised learning, reinforcement learning and hybrid algorithms. The project implements Neuroevolution of Augmenting Topologies (NEAT) and Deep Reinforcement Learning (DRL).

Supervised learning involves recognition of patterns and relationships from data which are labelled. Patterns and relationships are identified in order to predict the label for a data instance, which needs to be labelled. Artificial neural network (ANN) is a form of supervised learning algorithm which is used in AI game playing [3, Ch. 2, pp. 57–59].

In contrast to supervised learning, unsupervised learning techniques encompass the recognition of patterns and relationships from data which are unlabelled. Clustering and frequent pattern mining are two types of unsupervised learning techniques. They are used to learn the behaviour of a character or build upon existing game levels through the use of player data in order to balance out the game [3, Ch. 2, pp. 77–78].
The above-mentioned techniques are pattern recognizing algorithms and are not learning techniques, and thus do not show human like intelligence. They require huge amounts of data for supervised and unsupervised learning. In tree search algorithms, the AI does not learn from its actions, it only tries to follow an optimal sequence of actions.

Reinforcement Learning is another form of a machine learning technique which learns through continuous interactions between the agent and the environment. The agent at time t in a state s takes a possible action a and gets a reward r for carrying out the action. The agent needs to figure out best actions to take place which will maximize the sum of the rewards. [3, Ch. 2, pp. 70–71] [9]

Deep Learning is a type of artificial neural network machine learning algorithm which learns from experience by replaying a set of actions with variations to execute a particular task. It is called Deep Learning because it contains many hidden layers to allow learning. [10]
Deep reinforcement learning has been successful in the field of gaming. DRL is a combination of deep learning and reinforcement learning [11]. ***Mnih*** **et al.** [12] mention three notable challenges faced in the context of deep learning by reinforcement learning, which are stated as follows:

- Deep learning algorithms require huge amount of data to be labeled for training, but Reinforcement Learning algorithms learn through reward signals.
- Instances of data are considered to be independent in deep learning technique, in reinforcement learning we discover data which is likely to be related to each other.
- Deep learning requires the data distribution to be fixed and should not be altered when learning new behaviours, but for reinforcement learning data transforms as it learns.

The challenges above are attempted to be achieved through the use of a single recurrent neural network with raw pixel data, and controls as input, to discover control policies which should perform well in learning to play several games. The network is built with a modified version of a Q-learning algorithm, in which the weights are updated using stochastic gradient descent. To overcome data correlation and transformation that happens in Reinforcement Learning, we use a replay method to level the training data distribution by using random samples from previous conversions.
The parameters which approximate the value function are updated based on the sample of experience. The network learns from interacting with the environment. The experience sample for each time stamp is stored in a dataset. In the experience replay step, an experience sample is randomly chosen and is used to update the parameters above. The many advantages of applying this variation to Deep Q learning are:

- Weights are updated by experience which provide better data efficiency.
- Learning is done based on selecting random experience sample. This breaks the correlation between the data sample.
- This also smoothens the learning as its behaviour is the mean behaviour of the previous states.

Therefore, solves the three challenges faced in the deep Q learning approach to reinforcement learning [12].

Neuroevolution (NE) algorithms are hybrid techniques. One such popular Neuroevolution algorithm is called Neuroevolution of Augmenting Topologies (NEAT). NEAT evolves network structure and weights of the neural network through the use of genetic algorithm. The network information, which comprises of the node and connection genes, is contained inside a genome. Node genes represents the nodes in the input, hidden and output layer. The connection gene represents the linking of two node genes (output and input node), and it consists of a weight, flag value (dependent on whether its enabled or disabled) and an innovation number. In order to track the gene evolution history an innovation number is given to the gene. Each genome contains a random mutation rate for each type of mutation technique.

There are mainly four techniques of mutation in NEAT. A node can be mutated through randomly updating its weights. Node and connection genes can be mutated through changing the state, from disabled to enabled, or vice versa. The add node mutation inactivates a connection and creates a new node with one connection of weight which has the value of one. The other connection contains the weight of the previously disabled connection. Add link mutation is a technique which randomly adds a new connection between two nodes with weights between the range of negative two and two.

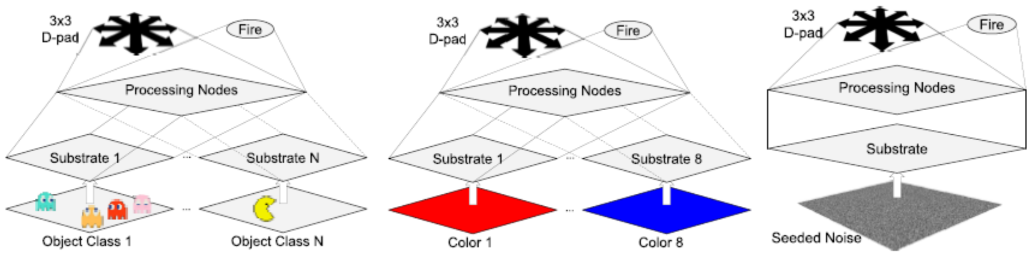
In NEAT, genome crossover is done in a meaningful way in order to protect genomes which are weak but contain useful topological information. These genomes are eliminated before their weights are optimized. In order to avoid the above-mentioned issues for genes, new node genes and connection genes are created during add node and add link mutation, followed by the incrementation of the global innovation number. The new innovation number is assigned to the new gene created and thereby helps in the tracking of gene evolution. Two genes with same innovation number must represent same topology.

Furthermore, the genomes which are similar to each other are grouped together into species. They are split into species by calculating the sum of disjoint or excess genes in each genome and the difference in weights between the identical genes. If the sum is below a given threshold then it is placed into that species. We split genomes into species so that they can fight amongst the genomes present in the species and eliminate the weak ones instead of the entire genome pool. This helps to retain new topological innovation in genomes which do not have a high fitness yet, due to insufficient evolution of weights [13].

The crossover happens between two genomes to create a new genome which contains the best genes from the parent genomes. The genes from the parent genomes are ordered based on their innovation number. For each innovation number, the gene from the most fit parent is chosen and if both the genomes have equal fitness, then the genes are randomly picked from either of the parents, which are then placed into a child. If a particular innovation number of genes is present only in one parent, then this gene is considered as a disjoint or an excess gene and are also placed in the new genome.

2.2 Related Works

Hausknecht **et al.** [7] assess neuroevolution algorithms on the basis of learning diverse Atari video games with minimal game domain knowledge. Four unique neuroevolution techniques were used for this purpose. Conventional Neuro-evolution (CNE) algorithm evolves weights of a fixed network topology artificial neural network (ANN) [14]. The next algorithm uses the Covariance Matrix Adaptation Evaluation Strategy (CMA-ES) to evolve weights in a fixed network topology ANN [15]. NEAT algorithm evolves both weights and topology of an ANN. Lastly HyperNEAT evolves the topology and weights of an indirect encoding known as Compositional Pattern Producing Network (CPPN) using NEAT [16]. To estimate the weights for the nodes in the layers adjacent to each other in the ANN, we use CPPN. The algorithms were applied using three different state input representation of the game, which were object, raw-pixel and noise-screen. The noise screen representation produces random activations. This is done to analyse if the algorithms are memorizing by assessing if the algorithms can figure out the correct actions from the input and ignore the random actions happening in the game.



(a) Object Architecture (b) Raw-Pixel Architecture (c) Noise-Screen Architecture

Figure 2.1: Architectures [7]

Figure 2.1 presents a graphical representation of the system architecture for different input representation. A total of 55 Atari games with different genres were chosen. A policy for each game was generated separately. This policy consisted of 150 generations and 100 individuals per generation. 15k episodes play per each game and an episode ran for a game interval of 13 mins. The game score was the fitness score of the individual at the end of each episode. The fittest individual at the end of 150 generations is considered as the score for the neuro evolution agent. Their technique is tested by using z-score for score normalization. This is done since each game has its own different game scores. Using this normalization, we know how many standard deviations a score is from the mean score. The performance of the algorithm is measured by comparing the game score which is normalized among the Atari games using z-score normalization.

The performance of the algorithms was analysed based on Analysis of Variance (ANOVA) statistical test with a Turkey's HSD significance test. From conducting the above experiment, they found out that for object representation, NEAT and HyperNEAT outperformed the fixed topology networks. For noise representation, NEAT performed better than the rest of the algorithms. For raw pixel input representation, only HyperNEAT was able to learn due to the indirect encoding in HyerNEAT. Even though HyperNEAT algorithm is more complex and contains indirect encoding, it did not seem to perform better than NEAT as long as the inputs were object or noise. Learning algorithms (NEAT and HyperNEAT) were compared with planning algorithm, random play and human high scores by experts. The planning algorithm gives better performance than the learning but both of them perform better than random play. Neuro evolution methods gave a better score than the high scores for three of the games but on average the score of the algorithms were beaten by the human experts.

The performance of the algorithm reduces depending on the generalization of the input representation. As Atari games are deterministic and do not require the history of the character actions to predict the next move and therefore all the algorithms implemented are feed forward networks. To apply the algorithm to non-deterministic games, ***Hausknecht*** **et al.** mentioned the possible implementation of recurrent networks. The objects were recognized manually for each Atari game, this is not very efficient for a large number of games or a game which contain many objects. The manual recognition raises the possibility of error in regards to identifying objects. Time consumption is another factor which could be taken into account for the manual recognition of objects for each game.

Mnih **et al.** [12] presents an alternative to HyperNEAT for Atari games. The paper incorporates deep reinforcement learning with Atari games. This algorithm contains a Deep Q-Network with an experience replay method [17]. This system was evaluated by calculating the mean of the maximum predicted Q-values of a set of randomly selected states. The rewards are discounted when the agent follows the policy on the particular state.
This deep reinforcement learning algorithm was compared to other Reinforcement learning algorithms such as the Sarsa(λ) algorithm [7] and the contingency algorithm [6], both of which are considered as top performing models. Both of these algorithms had more prior information of the game by having each of the 128 colours as a separate channel and conducted background subtraction in order to understand the given visual problem in a game. The deep reinforcement learning algorithm on the other hand learnt by RGB screenshots, which are converted into raw gray scale screenshots, and the detection of game objects is its own task to do. This algorithm performs better than the other reinforcement learning algorithms even with less knowledge about the game. The algorithm beat human players in almost half of the games. The ones it scored less than the human scores were primarily strategy games which require a lot of time to learn and play. The wide variety of possible moves in a full-fledged strategy game would cause the algorithm to take a significant amount of time to learn due to the complexity of each move, considering an evaluated decision of good and bad moves must be taken.

McPartland **et al.** [9] primarily aim to apply Reinforcement Learning in First-Person Shooter games. A complex AI bot is used which learns through navigation and gathering items in an user created three-dimensional environment similar to a maze. The AI bot also learns through combat scenarios. In order to learn the bot controllers, Reinforcement Learning uses Sarsa(λ) algorithm. The secondary aim brought forward by the author is to analyse a way to effectively create generalized autonomous bots through the use of combining Reinforcement Learning controllers.
The system was evaluated through three tasks; Navigation Task, FPS Combat Task and General Purpose Task. The navigation task was aimed towards testing how efficiently a bot could traverse a maze-type setting while gathering items. The aim of the FPS combat task was based on the ability of how well a bot could fight when trained against an opponent which was controlled by a state machine. This state machine bot was encoded to fire a shot whenever an enemy was within sight, while ensuring that the weapon was ready to fire. The state machine controlled bot keeps

enemies within range until elimination. The final task, general purpose task, was setup with four types of bots. One was a Reinforcement Learning bot, another was a state machine bot, and the remaining were non-reacting bots. The state machine controlled bot was encoded with aggressive behaviour, which explores the playing field, gathers items within range, and hunts down enemies when sighted. The non-reacting bots were placed to give the Reinforcement Learning bot targets to practice on and learn a generalized combat strategy. Three types of Reinforcement Learning models were used HeirarchicalRL, RuleBasedRL and Reinforcement Learning. All the bots were compared to provide average, lowest and highest results. The conclusions show that the hierarchical reinforcement learning and the rule based reinforcement learning models presented higher performance in combat and navigation is comparison to the flat reinforcement learning model.
The algorithms were run and tested within a user created game environment in which all bot spawn points, and item spawn points were the same throughout all the trials. The weapons have unlimited ammunition, with a 1 second delay rate between shots fired. We can say that the models were over-fitting since the game environment was static throughout all the trials, which decreases the chances of the AI actually learning. If any of these bots were placed into an actual game environment with dynamic respawn points and limited ammunition capacities with varying reload times from different weapons, the accuracy may possibly drop significantly.

Lample **et al.** [18] present an AI-agent which was used for playing FPS games, in a game mode which involved achieving the maximum number of kills by the AI-agent. The agent used Deep Recurrent Q-Network (DRQN) with Long Short-Term Memory (LSTM), and was trained in two phases, one being navigation, and other being shooting. The training was done through reward shaping and additional supervision from ViZDoom platform. The VizDoom platform is used for the game Doom. The internal configuration and ground-truth knowledge of enemies provided by ViZDoom is used during training.
The agent was evaluated against game bots which are built into the game, with and without using navigation. The evaluation was run for 15 minutes on each map, while for full death matches the evaluation was run with an average of 10 training maps and 3 test maps.
The evaluations cover the navigation aspect of DRQN, comparing scores to human counterparts, and game features aspect of DRQN. With navigation enhancements, the agent showed significant improvements to the number of items picked up and the kill/death ratio in comparison to having no navigation enhancements. The agent also outperformed humans, whose scores were averaged over 20 individuals. Using game features for training improved performance significantly with notable differences in the kill/death ratio.
The modified DRQN model, and an alternative model presented by ***Dosovitskiy*** **et al.** [19] known as direct future prediction, took part in a competition called the Doom AI Competition. The alternate model only took part in the full Deathmatch track and beat the DRQN model by over 50%. Direct future prediction was a simple model and did not require the additional supervision which is presented within the modified DRQN model.

***Nielsen* et al.** [20] aim to create agents which can effectively play a range of general strategy games. Six games are played, which are two player turn-based games on a two dimensional grid where the victor is the one which removes the pieces put forth by the opponent. Strategy Game Description Language (SGDL) is a game engine which is used to assert a variety of strategy games. Six architectures are used for agents, along with their variations. All architectures based on learning algorithms were put through substantial training for each agent. All architectures were given equal time for training.
The agents went through two types of evaluation, namely competitive evaluation and user interaction evaluation. In competitive evaluation, each agent was pit against another agent on a wide range of models. This helped analyse the general performance of all the trained agents. In user evaluations, human participants were asked to play against a minimum of two agents which use different game models, and then provide answers in relations to their preference between the two agents. This helped to provide data on which agent proved to be the best against humans, and which agents had the most satisfying interactions with the participants.
Many agents would prove to fare poorly against strategy games which have high branching factors. Branching factors are the number of potential moves per turn. High branching factors can prove to be very challenging for AI systems to overcome. Certain games like Civilization could prove to be very computationally expensive due to its sheer complexity. The paper mentions that this is a topic of research.

***Kunanusont* et al.** [21] create a learning agent which has the ability to learn from video game screenshots through the use of Deep Q-Network within General Video Game AI framework (GVG-AI). The Deep Q-Network method was implemented with experience replay on six games grouped into two categories, namely shooting games, and exit-finding games. The screen block size, dimensions and level dimensions were used as inputs to the algorithm. There is no restriction on time, which means that the algorithm terminates once the game ends. The algorithm is compared in performance by evaluating it against Monte Carlo Tree Search planning algorithm.
The authors did not train many games from the known 140 GVG-AI games due to the long amount of time taken to train each game. The comparison is biased since the Monte Carlo Tree Search had a time limit, while the algorithm used by the author has no time constraints. The Convolution Neural Network (CNN) is pre-trained before applying it to a game. In the event that the game is too large, the CNN work better if it could be expanded. When using General Video Games AI, the learning agent should be able to use its previous experience from other games, into a new game. This would be more efficient since the AI does not have to learn from the very beginning for each game, which is similar to human behaviour.
A similar approach is taken by ***Woof* et al.** [22] who describe an alternative state representation of General Video Game AI in which the input is sent as a game object to the Deep Q-Network. The authors compare state representation with two others, pixel representation and feature representation. Object state representation was shown to have a higher average score over the other two in games where the environmental ground-truth was available for extraction. The ground-truth in this context refers to the mechanics of the game and the relation

between the objects and their domain.

Chapter 3

Design

This chapter outlines the use cases, system architecture, activity diagram and graphical user interface. The use cases section outlines the use case for the application. The system architecture contains a diagram of how the different components of the application interact with each other and the data flow between them. The activity diagram shows the interaction between components. The final section portrays the interface design of the application.

3.1 Use Cases

The use case diagram aids in the visualisation of the functionalities of the system from a users' viewpoint. This diagram helps in identifying the actors and the tasks that are needed by the system to behave in a specific manner.

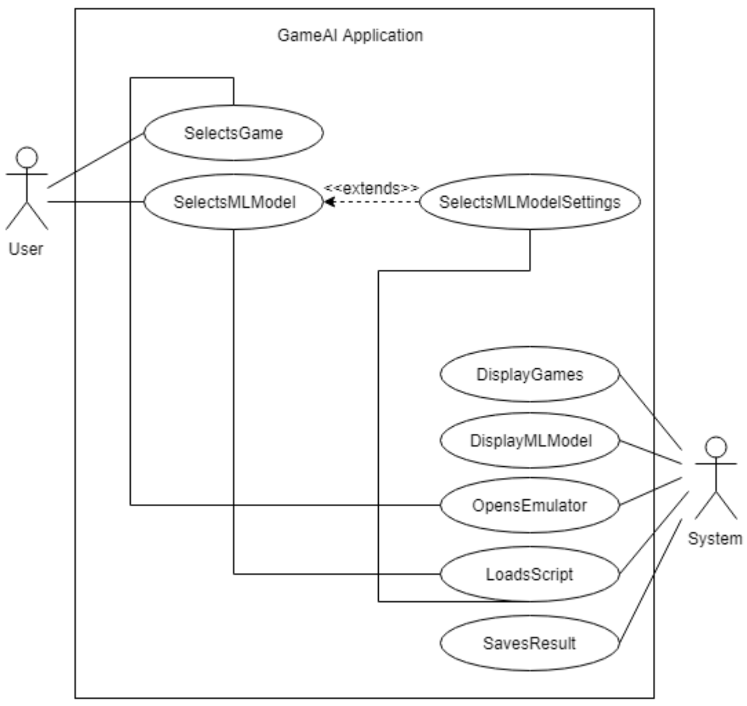


Figure 3.1: Use Case Diagram of the Application

The figure 3.1 describes the user and system requirements of the application. The main users of the application are identified to be AI researchers as they will utilize this tool to execute different machine learning models on games, which acts as a test-bed, and analyse the behaviour of the algorithms. The AI researchers should be able to select a machine learning algorithm and a game on which they want to execute the algorithm. Upon selecting a machine learning model, user might be given options to edit the settings of the model.
The system is required to display the list of games and machine learning models which are currently implemented within the system. The system should automate the process of opening the emulator and inserting the game. It should also be able to run the machine learning model script on the game and generate results in a file format.

3.2 System Architecture

The system architecture outlines the high level structure of the system, and describes how each component interacts within the system. System architecture helps us in understanding how the system functions. This section presents the current system architecture in detail, and provides an overview of the relations between the components.

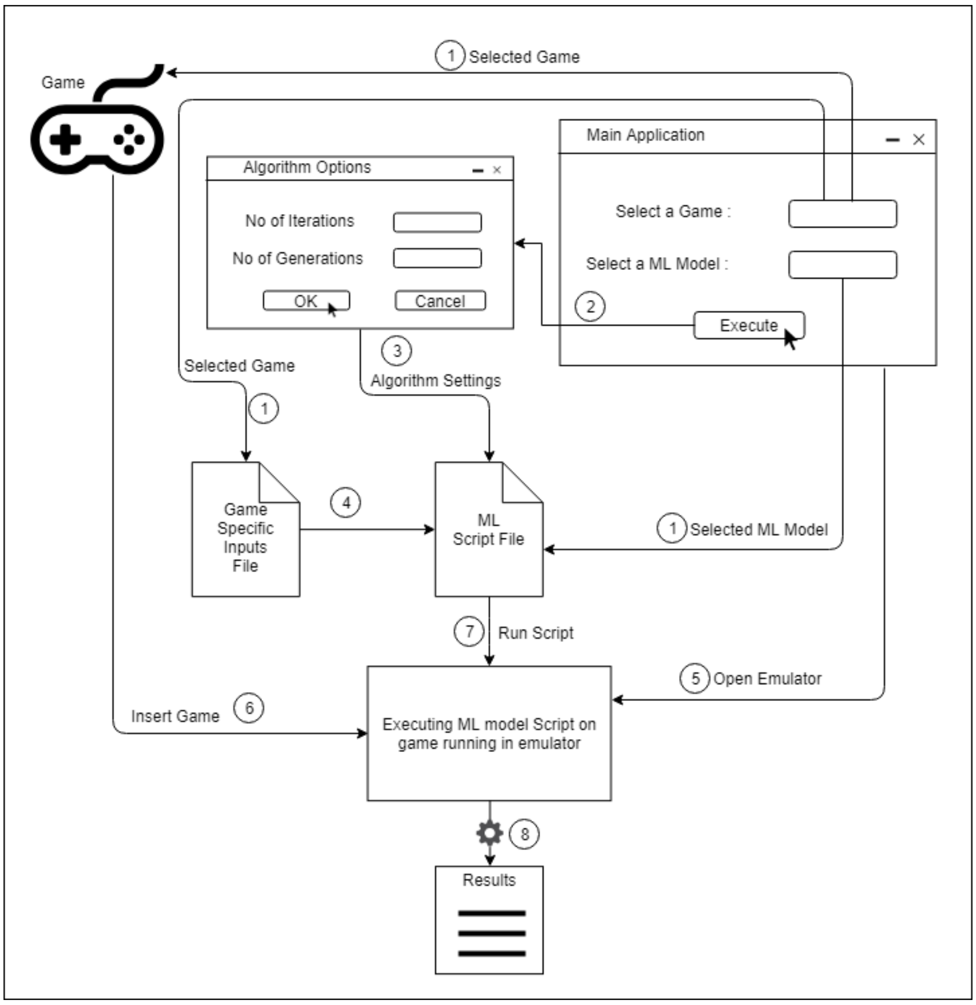


Figure 3.2: System Architecture of the Application

The figure 3.2 represents how the different components of the application interact with each other. When the user selects a game and machine learning model of his choice,another window may open with the options of editing the configuration of the ML model. Depending on the game, the script containing the game information and the user defined configurations is applied to the selected ML model script, which is then executed on an emulator which supports the selected game. The application opens the emulator and inserts the selected game. Then the model script is run on the game and finally generates the result data, once task is completed.

3.3 Activity Diagram

An activity diagram models and describes the systems' flow of control in detail. It assists in capturing the exception cases that can occur while performing the task and is useful during implementation.

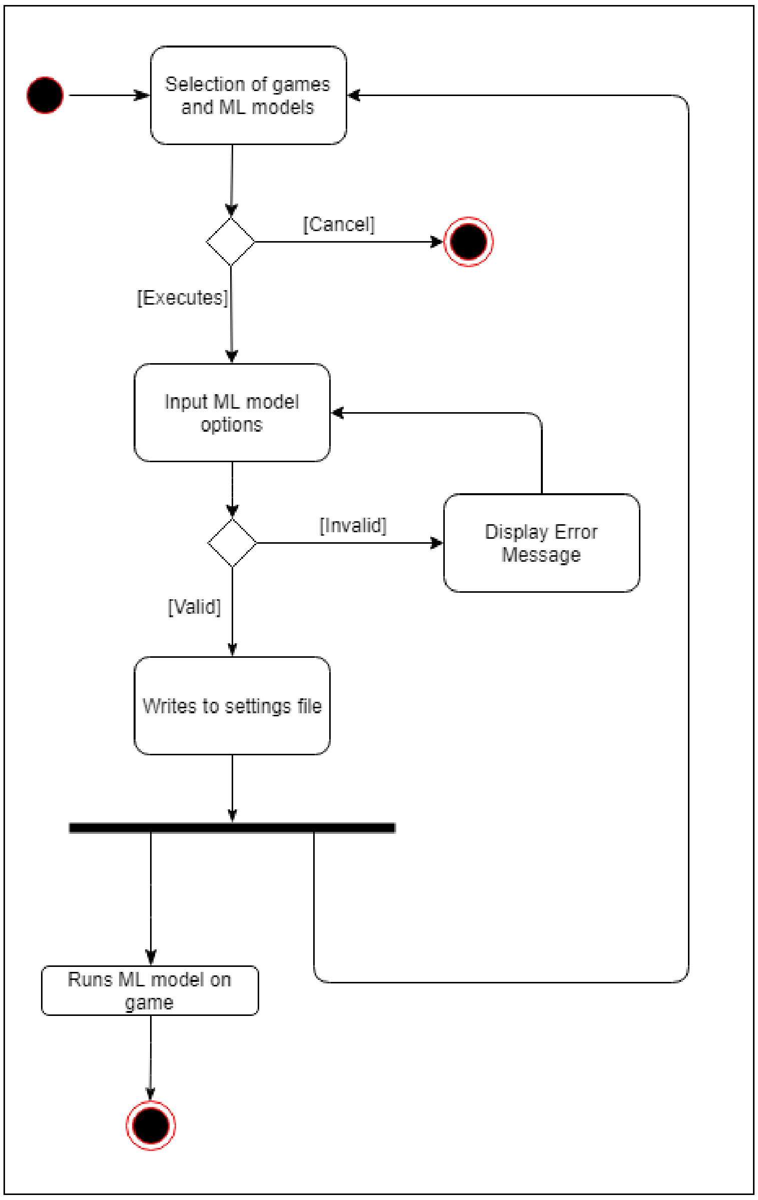


Figure 3.3: Activity Diagram of the Main Software Application

The figure 3.3 illustrates the detail process of the main application window. The window requires the user selection of a game and a ML model, once that is achieved the user either chooses to execute the model or cancels the window. If the the window is closed, the program is terminated. If the execute option is chosen, depending on the ML model, a window opens up which contains the default configurations of the ML model. These configurations can be modified by the user. In the event that the provided configuration parameters are invalid, it displays an error message and goes back to the configuration window. If all the configuration parameters are valid, they get written to a settings file and closes the configurations window. Finally, the machine learning model is run on the game in parallel to the running of the main application.

3.4 Graphical User Interface

The user interface design is an essential part of user experience. A good design has the ability to attract people and keep them engaged in a simplistic manner. The graphical user interface for this project is aimed towards simplicity as it is only for research purposes.

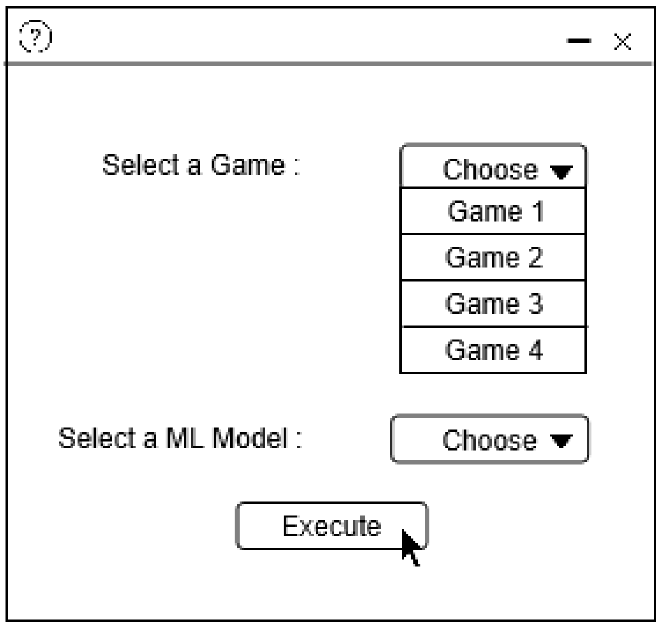


Figure 3.4: Graphical User Interface Design

The figure 3.4 describes a basic and simple straight forward GUI interface of the application. Users can select a game and a machine learning model from the drop-down menu containing the list of different games and ML models. Once the selection is made, the user clicks on the execute button. The rest of the process is carried out by the system. Therefore, the initial design of the application is assumed to be user friendly and easy to navigate and fulfil its purpose.

Chapter 4

Implementation

Chap Intro

4.1 Project Methodology

In the initial stage, an efficient project methodology must be elected for smooth and efficient development of the project. An Agile Methodology known as Scrum shall be utilized to develop this project until completion.
Scrum teams can be divided into two primary roles. First being a Scrum Master, whose role is similar to that of a coach or mentor. Second is a Product Owner, whose role is to help build a feasible and well planned product [23]. In regards to this project, the two roles have been assigned as follows:

- **Scrum Master** - Supervisor
  The supervisor takes on the role of a Scrum Master by verifying that the developer/researcher is free of any doubts and distractions which may directly affect the progress of the current task.

- **Product Owner** - Supervisor and Users
  The supervisor adopts the role of a product owner by ascertaining the successful completion of required functionalities. The users are also considered as a part of this role since they play a direct role to the development of the user interface through user experience feedback.

The scrum process has three primary artifacts. They are namely the product backlog, the sprint backlog, and the burndown charts. The product backlog holds a list of functionalities to be added to the system. The product backlogs are prioritized to be able to work on the most critical features first. One of the most efficient ways to create a sprint backlog is to store user stories. User stories are brief functional descriptions from a users perspective. A sprint backlog is created during the planning phase of meetings, which encompasses tasks which need to be done in order to successfully deliver the committed functionality for the current sprint. Burndown charts are used to visualize the amount of work remaining in a release or a sprint. This can prove to be effective in order to view that the plan is being executed as per schedule.

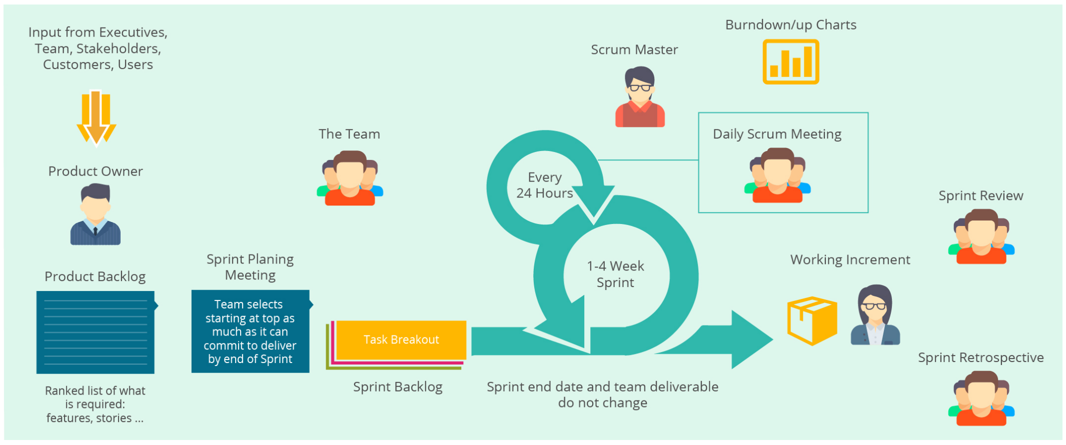

Input from Executives, Team, Stakeholders, Customers, Users
Product Owner
Product Backlog
Ranked list of what is required: features, stories ...
Sprint Planing Meeting
Team selects starting at top as much as it can commit to deliver by end of Sprint
The Team
Task Breakout
Sprint Backlog
Every 24 Hours
1-4 Week Sprint
Sprint end date and team deliverable do not change
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Working Increment
Sprint Review
Sprint Retrospective

Figure 4.1: Scrum Process [24]

The tasks were split into a small set of deliverables. Each task was delivered iteratively within a fixed time frame. These are also known as timeboxed iterations or sprints [24]. Each delivered task has a potential of being shipped out as a working increment. Meetings were held on a weekly basis with the Scrum Master to make sure that development is being carried out as planned. A great example of a Scrum process is shown in figure 4.1. Upon the release of the first working prototype, the Supervisor was asked to provide feedback on changes expected, if any. These changes were taken into account, and another iteration of the prototype was released to show the effective implementation of the changes.

Weekly meetings were held with the Supervisor in which ideas to extensively expand the project by adding a research aspect. Solutions for any issues faced during the implementation and testing of the project was also discussed extensively, with each being applied soon after a meeting was held. The project was implemented by following the project plan.

4.2 Technologies

- **BizHawk** This an emulator for Nintendo Entertainment System (NES) games. BizHawk version 1.11.3 emulator is an open-source software and was utilised in this project. It provides an interface for executing lua scripts, which made it easy to execute a machine learning algorithm script. The emulator allows users to save any particular state of the game, this feature was useful when reloading back to the initial state during the learning process. It even provides support for memory addresses, which is important when gathering game related information.

- **Pywinauto** Pywinauto version 0.6.5 is a Python module which is used to automate Windows GUI. Pywinauto allows mouse and keyboard clicks in an automated way to Windows OS dialog boxes and controls. This has proved essential for the automation of the emulator.

- **Numpy** Numpy version 1.16.2 is a Python module which allows the creation of multi-dimensional arrays and matrices. Numpy is primarily used in the application for machine learning models.
- **Keras** Keras version 2.2.4 is a neural network library written in Python. It is run alongside Tensorflow for quick experimentation on deep neural networks.
- **Tensorflow** Tensorflow version 1.13.1 is a very famous open source library for machine learning, primarily used for research and production applications. Tensorflow is used as a Python package in the project for deep reinforcement learning.
- **Pywin32** Pywin32 version 224 is a Python package which allows access to the Win32 API in order to create and manipulate system COM objects.
- **Opencv_python** OpenCV version 4.0.0.21 is a library which is aimed at real-time computer vision. It is used with deep reinforcement learning to capture elements of the screen which would potentially be used to track rewards within the algorithm, thus assisting in learning.
- **PyQt5** PyQt5 version 5.12.1 is a Python package which allows users to create an interactive Graphical User Interface. This package is primarily used to create the main interface for the application.
- **Win32gui** Win32gui version 221.6 is a Python extension similar to Pywin32. The main use of this package is to get the current *x* and *y* cursor positions on the screen.

4.3 Implementation Process

This section covers the implementation of the software and research components of the project.

4.3.1 Software Application Implementation

This section contains the implementation of the graphical user interface and the generic machine learning model implementation.

Graphical User Interface Implementation

This section primarily aims to provide the various steps and decisions which needed to be taken to create the main application interface. In this section, we dive into the aspects of code, and design implementations and their interactivity between graphical components.

All the application windows was created in Python, using a package known as PyQt5. The system has three PyQt5 windows. The main application window is the initial window which pops up once the application is run. Upon selecting a game, with the NEAT algorithm as an option, a second window shows up in which parameters for the algorithm can be modified. In the event that a user selects the NEAT algorithm with mutations option, another window shows up specifically for parameters which can be modified for that specific algorithm. Each of these, along with their components are described further along within this section.

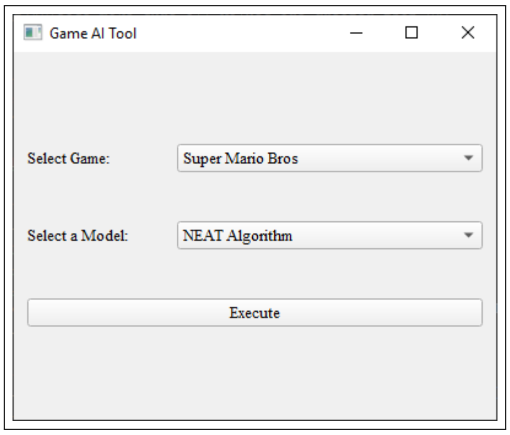
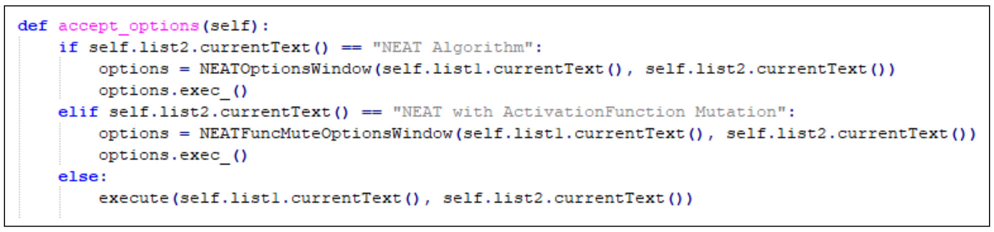


Figure 4.2: Main Application Window

Through the main application window (Figure 4.2) a user is able to select the game of choice, and a model of choice. Currently, the only available games are Super Mario Bros, Super Mario Bros 3, and Super Mario World. The models which can be chosen are also limited to the NEAT Algorithm, Deep Reinforcement Learning and NEAT with activation function mutation.

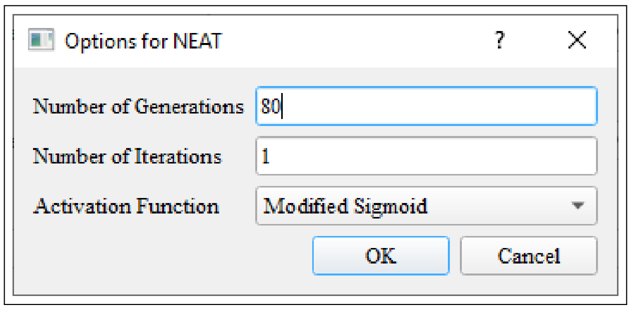


```
def accept_options(self):
    if self.list2.currentText() == "NEAT Algorithm":
        options = NEATOptionsWindow(self.list1.currentText(), self.list2.currentText())
        options.exec_()
    elif self.list2.currentText() == "NEAT with ActivationFunction Mutation":
        options = NEATFuncMuteOptionsWindow(self.list1.currentText(), self.list2.currentText())
        options.exec_()
    else:
        execute(self.list1.currentText(), self.list2.currentText())
```

Figure 4.3: Options Execution Code Snippet

Upon selecting a game and machine learning algorithm, the application checks if the NEAT algorithm or NEAT with activation function mutation algorithm are chosen. Figure 4.3 shows that once either of these conditions are met, their corresponding windows are executed. If neither is run (which means that deep reinforcement learning algorithm is picked) then the application does not open another window, and continues to the main block from where BizHawk is opened. The deep reinforcement learning algorithm does not use LUA, but is instead written in Python, and uses Tensorflow and OpenCV to collect reward information from the game, and enable learning.

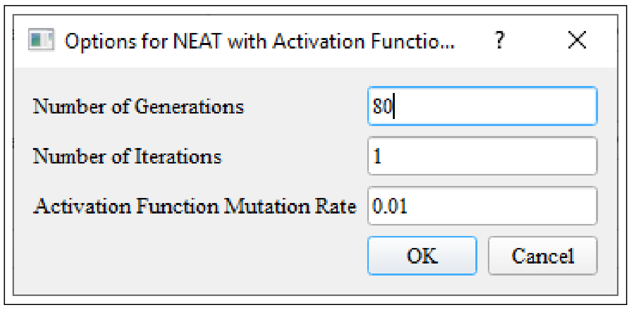
BizHawk is fully automated through the Pywinauto Python package file. The automation begins from picking the appropriate game, followed by opening the Lua Console and running the appropriate Lua Script for either NEAT or NEAT with activation function mutation.


Options for NEAT

Number of Generations	80
Number of Iterations	1
Activation Function	Modified Sigmoid

Figure 4.4: NEAT Options Application Window

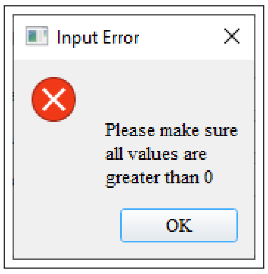
Figure 4.4 portray the NEAT Options window given that the NEAT algorithm option is chosen. Within this window, users are able to specify the generations, iterations, and which function they would prefer to use. The functions which can be chosen are modified sigmoid, sigmoid, tanh and ReLU.


Options for NEAT with Activation Functio...

Number of Generations	80
Number of Iterations	1
Activation Function Mutation Rate	0.01

Figure 4.5: NEAT with Mutations Options Application Window

Figure 4.5 on the other shows the NEAT with activation function mutation options window, from which a user is able to select the generations, iterations and the function mutation rate. Once appropriate values are filled in and 'OK' is clicked, the windows from figure 4.4 and figure 4.5 create an appropriate settings file in the BizHawk directory of the project.


Input Error

Please make sure all values are greater than 0

Figure 4.6: Error Window

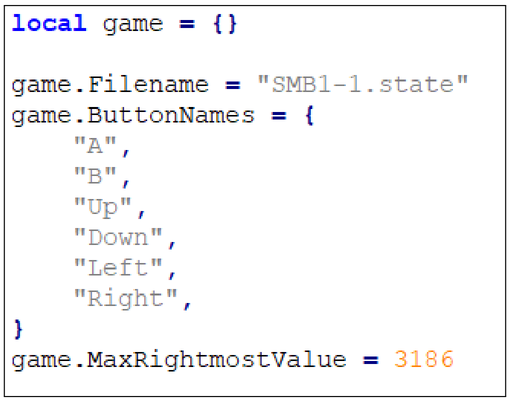
In the event that the values inserted in either options window is below or equal to the value of 0 an error message is displayed (Figure 4.6). This allows the application to make sure that only a suitable value can be given.

Generic ML Model Implementation

- **Neuroevolution of Augmenting Topologies algorithm (NEAT)**

  NEAT is a Neuroevolution algorithm which consists of a neural network which evolves its network structure and weights using genetic algorithm approach. The detailed concept of the algorithm is mentioned under Chapter 2 in section 2.1.3.

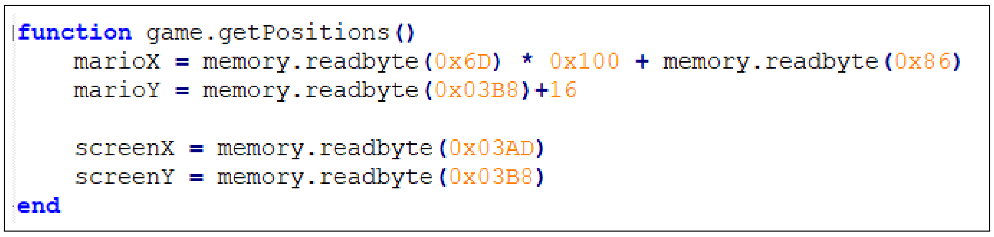
  The main NEAT code file was adopted from ***SethBling*** [25] and is free to use, which is implemented on two platform games from the Mario series. Modifications were made to the algorithm so as to make it generic, therefore accommodate other platform games. This was achieved by separating the game specific information from the file containing the NEAT algorithm. The games which are implemented are different from each other based on the number of player actions, the environment level map and the different types and number of enemies.



```
local game = {}

game.Filename = "SMB1-1.state"
game.ButtonNames = {
    "A",
    "B",
    "Up",
    "Down",
    "Left",
    "Right",
}
game.MaxRightmostValue = 3186
```

Figure 4.7: Game specific variables

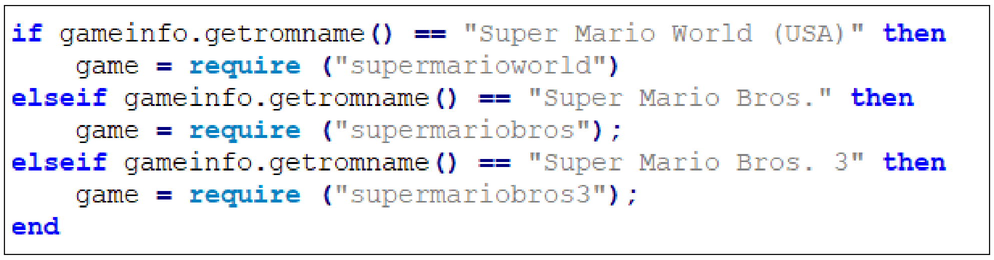
Figure 4.7 represents the variables specific to the game Super Mario Bros. The entire game specific file is wrapped into local game variables defined. The filename contains the initial state of the game and button names which are required to display the output of the neural network on the game screen. The maxRightmostValue variable stores the maximum fitness value in the fitness function defined in the algorithm.



```
function game.getPositions()
    marioX = memory.readbyte(0x6D) * 0x100 + memory.readbyte(0x86)
    marioY = memory.readbyte(0x03B8)+16

    screenX = memory.readbyte(0x03AD)
    screenY = memory.readbyte(0x03B8)
end
```

Figure 4.8: Game specific function

Figure 4.8 is an example of a game specific function which is also wrapped with the above mentioned game variable. The functions present in the game specific input files are getPositions(), getTile() and getSprites(). These functions provide the *x* and *y* positions of the player and enemies, and the layout map of the level.



```
if gameinfo.getromname() == "Super Mario World (USA)" then
    game = require ("supermarioworld")
elseif gameinfo.getromname() == "Super Mario Bros." then
    game = require ("supermariobros");
elseif gameinfo.getromname() == "Super Mario Bros. 3" then
    game = require ("supermariobros3");
end
```

Figure 4.9: Game specific data imports

The require statements in figure 4.9 are added to the NEAT ML model file. When the ML script is run on the Lua console, it checks the name of the game inserted into the emulator and imports the file which contains its game specific information. Thereby, making the algorithm generic. So to add another game which can run with the NEAT algorithm, all that is required is to create a game specific information file with the same variables and functions as defined in the '*supermariobros.lua*' and '*supermarioworld.lua*' files. Then adding a require statement of the specified file in the NEAT algorithm implemented file based on the name of the game.

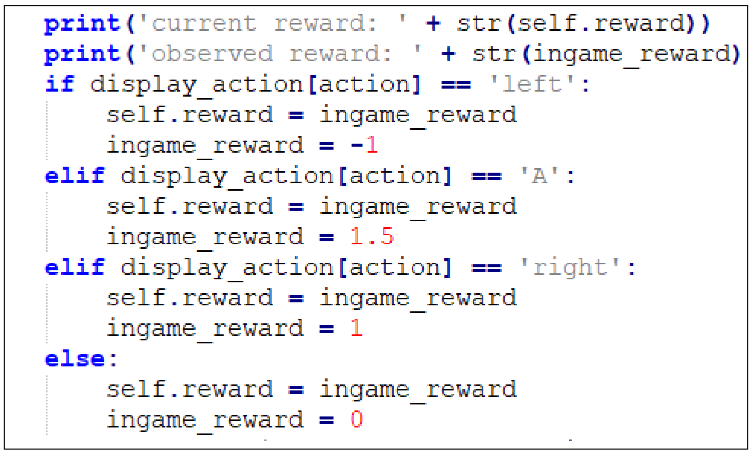
The memory addresses of the games Super Mario Bros and Super Mario World were already provided in the NEAT Lua code written by ***SethBling***. To ensure that it is easy to insert another platform game and run NEAT on it, we implemented another game from the Mario series, which is Super Mario Bros 3. We were able to run the NEAT machine learning algorithm on the above mentioned game following the above mentioned steps. The only problem faced was the lack of documentation on the game relevant memory addresses. The main resources for game relevant addresses were available in multiple sources, namely ***TASVideos*** [26], the Lua folder of the BizHawk emulator and the memory address searches feature of BizHawk.

- **Deep Reinforcement Learning (DRL)**

  DRL is a machine learning algorithm which combines deep learning and reinforcement learning techniques. It consists of a neural network with many layers for learning, and it learns through continuous interactions of the player with the environment with a reward system. Detailed description of this algorithm is discussed under Chapter 2 in section 2.1.3.

  The DRL algorithm code implementation was taken from ***Trivedi*** [27][28]. This code was implemented for FIFA PC game. This implementation is mostly generic and contains an API for the game related information.

  The game specific API should contain the following functions. The **observe function** which consists of the screen capture of the game environment. The **act function** which describes the controller keys mapped with the keyboard keys, which needs to be pressed and released after few seconds. The **reward function** which is where one would define the rewards for the learning. Finally the **is over function** returns a boolean indicating if the game is over or not.



```python
print('current reward: ' + str(self.reward))
print('observed reward: ' + str(ingame_reward)
if display_action[action] == 'left':
    self.reward = ingame_reward
    ingame_reward = -1
elif display_action[action] == 'A':
    self.reward = ingame_reward
    ingame_reward = 1.5
elif display_action[action] == 'right':
    self.reward = ingame_reward
    ingame_reward = 1
else:
    self.reward = ingame_reward
    ingame_reward = 0
```

Figure 4.10: Rewards function

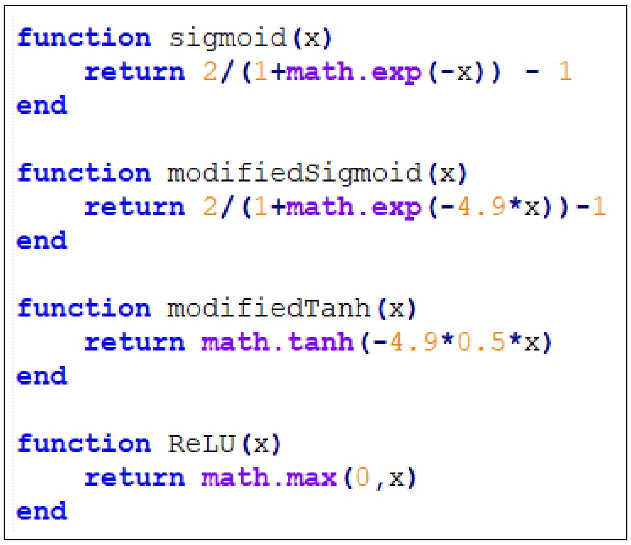
The API for the games super mario bros and super mario world was created following the above file function description. The reward functions are basic as there was no change in score when the player moves forward, therefore makes it difficult to calculate proper rewards and the rewards were given based on the actions as shown in figure 4.10.

4.3.2 Research Tasks Implementation

- **Implementation of NEAT with different activation functions**

  This is implemented by changing the modified sigmoid function which was the activation function for the original NEAT algorithm. This function was replaced by normal sigmoid, modified version of tanh and RELU activation functions. The maximum fitness at each generation was stored into a csv file in the main folder.

  Figure 4.11 shows the different activation function definitions implemented in the project:

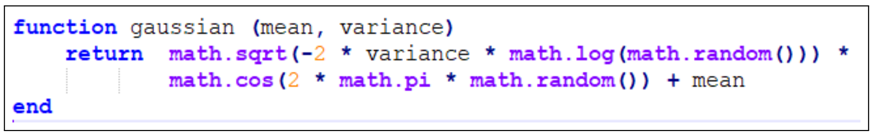


```lua
function sigmoid(x)
    return 2/(1+math.exp(-x)) - 1
end

function modifiedSigmoid(x)
    return 2/(1+math.exp(-4.9*x))-1
end

function modifiedTanh(x)
    return math.tanh(-4.9*0.5*x)
end

function ReLU(x)
    return math.max(0,x)
end
```

Figure 4.11: List of activation functions definitions [29]

- **Implementation of NEAT using an Evolution Strategy**

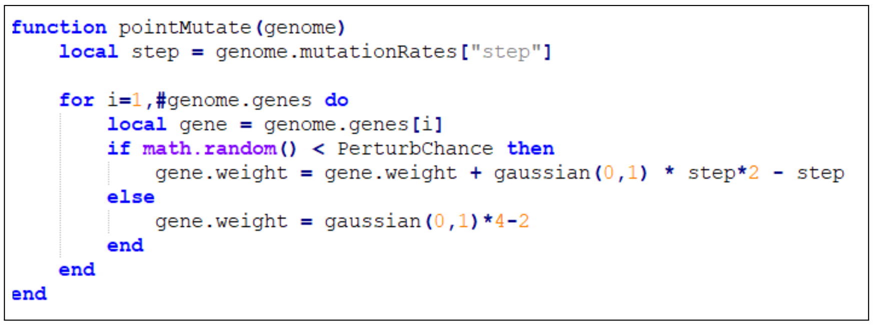
  This is executed by implementing a Gaussian or a normally distributed random variable during weights mutation. This is implemented for tanh activation function.



```lua
function gaussian (mean, variance)
    return  math.sqrt(-2 * variance * math.log(math.random())) *
            math.cos(2 * math.pi * math.random()) + mean
end
```

Figure 4.12: Gaussain function defination in lua

The lua implementation of the gaussian function definition displayed in figure 4.12 was taken from ***Code***[30]



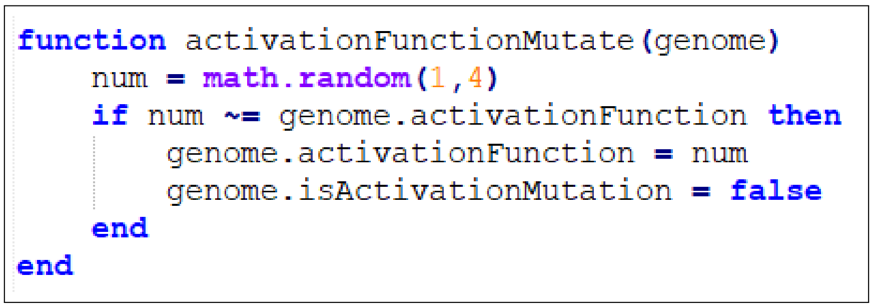
```lua
function pointMutate(genome)
    local step = genome.mutationRates["step"]

    for i=1,#genome.genes do
        local gene = genome.genes[i]
        if math.random() < PerturbChance then
            gene.weight = gene.weight + gaussian(0,1) * step*2 - step
        else
            gene.weight = gaussian(0,1)*4-2
        end
    end
end
```

Figure 4.13: Gaussain function implementation in lua

The figure 4.13 displays the implementation of the gaussian function on the weight mutation present in the point mutation function. The Gaussian function arguments are set as mean = 1 and variance = 0.

- **Implementation of NEAT with an activation function mutation**

This mutation feature is implemented by initializing genome with an activation function randomly, a boolean which states whether that genome has undergone activation function or not and an activation function mutation rate. If the genome undergoes activation function mutation once, then the boolean is set to false and it cant mutate its activation function again. The genome is always mutated to the activation function which is not its initial activation function. The implementation of the above described mutation function is shown in figure 4.14



```
function activationFunctionMutate(genome)
    num = math.random(1,4)
    if num ~= genome.activationFunction then
        genome.activationFunction = num
        genome.isActivationMutation = false
    end
end
```

Figure 4.14: Activation function mutation implementation

Chapter 5

Evaluation

This chapter has three sections. The first section describes the system requirements and how they are achieved. The second section has research experiments conducted, and the last section evaluates the usability of the graphical user interface.

5.1 System Requirements Evaluation

This section describes the completion and evaluation of the requirements that were necessary to be implemented by the system, so that the system displays its required functionalities.

Type	System/User	Description	Requirement	Status
Functional	System	The system shall interact with the emulator	S-FR-1	Completed/Tested
Functional	System	The system shall load a game on an emulator	S-FR-2	Completed/Tested
Functional	System	The system shall process a user selected machine model from a given list, on a game which is also from a given list	S-FR-3	Completed/Tested
Functional	System	The system shall allow users to modify settings of the algorithm	S-FR-4	Completed/Tested
Non-Functional	System	Functional Scalability	S-NFR-1	Completed/Tested

Table 5.1: Completed/Tested System Requirements

All the above requirements provide the expected working of the system and therefore are tested. For the non-functional requirement, addidtional functionality to save results in a CSV files is added.Therefore, it displays functional scalability.

5.2 Research Evaluation

All the three types of research variations to NEAT described under chapter 4 in section 4.3.2. The experiment was carried out by running the different code implementation of the research tasks on the Super Mario Bros game and storing the results in csv files. This experiment was run for thirty iterations for 80 generations each. Doing this on a single computer would have taken more than a long time to finish. The task was divided between six computers, with five iterations each. The learning performance is calculated by average of the maximum fitness obtained by the algorithms in each generations for thirty iterations.

- **Task 1 - Comparison between different activation functions**

  This task is about implementing different activation functions and if any could do better than the existing activation function for the NEAT algorithm.

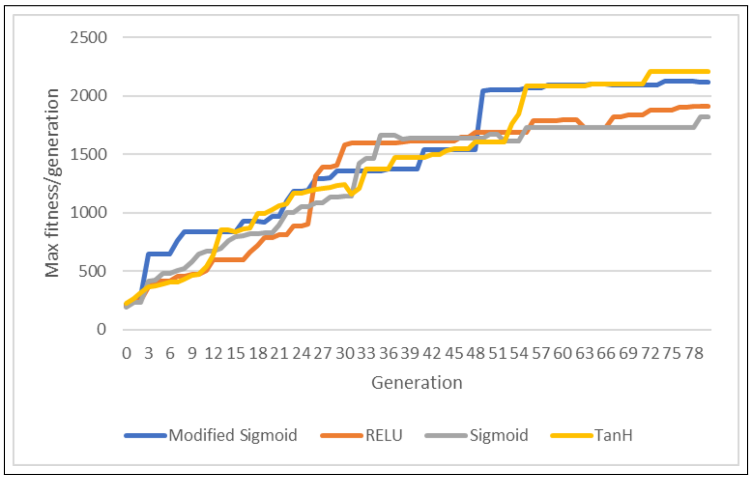


Figure 5.1: Comparison between activation functions

In figure 5.1, we notice that modified sigmoid starts off with a good learning rate at early generations, tanh and ReLU activation functions show minimum learning performance in the fifteen generations and sigmoid dangles as an average between them.Sigmoid and modified sigmoid takes comparatively more time to complete eighty generations than tanh or ReLU. Though tanh dint get a good start in the beginning, but it manages to reach the highest maximum fitness at the end of the execution. Sigmoid and relu are a weak activation function and provide less or slow learning. Tanh performs better when the total generations is eighty, but from the graph we notice that modified sigmoid will fare well if the generations stop at fifty.

- **Task 2 - Comparison between weights updated with normally distributed random variable vs default random variable**

  This task is about implementing a normally distribution random variable and comparing it with the original file but with activation function changed to tanh.

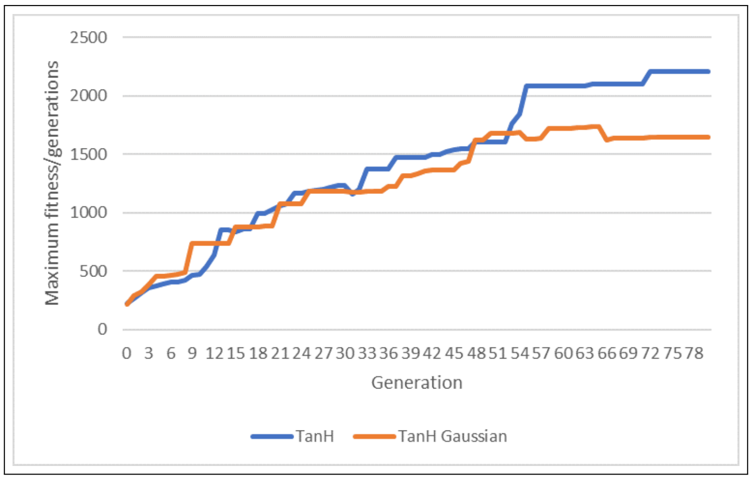


Figure 5.2: Comparison between activation functions

In figure 5.2, the normally distributed random variable produces a high fitness value in the first ten generations than the non normally distributed random variable. For the next twenty generations the two algorithms moved at same pace. Finally, a great difference exits between the normally distributed random variable leaning rate and the non normally distributed random variable. Therefore, the random variable which does not have a normal distribution is faster at learning.

- **Task 3 - Comparison between different activation function mutation rates**

  This task is carried out to check which mutation rate is best to our additional activation function mutation implementation.

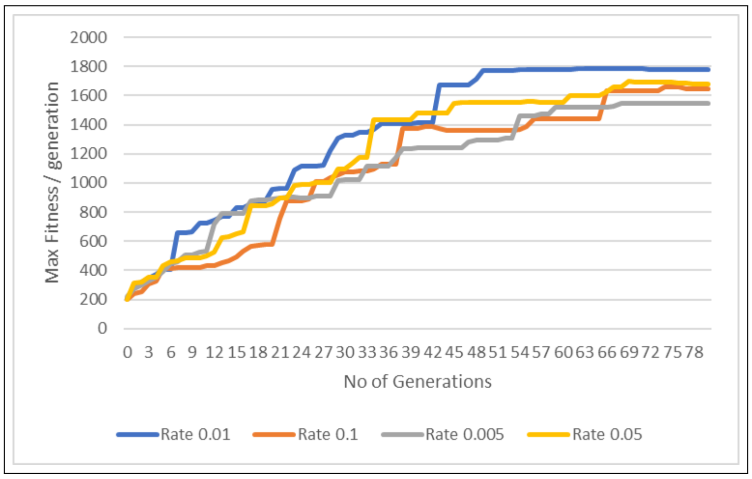


Figure 5.3: Comparison between activation function mutation rates

In figure 5.3, all the lines with different mutation rates overlapped each other for the first ten generations. The activation function mutation rate which provides the highest learning rate is 0.01. So, the best possible activation function mutation rates for obtaining higher fitness value is in the range from 0.01-0.05.

5.3 Usability Evaluation

The application interface was evaluated based on 10 individuals above the age of 18 with knowledge in Machine Learning and Artificial Intelligence. The candidates were very eager to test the functionalities of the application.

Type	System/User	Description	Requirement	Status
Functional	User	The user shall be able to select a game on which he wants to run a machine learning algorithm	U-FR-1	Tested
Functional	User	The user shall be able to select machine learning algorithm to execute on the game	U-FR-2	Tested
Functional	User	The user shall be able to modify settings of the ML algorithm	U-FR-3	Tested
Non-Functional	User	Interactivity	U-NFR-1	Tested
Non-Functional	User	Usability	U-NFR-2	Tested ($^1/_2$)
Non-Functional	User	Quality	U-NFR-3	Tested

Table 5.2: Tested User Requirements

The requirements specified in table 5.2 were tested through the use of a pre-survey questionnaire and post-survey questionnaire. The pre-survey questionnaire was aimed towards assessing the interest of the participants on machine learning models, and their interest with games. Following the pre-survey questionnaire the candidate was asked to run a series of tasks. The post-survey questionnaire on the other hand was aimed towards evaluating the functional and non-functional user requirements. **U-NFR-2** was only partially tested since the application does not have an help section.

The pre-survey questionnaire has seven questions. Only the last two questions of the questionnaire have a correct answer, while the rest are only general questions to assess the candidates knowledge. The questions are as follows:

1. Rate your knowledge of Machine Learning Models
2. Are you interested in learning about machine learning algorithms?
3. Have you ever heard of Neuroevolution of Augmenting Topologies?
4. Have you ever heard of Deep Reinforcement Learning?
5. Are you a fan of gaming?
6. What does the genre ’platform games’ mean?
7. Which of the following games are platform games?

Out of the above seven questions, we have chosen to present three results since they have very interesting metrics along with a description for each about what makes the result so interesting.

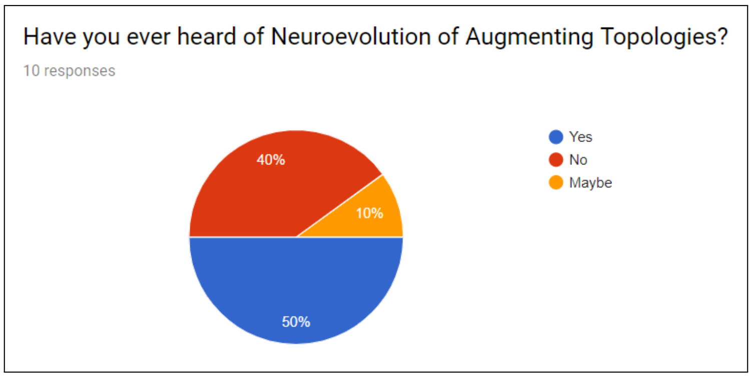


Figure 5.4: NEAT Pre-Survey Result

Figure 5.4 presents the results for a question which checks if the candidate has any knowledge of NEAT algorithms. We can notice that out of the ten candidates, half of them are aware of NEAT. This tells us that NEAT was moderately popular amongst the candidates.

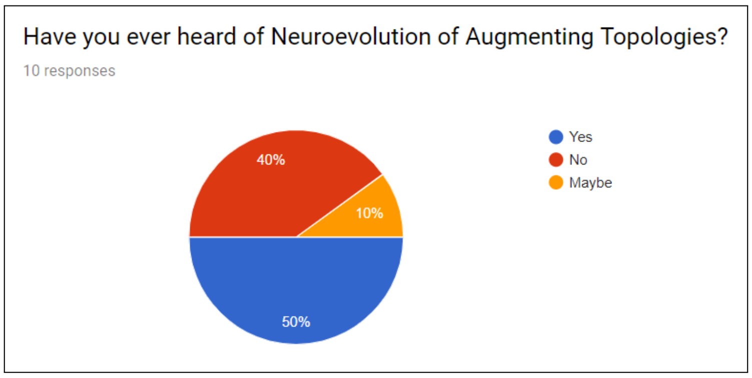


Figure 5.5: Deep Reinforcement Learning Pre-Survey Result

Apart from asking if they had any knowledge of NEAT the candidates were also asked if they had any knowledge of Deep Reinforcement Learning. Figure 5.5 proves to be interesting since it can be clearly seen that a majority of candidates know of Deep Reinforcement Learning. When comparing the results of figure 5.5 with figure 5.4 it can be seen that the candidates were a lot more aware of Deep Reinforcement Learning.

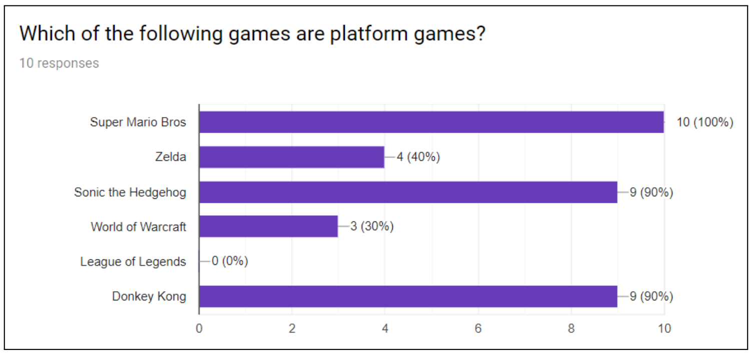


Figure 5.6: Platform Games Pre-Survey Result

The final result for the pre-survey (Figure 5.6) asks users if they were aware of which games were platform games, given the choice of five games. Out of the five games, only three games were actually platform games. The most interesting metric is that all the candidates who took part in the survey were entirely aware that Super Mario Bros is a platform game. The next two highest results are also correctly identified as platform games.

Once the pre-survey questionnaire was completed, the users were asked to run the application in a monitored environment. After completion, the candidates were asked to fill in a post-survey form which assessed the usability, and allowed us to place our user requirements to the test. The post-survey questionnaire consisted of five questions. The questions are as follows:

- What do you think of the design of the application?
- Please rate your experience of using the interface.
- Was the interface difficult to navigate?
- What did you find to be the most appealing aspect of the application?
- Are there any other features you would wish to see in a future release?

Out of these five questions, we take a closer look at three specific questions, out of which two are enough to accomplish the user requirements.

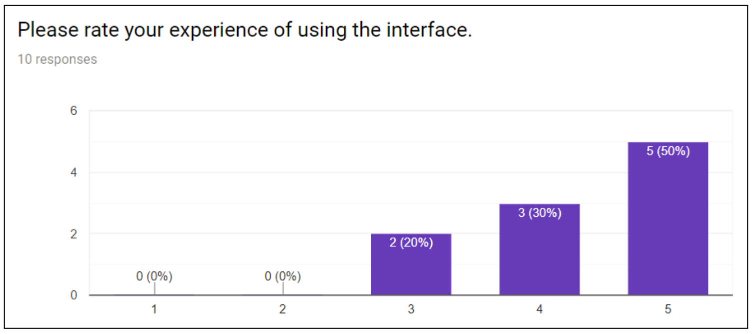


Figure 5.7: User Experience Post-Survey Result

Upon successful completion of the test, the candidates were asked about their user experience. Figure 5.7 presents the responses from the ten candidates, majority of whom seem to have had a very positive experience of using the interface. In regards to ’positive experience’, the general aim was for simplicity.

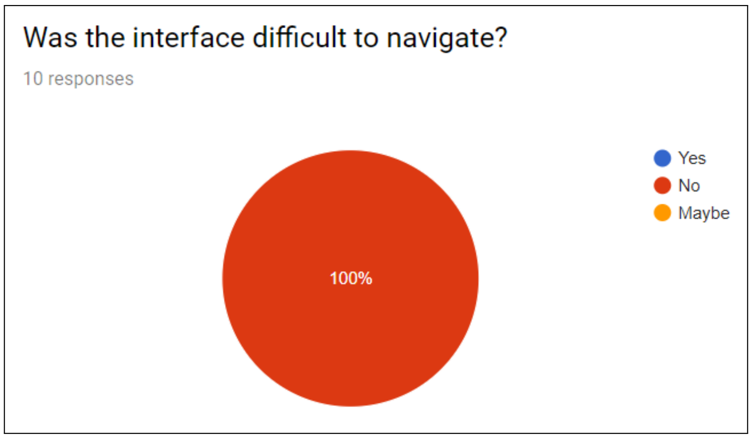
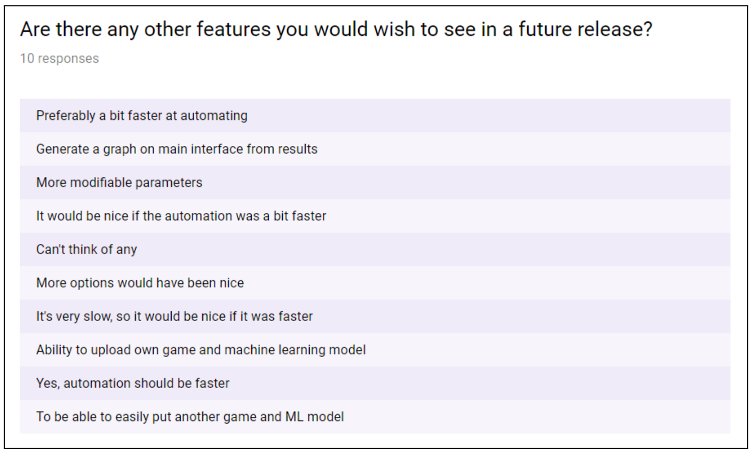


Figure 5.8: Navigational Difficulty Post-Survey Result

The candidates were also asked about how difficult it was to navigate from within the application interface. Figure 5.8 displays a stunning result in which all the users found it very easy to navigate through the system. This metric, along with the metric from figure 5.7 shows that the user functional and non-functional testing was completed successfully.



Are there any other features you would wish to see in a future release?

10 responses

- Preferably a bit faster at automating
- Generate a graph on main interface from results
- More modifiable parameters
- It would be nice if the automation was a bit faster
- Can't think of any
- More options would have been nice
- It's very slow, so it would be nice if it was faster
- Ability to upload own game and machine learning model
- Yes, automation should be faster
- To be able to easily put another game and ML model

Figure 5.9: Future Features Post-Survey Result

The final question the candidates were asked was about any features they would wish to see in a future release. Figure 5.9 shows that a majority of people wanted the speed of the automation of the system to be a bit faster, while others aimed towards being able to attach their own custom game and machine learning model. These results do not affect any user requirements, but were primarily added to analyse what candidates would be most interested to see, and provide possible innovation for future work.

Chapter 6

Conclusion

This sections outlines the achievements of the project. Its limitation in fulfilling expected functioning of the application. Finally, possible enhancements to the application.

6.1 Achievements

The application is able to automate the process of executing given machine learning models on user defined games. The machine learning models are constructed in a generic manner, which allows easy accommodation of different types of platform games. NEAT algorithm could be enhanced to improve fitness value faster by modification of existing activation function to a variation of tanh.

6.2 Limitations

The mouse positions used to open the lua console or load the state are dependent on screen size and resolution, which causes a problem when running on another system. There was lack of proper documentation in regards to the game relevant memory addresses in the case of NEAT. The rewards were being calculated on actions rather than the distance travelled. The score displayed (on the emulator) for Mario does not increase based on the distance of the player. The above two reasons caused problems while implementing a game on the generic machine learning model.

6.3 Future Works

The software can be improved up by uploading a ML model script from the user and running it against a user chosen game. As the implemented generic machine learning models can only be extended to platform based game, it is a limitation that could be exploited to other game genre.

References

[1] F.-H. Hsu, *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2004.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[3] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, Jan. 2018, ISBN: 9783319635187.

[4] J. Togelius, "Ai researchers, video games are your friends!" In *Computational Intelligence: International Joint Conference (IJCCI)*, vol. 669, 2016. arXiv: `1612.01608 [cs.AI]`.

[5] S. Karakovskiy and J. Togelius, "The mario ai benchmark and competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, 2012.

[6] T. Murphy VII, *The first level of super mario bros. is easy with lexicographic orderings and time travel... after that it gets a little tricky.* 2013.

[7] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *CoRR*, 2012. arXiv: `1207.4708`.

[9] M. McPartland and M. Gallagher, "Reinforcement learning in first person shooter games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 1, 2011. DOI: `https://doi.org/10.1109/TCIAIG.2010.2100395`.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[11] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. P. Liebana, "Deep reinforcement learning for general video game ai," Jun. 2018.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, 2013. arXiv: `1312.5602`.

[13] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/?stanley:ec02.

[14] J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria, “On the performance of indirect encoding across the continuum of regularity,” *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 3, pp. 346–367, 2011.

[15] N. Hansen, “The cma evolution strategy: A comparing review,” in *Towards a new evolutionary computation*, Springer, 2006, pp. 75–102.

[16] J. Gauci and K. O. Stanley, “A case study on the critical role of geometric regularity in machine learning.,” in *AAAI*, 2008, pp. 628–633.

[17] L.-J. Lin, “Reinforcement learning for robots using neural networks,” Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., 1993.

[18] G. Lample and D. S. Chaplot, “Playing fps games with deep reinforcement learning,” *CoRR*, 2016. arXiv: 1609.05521.

[19] A. Dosovitskiy and V. Koltun, “Learning to act by predicting the future,” *CoRR*, 2016. arXiv: 1611.01779.

[20] J. L. Nielsen, B. F. Jensen, T. Mahlmann, J. Togelius, and G. Yannakakis, “Ai for general strategy game playing,” *Handbook of Digital Games*, Mar. 2014. DOI: https://doi.org/10.1002/9781118796443.ch10.

[21] K. Kunanusont, S. M. Lucas, and D. P. Liebana, “General video game ai: Learning from screen capture,” *CoRR*, 2017. arXiv: 1704.06945.

[22] W. Woof and K. Chen, “Learning to play general video-games via an object embedding network,” *IEEE Computational Intelligence and Games Conference*, Jun. 2018. DOI: https://doi.org/10.1109/CIG.2018.8490438.

[23] *Scrum*, Mountain Goat Software. [Online]. Available: http://www.mountaingoatsoftware.com/agile/scrum (visited on 11/19/2018).

[24] *Scrum and kanban – are they that different after all?* Perfectial. [Online]. Available: https://perfectial.com/blog/scrum-and-kanban-are-they-different/ (visited on 11/19/2018).

[25] SethBling, *Mari/o machine learning for video games*. [Online]. Available: https://pastebin.com/ZZmSNaHX (visited on 02/14/2019).

[26] TASVideos, *Game resources*. [Online]. Available: http://tasvideos.org/GameResources.html (visited on 02/01/2019).

[27] C. Trivedi, *Using deep q-learning in fifa 18 to perfect the art of free-kicks*. [Online]. Available: `https://towardsdatascience.com/using-deep-q-learning-in-fifa-18-to-perfect-the-art-of-free-kicks-f2e4e979ee66` (visited on 02/24/2019).

[28] ——, *Deepgamingai_fifarl*. [Online]. Available: `https://github.com/ChintanTrivedi/DeepGamingAI\_FIFARL` (visited on 02/24/2019).

[29] E. Papavasileiou and B. Jansen, "The importance of the activation function in neuroevolution with fs-neat and fd-neat," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2017, pp. 1–7.

[30] R. Code, *Statistics/normal distribution*. [Online]. Available: `https://rosettacode.org/wiki/Statistics/Normal_distribution#Lua` (visited on 02/14/2019).

Appendix A

Setup & Execution of Application

The following steps must be followed in order to run the application:

Step 1: Ensure Python 3.6 is installed.

Step 2: Open the folder called ’Prerequisites’. Within this folder, there is a batch file named ’Install.bat’. Execute this file with administrator privileges and follow the steps within the installer.

Step 3: Ensure that a folder called ’Games’ exists in the ’Bizhawk’ directory. This folder houses the games needed to run the application.

Step 4: From the root directory of the project, open command prompt and type ’python interface2.py’.

The application can be downloaded from the following GitHub Link: —
The above mentioned link does not contain the ’Games’ folder.

Appendix B

Project Structure

Root Folder

- interface2.py (Python): This file contains the design and implementation of the main and options windows of the application.

BizHawk Folder

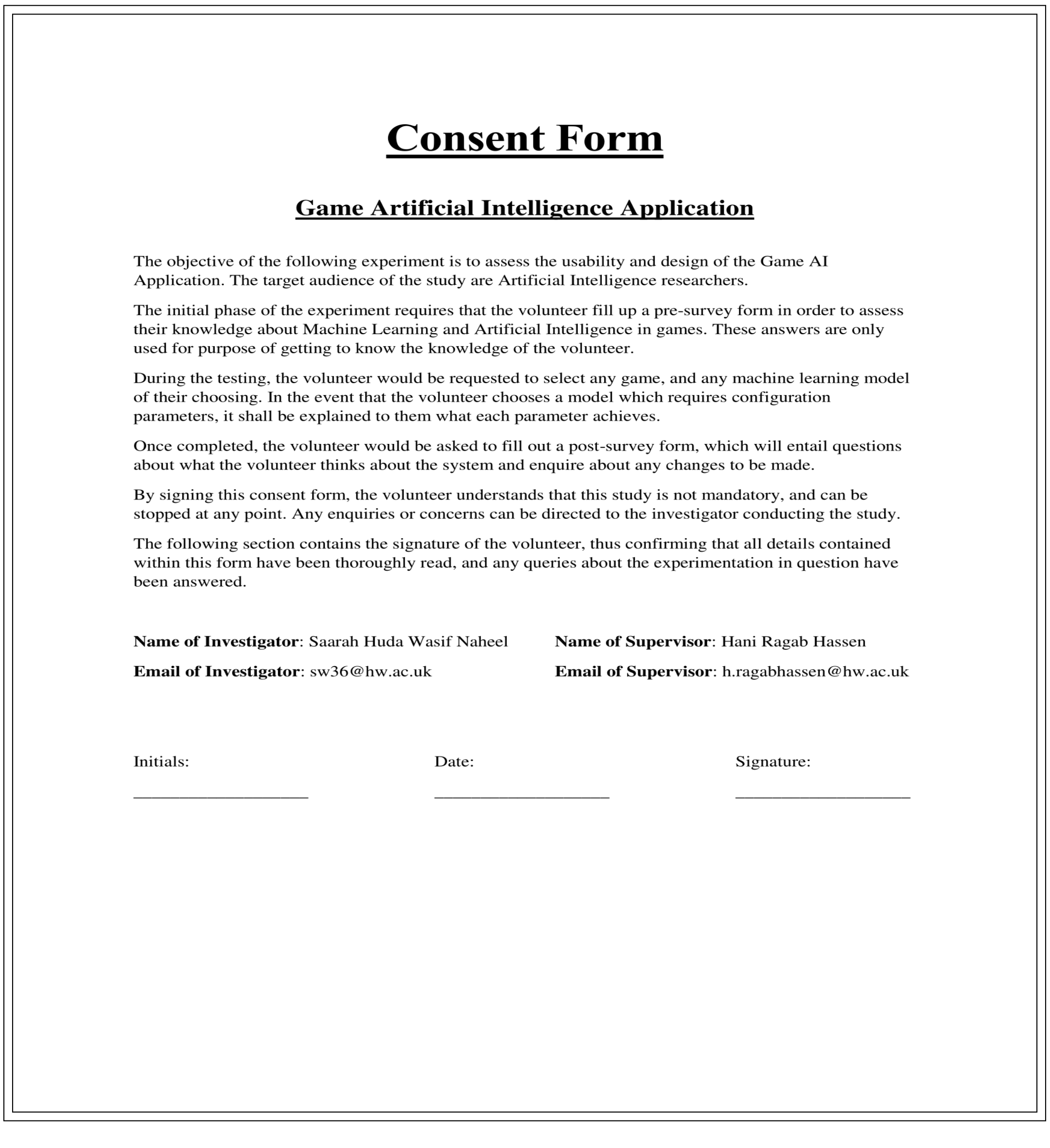
- neatevolve.lua (LUA): Contains a generic NEAT Algorithm. - neatevolve_ActFunc.lua (LUA): Similar to 'neatevolve.lua' but with specifiable activation function mutation rate. - supermariobros.lua (LUA): The controls and memory address locations for Super Mario Bros. - supermariobros3.lua (LUA): The controls and memory address locations for Super Mario Bros 3. - supermariobrosworld.lua (LUA): The controls and memory address locations for Super Mario World.

DeepReinforcementLearning Folder

- main.py (Python): Runs training for deep reinforcement learning on the chosen game (Super Mario Bros or Super Mario World). - MARIO.py (Python): Contains a class which acts as an API for the Super Mario Bros game by interaction through screen-grabs and key press simulation. - MARIOWORLD.py (Python): Similar features to 'MARIO.py', but meant for Super Mario World.

Appendix C

Consent Form



Consent Form

Game Artificial Intelligence Application

The objective of the following experiment is to assess the usability and design of the Game AI Application. The target audience of the study are Artificial Intelligence researchers.

The initial phase of the experiment requires that the volunteer fill up a pre-survey form in order to assess their knowledge about Machine Learning and Artificial Intelligence in games. These answers are only used for purpose of getting to know the knowledge of the volunteer.

During the testing, the volunteer would be requested to select any game, and any machine learning model of their choosing. In the event that the volunteer chooses a model which requires configuration parameters, it shall be explained to them what each parameter achieves.

Once completed, the volunteer would be asked to fill out a post-survey form, which will entail questions about what the volunteer thinks about the system and enquire about any changes to be made.

By signing this consent form, the volunteer understands that this study is not mandatory, and can be stopped at any point. Any enquiries or concerns can be directed to the investigator conducting the study.

The following section contains the signature of the volunteer, thus confirming that all details contained within this form have been thoroughly read, and any queries about the experimentation in question have been answered.

Name of Investigator: Saarah Huda Wasif Naheel
Email of Investigator: sw36@hw.ac.uk

Name of Supervisor: Hani Ragab Hassen
Email of Supervisor: h.ragabhassen@hw.ac.uk

Initials:	Date:	Signature:
__________________	__________________	__________________

Figure C.1: Consent Form Sample