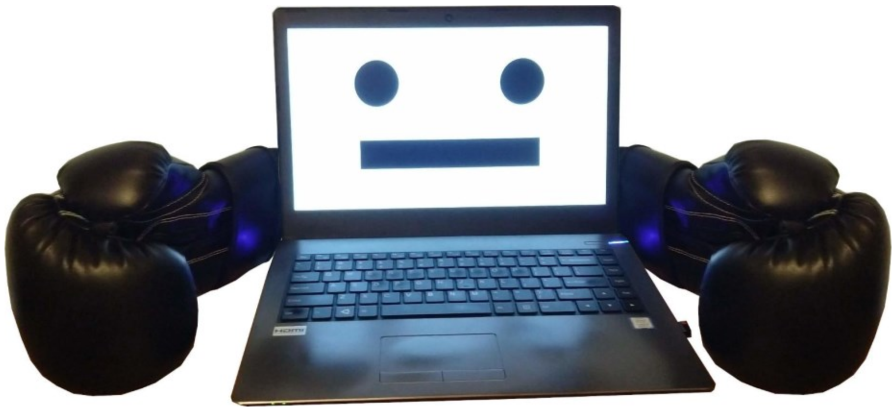
Machine Learning Arena:

Creating an ML Based Game



A Major Qualifying Project
Submitted to the Faculty of Worcester Polytechnic Institute

By

Grant Ferguson

Jordan Cattelona

Justin Kreiselman

Kyle Corry

Date: 12/11/2019
Project Advisors:

Professor Gillian Smith, Advisor

Professor Jacob Whitehill, Advisor

ABSTRACT

We present a new game, “MLA: Machine Learning Arena”, in which the player’s goal is to train a machine learning agent to win a boxing match. The game features multiple phases, fully animated characters for the player to control, and machine learning integration. We tackled several technical and design challenges in this MQP, including: 1) Communicating machine learning progress through UI elements to the user, 2) Training an effective model for the game agent despite poor training examples from users, 3) Explaining key ideas about machine learning to players with no background in the field. We conduct user testing with 27 human participants to determine if players feel that the ML is learning from them. We found that 85 percent of players were able to distinguish between a random agent and the trained agent. Finally, we offer suggestions for future development: 1) Find new ways to explain machine learning to players through either gameplay or UI metrics, 2) Sanitize the input to make machine learning feel more fulfilling to players, 3) Budget enough time to design, implement, and train machine learning models.

ACKNOWLEDGMENTS

We would like to acknowledge our Professor Advisors, Jacob Whitehill and Gillian Smith, for their oversight and advice on our project. We would also like to thank our human testers who helped to make the game experience better for any future players. Finally, we would like to thank the WPI Computer Science and Interactive Media and Game Development Departments, both for their support throughout the project and throughout our time at WPI.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Machine learning (ML) is a subset of artificial intelligence which infers intelligent behavior from examples rather than from explicit instructions of what to do. There are several major categories of machine learning algorithms - supervised, unsupervised, and reinforcement learning - of which we explored both supervised and reinforcement learning. In reinforcement learning, an agent will learn how to take actions to maximize a reward. A reward is some value that can be mathematically calculated in a way determined by the developer. In supervised learning, the agent will derive a function to map the given inputs to their corresponding outputs. Then, once presented with new input, the agent can infer how to act from the connections taught to it. When used in video games, these techniques have distinct applications. Reinforcement learning can be used to create a mapping from observations to actions via agents playing a game repeatedly, such as in DeepMind's "AlphaGo" program [34]. Supervised learning can be used to record gameplay from a human player and optimize a policy to mimic their behavior. This strategy was used to form the initial behaviors of "AlphaStar," DeepMind's AI designed to play *Starcraft 2* [38].

In this project, the goal was to create a video game that was enhanced by interaction with a machine learning agent (called Roboxer). By making such an agent integral to the function of the game, the player's experience will be defined by their interactions with Roboxer. To serve this goal, we sought to make the experience of training and interacting with Roboxer as enjoyable as possible, as well as to provide insight into the function of machine learning algorithms in video games.

Our team faced multiple challenges in incorporating machine learning into a video game. Many of these difficulties had their roots in communicating information to the player. Informing the player what Roboxer was learning and guiding the player how best to teach Roboxer became heavily scrutinized problems. Because the player's goal was to train a boxer to win a fight, we needed to develop a method of showing their progress. Simply showing that Roboxer was learning from player actions wasn't enough when there was a chance that the player was training them to be a poor fighter. Not only did this information need to be brought across to the player, but it needed to be communicated clearly and efficiently during the middle of the game. Other difficulties arose while trying to make training Roboxer satisfying to the player. It was difficult at first for players to see their impact on the Roboxer's behaviors, and their enjoyment dropped.

The discovery of and solutions to these problems are discussed in chapter five of the report, where we detail the development process.

2 BACKGROUND

2.1 The Fighting Game Genre

Fighting games are a genre of video games in which players control characters to compete in fighting matches against one another or against computer-controlled players. In fighting games, characters use a wide range of fighting styles from hand to hand combat, to weapon-based combat and also magic-based combat. Within the fighting game genre, there are multiple sub-genres such as arena fighters, tag-team fighters, 3D fighters, and 2D fighters. Arena fighters are fighting games in which the player-controlled characters can freely move in a 3D space to fight one another such as *Dissidia Final Fantasy NT [36],* and *Naruto: Ultimate Ninja Storm [12].* Tag team fighters are fighting games in which the player controls a team of two or three characters that can be swapped out during the match. Notable tag team fighters include *Marvel Vs Capcom 2: New Age of Heroes [9]* and *Dragon Ball FighterZ [2].* 3D fighters are fighting games in which the player-controlled characters move in a 2D plane for a majority of the match while having three-dimensional movement options such as sidestepping to avoid incoming attacks. 3D fighting games include *Tekken* [4] and *Dead or Alive [37].* 2D fighters are fighting games in which the player-controlled character is locked in a 2D plane of movement, usually only being able to move left and right. Some 2D fighters are *Mortal Kombat [3], Street Fighter [8],* and *Divekick [21].* We position our game as a 2D fighter because the game is confined to a 2D movement plane. The 2D fighter game that has most inspired this project's mechanics and style is *Punch Out!! [41].*

2.1.1 Punch Out!!

Punch Out!! is a game created by Nintendo in 1987 in which the player controls a boxer rising through the ranks of the world circuit boxing tournament. The game is played over 23 matches with the player facing off against a wide range of characters each with different characteristics and fighting styles. In each match, the player will have three rounds to knock out the opponent. This can be achieved by using a combination of attacks including body blows, punches to the face, dodging, ducking, blocking and an uppercut [41]. The player performs these actions by pressing the A and B buttons and the directional pad on the Nintendo Entertainment

System controller to punch and dodge. The full controls of the game can be seen in Figure 1 below.

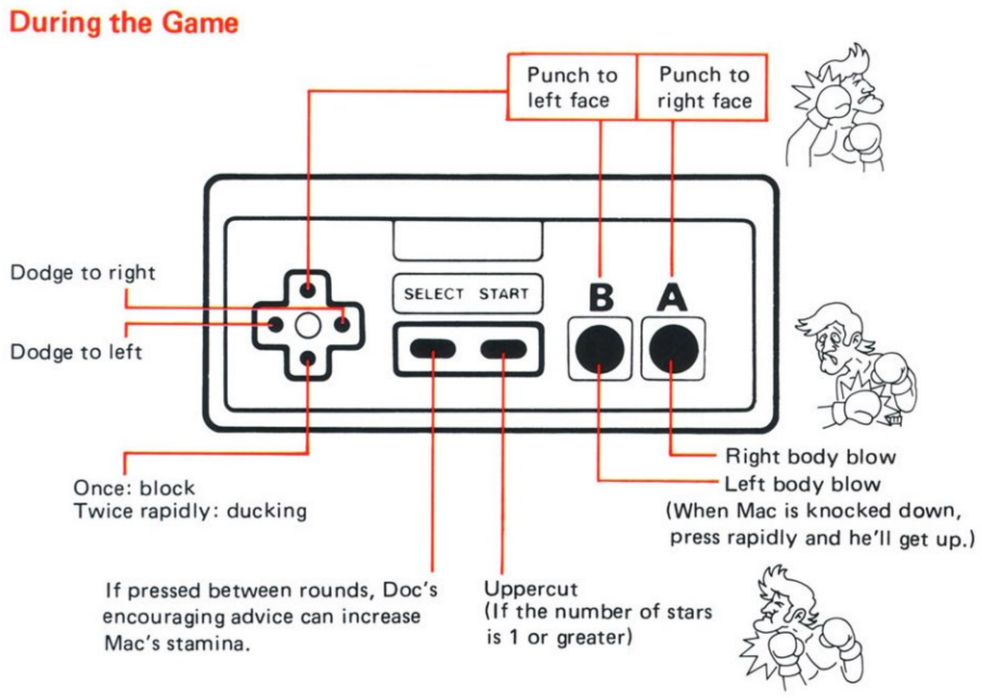

During the Game
Punch to left face
Punch to right face
Dodge to right
Dodge to left
SELECT START
B A
Once: block
Twice rapidly: ducking
Right body blow
Left body blow
(When Mac is knocked down, press rapidly and he'll get up.)
If pressed between rounds, Doc's encouraging advice can increase Mac's stamina.
Uppercut
(If the number of stars is 1 or greater)

Figure 1: *Punch Out!!* control page [41]

Punch Out!! was used as the main source of inspiration for our project's gameplay mechanics, style, and controls due to its simple, yet effective design that works well for training a boxer with machine learning. This is because in *Punch Out!!* players do not move around the ring which reduces the state space we need to consider for the machine learning system. This also allows us to have the players only focus on the core actions of the fight which are the punches and dodges. The gameplay mechanics are inspired by the multiple rounds the player would play through to win the tournament. This influenced our decision to use boxing as the fighting style for the project.

2.2 Machine Learning Techniques in Video Games

Machine learning (ML) can be used in many different components of a video game, such as human substitution, dynamic difficulty adjustment, player adapted game elements, and enjoyable opponents [35]. Most games incorporating machine learning have done so by training an ML model with human examples (imitation learning) or via playing the game (reinforcement learning) rather than a developer needing to write a preset list of rules [24,25]. To add ML into a game, a developer must allow a game agent to collect state observations, perform actions, and have a predefined goal that it should reach [35]. A downside to using ML models is that they typically require a large amount of training data and training time to converge on an optimal solution [11].

Reinforcement learning involves an agent acting within the game environment to find optimal strategies that maximize a developer or player specified reward [35]. To tune the underlying ML model, an agent plays a game multiple times and converges upon a local maximum determined by positive and negative rewards. On the other hand, imitation learning (IL) is a form of supervised learning in which play traces from a human player are gathered and used as training data. When given enough play traces, imitation learning can train an agent to behave similarly to a human player [20]. Our game uses imitation learning as the primary learning technique, as it fits into the goal of having a human player teach an AI how to fight. Early versions of our game also contained a training phase that was created using reinforcement learning, so the boxer could pick up moves of its own; this phase was removed in later versions of the game in favor of only using IL (see Section 5.5).

Human substitution has received a large amount of publicity in the past few years as algorithms such as Deep Reinforcement Learning have been able to produce agents with an expert-level performance for some games [34]. Researchers have used ML techniques to play games such as a variety of *Atari* games [27], *DotA 2* [10], *StarCraft [30,38]*, *Go [34]*, capture the flag [23], and shooting games [31,32]. In the industry, human substitution can be seen in *Forza Motorsport* Drivatars, which use imitation learning to drive with the same style as the player and substitute for them [26].

Dynamic difficulty adaptation (DDA) involves a game agent scaling its difficulty to meet the skill level of the player. ML researchers have found recent success by applying RL to the problem which will approximate the skill level of the player and adjust the game AIs accordingly [17,22]. Many production video games, including *Crash Bandicoot* [15,42] and *Left 4 Dead* [28], use a form of DDA which does not specifically use reinforcement learning algorithms but has the effect that the game's difficulty fits the current player over time. Using ML in place of traditional

DDA has the potential to produce a custom DDA for each player which can adapt to their skill level and behavior; there are not any prominent industry examples as it is still in the realm of research and many of the existing DDA algorithms are suitable for players [29]. On the other hand, ML has been used to adapt game elements to better fit the player, for example, *Galactic Arms Race* uses neural networks to create unique weapons for each player based on their playstyle [18].

Many video game developers have been researching ML approaches to make more interesting and enjoyable opponents. For example, *Colin McRae's Rally 2* utilized imitation learning to control the driving of the other racers in the game, which allowed the cars to feel more challenging to players [6]. *Race for the Galaxy* is another example that uses a neural network that plays the card game as an opponent to the player to “offer new challenges over and over again” as its behavior is less predictable than that of an expert system [13]. Other large game developers are researching the improvements that ML could bring to their games, such as *Battlefield*’s efforts using imitation and reinforcement learning to create challenging enemies and *Blade and Soul*’s fighters which use reinforcement learning [11,14]. Overall, ML can make agents which make decisions that are less predictable/scripted than heuristic-based agents [13].

2.2.1 Popular Video Game ML Algorithms

The Unity framework used by our game, ML Agents, provides several learning algorithms frequently used in games: Proximal Policy Optimization (PPO) [33] - a reinforcement learning algorithm - and two imitation learning algorithms: Generative Adversarial Imitation Learning (GAIL) [19] and Behavioral Cloning (BC) [40]. Effectively using these algorithms in relation to a boxing game was a major goal of our project, and we were able to experiment with all three.

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that is used to train a neural network, known as the policy network. PPO uses a metric known as the Q-value, which is a prediction of the reward value at a state given that a specific action was taken [33]. At each timestep, a value, known as the advantage, is calculated as the difference between the predicted (Q-value) and actual rewards. A ratio is then calculated between the new and old policies’ outputs. The ratio and advantage are used to clip the objective function of the neural network, reducing the likelihood of the gradient steps from overshooting the goal. Stochastic gradient descent is then run on the policy network. This algorithm has been shown to outperform other gradient-based reinforcement learning methods [33]. *OpenAI 5*, the *DotA* playing RL agents, were trained using self-play and large-scale parallel PPO algorithm. The *OpenAI 5* researchers used PPO to deal with long time horizons and data from “180 years worth of games against itself every day” [10]. In early versions of our project, we used this algorithm along with GAIL to handle the training phases of the game (see Section 5.2 and 5.3).

Generative Adversarial Imitation Learning (GAIL) is an imitation learning algorithm that can train a neural network using reinforcement learning techniques. Agents trained using GAIL demonstrate human-like skills with far fewer play traces and training time than traditional reinforcement or imitation learning algorithms [25]. Internally, GAIL is based on a reinforcement learning algorithm and built on top of a neural network to act as a policy. GAIL works by training two neural networks. One network, the “discriminator,” is used to distinguish between the training agent and an expert agent (from a recorded demo) and will produce a real value that describes how much the two agents differ. The value from the discriminator is used as a reward signal to the policy network, which will use an algorithm such as PPO to train. As the policy network trains to match the expert agent, the discriminator will be training to better distinguish between the two agents using a supervised learning approach [19]. GAIL can be combined with other reinforcement learning approaches by using the value of the discriminator network as a reward during training. A downside of GAIL is that it requires play traces to train the discriminator network, which need to be recorded before it can train the policy network. We were unable to use GAIL in our final game because of its pre-recorded play trace requirement and we wanted our agent to learn in real-time for the player to see; instead, we used Behavioral Cloning.

Behavioral Cloning (BC) is an imitation learning algorithm that can be used to copy the behavior of an agent in real-time. BC records input-output pairs from one agent (which may be human-controlled) and conducts supervised learning through mini-batch gradient descent on a neural network. BC can also work in real-time through the use of an experience buffer; the experience buffer is a collection of teacher input-output pairs. As a human plays the game, the experience buffer will grow, allowing BC to train on more data. If the experience buffer is cleared, BC can be used to learn a new behavior within a few training iterations (possibly forgetting the old behavior in the process) [40]. BC cannot be combined with reinforcement learning approaches such as PPO. In earlier versions of our game, we were unable to use BC because it conflicted with PPO, but after we removed the defensive phase of training we found BC to be a good fit for our game (see Section 5.5).

2.3 Choosing Machine Learning Over Traditional AI

There are many options for creating artificial intelligence agents for use in video games. Certain techniques are better suited than others for some tasks, but often multiple approaches are viable for any given task. This raises the question as to why our team decided to pursue machine learning over traditional game AI techniques. Because our team wanted to make interaction with an AI agent a central part of the gameplay mechanics, and because we focused on the idea of teaching an agent, a machine learning approach was selected. Arguably, other methods could have been used to replicate the experience of training an AI agent. A state machine could be used to switch the agent between levels of training depending on how long a player had spent with the agent. However, this approach would make it far more difficult to express each user's unique method of training. Multiple end states could be programmed that match common strategies of playing, but the complexity of the state machine would rise as we tried to individualize the experience. Pursuing machine learning allowed us to create a system that could express individualized output for every user.

Machine learning certainly has downsides compared to other techniques. While no AI system is completely predictable one hundred percent of the time, having a state-space hard coded by the developer limits the possibilities of what the agent could be "thinking." If a machine learning agent does something unexpected, it can be more difficult to understand why it made the decisions it did, and at what point it learned to make those decisions. In addition, training time is an incredibly important factor. Depending on the method used, training an agent can take several hours, but even a few minutes spent doing nothing or doing something boring could kill a player's investment in the game. The challenges we faced during the project were to maximize the benefits of choosing machine learning while minimizing the risks and drawbacks.

3 MACHINE LEARNING ARENA

This chapter describes the final game delivered as a result of this project. The final game executable can be installed by following the instructions in Appendix D. Certain terms are consistently used to describe aspects and components of the project. They are as follows:

- **Coach**: The human player sitting at the computer
- **Roboxer**: The ML Agent the coach is attempting to train
- **Enemy / Final Boss**: The AI opponent that Roboxer will fight
- **Training Phase**: Game time that the coach spends training Roboxer
- **Fighting Phase**: Game time where Roboxer fights the Enemy

A complete glossary of terms can be located in Appendix F.

3.1 Experience Goal

The intended player experience that we present here was developed through all of the design and development that will be described in the following sections.

The experience that we seek to give players is the satisfaction of training a machine to fight. The way that we gave players that experience was by using machine learning to give an avenue for the computer to pick up on what the player is doing, as well as crafting a system around that machine learning component as a scaffold. The reason that we think this was the best way to give players this experience was that phrasing the learning as a game not only gets players invested and interested in machine learning, but also does it in a way that players can understand. The reason why this was the experience goal we set for ourselves and this project is that when we looked at what would make players interested in a game and in machine learning, we saw that different parts of real-life training would be the most adaptable to a video game. Of those types of real-life training, boxing training is one that is somewhat ingrained in the public area of knowledge, from movies and other older games that use boxing as a backdrop for their stories, not to mention the history of boxing as a sport.

3.2 Audience

Our final intended audience has two components, each we developed after our many rounds of testing and seeing who was interested in playing our game. The first segment of our audience is those who are knowledgeable about machine learning from an academic background. With our game having a focus on the machine learning element, having players who understand that on a deeper level helps to give them a better experience when playing. This segment of the audience was much smaller than the second grouping that we had during playtesting and iterating.

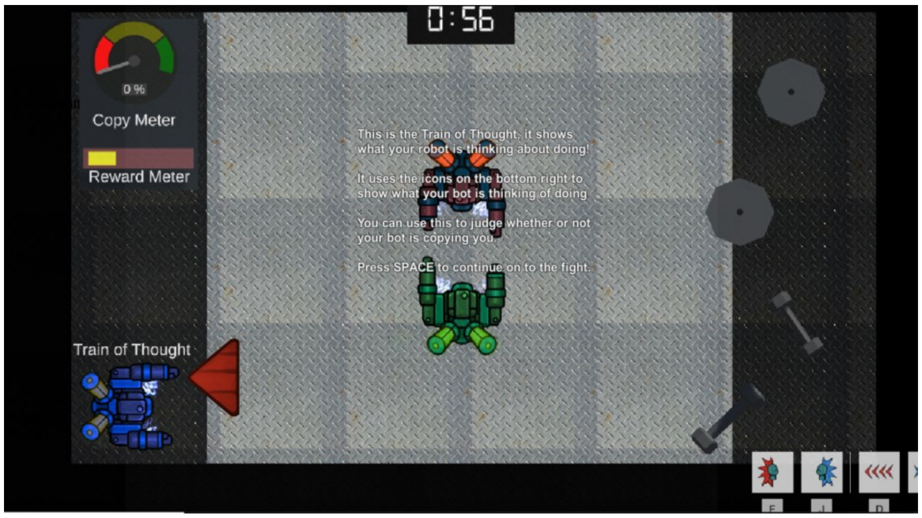
Our second segment of the intended audience is what we can call our general testing audience. This includes players who have played games or play games somewhat regularly, however, they do not have experience in developing AI or machine learning in other contexts. This audience includes the bulk of the testers that had played our iterations, and includes the majority of our expected audience for the game going forward.

3.3 Gameplay

3.3.1 Phases

The main phases of the game include 4 parts;

1. Tutorial. Takes the player through the basic concepts of the game as well as the basic controls. The players are given a screen to input each control once, to learn how to punch and dodge to each side. After those screens, the player is prompted to read through explanations of other game concepts, such as the learning mechanics and UI elements that show the learning components (see Figure 2).



This is the Train of Thought, it shows what your robot is thinking about doing!

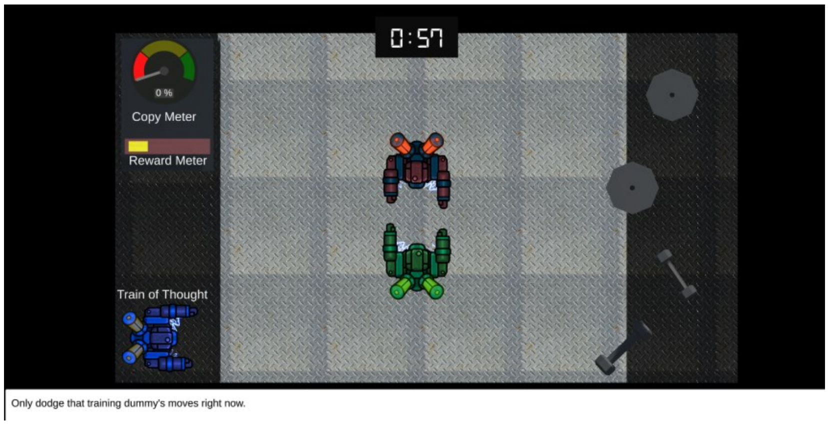
It uses the icons on the bottom right to show what your bot is thinking of doing

You can use this to judge whether or not your bot is copying you.

Press SPACE to continue on to the fight.

Figure 2: Training stage

2. Guided Training. This includes a short section directly as the player enters this area, where the enemy training dummy will only punch for a short time, with a text box explaining to players to dodge these attacks without punching (see Figure 3). After this, a similar text box will appear explaining that the players are to now throw sequences of punches at the training dummy. The training dummy does not punch back during this session.



Only dodge that training dummy's moves right now.

Figure 3: Guided training stage

3. Free-Play Training. The final part of the game is a free area for players to attempt to train Roboxer to the best of their ability, by both throwing punches and dodging incoming attacks. There is little explanation given during this section

relative to the guided phase. After the time runs out on this section, the players are prompted to either continue to the final area of the game or go back to train once again (see Figure 4).

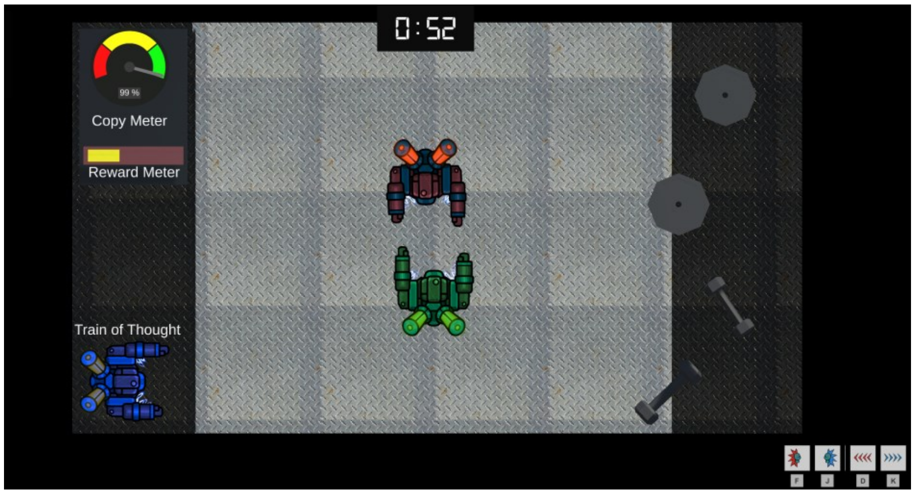


Figure 4: Free-play training

4. Match. The final area of the game is the match itself, where the player's agent takes on the computer-controlled enemy AI (see Section 4.4.2) without input from the players during the fight. This serves as a capstone to the game to show what Roboxer has learned from the player. The match itself takes place on the final match location and each boxer has health bars, to make sure the players know clearly how well their bot is doing (see Figure 5). When the health of a boxer is depleted, they are knocked-out and the winner is declared, thus ending the game.

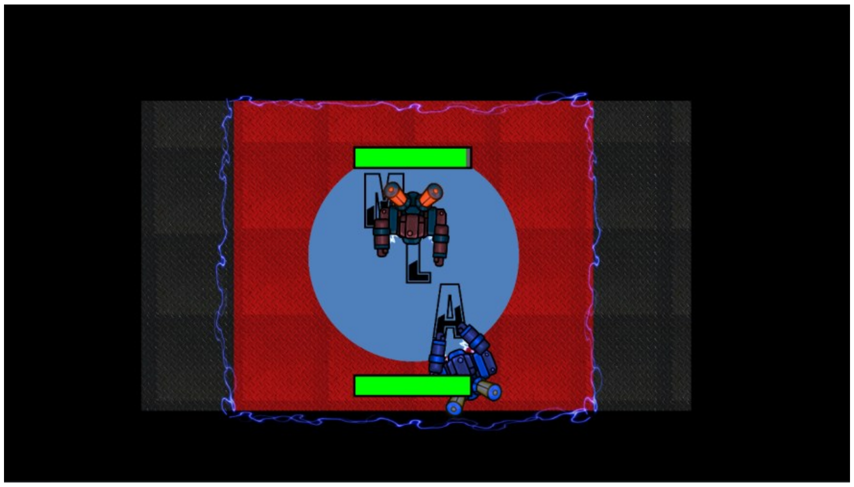


Figure 5: Final fight stage

3.3.2 Mechanics Design

Our game features four possible actions that a player can perform: a left and a right variant of punching and dodging. When hit by a punch, a boxer will take damage and flash red to indicate that they have taken a hit. All damage amounts are the same (10HP) in our game, and all players have 100HP, except for the final boss which has 400HP. The player can also attempt to dodge an incoming punch by dodging toward the opposite direction of the incoming punch. If the player dodges in the same direction as an incoming punch, then they will receive damage. We allow players to hold a dodge indefinitely to reduce the need for exact timing.

3.3.3 Game Controls

The controls for the game use the D, F, J, and K keys on the keyboard to allow players to perform punches and dodges. The D and K keys are used for dodge left and right respectively (can also be held to extend the duration of a dodge) and the F and J keys are used to punch left and right respectively. Those keys were chosen because when a person uses a keyboard properly the hand resting position has the fingers over those keys. They were also chosen to help differentiate the left and right side actions by having the keys be separated on the left and right side of the keyboard. Also, most US keyboards have indentations on the F and J keys. We use this to have the players quickly get back into position if they need to remove their hands from the keyboard for any reason without needing to look away from the screen.

3.4 Narrative

The narrative we constructed for the project was designed to serve the previously stated experience goal. In short, the intent was to provide the player with the experience of teaching a student and seeing their own strategies emerge in the actions of their trainee. To that end, a base narrative was created involving a robot boxer and their protégé (called Roboxer). The player controls a robotic fighter past their prime. They take up an apprentice and teach it to fight in their stead. After enough training, the apprentice attempts to win its first official fight.

Due to a desire to focus on the technical nature of the project, less time was spent developing and communicating the narrative. We knew that the other aspects of the game would be far more demanding, so the narrative was kept light. However, we still wanted to make sure that a light narrative was communicated effectively and served the experience goal. The initial premise is explained during the tutorial that teaches the player the rules. The gym environment during the training phase helps to cement the idea of working out and training. The final arena suggests a climactic conclusion to the training given up to that point.

In addition to the base narrative we laid out for the players, the machine learning aspect of the game would provide its own narrative elements. When an AI agent in a game takes an action, players tend to attribute that action to some underlying motivation or reasoning [16]. This is discussed in the article "Anthropomorphism and AI" as "The Mirror Effect." Our thought was that by having the ML agent learn from the player and take actions that were noticeably inspired by the player's training, the experience would tie into the narrative of training an apprentice. By seeing the Roboxer fail or succeed, the players could create a narrative in their heads as to the reason the Roboxer failed or succeeded. Thus, even though the player is only explicitly given a small bit of narrative, they can infer more from the state and progress of gameplay.

3.5 UI Design

The UI consists of the "Copy Meter", the "Reward Meter", and the "Train of Thought" on the left side of the training screen. The training screen has the player and the training dummy at the center of the screen as seen in Figure 6. This was done to give players the ability to focus on the fight at the center of the screen while being able to see the meters and information given on the left side of the screen. The training screen also contains a small set of icons on the bottom right of the screen that shows the controls of the game, with the keys and the icons that match those appearing in the train of thought element.

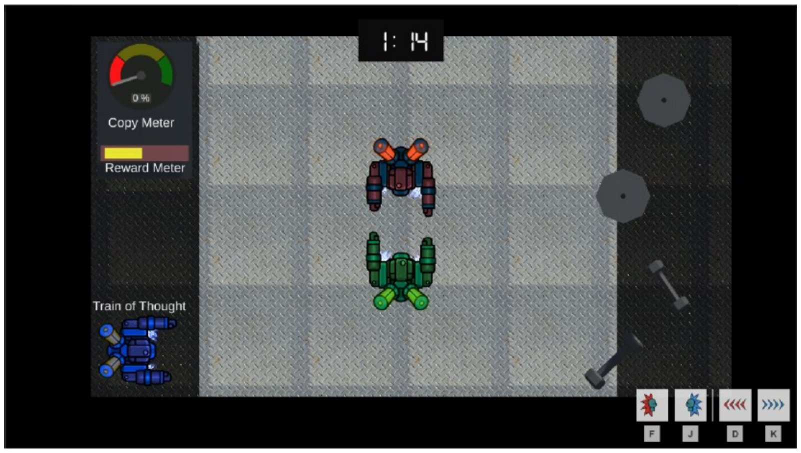


Figure 6: The training phase

3.5.1 Copy Meter

The copy meter shows how much Roboxer has learned from the player (using cross-entropy training loss - see Section 5.5.5). This is represented by a dial and a percentage as seen in Figure 7. The dial will light up when the percentage reaches a certain threshold. This was designed to allow the player to quickly glance over and see a general result while maintaining focus on the fight.

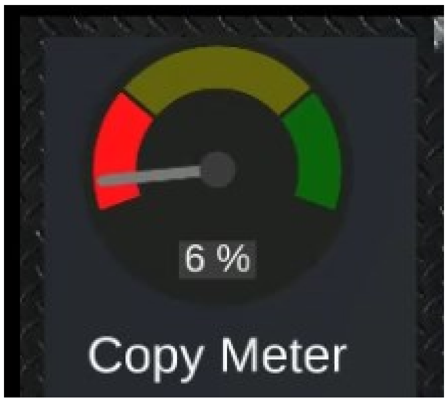


Figure 7: The copy meter

3.5.2 Reward Meter

The reward meter shows how much reward is given to Roboxer from the player's inputted action. This is represented as a yellow meter that increases in length when the inputted player action gives a high reward value as seen in Figure 8. This was designed to allow the player to see how well Roboxer is learning from the player.

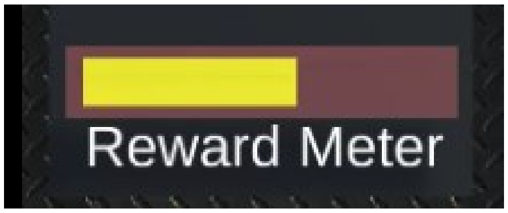


Figure 8: The reward meter

3.5.3 Train of Thought

The train of thought shows the actions that Roboxer is thinking to perform. This is represented by four icons moving up on the screen as seen in Figure 9. The four icons are the icons used for the controls and represent the left and right variants of punching and dodging and can be seen in Figure 10. The left and right variants are organized based on the color: left is colored red and right is colored blue. They are also organized by the side the icons spawn on relative to the center of the train of thought. These icons were designed to allow the player to quickly recognize what type of action their agent is currently trying to use while not fully diverting the attention away from the training in the center of the screen.

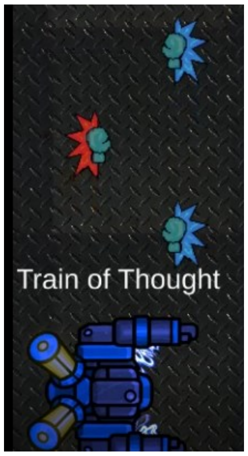


Figure 9: The train of thought

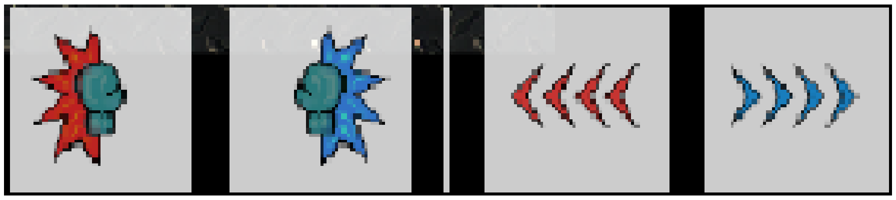
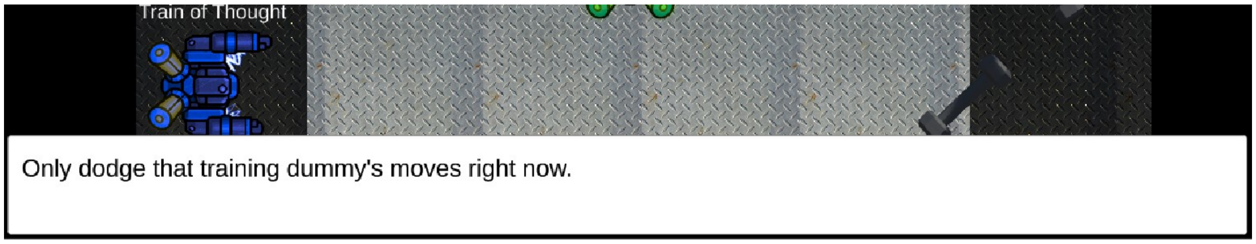


Figure 10: The train of thought and control icons

3.5.4 Coach Dialog Box

To assist with the transitions between phases, a dialog box is shown, which allowed the coach to speak to Roboxer as seen in Figure 11. This gives players some context into what they are supposed to do during each phase of the game and avoids harsh transitions. The dialog box works in tandem with the training dummy (see Section 4.4.1).



Only dodge that training dummy's moves right now.

Figure 11: The dialog box used to convey instructions to the player during the game

3.6 Visual Design

To reflect the theme of the project, using machine learning to train a boxer to fight, we wanted to incorporate machines into the level and character designs. The usage of machines in our game visuals was a way to imply the use of machine learning within our game.

3.6.1 Environment Art

To reflect the theme of the project, the environment art has been created to recreate what a robot boxing gym would look like. We used metal plates to simulate a training mat (see Figure 12). We used existing assets for the metal floor textures[1]. The texture was then taken into Photoshop to be adjusted to create two different color tones to help differentiate the mat from the

[1] https://www.deviantart.com/tmm-textures/art/Metal-Floor-42059858 Freeware

normal gym floor. This was also used to create an area for our UI elements that do not interfere with the characters on screen. To increase the feeling of the area being a gym, we included free weights and dumbbells scattered around the area. We also used existing assets for the gym equipment models[2].

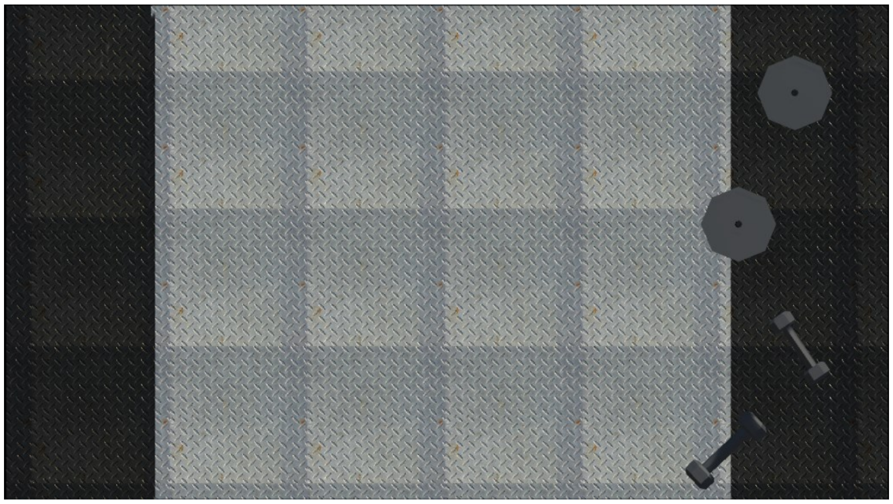


Figure 12: The final release of the training area

For the final fight arena, we used a similar arena used in the training section of the game. The gym equipment was removed, and a lightning square was added to simulate a fighting ring. The lightning[3] was used from existing assets along with the font[4] used in the center logo of the ring. The final fighting arena can be seen in Figure 13.

[2] https://jprinsloo.itch.io/free-low-poly-gym-pack License: CC0: Public domain, completely free to use in both personal and commercial projects
[3] https://assetstore.unity.com/packages/tools/particles-effects/lightning-bolt-effect-for-unity-59471 (c) 2016 Digital Ruby, LLC
[4] https://www.fontspace.com/digital-graphics-labs/gunmetal Freeware

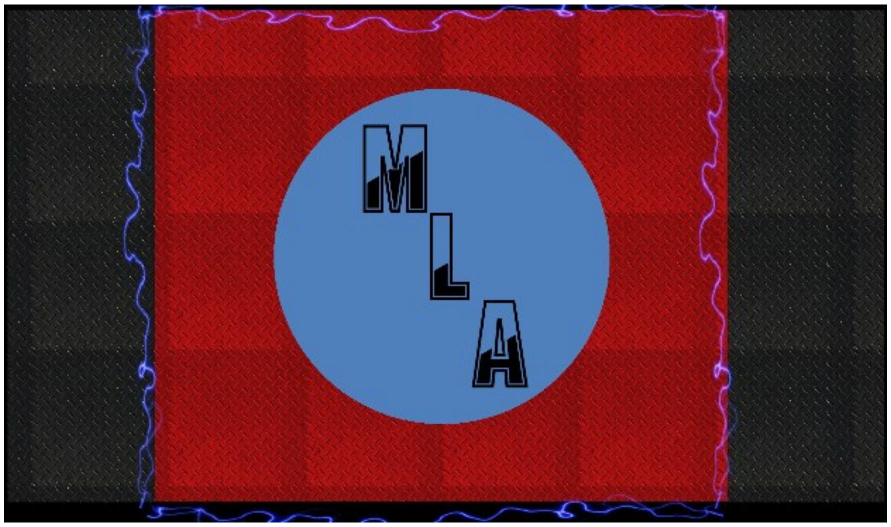


Figure 13: The final release of the fighting arena

3.6.2 Character Art

The character art is using a package from the Unity Asset Store called "Armored Soldiers and Monsters 2D" [5], and uses a singular part of the asset package, being one of the armored soldiers, shown in Figure 14. This sprite group is used in all of the characters and is recolored for different uses. The coach uses a green material, Roboxer a blue one, and the enemies use a red material. These materials overlay on top of the sprite. The animations of punching, dodging, and idling were created with the Unity Animation window. Specific parts of the sprite could be manipulated along a timeline to create complete animations. Unity's Animator window allowed for the construction of state machines that move from one animation to the other. The coach's and Roboxer's animations were meant to be quick, so a player would not be locked into them for long. The coach can dodge and immediately return to normal, or the coach can hold in the dodge position. This is because the animations for dodging out and returning were separated. The enemy dodge is one single animation, as the enemy never needs to hold its dodge position. In addition, the enemy punches were designed to be long and drawn out, with a large windup. This gives players ample opportunity to see the punch coming and react.

[5] https://assetstore.unity.com/packages/2d/characters/armored-soldiers-and-monsters-2d-top-down-133669 Unity Asset Store EULA

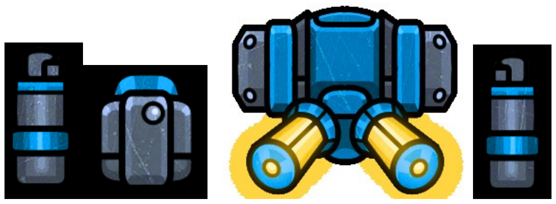


Figure 14: The character sprite

3.7 Audio Design

Audio was designed to provide feedback and confirm what was happening for the player. To that end, feedback sounds were created for punching and dodging. While the audio is sparse, and more time was spent developing the technical aspects of the game, the feedback given by these sounds is still important. For punching, a variety of hit sound effects were used. At any point, one of four sounds will be emitted on a successful impact. Dodging also has four different sound effects. As these actions were the most important actions that a player can perform, it was important that the sound effects would not be grating, as they will be played often. This is why four varieties of each sound were created. The sounds were sourced from sound effect collection videos on YouTube [6] [7].

3.8 Conclusion

This is the final build resulting from two terms of design and testing. It successfully utilizes machine learning to create an entertaining and unique experience. For the details of the technical implementation of the final build, readers can refer to chapter 4 of the report. To see the design process, and how the game has changed from the original proposal, readers can refer to chapter 5 of the report.

[6] https://www.youtube.com/watch?v=Gr2dxMzejQI YouTube
[7] https://www.youtube.com/watch?v=iPToKmyZi74 Free (Lukas Eriksen)

4 TECHNICAL IMPLEMENTATION

4.1 Machine Learning

Our game's machine learning (ML) component was responsible for dictating the actions of our game's main character, Roboxer. One of the goals of the ML was that the trained AI must behave like the player; if a player only did left punches, so should Roboxer. In order for the game to not feel dull, Roboxer must also be able to imitate the majority of the player's behavior within one minute. With these goals in mind, we designed the inputs of the model to be able to tell the current state of Roboxer and the opponent and the recent punch history of Roboxer. Having access to the current state of the opponent will allow the player to sense when they are about to be punched and act accordingly. Roboxer's current state may be used to see which actions are available to be performed while the recent punch history will allow it to learn a sequence of punches. The recent punch history is implemented as a finite state machine which encodes a sequence of punches up to a length of three (see Figure 15).

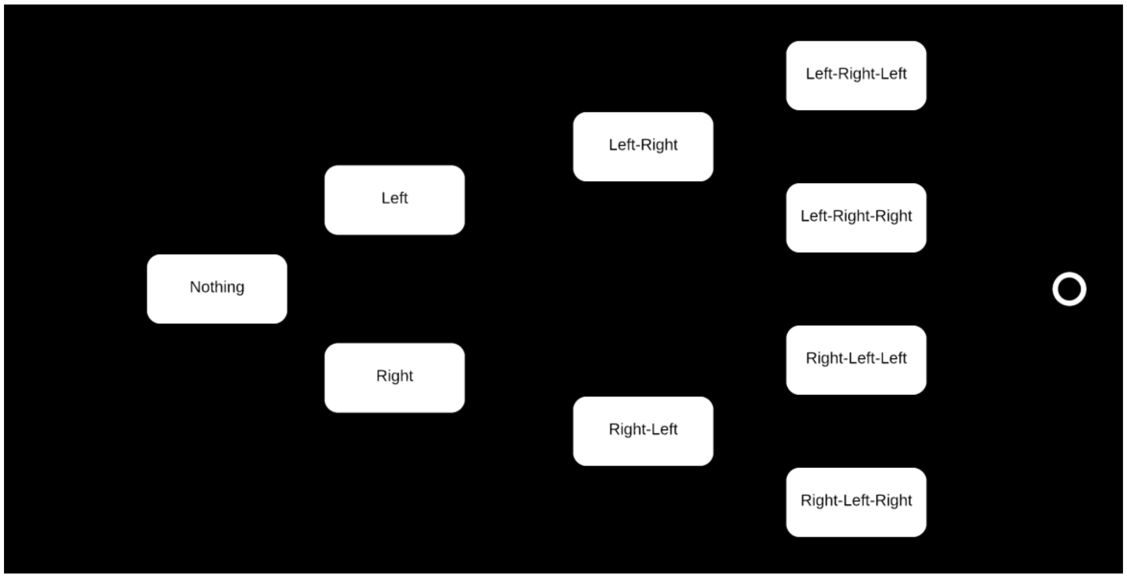


Figure 15: Punch combo FSM

The underlying machine learning model is a fully-connected feed-forward neural network with one hidden layer, which uses a softmax activation function on the output layer to act as a classifier (layer sizes: 36-128-5). The single hidden layer has 128 neurons, which was a value that we determined to be optimal for this game (see section 4.5.4). The hyperparameters for the network can be found in Appendix H. Each output neuron corresponds to an action available in the game (left punch, right punch, left dodge, right dodge, and do nothing). Given that the softmax activation function is in use, we also have access to the confidence values for each action; at the start of training, low confidence actions are ignored to give the illusion of Roboxer knowing no moves until it is trained.

To train the neural network, we are using Behavioral Cloning (BC) during the training phase with the coach set as the teacher and Roboxer as the student. There are two BC powered AIs in our game, one is visible to the player and the other is fighting a hidden training dummy to generate the "thought timeline". The visible Roboxer does not fight until the final phase of training where it shows what it learned; both the visible and hidden Roboxer share the same brain. During the actual match, the teacher is disabled so Roboxer will stop training and use the latest trained model.

We also sanitize the input from the player to avoid training on actions that would be harmful to our experience goals. The following scenarios are filtered out: throwing a punch while the opponent is punching, dodging while the opponent is doing nothing or dodging, and hitting a key while performing an action. The player is still able to perform the filtered moves during training, but the machine learning system will not use them for training data.

4.2 Game Mechanics

The number of mechanics in the game were deliberately kept small to make it easy for a machine learning agent to pick up and learn. The main script that handles the boxer mechanics is called "Boxer." A boxer is capable of performing two major actions; punching or dodging. A boxer can choose to perform either of these actions to the left or right, giving a total of four action options at any given time. The boxer script has a function to take in and process input from a given source called "AgentAction()". By interpreting the information given in AgentAction, the boxer knows to punch, dodge, or do nothing. Each boxer can have its input specified as either a machine learning agent, a scripted artificial intelligence, or the player's keyboard. This allows all three different sources of input to operate on the same principles inside the boxer, without having to create individual functions for each. Each boxer also has a reference to the opponent it is currently fighting with, so it can tell when it has hit or been hit.

The boxers also possess a pool of health for the final fight. Every time they take a hit from their opponent, either by not dodging or dodging in the wrong direction, their health pool is depleted. When their health pool reaches zero, they are unable to continue, and their adversary wins the fight. While this mechanic has no bearing during the training, it's crucial during the fight so an outcome can be determined. Because a boxer has a reference to its opponent, it knows when its opponent is punching. It can thus determine the point at which damage is registered if it is not dodging in the correct direction, and deduct from its own health pool accordingly. The connection between the two boxers is done using a `Match` class, which allows fights to be started and stopped and connects the punch events.

4.3 Phases

Because there are distinct mechanical phases of the game (see Section 3.3.1), handlers needed to be put in place to control the flow of game logic. Each phase was implemented as a game object in Unity to allow easy enabling/disabling while not active. In addition, an overall game handler was created to control the shifting of phases and the flow of control to each individual handler. The overall handler was referred to as the `GameHandler`. The `GameHandler` script acts as a state machine, keeping track of the current phase of the game and for how long the game should remain in that state (see Figure 16). Once the game timer reaches zero, the `GameHandler` moves to the next state, activating the appropriate phase handler.

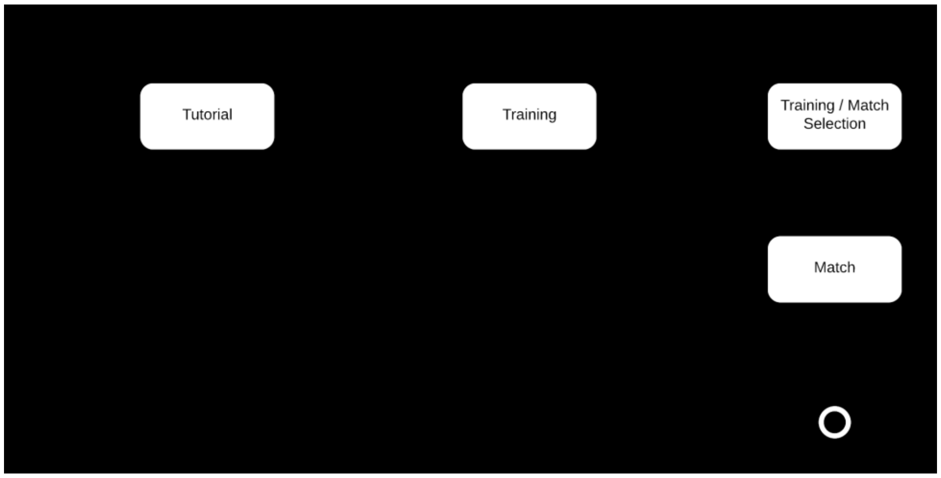


Figure 16: `GameHandler` FSM

Because the final game has two distinct phases, two phase handlers were created. These handler scripts were named `OffensiveTrainingMatchHandler` and `MatchGameHandler`. The first of these handled control of the initial phase of the game, when the player trains Roboxer. The script acts as another state machine that controls how the training phase of the game proceeds. The script's initial state is to wait for a command by the player to begin training. Once the player hits the spacebar to begin training, the handler starts paying attention to how much time has passed during the phase. At designated times, the handler will show dialogue from the coach and adjust the behavior of the training dummy to guide the training process. Once the timer reaches zero, the handler demonstrates what the AI agent has managed to learn from the player. After demonstrating Roboxer's capabilities, the player is offered a choice to continue training or proceed to the match. Should the player choose to continue training, the handler reverts its state to the beginning of training, and the process begins anew. If the player chooses to move to the final fight, a message is sent to the `GameHandler` script, which passes control over to the `MatchGameHandler` script.

In the `MatchGameHandler` script, the initial state is to wait for a prompt by the player, similar to the `OffensiveTrainingMatchHandler` script. After the player presses the spacebar, the handler begins the match. Instead of paying attention to the time, like the training handler did, the match handler pays attention to the health of each boxer. Once the health of either boxer is depleted, the match handler stops the fight and displays a corresponding victory message. Because the fight itself is less complicated than the training, the match handler can be a much simpler state machine, only needing to tell when an opponent is defeated. The completion of the match means that the game is over, and the player can quit.

4.4 Training Dummy and Final Boss AIs

Our game features two opponents for the player / AI to face, the first is the training dummy which is present in the first phase of the game and the second is the final boss which is the second phase of the game. Both opponents use a state machine to determine their actions and follow a predetermined sequence of moves with some minor exceptions for the final boss.

4.4.1 Training Dummy

The training dummy is the main opponent of the first phase of our game. The dummy follows a scripted set of moves, does not adapt to any input from the player, and does not get knocked out. The dummy has three different move sequences that it can follow: left-right punches (sequence 1), left-right dodges (sequence 2), and left-right punches followed by a left dodge (sequence 3). The dummy starts in sequence 1 and the player is encouraged to only dodge the incoming punches. This trains the ML model to associate being punched with the need to dodge. After 30 seconds of dodging, the dummy switches to sequence 2 and the player is encouraged to perform punches. The dummy can dodge in this stage, but the dodge is time-based rather than sensing the player's punches. This stage is useful for the ML model to learn sequences of punches and provides the highest level of player customization in our game. Sequence 2 lasts 30 seconds before switching to sequence 3. This final sequence provides a combination of punches and dodges and the player is encouraged to fight the dummy as if it were an opponent. This phase reinforces the punches and dodges that Roboxer learned in the previous sequences as well as how to prioritize dodges/punches when an incoming punch occurs while the player is trying to perform a punch combo. This sequence is interrupted after 30 seconds and the player is swapped out with Roboxer, which then fights the training dummy in sequence 3 to demonstrate to the player what it learned. Roboxer fights for 8 seconds before the inter-match screen appears.

Internally the training dummy is composed of two finite state machines (FSM) (see Figure 17), since FSMs make it possible to easily manage various AI behavior states [7]. The first FSM controls which sequence the training dummy is following and transitions based on time passed. The second FSM controls which action to take in a sequence. This FSM (referred to as the action FSM) takes a list of moves, including being inactive, to perform as input and cycles through it. The action FSM monitors the state of the boxer and will only perform the next move in the sequence if the previous move has finished and the next move isn't on cooldown still; this ensures that the training dummy will cycle through all actions in the proper order.

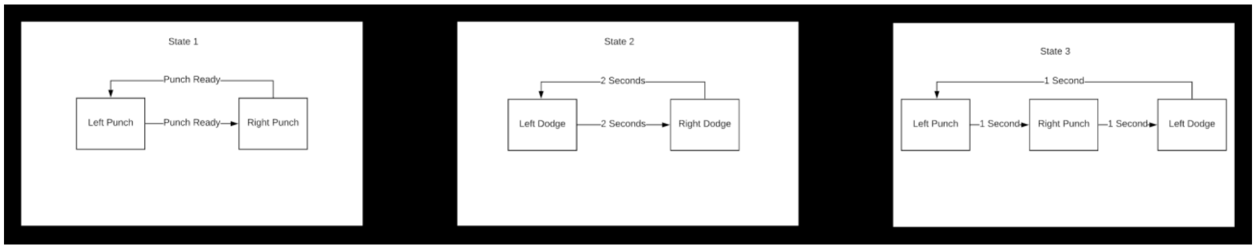


Figure 17: The training dummy FSM

The class containing the training dummy's logic inherits from the `Decision` class provided by ML Agents. This class can be used with a Heuristic Brain and attached to a `Boxer` in place of the learning or player brain. Because the training dummy only affects the brain of the `Boxer`, the game mechanics automatically apply to the dummy. During the training phase, there are also two training dummies following the same move sequences; one training dummy is visible to the player and the other is hidden and only used by the hidden Roboxer to generate the move timeline and reward meter.

4.4.2 Final Boss

The final boss of the game is what the player's AI agent fights during the final match. The boss is a buffed version of the training dummy which also gains the ability to dodge incoming punches with a small percent change and increased hit points. Like the training dummy, the boss follows a preset sequence of punches, but the boss has more sequences to choose from. The boss is designed so that it can beat poorly trained agents.

5 GAME DESIGN PROCESS

The ideas for our project were developed prior to its start, and after which we were able to work on the implementation of the game for close to six months. We created a timeline which detailed broke our project into two major milestones: Alpha and Beta release. We set the target for our Alpha release to be at the end of the first term of the project (7 weeks into the project). The goal of the Alpha release was to have all three game phases completed with working machine learning. The Beta release's goal was to have all components functional and a polished UI and game experience. This release was set to be 4-5 weeks after our Alpha release. In between our two releases, we redesigned several of our UI elements related to displaying training progress, added audio, and redesigned our ML system to accommodate the removal of the defensive training phase.

Throughout the development of our project, the core gameplay mechanics (punches, dodges, damage) and environmental/character art stayed relatively unchanged. The ML system (and associated UI) on the other hand received significant changes multiple times a month in an attempt to make it learn faster while meeting the expectations of our playtesters. We were able to handle these changes and integrate them with the rest of the concurrent development because we followed a relatively fast 1 week iteration/sprint with development stand-ups nearly every day of the week. This chapter documents the major milestones of our project in which significant changes were made (see Table 1 for an overview).

Table 1: Milestone timeline

Milestone	**Dates**	**Major Changes**
Summer	5/24/2019 - 8/30/2019	The game idea was decided upon and a mechanics and ML prototype were created.
First ML Build	8/30/2019 - 9/6/2019	Machine learning was incorporated into the boxing game prototype.
Full Game Flow Build	9/6/2019 - 9/20/2019	All major phases were added, machine learning was improved, and training progress visualizations were added.
Alpha Build	9/20/2019 - 10/4/2019	Training progress visualizations were added.
Beta Build	10/4/2019 - 11/22/2019	Removed defensive training phase, added animations/audio, added a new training dummy and enemy AI, and improved machine learning.
Post-Beta Build	11/22/2019 - 12/1/2019	Improved machine learning and enemy AI.

5.1 Summer Milestone

During the several months leading up to our project, we generated game ideas, created concept art, and experimented with some game mechanics.

5.1.1 Game Ideas

Before coming up with ideas for our project, we decided to do some research to see what types of games were being made using machine learning, both currently and in the future or coming up. One such game that we looked into was *Hello Neighbor [39]*, a game that uses machine learning to create an ever-increasing maze for the player to have to navigate through, and that gets made precisely to counter the player's previous strategies. This form of antagonistic machine learning we found interesting, however, we thought that a cooperative experience would suit our style of development better.

We came up with a couple of ideas for what we wanted our project to be, but ultimately decided on creating a fighting game in which the player would train a boxer. We decided upon training a boxer because we believed that our other ideas were scoped too big or not interesting enough to stay motivated. The ideas can be found in Appendix A. We also believed that by training a boxer we would be able to use reinforcement learning, which we all wanted to do.

5.1.2 Concept Art

As we began to create the visual identity of the game, we brainstormed multiple ideas for what our game should look like. The first idea was to replicate the look of the game *Punch Out!!* This would have the game be isometric and show a lot of the details of the characters, as seen in Figure 18.

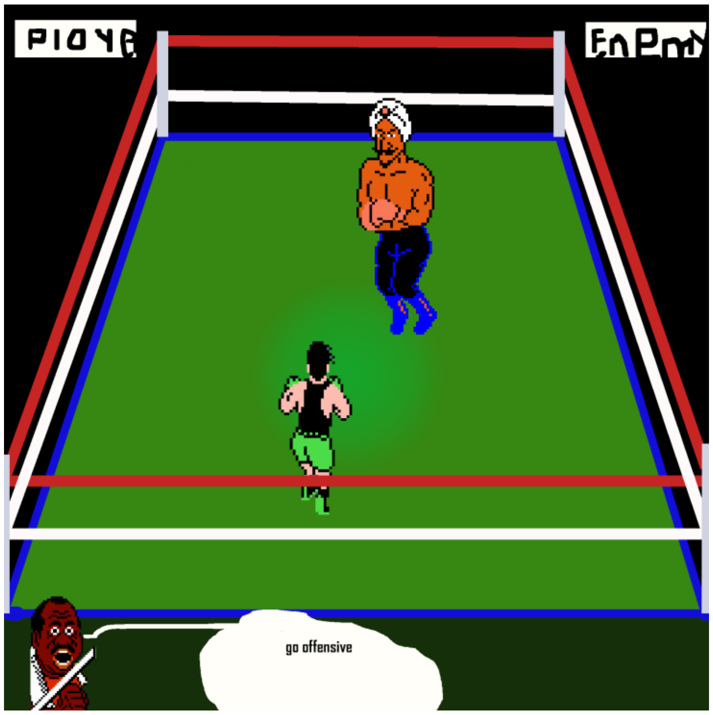


Figure 18: Isometric perspective based on *Punch Out!! [41]*

The next idea would also follow the isometric approach but would deviate away from the visual style of *Punch Out!!* as seen in Figure 19. This helped us to start thinking of our own visual identity for the game without fully depending on the game that inspired the project. This version also allocated some space for where some basic UI could be included such as health bars and a timer, represented by the black boxes at the top of Figure 19.

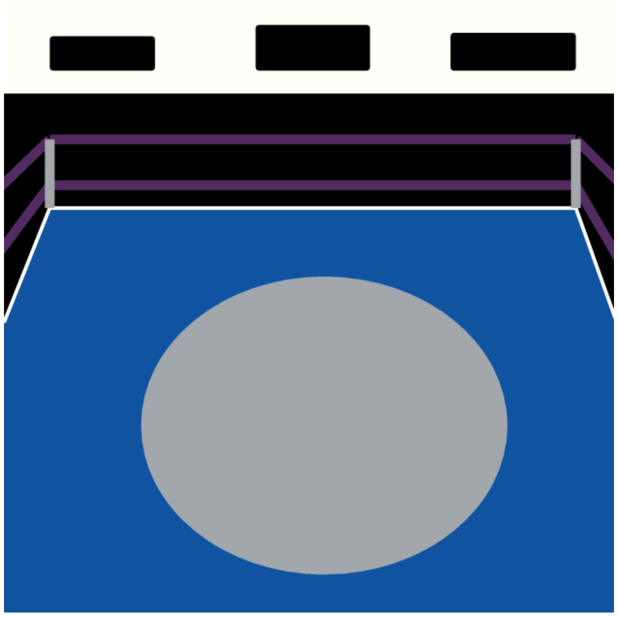


Figure 19: Isometric perspective idea 2

The next idea deviated from the isometric view to switch to a top-down perspective as seen in Figure 20. This idea would be what the team prefers due to the ability to use simpler character art, focusing on the top of the heads of the fighters instead of the character's face. This would allow the team to play to its strength with no artists on the team.

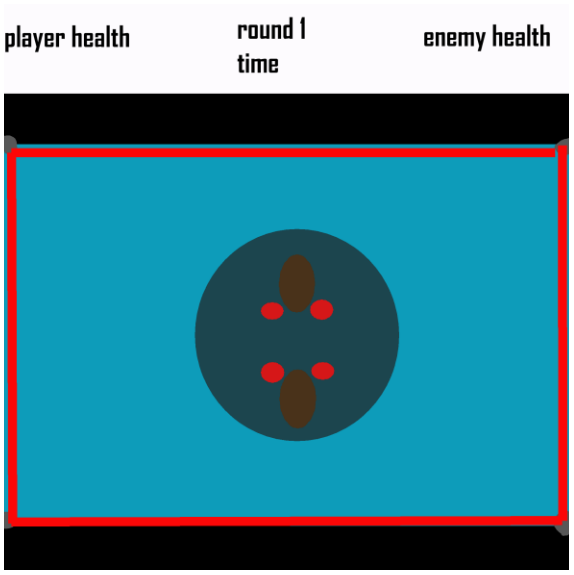


Figure 20: Top-down perspective

Once we finalized the idea of the game being about robotic boxers, we recreated the top-down view to reflect a more dark and electric feeling arena as seen in Figure 21. This theming was selected because we originally thought about incorporating multiple fights into the game in which the player and Roboxer would start with underground street fights and work their way up to the top of the robot boxing scene. We eventually decided to cut the idea of multiple fights to keep our scope at a reasonable size for the project. As a result, this was the only fight stage created at this point and would be used as a placeholder asset.

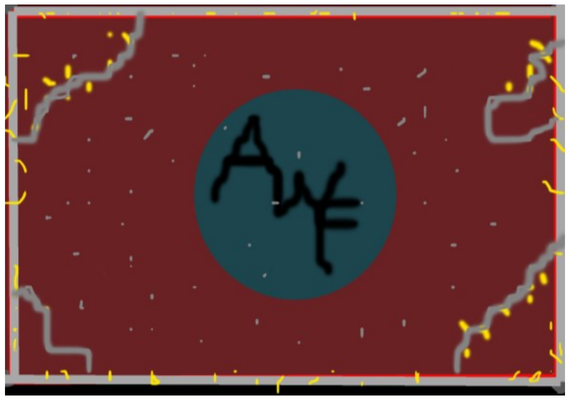


Figure 21: Robotic boxing fighting arena

The initial concept art for the characters, in Figure 22, illustrates the early ideas that we created for the characters and possible final looks for the game. These sprites were created in 64x64 pixel space and attempted to showcase how some aspects of the game mechanics might have been developed. For instance, one of the fighters below has lightning for arms instead of metal, and that carried over into the final game.

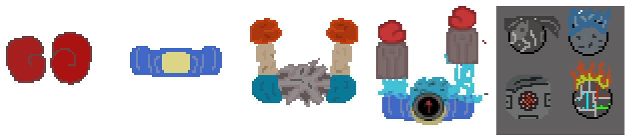


Figure 22 : Old sprites

5.1.3 Game Mechanics Prototype

This version of the initial design of the game had a base set of moves that included a left and right punch, a left and right dodge, as well as a block. There were no animations in the prototype, so the punches shoot straight out from the character and the dodges teleported the player to the side. The punches were also designed to broadcast an event when thrown so that it would be easier to add a damage mechanic in the future. The dodges and block could also be held so the player would stay in the dodge/block position. These mechanics served as the basis for our game in the later iterations.

5.1.4 ML Agents Prototype

We began our project by researching how to use machine learning within Unity and discovered the ML Agents toolkit. The toolkit provides a way to train in-game agents using reinforcement or imitation learning as well as using a pre-trained model. Internally, ML Agents uses TensorFlow and has implementations for a variety of algorithms such as PPO and GAIL. The Unity SDK for ML Agents is composed of several base classes and game assets which can be extended by the game developer.

The core of ML Agents is the `Brain` object, which can be either a Player Brain, Heuristic Brain, or Learning Brain. The brain acts as a controller for an `Agent` object, which is the base class that needs to be extended by game agents. Agents collect observations about their environment and use their brain to determine the proper action to perform. For ML Agents to reset the game environment after completing a trial, the agent must be contained within an `Area` game object. The area contains all game objects that the agent can interact with during a single trial; for example, in our game, the area contains the ring, agent, and opponent. Finally, to train the brains, an `Academy` game object needs to be created at the root of the scene. The academy contains a list of the brains that will be used and it can mark which ones need to be controlled through the TensorFlow interface. Figure 23 shows the class diagram for ML Agents.

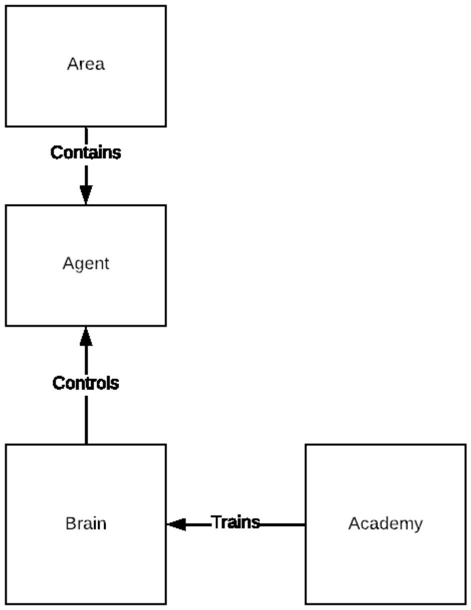


Figure 23: ML Agents class diagram

To learn how to use this toolkit, we created an archery game using ML Agents (see Figure 24). The demo game had a very limited action space and it was easy to derive a mathematical function between inputs and outputs. In our testing, we found that ML Agents requires an external script to be running prior to the start of the game, which will enable the training of the ML model. The training parameters were determined by a configuration file that we copied from the ML Agents examples. From creating this prototype, we learned that ML Agents makes it easy to integrate ML into a game, and it took only 3 minutes for the archer to perform without any noticeable flaws.

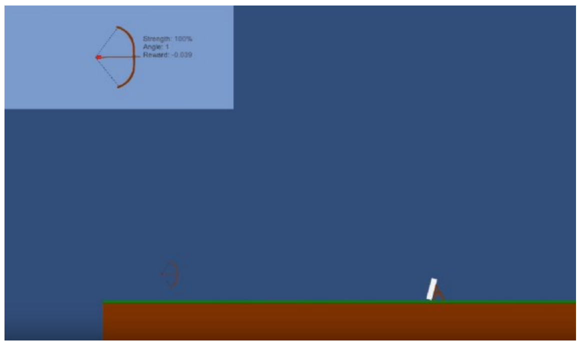


Figure 24: The archery prototype

5.2 First ML Build

This was the first version of our project which used ML Agents. The change to ML Agents required several code restructures (see Section 5.2.3) but was implemented in a few hours due to the previous testing with the archery game. This version proved that given enough time, an agent trained through RL could play the game, though there were still concerns about whether an AI could learn fast enough for a playable game. The first build can be seen in Figure 25.

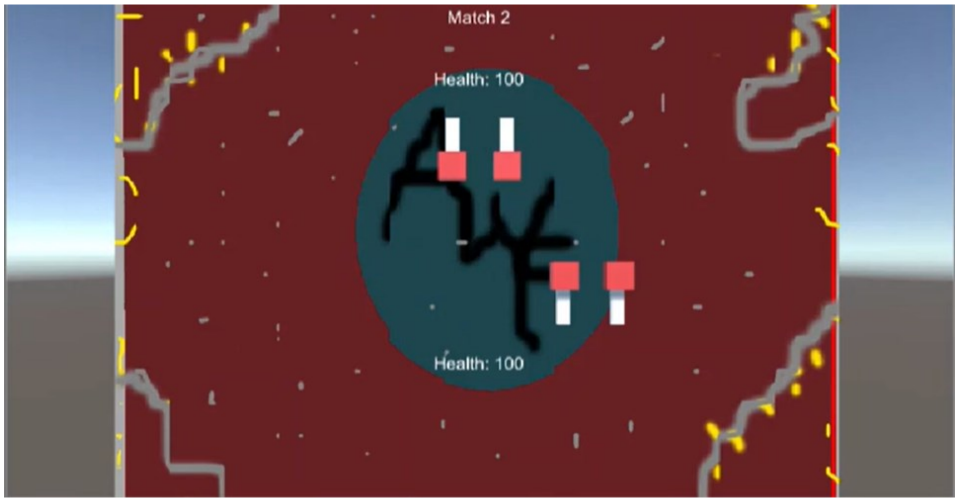


Figure 25: The first build of the game with ML Agents

5.2.1 Game Idea Changes

The layout document of how we imagined the game flow would be can be found in Appendix F. It outlines the 3 distinct phases that we wanted to create for our game: an offensive training phase, a defensive training phase, and a final boss fight. The offensive and defensive phases would consist of the player acting as the coach and teaching a robot fighter, called Roboxer (name decided on late in the project).

5.2.2 Health and Damage Mechanic

Our previous build of the game featured the ability to punch and dodge, but neither had any effect in the game other than being visually different. We created a simple solution whereby an event was broadcast when a punch was thrown and the `Arena` (see Section 5.2.3) would listen to the punch events of both fighting boxers. The Arena then calls the `Boxer.OnPunched` method would then be called on the recipient boxer and the damage logic determines the outcome of the punch. A punch can either deal full damage, be dodged or blocked, or result in a KO, depending on the current level of health and which direction the boxer is dodging. We also added support for different types of punches through a `Punch` class, though we only implemented a weak punch at this stage of the game.

5.2.3 Adding Machine Learning

Using the ML Agents archery prototype as a template, we converted the Boxer into an Agent, added the Academy object, and made the Arena extend the Area class. By having the Boxer class extend Agent, it gained the ability to use a Brain to gather observations and decide on actions. Three Brains were created: a player brain, a learning brain, and a heuristic brain so we can simulate a player for testing purposes. The configuration file from the archery demo was copied over to this project without any changes. The AI collected 16 frames worth of the observations documented in Table 2, because we felt that this would be enough to capture the last move of both Roboxer and opponent. The model was trained using PPO. It used a feed-forward neural network with two hidden layers, each containing 256 neurons and a softmax classifier on the output layer.

The rewards were added within the Boxer class to give a positive reward when Roboxer lands a punch and a negative reward when it gets punched. Positive rewards were also given for when it successfully dodges and a negative reward was given if the opponent successfully dodges. We created the defensive training phase first, given that it would be a good way to test if the ML was working and it was relatively easy to create a mock player. The mock player brain was created using a simple state machine that cycled through a predetermined sequence of moves. The mock player and Roboxer were made opponents and the ML was allowed to train. In our testing, it took almost an hour for Roboxer to be able to perfectly dodge the mock player's moves and to throw punches while not dodging. This duration was much too long for our game requirements, so reducing the training time became a focus of the next iteration.

Table 2: Observations collected by our original boxer agent

Observation
Agent’s health (between 0 and 1)
Opponent’s health (between 0 and 1)
Agent’s move state (Left punch, right punch, left dodge, right dodge, nothing)
Opponent’s move state (Left punch, right punch, left dodge, right dodge, nothing)

5.2.4 UI Design

The initial UI designs for the first ML build were minimal in terms of implemented assets, display elements, and concepts. The UI included the arms and hands of both the player and enemy as well as health displayed as a text field above and below the characters. The match was also displayed at the top of the screen. More elements would be added later, but at this point in development, more work was going into the planning for the coming stages than the implementation of UI elements.

5.2.5 Early Character Designs

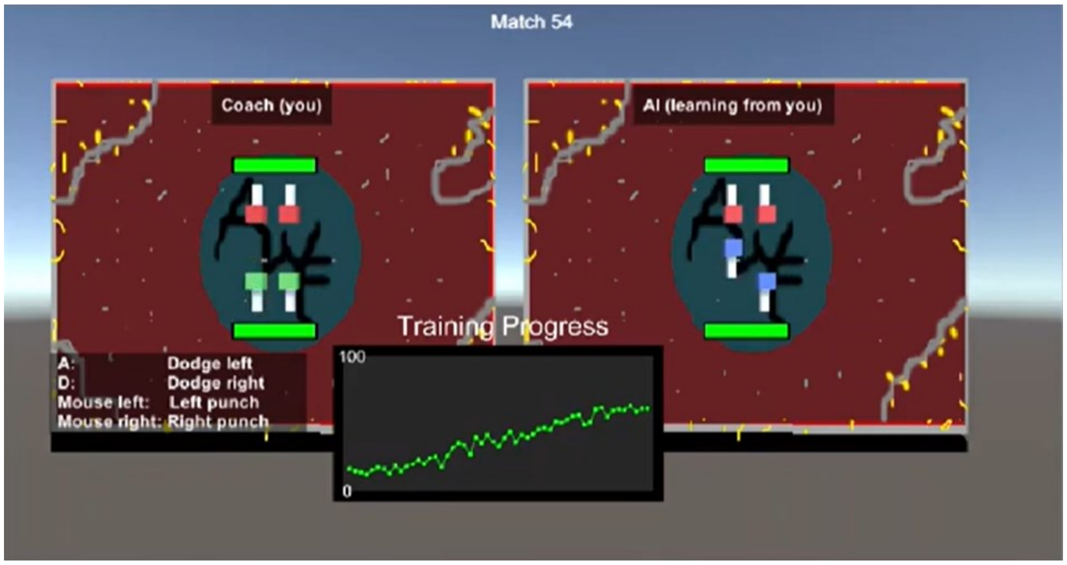
At this stage of development, the characters were simply composed of the arms and hands (3D blocks), as seen in Figure 25, each moving forward on the screen to indicate the punch and back to indicate the dodge state. At this point, we decided to look at 2D sprite sheets to see if we could use premade assets for our game, given that we didn’t have an artist in our group and wanted all visual assets to look professional.

5.3 Full Game Flow Build

This was the first version of our project which contained the offensive training, defensive training, and the match. This version proved that our game idea was viable and machine learning could be used during each training phase.

5.3.1 Game Flow

With the defensive training in place, we created the offensive training phase which added another two agents, one being the human-controlled coach and the other being a training dummy. The learning brain was also paired against a training dummy and its weights were updated every 8 seconds, which was the duration of the training dummy's move cycle. Roboxer was visible to the player for the duration of training and it could be seen performing its moves against its training dummy, as seen in Figure 26. The behavior controlling the training dummy was the same as the mock player in the previous build of the game. The changes to the ML system are documented in Section 5.3.4. We also added the final match, which paired Roboxer against the training dummy; this enemy AI would need to be changed in future iterations, we used the training dummy for development convenience. It was noted at this point in the project that the training dummy and enemy AI would need more complex behavior to enhance the playing experience. Our goal for the final training dummy was to give the players an enjoyable experience training their AI by fighting a realistic seeming opponent. The current training dummy (and the enemy AI) only performed a handful of moves and was very predictable.



A: Dodge left
D: Dodge right
Mouse left: Left punch
Mouse right: Right punch

Figure 26: The imitation learning stage of the first full game build

5.3.2 Phase Swapping (Offensive, Defensive, Match)

In this stage of development, we added the different phases of training into the game, which includes an offensive training stage, a defensive training stage, and finally an actual match against an AI enemy. We created this by making three game objects, representing each phase, and a camera that could move between them. This update allowed us to test how our ML system would perform with both the offensive and defensive training stages as well as the final match. We were able to experiment with both reinforcement learning and imitation learning as we now had both training sessions available to us. This would be updated further on in the alpha and beta versions of the game as well as changed later in the development cycle.

5.3.3 Game Mechanics Changes

During our first few plays of the game, we found that the hold to dodge was a bit awkward since the player could not punch while dodging. We preferred that players remain unable to punch while dodging, so we removed the ability to hold a dodge for a variable length (added back in Section 5.5.2). To do this, we implemented the same logic as a punch whereby the action could only be performed for a set amount of time. We also implemented cooldowns on both dodging and punching to ensure the moves would not be performed too fast, through the use of a state machine. If a move is on cooldown and the boxer attempts to use it, their action will be ignored.

5.3.4 Machine Learning Changes

In an attempt to reduce the training time, we reduced the complexity of the training space by removing the continuous health inputs and stacked observation vectors. We wanted Roboxer to learn combos, so we created a state machine that tracked the player’s combo. The state machine is encoded into a bit array and used as input to the ML model. These changes led to a significant improvement in the learning time of Roboxer, learning to visually resemble the mock player in about 2 minutes as compared to over an hour with the previous ML inputs.

The addition of the offensive training allowed us to experiment with imitation learning. We started with GAIL but soon realized that it would require a pre-saved play trace to function, which was not feasible for our game. A second solution was created which positively rewarded Roboxer every timestep that their actions matched that of the coach, and negatively rewarded them if they differed, as seen in Figure 27. We found that this approach led to Roboxer copying the behavior of the coach fairly quickly.

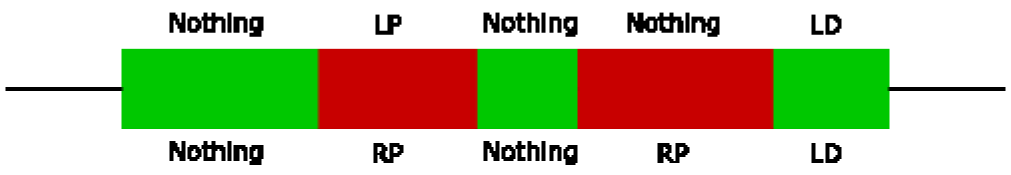


Figure 27: A timeline of rewards given to Roboxer. The coach's moves are shown on top while Roboxer's moves are shown on the bottom. Green indicates the time when a positive reward was being given while red indicates a negative reward.

5.3.5 Visualizing Training Progress

We decided that it would be beneficial for a player to see how much progress they have made in training, so they know whether they are actually training the AI or would need to modify their behavior (if the progress was not increasing). To do this, we created a 6 segment progress bar that was displayed on the side of the screen as seen in Figure 28. The value displayed by the progress bar is the normalized cumulative reward value calculated using Formula 1. This value was calculated at the end of each 8 second training period.

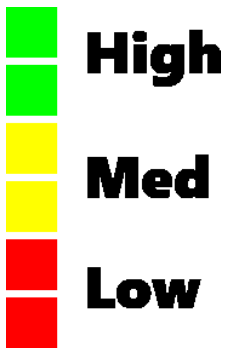


Figure 28: The six segment training progress bar

Formula 1: The normalized reward based training

$$(currentReward - minReward) / (maxReward - minReward)$$

We also decided that it may be helpful to see how the cumulative reward changed over time, so a graph was made and placed in the bottom left of the game's UI during the training phases as seen in Figure 29.

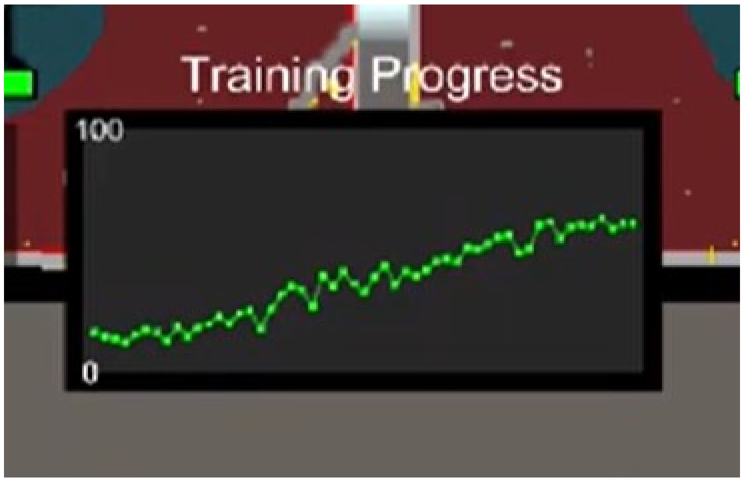


Figure 29: The training progress graph

5.3.6 Art Changes

We wanted different environments for the training phases to help differentiate the type of training the player will be doing. There would be three environments for the game: defensive training, offensive training, and the final fight environments. Figure 30 shows the first iteration of the defensive training environment. This rendition included a blue mat for the training ring and wires to connect to the boxer to track the training process. This design did not work out so well due to the conflicting color scheme resulting in some users feeling discomfort.

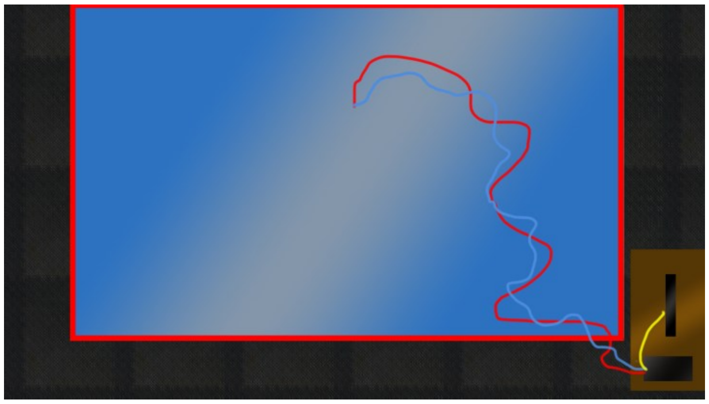


Figure 30: Defensive training environment iteration 1

For the next iteration, shown in Figure 31, we changed the mat and color scheme to have more of a factory feel to it since heavy machinery would be using the space. We added some gym equipment to make the space feel more populated, however, the simplified art style of the weights and tires conflicted with the semi-realistic background. Also added was a metal plate pattern that was found online[8].

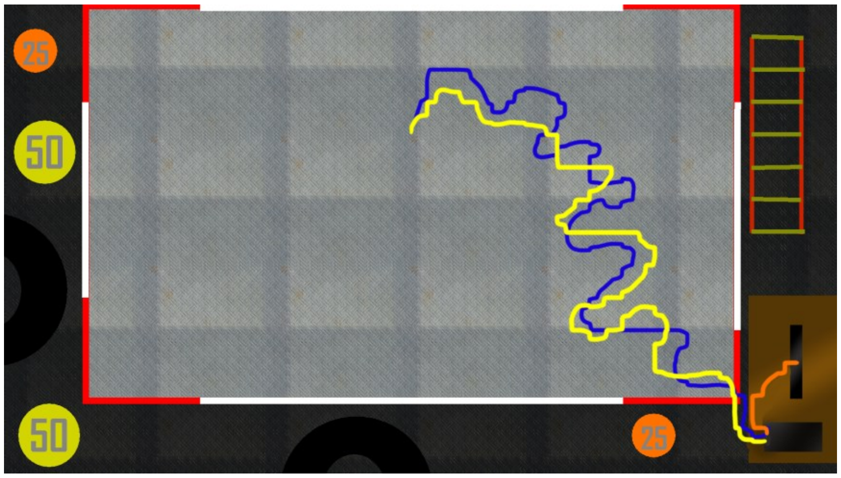


Figure 31: Factory defensive training arena

In the next iteration, we improved upon the weights and equipment to make them more recognizable and made it feel more like a gym as seen in Figure 32. We found the assets online with the Low Poly Gym Pack[9]. We then rendered the assets into 2D via manipulation of the render camera angle. An issue with these assets was that the equipment is unrecognizable in the top-down view. We also removed the wires to prevent animation issues from having the boxer agent moving and become disconnected from the wires.

[8] https://www.deviantart.com/tmm-textures/art/Metal-Floor-42059858 Freeware

[9] https://jprinsloo.itch.io/free-low-poly-gym-pack License: CC0: Public domain, completely free to use in both personal and commercial projects

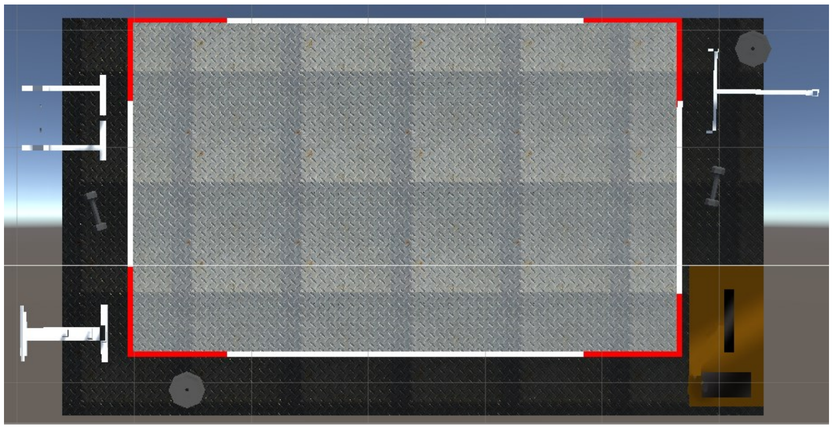


Figure 32: Updated gym equipment

After this iteration, we altered the gameplay loop to remove the defensive training phase from the game.

The offensive training area followed a similar iteration path as the defensive arena. In the first iteration, we tried to include an interconnected area between the offensive and defensive training areas. This iteration also included the blue training ring section and the wires from the first defensive training area iteration as seen in Figure 33 below.

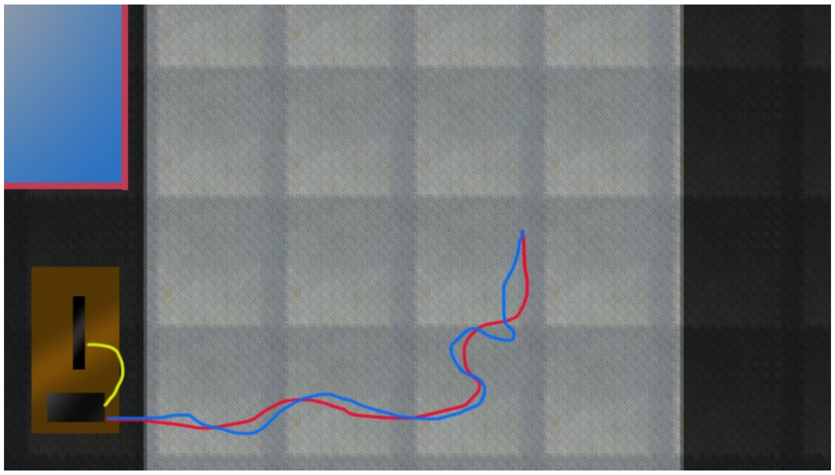


Figure 33: Initial offensive training area

This iteration included replacing the blue training mat ring with the steel plate floor to keep the environment consistent. We populated the gym with treadmills to add more gym equipment, however, the implementation would not work well due to the harsh difference in art style between the floor and the hand-drawn treadmills. We then added some bolts to the metal plates to make the repeating pattern seem less jarring (see Figure 34). This would provide the opposite of what we wanted to achieve.

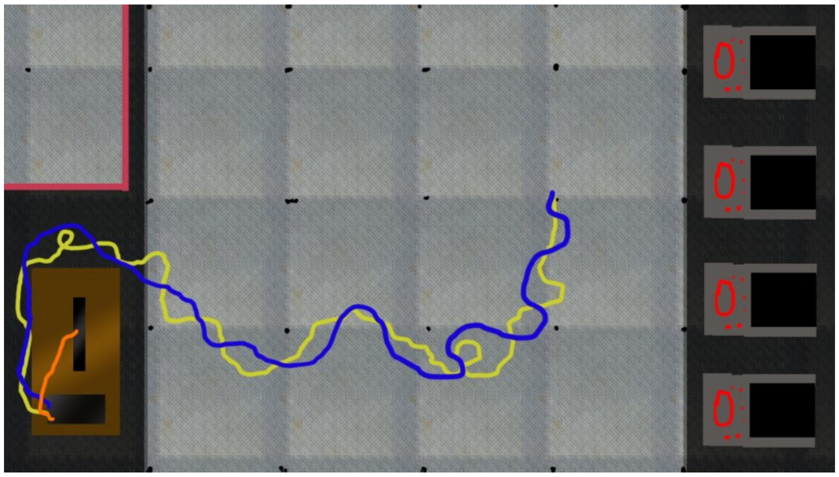


Figure 34: Updated offensive training area

In the next iteration, we removed a majority of the conflicting art styles. The treadmills, wires, and bolts were removed and the 3D modeled weights would be added to the training area. The figure below (Figure 35) shows the iteration before the inclusion of the 3D modeled weights.

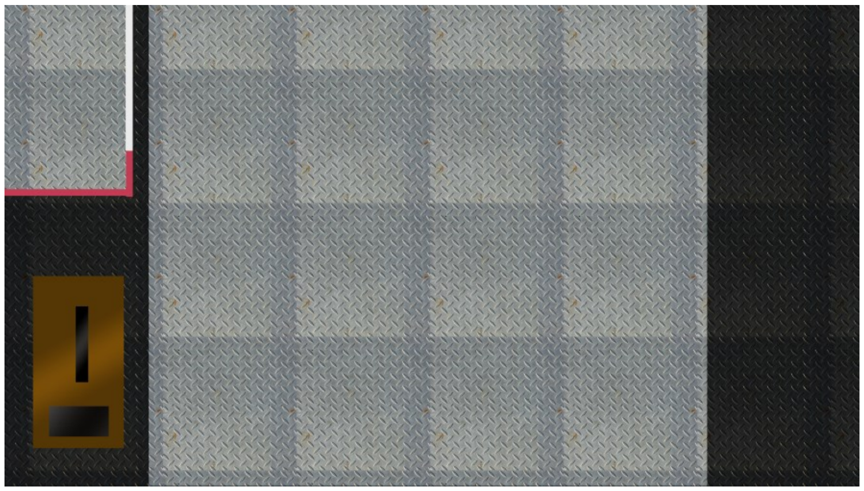


Figure 35: Offensive training area (no weights / wires)

5.3.7 Sprites

This version of the game finally included sprites that actually showed the body of the player and enemy inside of the unity project itself, but not in the game itself. Given that we didn't have an artist on our team, we decided it would be best to use premade sprite assets for our characters. At this point in the game design process, we didn't decide on a specific character sprite but decided to use something from the Armored Soldiers[10] sprite sheet below in Figure 36.

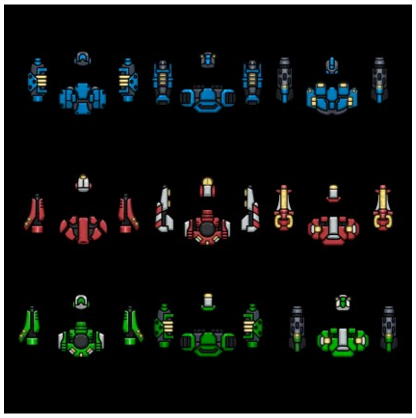


Figure 36: The Armored Soldiers sprite sheet

5.4 Alpha Build

The Alpha build of our game included most of our major game mechanics, phases, and fully working machine learning. This was the first version of the game that we conducted playtesting with.

5.4.1 Machine Learning Changes

For the Alpha build, we experimented with several changes to how rewards were given during imitation learning to see if we could obtain any noticeable improvements in our training. The new reward functions included one that gave a reward based on the last move performed by the coach and Roboxer and the time since. The other gave a reward if the most probable action of the coach and Roboxer were the same. To measure any improvements, we used the DTW score with a mock player as the coach (see the following section). After repeated attempts, we noted that the DTW score varied too much and we would re-evaluate these when we came up with a

[10] https://assetstore.unity.com/packages/2d/characters/armored-soldiers-and-monsters-2d-top-down-133669 Unity Asset Store EULA

better metric. We also found that increasing the reward size at each timestep led to faster training times.

We also used this iteration to refactor some of our existing code. One notable improvement was the way we structured our ML system. Previously, the Arena controlled when all boxers would train as well as the logic in the match, but this functionality was split up. The `Boxer` class obtained the ability to train, but this logic was offset to the `GameHandler` classes which we made to control each phase of our game. The logic which connected the punches between the two boxers was moved into a separate Match class which extended ML Agent's Area base class. The Match class also can start and stop fights, which made working on the imitation learning phase much easier as we could make Roboxer stay still while the coach fought.

From a gameplay perspective, we found that having Roboxer training on the screen next to the coach was visually distracting, so we created a hidden Roboxer and training dummy pair which were training in the background. The on-screen Roboxer was not fighting and positioned to watch the coach demonstrate moves. The player was given the option to press SPACE to let Roboxer fight to see how much they learned.

5.4.2 Visualizing Training Progress

A major focus of the Alpha build of our game was how to visualize training progress in a way that is both accurate and easy to understand for players with no ML experience. The previous indicator in our game used the cumulative reward, which essentially measured a rough percentage of the time where the coach and Roboxer were performing the same moves during a cycle. This indicator had several drawbacks such as having inexact upper and lower bounds and the need for Roboxer and coach to be in perfect sync. To improve this progress indicator, several metrics were created to see how reliable and effective each was.

The first new metric was sequence matching, which just calculates the percentage of moves in a cycle the coach and Roboxer performed in common (see Figure 37). This metric was less reliant on time but was heavily dependant on the order of the coach's and Roboxer's moves being in order. We found that if the two boxers were out of sync the metric would report a score close to 0 even if both were doing the same move sequences. A potential solution for this was to factor in the possibility that the moves were out of sync because the Roboxer skipped a move early on. This led to the next metric, dynamic time warping or DTW (see Figure 38). DTW is a technique to measure the distance between two time-ordered datasets, where the distance is a metric that factors in time differences, moves being skipped, or the wrong move being performed. This metric performed better than the sequence matching during our testing. When both the coach and Roboxer were performing the same move sequence, but not in perfect sync the sequence matching's score varied from between 0 and 1 depending on how out of sync the two boxers were while the DTW score remained close to 1. This was the metric used during our Alpha testing. One drawback that we noticed with the DTW was that if given an AI which performs random moves, the metric varied dramatically, which could potentially confuse a player. A proposed solution to this which we implemented in the Beta release was to factor in the model's action confidences through a metric such as cross-entropy. The training progress metric for the defensive training phase was not modified during this build.

LP - RP - LD - RP

LP - RP - RD - RP

Figure 37: Sequence matching showing a score of 0.75

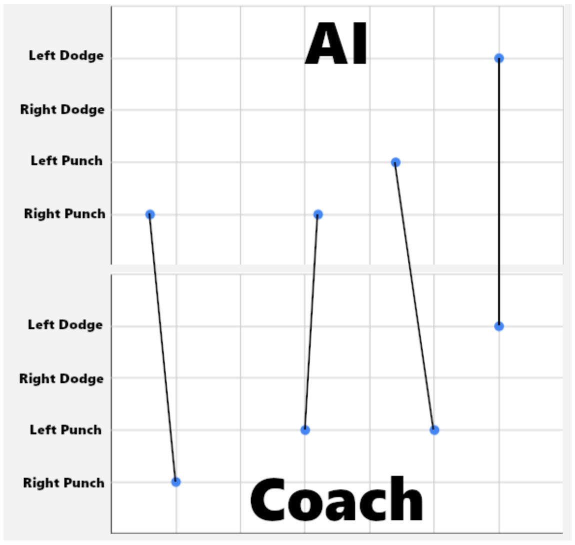


Figure 38: Dynamic time warping in action

Prior to our Alpha release, we also spent time determining how to best visualize the training progress metric. Through our playtesting, we determined that people were having a hard time understanding the graph and recognized that this visualization would need to be replaced, though this was held off until the Beta build for our testing purposes (the graph was useful when modifying the ML). The stacked progress lights developed as our first progress indicator was visually unappealing, so a progress bar was developed in the shape of a brain with the hope that players would see this as an indication of moves learned. The brain progress bar filled and changed color in relation to the DTW score during offensive training and the reward during defensive training, as seen in Figure 39.

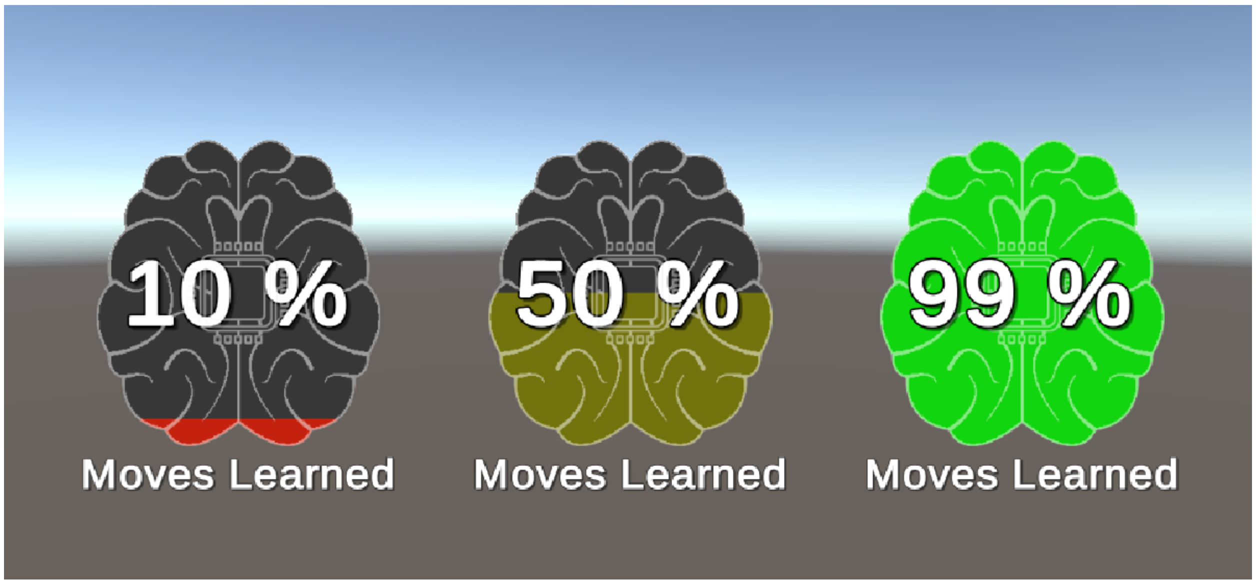


Figure 39: The brain-shaped training progress indicator

In addition to this redesign, a new progress indicator was added: the train of thought (Figure 40). The train of thought is a series of bubbles on the right side of the screen that shows what action Roboxer is "thinking" about performing. The bubbles were made of four colors with two-letter initials to distinguish them. The colors were blue, green, purple and red with the initials LD, RD, LP, RP to represent left dodge, right dodge, left punch and right punch respectively. We wanted to include the train of thought so the player would be able to see a history of what Roboxer was thinking about instead of having the player to focus on Roboxer performing the moves and memorizing the order on top of performing the moves themselves.

The train of thought would begin by spawning the thought bubbles at the bottom of the screen and would shift up when a new thought bubble is added. When the bubble reaches the top of the screen the bubble would disappear. This would allow for 9 bubbles to appear at a time.

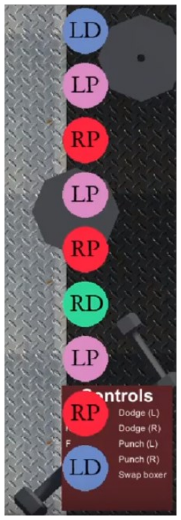


Figure 40: Train of thought initial design

5.4.3 Adding Animations and Character Sprites

In the alpha build we were also able to add animations to properly convey the action in the game, based on some simple punching and dodging. We did this using the sprites we had decided on, being the Armored Soldiers pack referenced in the appendices. We added in punching and dodging animations using the assets, each taking approximately 1 second to complete so that people had to finish an action before attempting another.

The assets we decided on, from the pack described above (Figure 36), were a singular sprite section of the robots (Figure 14), and we then used materials to color them differently to distinguish the player from the training dummy from Roboxer. In this way, we created a sort of internal consistency with a lot of mirrored elements, so that no players would think that the enemies had any different abilities or moves than the player did.

5.5 Beta Build

The Beta build of our game was the version we tested during AlphaFest.

5.5.1 Game Flow Changes

The first of the many large changes to the beta was the addition of a tutorial, which included instructions on how to control the character as well instructions on what the various aspects of the game were in relation to the icons and UI elements were seen on screen. We also added a dialog box that allows the coach to display text to the player, which we used to create a more guided training session (see Section 3.5.4).

During our Alpha playtesting, we observed that players threw punches randomly and missed many dodges during training. When asked to explain if they were trying to perform a specific punch combo, several playtesters stated that they were just trying to hit the dummy. With this in mind, we decided to remove health bars from the training scene so that players don't have the goal of knocking out the dummy, but rather focus more on the progress meters. During this iteration, the dummy also underwent some drastic changes to enhance the play experience of our game. A major design goal of the new dummy was to ease the player into the game by focusing on specific moves in a way that the ML would be able to pick up quickly, which was done through the addition of multiple dummy states. The first state only punches and the player is told through the dialog box to perform dodges. This lasts for 30 seconds before the dummy starts performing only dodges with long periods of doing nothing. The player is encouraged to perform some punch combos during the next 30 seconds. After that stage, the dummy combines punches and dodges and the player is encouraged to continue performing their combos as well as dodging incoming punches. A complete move state diagram of the dummy can be seen in Figure 17 (Section 4.4.1). The time between the dummy's moves was significantly increased since the Alpha build as well to reduce the pace of training.

We received feedback from many players during testing that it was not obvious what they had managed to teach the Roboxer. Players felt as though they had an impact, but could not see the specifics of what they had done. We decided to re-evaluate the defensive training phase. The nature of the learning in this phase made it so the Roboxer would not learn to do what the player was doing, they would learn *responses* to what the player was doing. Thus when the game transitioned to the final fight, there was no obvious repetition of the player's strategies. The decision was made to remove the defensive training entirely and focus on the offensive training. This way, training would shift towards copying the actions of the player, and players would be able to see their own actions present in the Roboxer with far greater clarity.

5.5.2 Hold to Dodge

After some more feedback from playtesters that the dodges were hard to perform properly because of timing as well as the feeling that they should be able to hold the keys to keep dodging, we decided to add the hold to dodge feature back. This required a slight change in the animation which locked the boxer in place while the key was held down. We also decided to only implement this for the player, so that we would not have to make any drastic changes to the ML system at this stage of the project.

5.5.3 Machine Learning Changes

During our Alpha iteration, we determined that using the dynamic time warping score to evaluate our ML models was too inconsistent for any accurate measurements (see Section 5.4.2). A solution to this was to use cross-entropy, a standard in machine learning, to measure training loss. To implement cross-entropy, we needed to modify the source code of ML Agents to send the confidences and training loss to the Unity game. With access to the confidences, we also decided to ignore low confidence moves at the beginning of training to avoid confusion among players when Roboxer began to perform random moves. Using the cross-entropy performed much more stable measurements than the DTW scores and when comparing hyperparameters we were able to see performance differences over repeated trials.

Around the same time as the implementation of cross-entropy, we removed the defensive stage of training. This change meant that we would only be performing imitation learning in our game, so we decided to try out Behavioral Cloning (BC). BC is designed for imitation learning and once implemented it showed drastic improvements over our custom imitation learning solution. BC is a supervised learning algorithm, so it needs a collection of samples from the player to train on. With a supervised approach we faced two issues: punches tend to occur much more often than dodges and some players may hold keys down when performing an action while others may press only once. To handle these cases, we removed the Player Brain and created a new Heuristic Brain that mapped the keys to actions. In this new brain, we standardized the holding behavior of moves between all players as well as increased the number of dodge samples to roughly equal that of the punches.

5.5.4 Hyperparameter Tuning

Initial testing of the custom imitation learning algorithm showed that it would produce a cross-entropy of around 1.3 after 2000 training steps (see Table 2 for details on the test player agent). In our testing, Roboxer was unable to perform a combination of two or more punches and produced cross-entropy values over 6 in these cases. On the other hand, Behavior Cloning proved that Roboxer could learn sequences of punches while achieving cross-entropy values of less than 1. This algorithm better fit our gameplay design due to its ability to learn more complex moves, so we moved on to hyperparameter tuning.

Hyperparameters are variables in a machine learning algorithm that must be configured externally rather than learned from data [1]. To achieve the best performance from an ML model, hyperparameters must be tuned. To conduct hyperparameter tuning we needed a testing environment that would produce reproducible results and allow us to compare different combinations of hyperparameters. To eliminate human-introduced noise, we created an agent which would act as the player and perform a predetermined sequence of punches and dodges. Five variants of this agent were created as documented in Table 3. Cross-entropy was used to evaluate the performance of the model; this loss value was automatically generated by ML Agents. Training was run for 2000 steps and repeated on all variants of the player agent. The average cross-entropy of all five variants at the end of training was used to represent the performance of the model.

Table 3: Test player agent variants

Variant	**Behavior**
1	Left punch
2	Right punch, dodges
3	Left punch, right punch
4	Right punch, left punch, dodges
5	Left punch, right punch, left punch

The hyperparameters were tuned one by one, and a significance test was normally conducted with a relatively high or low parameter value to measure a significant change in performance. Our threshold for significance was a change of greater than +/- 0.1 for the average cross-entropy across all five variants. The hyperparameter values were selected by hand using recommendations from ML Agents documentation as starting points. Since the hyperparameters were tuned one after another, the previously tuned hyperparameter was brought along into the next test. Appendix C shows the detailed results of hyperparameter tuning and the graph in Figure 41 shows how the performance of our model was affected by the tuning.

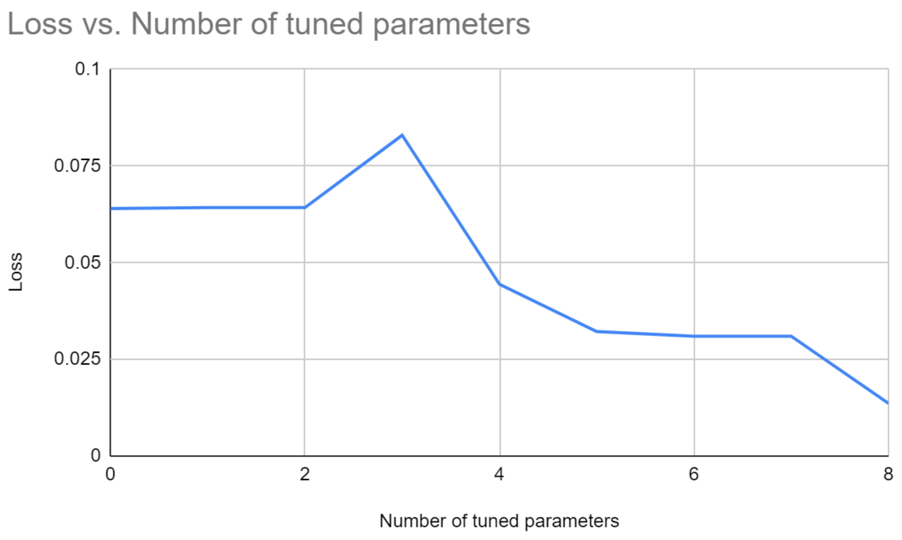


Figure 41: The result of hyperparameter tuning on model loss

5.5.5 Training Progress Visualization Changes

After completing our Alpha playtesting we determined that the current progress indicators were still confusing for players. Given that the ML subsystem now had access to the underlying action confidences, we were able to implement a cross-entropy based metric. This metric calculated the average cross-entropy of a cycle by calculating the negated log of the correct move's confidence, where the correct move was decided to be the current move of the coach. This metric had similar drawbacks to that of the time matching score in our early game builds because it used the coach's current move as the truth vector. ML Agents can generate a TensorBoard summary to display training loss, and it was noted that the cross-entropy loss presented there smoothly decreased to 0 as training progressed. This also proved to be a more accurate metric for hyperparameter tuning as the values were much more consistent and better-reflected training loss than the in-game calculated metric (see 4.1 for details on hyperparameter tuning). The source code for ML Agents was modified again to send the current training loss to the Unity client. Once the loss was obtained, it needed to be converted to a value between 0 and 1 using Formula 2. This value was then used as the new metric for our progress bar, and it now provided an accurate representation of training progress from a machine learning standpoint.

Formula 2: A conversion from loss to training progress

$$e^{-Loss}$$

After our Alpha testing, there was some concern that the brain progress indicator did not fit the game's theme and its descriptive text may be confusing to players. To make this progress indicator fit better with our theme, it was converted to a smoothed dial. As the training progress increases, the dial moves to the right and the light under the needle will turn on to be easier to see in a player's peripheral vision, as seen in Figure 42. The text describing the dial was changed from "Moves Learned" to "Copy Meter", which better reflected what the meter was attempting to convey to players.

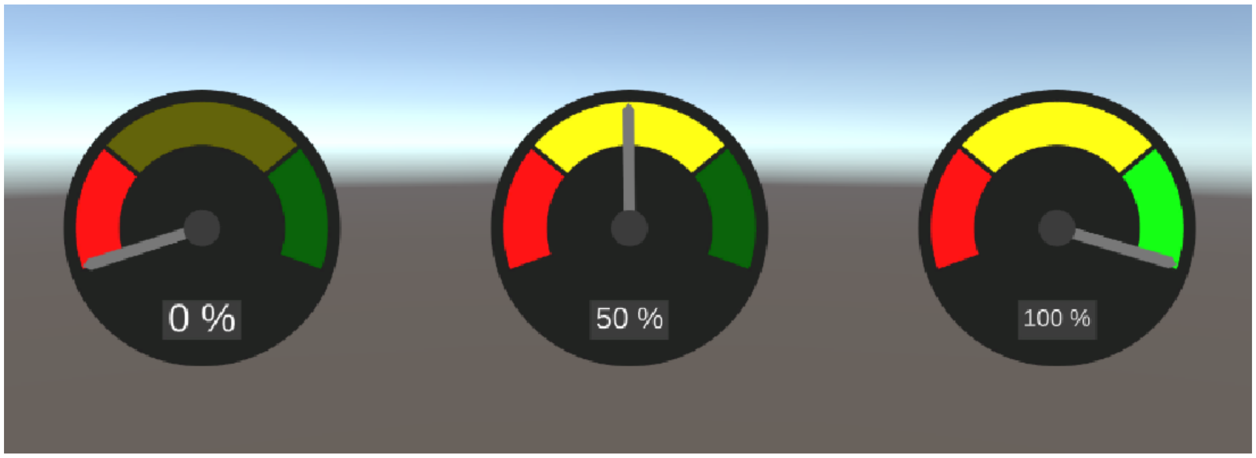


Figure 42: The copy meter

The other progress indicators also received some redesigns. This includes the train of thought which received an updated look and placement. As we tested with the train of thought some players expressed how difficult it was to quickly look over and understand the symbols. To alleviate this we created new symbols for the possible actions. To represent the dodges we created arrows to point in a specific direction depending on the dodge. For the punches we used a boxing glove with an impact on the side that is used. To help differentiate the symbols from their left and right counterparts, we gave the left side icons a red color scheme and the right side icons a blue color scheme. We also separated the icons to be on a specific side of the area so the left icons would appear on the left side and the right icons would appear on the right side of the train of thought area as can be seen in Figure 43 below. The location was moved from the right side of the screen to the left side of the screen to reflect the updated Agent AI location. This would help us to try to connect the agent to its train of thought and to keep the UI Training visualizers closer together to prevent players from looking all over the screen to receive information.

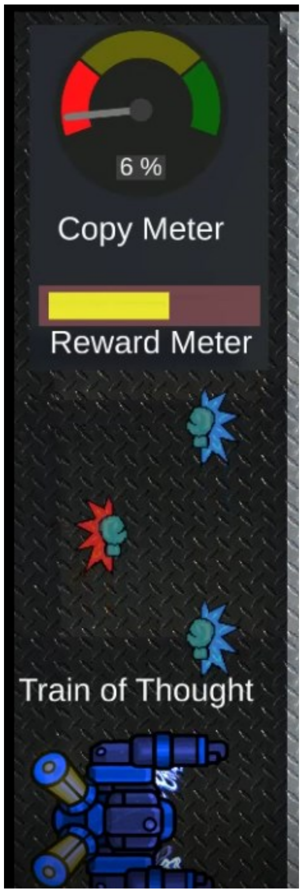


Figure 43: Updated train of thought

The reward meter was added to show how much Roboxer was being rewarded. It went through several design iterations. The initial idea was to have something like a light bulb. As the reward increased, the lightbulb would increase in brightness, and slowly diminish over time. This design wound up being too imprecise, as it was difficult to get a sense of context with the brightness. Instead, a design was chosen to represent the reward meter as a sort of bar graph. Rather than increasing in luminosity, the bar graph would elongate and shrink depending on how much Roboxer was being rewarded, as seen in Figure 34 above. Having a physical object change in size with an obvious maximum and minimum wound up being a far more effective design than a lightbulb changing its luminosity.

5.5.6 Improved Enemy AI

The previous enemy AI was the same as what the player would see the training dummy perform in the training session. The enemy AI was improved to provide more of a challenge to the player. The final boss of the game is what Roboxer will fight during the final match. The boss follows a scripted series of actions with some input for dodges based on Roboxer's actions. When the boss is a state in which the punch and dodge cooldowns are zero and the player's agent is punching, the boss will have a 25% chance to dodge correctly, a 25% chance to dodge incorrectly and a 50% chance to ignore the incoming punch and perform the next action in the rotation. This was done to allow the enemy to appear to dodge naturally instead of using predetermined timings since we will not know when Roboxer will choose to attack and this may break the immersion of the player.

The boss is programmed to follow a set series of actions and repeats after it has gone through all available actions. The series of actions taken is meant to mirror simple combos the player may have taught Roboxer. These include left punch, left punch, right punch; right punch, right punch, left punch; right punch, right punch, right punch; and left punch, left punch, left punch. These actions were chosen to mimic what some players may choose to train their ai agent with and to provide a variety of actions that appear to not be random. The enemy AI also received more hit points to have the fight last longer. The fights originally did not last long with the winner most of the time going to Roboxer. With increased health, the matches last significantly longer and are more interesting.

5.5.7 Adding Audio

Audio feedback was an important part of the game to make actions feel more impactful. Before audio was added into the game, it wasn't always clear when a punch made contact with the enemy. The structure of the audio scripts allows for choices of a series of different audio cues for different game events, such as punching, dodging, or being hit. The combination of these sounds help the game world feel more realized. The specific sound choices are detailed in section 3.7.

5.5.8 UI Changes

After receiving feedback from playtesting sessions we found out that some of the background is not needed and was causing confusion for some players. The main parts of concern were the PC desk area in the bottom left area of the stage and the defensive training ring in the top left. The PC desk area was removed and this allowed us to move Roboxer to that location along with the Train of Thought. Removing the ring in the top left corner allowed us to also place the copy meter and the reward meter (Figure 44).

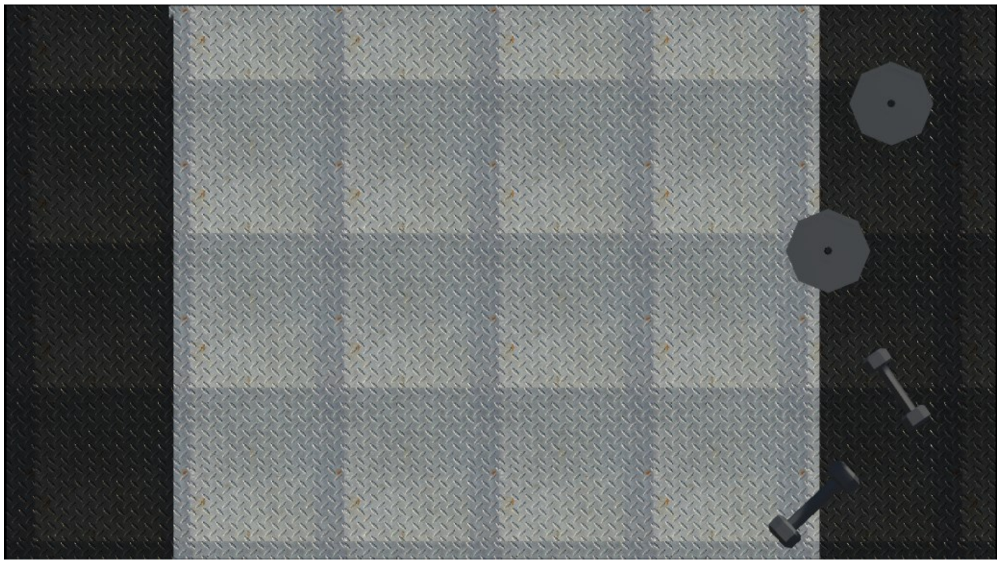


Figure 44: Final training area with no UI elements

The final match area was also changed to match the same art style as the training area. This also changed the logo in the middle of the stage to MLA to match the name of the game (Figure 45).

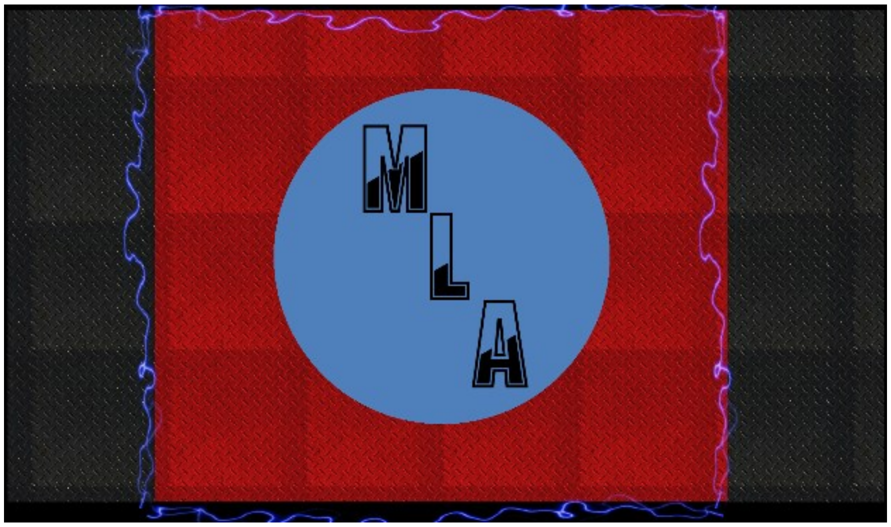


Figure 45: The final release of the fighting arena

A new control panel was added to the game to remind the player of the keybindings as well as associate the timeline icons with actions. The panel replaced the text-based one and took visual inspiration from the popular video game *Overwatch* [5]. The updated control panel can be seen in Figure 46.

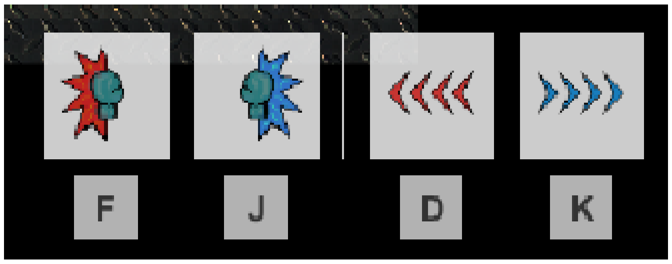


Figure 46: Updated control panel

5.6 Final Build

5.6.1 Machine Learning Input Sanitization

At AlphaFest, we noted that all of our playtesters were having trouble getting Roboxer to both dodge and punch. From our observations, the players did not provide consistent enough input for an ML model to properly learn a policy. For example, we observed that many players punched the training dummy while it was throwing punches and only occasionally dodged. This behavior seemed to cause Roboxer to only learn punches (and the opposite if players dodged too much), and our playtesters did not think it learned from them very much. Many of our playtesters also stated that they thought Roboxer should have dodged at least a few times, even though they only showed it a handful of successful dodges. To address this, we decided to filter the player's input before training on the data. More details about the filter can be found in Section 4.1. After conducting several more playtests with the filter in place we determined that players enjoyed the experience and felt that Roboxer learned from them. By doing this, we discovered that a balance needs to be achieved between having a "pure" machine learning game, which trains off of anything the player does, and a fun game that allows for learning characters, but is a bit scripted.

5.6.2 Final Boss Changes

At AlphaFest, we noted that our final boss would constantly dodge and perform a small number of punches. We also noticed that the final boss would perform these actions very quickly with little to no delay between each action. To remedy these issues, we changed the percentages for the final boss to dodge incoming punches. Now the final boss will dodge less often and punch more when it is being attacked. We also increased the duration of the final boss's nothing action. This allows the Roboxer to get some hits in without needing to constantly dodge an incoming punch. The nothing action happens after a string of punches from the final boss. This creates a cycle of dodges and punches for Roboxer to navigate in order to defeat the final boss.

5.7 Conclusion

Over two terms, our game underwent significant development work and there were many major changes. The areas which changed the most were the machine learning system, the training phase, and the training progress indicators. The machine learning system changed every time we modified a game mechanic or changed the game flow. This ensured that the ML was optimized for the latest version of our game. We also made a lot of tweaks to how the ML system learned and how fast it could pick up moves from the players. Our training phase also saw a lot of changes, most importantly, we removed the defensive training session. Through playtesting, we found that this session caused Roboxer to learn moves that the player did not teach it and our testers did not believe it learned from them well. As we conducted more playtests, our visual progress indicators saw a significant amount of changes to become clearer to players.

While working on this project, our goals also evolved to reflect the feedback we received, as well as our timeline. We found that we spent a significant amount of effort trying to communicate ML concepts to players through our visual indicators. In total, we created eight UI elements to display progress, three of which made it into our final game. A goal of our project became creating a way to convey ML information to players who may not have an ML background. We also moved our focus to creating a technical demo rather than a game with a full story. This change was necessitated by our accelerated timeline and difficulties we were having with getting an ML system that players found enjoyable.

We were faced with several challenges during our time developing this game, such as: players not understanding our training indicators, receiving poor training data, and speeding up the training time. Information about the challenges we faced regarding our training indicators can be found in Sections 5.3.5, 5.4.2, and 5.5.5. Our solution to receiving poor training data is documented in Section 5.6.1 and the process we used to speed up training time is seen in Sections 5.3.3, 5.3.4, 5.4.1, and 5.5.4.

6 TESTING

6.1 Methodology

Our goals for the project are focused on creating a game that combines human interaction and artificial intelligence interaction through machine learning. The game is made in the Unity game engine and has the player training an artificial intelligence character to compete in boxing matches against other characters. The game is split into two gameplay phases: training phase and combat phase. During the training phase, the player trains Roboxer to perform boxing maneuvers by performing the actions themselves and then watches Roboxer perform the actions learned. The player will then have the opportunity to train Roboxer more or to send them to the final fight.

When training is completed, the next gameplay phase will start. For this gameplay phase, Roboxer competes against an enemy in a boxing match. During this phase, the player takes a passive role and watches Roboxer compete. Roboxer's performance will depend on the training it receives from the player.

We obtained feedback on our game throughout the design process and started by receiving feedback on a basic UI mockup of the game which includes a storyboard for background details. We asked survey participants whether the game story and UI makes sense and if they would find the game interesting. This feedback was used when creating our initial prototype of the game.

Once we had a working prototype, we conducted live gameplay testing with several students by holding a playtesting session in the Zoo Lab. This was promoted by sending an email to the WPI IMGD course mailing list and offered IMGD playtesting credit. We had several computers set up in a WPI computer lab with our game and asked the participants to play the game using only the game's instructions. After completing the game, we conducted in-person surveys with the participants to get feedback on the experience. Some of the questions that were asked include:

- Was the playing experience enjoyable?
- How easy was it to train the fighter?

- Does the information on the screen clearly describe what to do? If not, what should be changed?
- Did you like training the fighter?
- Did the fighter seem intelligent?
- Did the fighter do anything that you did not expect? Please describe.
- Did you find the combat phase interesting to watch?
- What would you recommend changing in the training phase of the game?
- What would you recommend changing in the final fight phase of the game?

We also asked some for further details on the responses to the above questions and recorded the responses through writing. The responses were analyzed after the testing was completed, and we made modifications to our game accordingly. We had multiple rounds of playtesting as we developed the game. Our goals for each playtesting session were different, and our objectives are detailed in the experiment subsections of the following sections. The testing took place throughout B term, and the UI mockup feedback took place in the middle of A term.

6.2 UI Mockup Survey

6.2.1 Experiment

The UI Mockup survey was given out over the course of a week. Participants were given a Google survey containing mockup images of each of the gameplay phases and questions. The survey can be seen in appendix G.

6.2.2 Results

Most of the people queried found the idea of training a machine learning agent to be engaging. However, there were concerns about the UI elements presented. 50 percent of responders thought it wouldn't overwhelming to divide their attention between the UI elements and the training of the Roboxer, while the other 50 percent thought it would either be overwhelming or were unsure either way. In addition, 71 percent of responders reported that they either had not or were not sure if they had ever played a game that involved taking player input and using it for some type of AI training. Our full findings, including graphs of the responses, can be found in Appendix B.

6.2.3 Discussion

For our UI Mockup Survey we wanted to see how some initial concepts thoughts for our game would appeal to players. For the main menu we found that players did not have a positive reception to it. They stated that the main menu would mostly tell the player about the game and not show what the game is about. For the offensive stage, surveyors liked the layout off the offensive stage but did not like the content that was shown. For example, surveyors did not like the phone UI to keep track of move combos and other information. For the defensive training, surveyors like the simplicity of the stage, but felt that the screen was too cluttered. For the Tournament Stage, players liked the shouting and tournament mechanics but found that the layout of the screen was too confusing. From these results we decided to do a full UI redesign. This included, removing the main menu, removing the reinforcement buttons and separate views from the gameplay area to the UI area.

6.3 Alpha Playtesting

6.3.1 Experiment

Alpha playtesting was conducted over the course of one week. Testers were given an explanation of how the game worked, as there was no tutorial implemented in the Alpha. Testers were told to attempt to train the Roboxer in the first two phases, then observe the results in the fighting phase. Following the conclusion of the fight, testers were asked whether they felt they felt as though they taught the Roboxer anything. Testers were also asked about the effectiveness of the UI elements, the layout of the controls, and the enjoyability of the experience.

6.3.2 Results

We found that 80 percent of players felt as though they had an impact on the actions of the Roboxer. 40 percent of testers did not like or did not notice any of the UI elements, while 60 percent like some or all of them. 80 percent of players found training the boxer to be enjoyable. When asked what they felt they managed to teach the Roboxer, testers responded that they recognized some of the actions they performed, but not others.

6.3.3 Discussion

The results of Alpha playtesting left us with several key points to consider moving forward. Most pressing was the information that players weren't entirely certain what they had managed to teach the Roboxer. Although they felt as though they had an impact on the actions the Roboxer performed, the exact effects of their training were not clear enough. As a result of this feedback, the decision was made to remove the second phase of training from the game. A full discussion of this decision can be found in section 5.5. The other biggest takeaway from alpha testing was that the UI needed to be improved. Forty percent of testers reported that they did not like any of the UI elements. When asked about their responses, some of those testers reported that they did not even notice the UI elements. Following Alpha testing, an effort was put into revisiting the UI of the game, also detailed in section 5.5.

6.4 Post-Alpha playtesting

6.4.1 Experiment

Post-Alpha playtesting was conducted over a week with changes being made to the game with each new session. For our testing sessions, we had playtesters play the game and asked them questions about certain game mechanics and features that we wanted to improve upon rapidly.

6.4.2 Discussion

Post-Alpha playtesting occurred sporadically throughout the time period between Alpha and Beta testing. The main purpose of post-alpha testing was to determine the effectiveness of both the removal of the defensive training phase, as well as the effectiveness of the new UI designs. Each test would be followed by iterative updates, so no two testers during this time period received the same build. Over the course of testing, updates were made to the position and style of the UI elements until they reached the design used in Beta Testing. In addition, a new tutorial was developed and tested during this time so that no explanations would need to be given to the testers.

6.5 Beta / Alphafest Playtesting

6.5.1 Experiment

Beta testing was conducted over the course of one week, and was concluded when we brought the build to Alphafest. We had several objectives for beta testing. As with our previous tests, we wanted to determine how much influence a player felt they had on the Roboxer. However, an effort was made specifically during beta testing to determine the effectiveness of the newly added UI elements, the tutorial section, and the coach's dialogue throughout the training session. For the tests, two versions of the beta build were created. One contained coach dialogue throughout the training phase that encouraged specific training patterns while the other had no such direction. Testers would be given a build at random, and their actions during training would be watched closely. No instructions for the game were given to the testers, apart from an explanation that they were going to be attempting to train a machine learning agent to win a boxing match.

6.5.2 Results

With a total of twelve testers, there was no discernible difference in the behavior of fully trained agents between testers who played the build with directed training and those that played without. During training, those with the directed training build either followed the training suggestions for only a few moments, or ignored the suggestions entirely. No testers followed the suggestions throughout the training process. None of the testers reported that they felt as though they had successfully trained Roboxer. All testers understood the controls after finishing the tutorial.

6.5.3 Discussion

We were very interested in seeing the responses to the UI elements during this round of testing, as serious effort had been made to improve them and provide more information to the player. We were very happy with the responses to the tutorial, as no one expressed or showed confusion with controls or objectives after the tutorial was completed. However, we were surprised at the results of the directed vs. non-directed training. We had originally introduced it as a way to guarantee some level of competence from a fully trained agent. We feared a possible downside of the directed training would be the removal of individual personality from full trained agents, and that all testers who followed instructions would wind up with a very similar Roboxer. However, we were surprised to see that those given training suggestions almost completely ignored them. As a result, their Roboxers were very poorly trained at the end, similar to the testers who had played without training suggestions. This tied into the feedback players gave indicating that they felt they hadn't trained the Roboxer at all. We deliberated several options to help guide players towards a well trained Roboxer. One of the options considered was to make the tutorial heavily guided, to the point where incorrect actions would be locked out to the player. This was deemed to be too restrictive, and instead, the decision was made to "sanitize" user input, so the Roboxer would only train off of good actions. This process is detailed in chapter 5.6 of the report.

6.6 Concluding Experiment

6.6.1 Experiment

For our concluding experiment, we tasked players to identify the difference between a random agent and a learning agent. This would give us insight into whether people could actually tell that Roboxer was learning from them rather than just coincidentally performing similar moves. Before playing, we gave our subjects tips on how to train Roboxer and told them that they would be playing two games: one with a random AI and one with a learning AI. We disabled the training progress indicators prior to having our subjects play the game, so that we could just focus their attention on Roboxer's behavior. We then chose an AI (random or learning) at random for them to play with. After completing the first game, we had them play a game with the other AI and then asked them to identify which was random and which was learning from them.

6.6.2 Results

In total, we had 27 subjects for our concluding experiment. To know if our learning AI could be distinguished from a random agent, we conducted a significance test with the null hypothesis of $P = 0.5$ (randomly guessing). After completing the experiment, we had 23 out of 27 correctly identify the learned agent and can reject the null hypothesis on a 0.01 significance level (P-Value = 0.0003). From the qualitative elaboration from players, we learned that they were able to tell which one is the learned agent because they recognized some of its moves as their own. For those that incorrectly identified the random agent as the machine learning agent, different reasons were given. One player felt that the rapid, “twitchy” actions of the random agent were closer to how they played. Another player reported that the actions of the trained agent were “too good,” and so the random Roboxer must have been the one that learned from them.

6.6.3 Discussion

Our results indicate that players can tell the difference between a random and learned agent and feel that the learned agent picks up their moves. The reasons they gave occasionally varied, but by far the most common justification for their decision was recognizing their own behaviors in the Roboxer. However, the choice justifications given by the testers who guessed incorrectly bring up interesting points for discussion. One of the players found the actions of the random agent to match more with how they played during training. This could indicate that there are certain behaviors that our algorithm doesn’t replicate as well, such as the “twitchiness” described by the player. The player who reported the trained agent being “too good” tested the learned agent first, and the random agent second. They said that they were still getting used to the controls and concepts of the game in the first run, but the Roboxer won anyway. As a result, they felt as though the actions taken by the Roboxer did not reflect the (in their opinion) poor training given. This goes back to the problems posed by sanitizing input. By removing the bad training data, many more players wind up with a well-trained agent that can win a fight. While it is still possible to train the Roboxer to make mistakes, it is far more likely to succeed. For some players who do not feel as though their training strategies are adequate, it can be jarring to see a well trained Roboxer emerge regardless.

7 POST-MORTEM DISCUSSION AND RECOMMENDATIONS

If we could restart the project with the knowledge we have now, we would have done more research at the beginning of the project into imitation learning and ways to communicate and explain ML concepts to players. These areas turned out to be our biggest challenge in the later stages of our project. We also believe that we could have produced a more complete game if given another term or semester, as our timeline was rushed and we had to divert focus away from polishing our game's details.

7.1 Team Dynamics

We felt that our team dynamics were very successful throughout the length of the project. We constantly posted updates via text communication and in-person or online meetings on four days of the week. We were able to divide up the work in a way that was reasonable for everyone and their schedules. Kyle focused on implementing and refining the machine learning system. Jordan worked on the fighting mechanics and animations. Grant and Justin primarily focused on the UI elements and game flow. Everyone also contributed to systems outside of their primary focus areas such as creating the enemy AI and ML progress indicators. Overall, we felt that we had a strong team and were able to complete a lot of work in a limited time; it was useful to have everyone responsible for different focus areas.

We believe that our communication was great but it could be improved in several ways. One of our members was a commuter student, so it was harder to organize in-person meetings as we had to work around his schedule. For the most part, we conducted online meetings, but some things such as code reviews and collaborative coding would have been easier if everyone was available to meet on campus. Another area of improvement would be a better use of Trello to track tasks, as there were several points in the project where we were unsure of who, if anyone, was working on certain tasks. If we all had consistently used Trello to assign tasks and update their statuses, we could have mitigated this issue and been more knowledgeable about the status of our tasks.

For the most part, we all wrote clean code and made small commits often. But, during the end of our project, we began having multiple merge issues related to the main scene in Unity. This file is in a computer-readable format that is hard to interpret, so when merge conflicts arise in this file, they are hard to manage. There were several times when we lost some work due to incorrect merges and we lost several hours trying to restore the changes. We believe that these conflicts could be reduced if we followed some of the best practices in Unity, such as creating more prefabs. We only used a single scene so that we could keep the Academy object alive for ML Agents to run, but there may be a way to retain a game object between scenes that would have allowed us to split our scene into sub-scenes. Other than that, we could have benefited from occasional refactors (which we did, but not often) that would have made our codebase easier to change.

7.2 Time Management

Given that we were completing our project in an accelerated time frame, we were able to successfully manage our time to ensure that all major components were completed. We began at the start of the summer break so that we would be in a good starting place when we met with our advisors in A term. We decided that by the end of the summer we would have a finalized idea for our game and prototypes of the fighting mechanics and ML. Once the project was officially started, we used Trello to assign and manage tasks for the week and were on time with most of our iterations. At the start of the terms, we outlined deadlines for major releases of our game such as the Alpha and Beta releases. We were able to meet all of our deadlines and complete most of our major components. The only system we were unable to complete was the combo mechanic.

As stated in Section 7.1, we believe that using Trello to track all of our tasks would have made our time management a bit better. Doing a two-term MQP was challenging, and due to the time restrictions, we had to prioritize implementing certain features/story elements over others. We recommend outlining exactly what a project will accomplish and what will be left out at the start of a shortened MQP, rather than doing so as the project progresses.

7.3 ML-Based Games

We learned that creating a machine learning based game is difficult. Implementing the actual ML into the game using ML Agents was very straightforward (took under an hour), but a lot of tuning and game design is needed to do this successfully. We spent well over 40 hours tuning the ML hyperparameters and inputs to allow Roboxer to be trained in under 1 minute of gameplay. In order to get our ML system to train quickly, we also needed a reduced action space for our game, thus we gave Roboxer only five possible actions (2 punches, 2 dodges, and do nothing). Then, we were challenged with receiving poor input data from players and needed to design an input sanitization filter. On top of this, we needed to create ways to present the ML concepts to players without confusing them.

From creating our game, we recommend allocating a large chunk of time to tuning and refining the machine learning in a game. If training time is a factor in the game, then a simplified input/output space will be beneficial. We also recommend researching and experimenting with ways to convey ML concepts to players, as this took us weeks before we had players say that they were useful, and we still could make a lot of improvements to their understandability.

7.4 Game Story

Our original game story was much more involved than the version unfolded by our game. The original story was about an older boxer (called Coach), who in a fight became injured and retired. A young boxer (called Al) came to Coach asking to become their student, and convinced Coach to mentor them. They schedule Al for a fight against a tough opponent and Coach begins the training sessions. During the training, Coach receives details about the upcoming fight, including what the opponent's favorite combo is. Coach then uses this information to train Al how to defeat the opponent. None of this story made it into our final game, though we did settle for a stripped-down version (Coach teaches Roboxer during training, then Roboxer goes into the fight after).

We felt that the storyline of our game was lacking and it felt more like a tech demo than a normal video game. If we had more time, we could have refined the story and included more narrative elements into our game. We believed our characters lacked personalities and backgrounds, plus all the sprites were just different colors of the same asset. We would have liked each of our sprites to be different, so it would be easy for the player to identify them and it would give us a chance to add more personality to them; for example, we wanted to make the coach sprite look battle-worn and rusty. In our game, we were also missing content related to our game's backstory, as stated earlier. If we had another term, we probably would have been able to give our game a story that would make it feel much more polished.

7.5 Playtesting

We found that it was very easy to get playtesters for our game, especially during the last few weeks of B term. We recruited playtesters by sending an email to the IMGD courses alias and asking them to show up in the Zoo Lab between 6 and 8 PM on weekdays. We believe it is a combination of the time of day and the newly added playtesting requirement for IMGD courses that allowed us to get many student testers.

Toward the end of the project, we began doing daily playtesting sessions which we found to be valuable for our development. Going back, it would have been beneficial to do multiple playtests a week throughout our entire project. There were several design issues and bugs that we would have been able to identify through early playtests which would have simplified future development. For example, most of the game was developed while we did not have the hold to dodge mechanic, but playtesters during the final months of our project felt that it was needed. This change took a while to implement and tune with the ML system because it was added so late in our project.

7.6 Recommendations and Future Work

After the completion of our project, the following recommendations were created based on the challenges we faced during the implementation of a machine learning based game. Our recommendations include further study of conveying machine learning to players, improved algorithms for creating ML training data from inconsistent user input, and further exploration of how ML can be incorporated into video games.

7.6.1 Conduct Further Research into Conveying ML Concepts to Players

We had mixed success when conveying how well training was going to players, and believe that further research is needed in this field. Throughout the course of our project, we experimented with visualizations of training loss which can be a good indicator of how well an ML model was made to fit the training data. We were able to create indicators that displayed a normalized version of training loss and players generally understood that they should try to maximize that. We struggled with letting players know how they could maximize those meters.

We added a tutorial and dialog box during gameplay that displayed tips on how to improve training but players did not read them, or only followed them for a short duration. We also attempted to convey what the ML model's most probable actions were at each step of training through the thought timeline, but many players interpreted this as the moves they should be performing, rather than what Roboxer was thinking; this UI element could be improved through better explanatory text and a more descriptive title. Something that we also noted was that some of the visualizations that we, as developers, found most useful such as the graph were disliked by players because they were "confusing" or hard to keep track of. In all, we believe further studies are needed to better convey machine learning concepts to players unfamiliar with the field so that players can have a better understanding of how well they are training machine learning characters.

7.6.2 Improve the Robustness of the ML System to Handle a Wider Variety of Player Behavior

Prior to filtering player-generated training data, many of our playtesters were unable to successfully teach Roboxer how to fight and they believed that it only learned a fraction of what they taught it. From this testing, we learned that a major factor in this issue was that players were not consistent with their moves and missed many dodges. To resolve this, we implemented a filter that removed training samples that we determined were harmful to successfully training an ML model. Upon testing this new filter, our players felt that they were able to successfully train Roboxer, though some felt that it learned to be too good at the game. We recommend researching better training data filters for video games to improve the AI's ability to mimic the player.

We also believe that the amount of poor training data can be reduced through improved onboarding and guided gameplay modes, such as our Beta build's training dummy. We recommend that further work is conducted to identify and implement ways to encourage players to remain consistent with that moves and provide better training data overall.

7.6.3 Explore Other Uses for Imitation Learning in Video Games

As we demonstrated in this project, ML can be used to imitate player behavior and create a unique character for them to interact with. Based on play traces collected from the player, an ML system could be used to improve NPC and opponent behavior for a more custom-tailored experience. For example, in a boxing game a strong opponent could train (with reinforcement learning) using play traces collected from earlier fights to predict the player's actions and provide a challenge. Another example could be an NPC that uses an ML based recommender system to sell the player in-game items based on their past transactions. We recommend that a game is created in which machine learning based opponents adapt to the way a user is playing.

During our project, we noted that our game could be a good way to teach people about machine learning concepts such as input consistency/quality and training loss. Using a video game to teach ML concepts allows a player to experiment without fear of being penalized in real life - they can just restart the game if they train an AI with poor data. Our game encourages players to be consistent with their behavior, as this produces a better-trained model; other games could use similar approaches to teach players that ML algorithms work best with orderly training data. We recommend conducting further studies on how video games can be used to teach ML.

7.6.4 Continue the Development of Machine Learning Arena

There were several features which we were unable to get in the game and a few that we feel could be improved. Our game has the code in place for a combo system, but there is no visual indicator for it or bonus for performing a combo. We recommend that this be added to the game to improve the gameplay experience. We also believe that our input sanitization can be improved to handle cases where the player does want to train Roboxer to perform poorly (punch while being punched or dodge when it is not necessary). We recommend that more development time is spent on this filter and how it impacts a player's experience. Our game was also presented in a tech demo state, lacking most story elements. This makes it a less enjoyable experience for players, so we recommend that more time be dedicated to improving the gameplay experience. During our final presentation, we also noted that many people enjoyed watching other people play and try to train Roboxer. It may be beneficial to look into ways to make this game cooperative, or a spectator sport-like game to leverage the excitement we saw.

7.6.5 Three Tips for Game Development

We all learned a decent amount about Unity during the course of this project. Several tips that we have for new developers are: 1) Use trigger events for animations, 2) Utilize prefabs and scenes, and 3) Modify ML Agents rather than implementing custom machine learning code. Animations can make a game feel more polished, and we found that development is much easier when using events during animations to trigger actions in code. Having the animations trigger events ensures that the animations will stay in sync with the game logic and tweaks made to timing can be done in the Unity editor. Using prefabs allows developers to quickly modify game objects by modifying values in a single place. In our project, this was useful for our boxers, as we could just modify the prefab to change traits such as HP and damage. Prefabs also make it easier to use a source control system such as Git. Finally, we recommend that developers use ML Agents as a starting point for experimenting with different ML algorithms rather than creating something from scratch. The source code for ML Agents is easily accessible and relatively easy to modify. ML Agents handles the connection between Unity and an external Python/Tensorflow library for developers already. A downside of doing training in real-time is that having an external Python script makes creating an executable a non-trivial task. This is because the game needs to ensure that the correct version of Python is available with the necessary packages to run; on some users' computers, Python may already exist and requesting a different version or that the user install packages may not be feasible. To install a game with an external script, we recommend finding a way to package Python with it and installing it in a way that it does not conflict with existing Python installs. ML Agents does not have a problem being packaged if a pre-trained model is being used.

8 CONCLUSION

The goal of this project was to create an experience for players in which they could train a boxer how to fight and defeat a simple AI boss. Machine learning can be used to learn unique behavior from each player and customize a gameplay experience to match how they play. Our project demonstrates that a game centered on machine learning can be a challenging, yet enjoyable experience for players. We were also faced with several challenges in creating a machine learning based game, such as communicating machine learning progress to players, and obtaining good training data during gameplay. This paper documents how we attempted to resolve these issues, and may serve as an inspiration to future projects wishing to explore similar topics.

From our project, we learned that imitation learning can provide players with a unique gameplay experience. We experimented with multiple imitation learning algorithms and found the best results with Behavioral Cloning. We also learned that conveying ML information to users is vital, and as a result, we created multiple UI elements related to training progress. The techniques used by our team to create a machine learning based game can be built upon by other projects. We recommend that further research is conducted to explore imitation learning within video games and how to convey the underlying ML to players.

Overall, this project explored building a game focused on machine learning and how to convey that to players. Machine learning is relatively new to video games, and there are plenty of unexplored areas that have the potential to impact the future of how we interact with video games. We hope to see more video games that utilize machine learning in innovative and entertaining ways.

9 REFERENCES

[1] Samarth Agrawal. 2019. Hyperparameters in Deep Learning. *Medium*. Retrieved November 13, 2019 from https://towardsdatascience.com/hyperparameters-in-deep-learning-927f7b2084dd

[2] Arc System Works and Bandai Namco Entertainment. *Dragon Ball FighterZ*.

[3] Avalanche Software and Midway Games. *Mortal Kombat*.

[4] Bandai Namco Studios. *Tekken*.

[5] Blizzard Entertainment. *Overwatch*.

[6] Mat Buckland. Jeff Hannan. *AI Junkie*. Retrieved September 26, 2019 from http://www.ai-junkie.com/misc/hannan/hannan.html

[7] Paris Buttfield-Addison, Jon Manning, and Tim Nugent. 2019. *Unity Game Development Cookbook: Essentials for Every Game*. O'Reilly Media, Incorporated. Retrieved from https://play.google.com/store/books/details?id=j4TEtAEACAAJ

[8] Capcom. *Street Fighter*.

[9] Capcom and Backbone Entertainment. *Marvel Vs. Capcom 2: New Age of Heroes*.

[10] Brooke Chan. 2018. OpenAI Five. *OpenAI*. Retrieved September 19, 2019 from https://openai.com/blog/openai-five/

[11] Jinyun Chung, Seungeun Rho, and NCSOFT. Reinforcement Learning in Action: Creating Arena Battle AI for "Blade and Soul." In *Game Developers Conference 2019 AI Summit*.

[12] CyberConnect and Namco Bandai Games. *Naruto: Ultimate Ninja Storm*.

[13] Theresa Duringer. 2017. Race for the Galaxy AI. *Temple Gates*. Retrieved September 21, 2019 from https://www.templegatesgames.com/race-for-the-galaxy-ai/

[14] Electronic Arts. 2018. Teaching AI-agents to Play Battlefield 1. *Electronic Arts Inc.* Retrieved September 21, 2019 from https://www.ea.com/en-gb/news/teaching-ai-agents-battlefield-1

[15] Andy Gavin and Jason Rubin. 2011. Making Crash Bandicoot – part 6. *All Things Andy Gavin*. Retrieved September 21, 2019 from https://all-things-andy-gavin.com/2011/02/07/making-crash-bandicoot-part-6/

[16] gk_ and Jason Hreha. 2017. Anthropomorphism and AI. *Medium*. Retrieved December 11, 2019 from https://becominghuman.ai/anthropomorphism-and-ai-151ed0b17dad

[17] F. G. Glavin and M. G. Madden. 2018. Skilled Experience Catalogue: A Skill-Balancing Mechanism for Non-Player Characters using Reinforcement Learning. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. DOI:https://doi.org/10.1109/CIG.2018.8490405

[18] Erin Hastings, Ratan Guha, and Kenneth Stanley. 2009. Evolving Content in the Galactic Arms Race Video Game. *2009 IEEE Symposium on Computational Intelligence and Games (CIG)* (2009). Retrieved from https://eplex.cs.ucf.edu/papers/hastings_cig09.pdf

[19] Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. *arXiv [cs.LG]*. Retrieved from http://arxiv.org/abs/1606.03476

[20] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation Learning. *ACM Computing Surveys 50*, 1–35. DOI:https://doi.org/10.1145/3054912

[21] Iron Galaxy. *Divekick*.

[22] M. Ishihara, S. Ito, R. Ishii, T. Harada, and R. Thawonmas. 2018. Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors. In *2018 IEEE Conference on Computational Intelligence and Games*

(CIG), 1–8. DOI:https://doi.org/10.1109/CIG.2018.8490376

[23] J. Ivanovo, W. L. Raffe, F. Zambetta, and X. Li. 2015. Combining Monte Carlo tree search and apprenticeship learning for capture the flag. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 154–161. DOI:https://doi.org/10.1109/CIG.2015.7317914

[24] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. 2018. Unity: A General Platform for Intelligent Agents. *arXiv [cs.LG]*. Retrieved from http://arxiv.org/abs/1809.02627

[25] Vadim Karavaev, Tatyana Kiseleva, and Oksana Orlinskaya. 2018. Simultaneous use of Imitation Learning and Reinforcement Learning in artificial intelligence development for video games. In *Integrating Research Agendas and Devising Joint Challenges*, 154–161.

[26] Manteomax. 2019. Drivatars. *Forza Motorsport Support*. Retrieved September 12, 2019 from https://support.forzamotorsport.net/hc/en-us/articles/360005302934-Drivatars

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv [cs.LG]*. Retrieved from http://arxiv.org/abs/1312.5602

[28] Gabe Newell. 2012. Gabe Newell Writes for Edge. *Edge Magazine*. Retrieved September 21, 2019 from https://archive.is/20120909153756/http://www.next-gen.biz/opinion/gabe-newell-writes-edge

[29] Rasmus Nordling and Robin Rietz Berntsson. 2012. AI in Neverwinter Nights using Dynamic Scripting. Retrieved September 26, 2019 from http://www.diva-portal.org/smash/record.jsf?pid=diva2:831226

[30] I. Oh and K. Kim. 2015. Testing reliability of replay-based imitation for StarCraft. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 536–537. DOI:https://doi.org/10.1109/CIG.2015.7317899

[31] Daniele Piergigli, Laura Anna Ripamonti, Dario Maggiorini, and Davide Gadia. 2019. Deep Reinforcement Learning to train agents in a multiplayer First Person Shooter: some preliminary results. *2019 IEEE Conference on Games (CoG)* (2019). Retrieved from http://ieee-cog.org/papers/paper_86.pdf

[32] Marco Pleines, Frank Zimmer, and Vincent-Pierre Berges. 2019. Action Spaces in Deep Reinforcement Learning to Mimic Human Input Devices. *2019 IEEE Conference on Games (CoG)* (2019). Retrieved from http://ieee-cog.org/papers/paper_22.pdf

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv [cs.LG]*. Retrieved from http://arxiv.org/abs/1707.06347

[34] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (October 2017), 354–359. DOI:https://doi.org/10.1038/nature24270

[35] István Szita. 2012. Reinforcement Learning in Games. In *Reinforcement Learning: State-of-the-Art*, Marco Wiering and Martijn van Otterlo (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 539–577. DOI:https://doi.org/10.1007/978-3-642-27645-3_17

[36] Team Ninja and Square Enix. *Dissidia Final Fantasy NT*.

[37] Team Ninja and Tecmo. *Dead or Alive*.

[38] The AlphaStar team. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II.

Deepmind. Retrieved September 19, 2019 from https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

[39] tinyBuild and Dynamic Pixels. *Hello Neighbor*.

[40] Unity Technologies. *Training with Behavioral Cloning*. Retrieved November 20, 2019 from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Behavioral-Cloning.md

[41] 1990. *Punch-Out!! Featuring Mr. Dream*. Nintendo of America Inc. Retrieved from https://www.nintendo.co.jp/clv/manuals/en/pdf/CLV-P-NAATE.pdf

[42] Dynamic Difficulty Adjustment. *Bandipedia*. Retrieved September 21, 2019 from https://crashbandicoot.fandom.com/wiki/Dynamic_Difficulty_Adjustment

10 APPENDICES

Appendix A: List of Project Ideas

- A prison escape game
    - Co-op with an RL agent
- Keep balls in the air with two paddles, one being AI controlled
- **A fighting game where you train an AI**
- An endless runner game
- A battle royale with the player vs ML agents

Appendix B: UI Mockup Survey Results

Main Menu

- Main menu not that popular, only a few are positive about it
- The explanation does help manage expectations though
- Having more to work with on the menu would give people more definite opinions
- However when asked if they dislike it, most said they did not like the "telling and not showing"
- People want the mechanics explained, some story as well as other info on the main menu rather than an explanation of the game.

Offensive Stage

- Most people like aspects of the page, the only thing was keeping track of doing the moves and also the reinforcement buttons being too much
- What people don't like: a couple didn't like the phone, some said having the phone being too far away from the screen and reinforcement buttons, too cluttered.
- People want more info on what the AI was learning and how it's doing.

Defensive Stage

- People liked the centered part of this stage as well as the more simplistic graphics
- People really want the phone to stay a consistent size.
- People also still felt that the screen was cluttered

- Some fixes are probably going to be implemented with more graphics and more spaced out UI

Tournament Stage

- The shouting and tournament mechanics are both well received.
- Some of the things that need to be made clearer
    - The two timers
    - The radio buttons vs. keys
    - How the AI is doing in its internals
    - Also making the fight area bigger
- People also wanted a way to see who is winning more clearly than just the health bar

Overall Notes

- Many of the people have played fighting games before
- And most have not played something with an AI component
- The games include *Super Smash Bros* (I'm assuming the amiibos) as well as *Samurai Showdown*

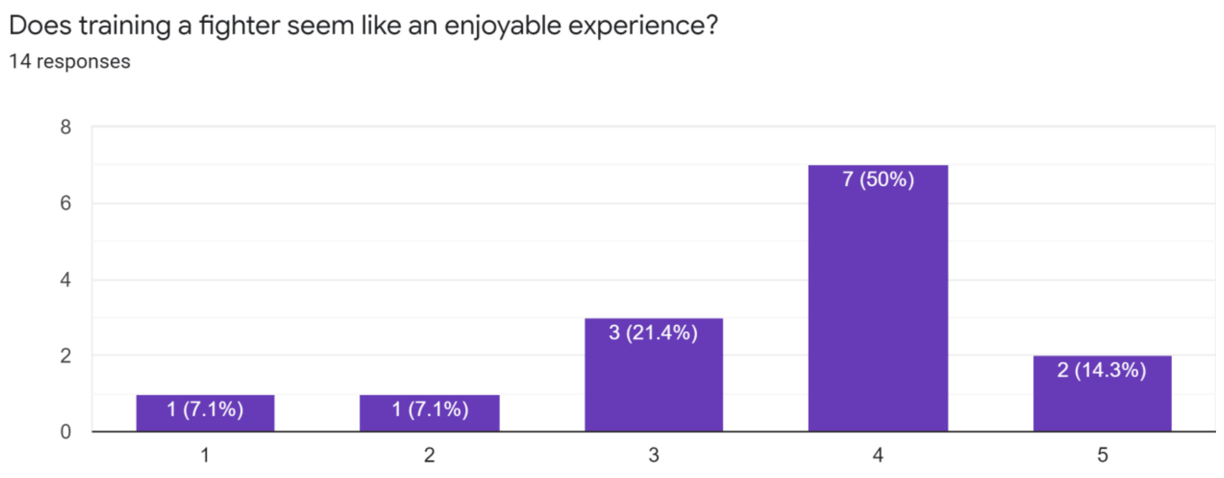


***Figure* 47**: UI survey: game enjoyable

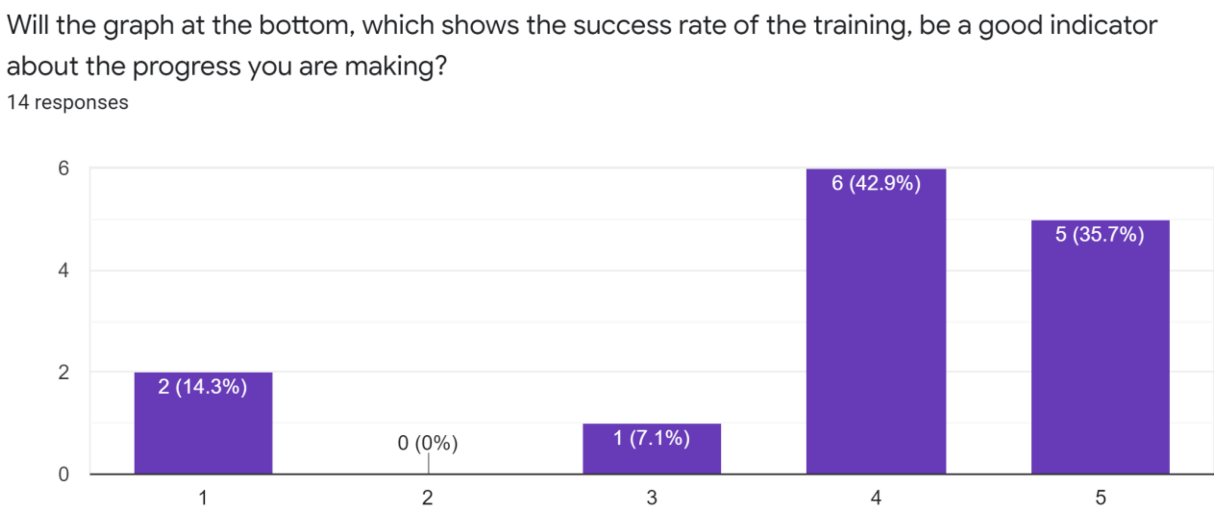


Figure 48: UI survey: training progress graph

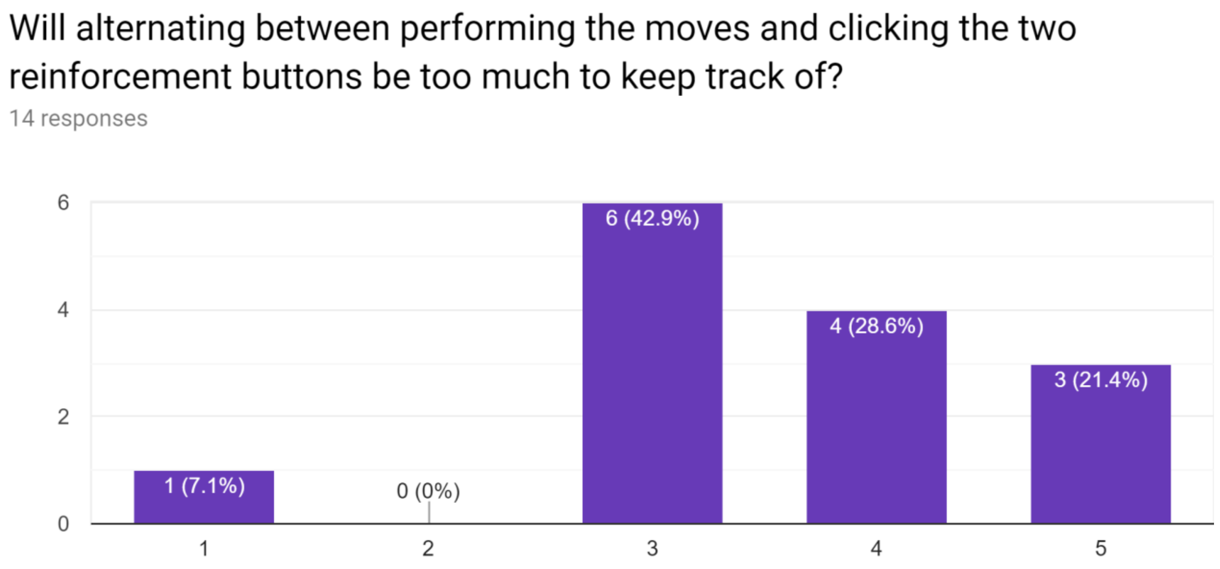
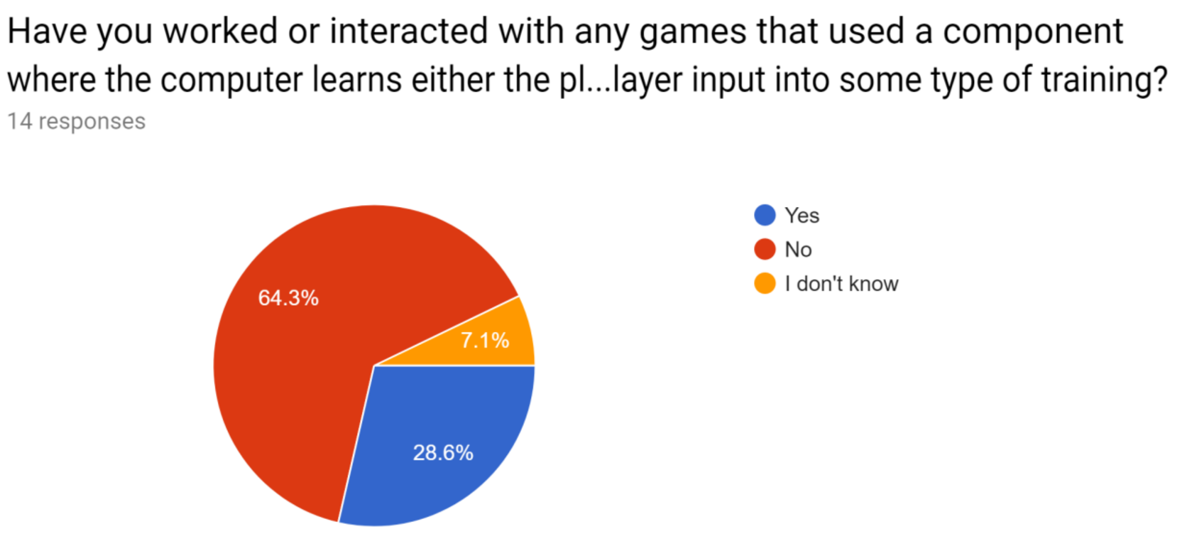


Figure 49: UI survey: reinforcement buttons


Have you worked or interacted with any games that used a component where the computer learns either the pl...layer input into some type of training?

14 responses

Figure 50: UI survey: games played

Appendix C: Hyperparameter Tuning

Table 4 documents the results of hyperparameter tuning on each hyperparameter.

Table 4: Hyperparameter tuning results

Number of layers						
Parameter value	1	2	3	10		
Average loss	0.064265	0.064027582	0.166018602	4.984400088		
Learning rate						
Parameter value	0.0003	0.001	0.01	10		
Average loss	0.0719788	0.064265	0.772394348	368		
Hidden units						
Parameter value	4	32	64	128	256	2048
Average loss	0.333	0.153	0.083	0.083	0.064	0.054
Batch size						
Parameter value	8	16	32	128		

Average loss	0.044	0.083	0.115	2.454		
Buffer size						
Parameter value	100	200	400	600	1200	
Average loss	0.022	0.044	0.032	0.030	0.051	
Batches per epoch						
Parameter value	5	10	20			
Average loss	0.032	0.031	0.026			
Stacked vectors						
Parameter value	1	2	16			
Average loss	0.044	0.031	0.013			
Time horizon						
Parameter value	16	32	64	2048		
Average loss	0.021	0.014	0.031	0.733		

Appendix D: Installation Procedure

The current build of our game requires Windows.

1. Download the game from https://github.com/jacattelona/PunchOut/releases/tag/v1.0
2. Install the required Python version:
    a. Python 3.5.X - 3.7.X
    b. Ensure python is added to the path
    c. You can also run the **install-python.bat** script from the zip file
3. (Optional) Create a virtual environment within Python for ML Agents
4. Install ML Agents

    a. Run the **install.bat** file from the downloaded zip

5. Run the game
    a. Run the **play.bat** file from the downloaded zip

Appendix E: IRB Application

Machine Learning Techniques for Game Artificial Intelligence
9/2/19

Mission Statement and Objectives

Our project is to create a video game that uses machine learning as a primary gameplay mechanic. The game will involve the player training an artificial intelligence character to compete in virtual, low-fidelity boxing matches against other artificial intelligence characters.

Methods Listing

- Observe testers to see their behaviors during play
- Survey of playtesters to obtain players' opinions about our game.

Notes on IRB Application

No risk to human subjects

Methodology Chapter Draft

Our goals for the project are focused on creating a game that combines human interaction and artificial intelligence interaction through machine learning. The game will be made in the Unity game engine and will have the player training an artificial intelligence character to compete in boxing matches against other characters. This will be split into two gameplay phases: training phase and combat phase. During the training phase, the player will be training an artificial intelligence character to perform boxing maneuvers by performing the actions themselves and then watching the artificial intelligence character attempt to perform the actions the player inputted. The player will then have the opportunity to approve or disapprove the action(s) the artificial intelligence character performed. This will allow the machine learning algorithm to apply a reward to the artificial intelligence character to remember the maneuvers. Afterward, the game will transition into the defensive training phase. For this portion of gameplay, the player will be attacking the artificial intelligence character to teach it to defend and dodge against certain moves. The artificial intelligence character will learn to dodge against attack by not taking damage. Successfully defending the attacks will apply a reward to the artificial intelligence character to favor certain defensive techniques.

When training is completed, the next gameplay phase will start. For this gameplay phase, the artificial intelligence character will compete against another artificial intelligence character in a boxing match. During this phase the player will be taking a passive role and watching the artificial intelligence character they trained to compete. The artificial intelligence character will either win or lose depending on how it performs and the player's training regiment.

We plan on getting feedback on our game throughout the design process and will start by receiving feedback on a basic UI mockup of the game which will include a storyboard for background details. We will be asking survey participants whether the game story and UI makes sense and if they would find the game interesting. This feedback will be used when creating our initial prototype of the game.

Once we have a working prototype, we will conduct live gameplay testing with several students via tabling in the Campus Center. This will be promoted with flyers and we may have candy present for study participants. We will have several computers set up in a WPI computer lab with our game and ask the participants to play the game using only the game's instructions. After completion of the game, we will conduct in-person surveys with the participants to get feedback on the experience. Some of the questions that would be asked are the following:

- Was the playing experience enjoyable?
- How easy was it to train the fighter?
- Does the information on the screen clearly describe what to do? If not, what should be changed?
- Did you like training the fighter?
- Did the fighter seem intelligent?
- Did the fighter do anything that you did not expect? Please describe.

- Was the fighter able to defend itself during the training phase?
- Did you find the combat phase interesting to watch?
- What would you recommend changing in the training phase of the game?
- What would you recommend changing in the combat phase of the game?

We may ask for further details on the responses to the above questions and will record the responses through writing. The responses will be analyzed after the testing is complete, and we will make modifications to our game accordingly. We expect to have one or two rounds of this testing as our game becomes more complete. The testing will take place at both the start and end of B term, and the UI mockup feedback will take place mid to late A term.

Sample Questions

- Was the playing experience enjoyable?
- How easy was it to train the fighter?
- Does the information on the screen clearly describe what to do? If not, what should be changed?
- Did you like training the fighter?
- Did the fighter seem intelligent?
- Did the fighter do anything that you did not expect? Please describe.
- Was the fighter able to defend itself during the training phase?
- Did you find the combat phase interesting to watch?
- What would you recommend changing in the training phase of the game?
- What would you recommend changing in the combat phase of the game?

Appendix F: Original Game Structure

Glossary of Terms

Coach: The human player sitting at the computer
Roboxer: The ML Agent the coach is attempting to train
Enemy / Final Boss: The AI opponent that Roboxer will fight

Training Phase: Game time that the coach spends training Roboxer

Fighting Phase: Game time where Roboxer fights the Enemy

Training Phase (AKA Phase 1)

- Coach is given information (in the form of text or a video) about the Enemy that Roboxer will fight in the next phase
    - Patterns the Enemy follows in attacking or defending
- Coach can use this information to better train Roboxer
- Training Phase A (Imitation)
    - Coach will perform maneuvers, agent will attempt to replicate them
    - “Remember these techniques during the fight!”
    - Coach informs agent when they are imitating correctly (positive and negative reinforcement)
- Training Phase B
    - Coach acts as a mock enemy, repeating a sequence of attacks
    - “Learn how to defend yourself against these moves!”
    - Agent is reinforced automatically. They know getting hit is bad, and that blocking or dodging is good

Fighting Phase (AKA Phase 2)

- Agent is placed against an Enemy, the fight begins
- The Coach has no direct control over the fight
- Three Design Options
    - Design 1 (Passive)
        - Each fight is split into multiple “rounds”, like a real boxing match
        - In between rounds, the coach can instruct Roboxer to follow a different preset strategy
        - “The Enemy looks very aggressive, you should block a lot to tire them out”
        - “The Enemy looks tired, now you should go on the offensive”
    - Design 2 (Active)
        - There are no rounds in the fight

        - Coach can shout instructions from the sidelines, telling Roboxer to follow a different preset strategy
        - Put a cooldown on this ability, so it can't be used every moment
    - Design 3 (Powerless)
        - The Coach has no way to impact the fight whatsoever
        - The Coach simply watches the fight play out

Upon Completion of the Fighting Phase, we go back to Training Phase, with information for a new Enemy. The pattern of the game is (Training, Fighting), (Training, Fighting), (Training, Fighting)

Appendix G: UI Mockup Survey

Introduction

The following images you are about to see are mockups for the UI for a new game. This game will be based on the idea of training an Artificial Intelligence to fight in a fighting game. There will be 3 distinct phases shown here, the two training phases as well as the final fighting phase. There will also be a main menu mockup that will have some base questions.

Main Menu

The first page that we have as a mock-up is the Main Menu. This will be the first screen that the players seen upon booting up the first versions of the game. It will give a brief description of the game as well as what the player should expect while playing the tech demo.

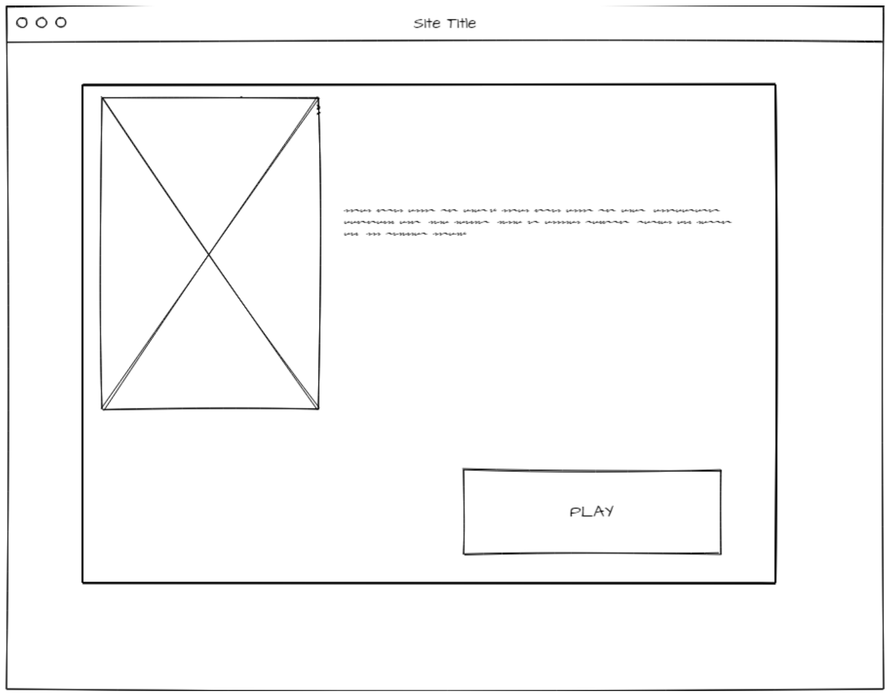


Figure 51: UI mockup survey main menu

- Do you like the idea of an explanation of the game rather than the standard main menu?
- Does an explanation help manage your expectations about the game?
- Does having a singular image help direct your expectations toward the game?
- What design choices do you particularly like about the page?
- What design choices do you particularly dislike?
- What design choices do you particularly dislike?

Offensive Stage

The second page that we have as a mock-up is the Offensive Stage. This will be the player (labeled "Coach" here) showing the Artificial Intelligence (labeled "Fighter") some basic moves. This will entail the player doing a move, the AI mimicking that move, and then the player hitting either "Good Job" or "Not Quite". The phone on the left-hand side will be a set of moves for the player to do, and for the AI to react to.

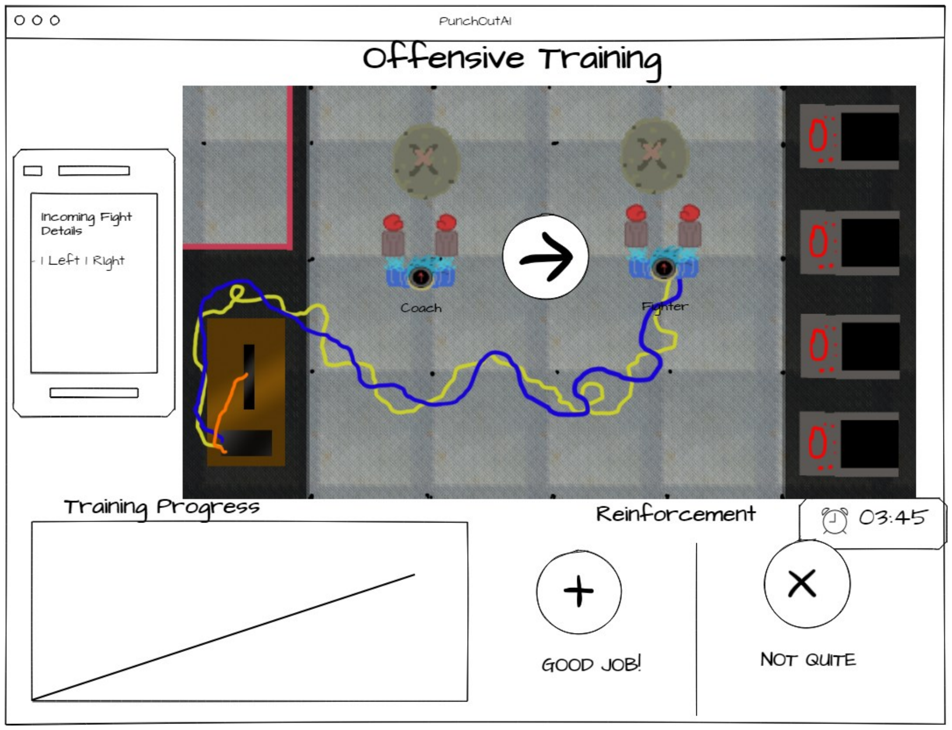


Figure 52: UI mockup survey offensive training screen

- Do you like the idea of getting the directions from a phone?
- Does training a fighter seem like an enjoyable experience?
- Will alternating between performing the moves and clicking the two reinforcement buttons be too much to keep track of?
- Will the graph at the bottom, which shows the success rate of the training, be a good indicator about the progress you are making?
- Does the central arrow clearly depict whose turn it is?
- What design choices did you particularly like about the page?
- What design choices did you particularly dislike about the page?
- What information would you like to see when training an AI to attack?

Defensive Stage

The third page we have is the Defensive Stage. In this stage, rather than the player doing moves and the AI emulating them, the AI will be attempting to learn how to block or dodge moves the player throws at them. The moves will once again come in through the phone on the left, and the graph will show the same data as above.

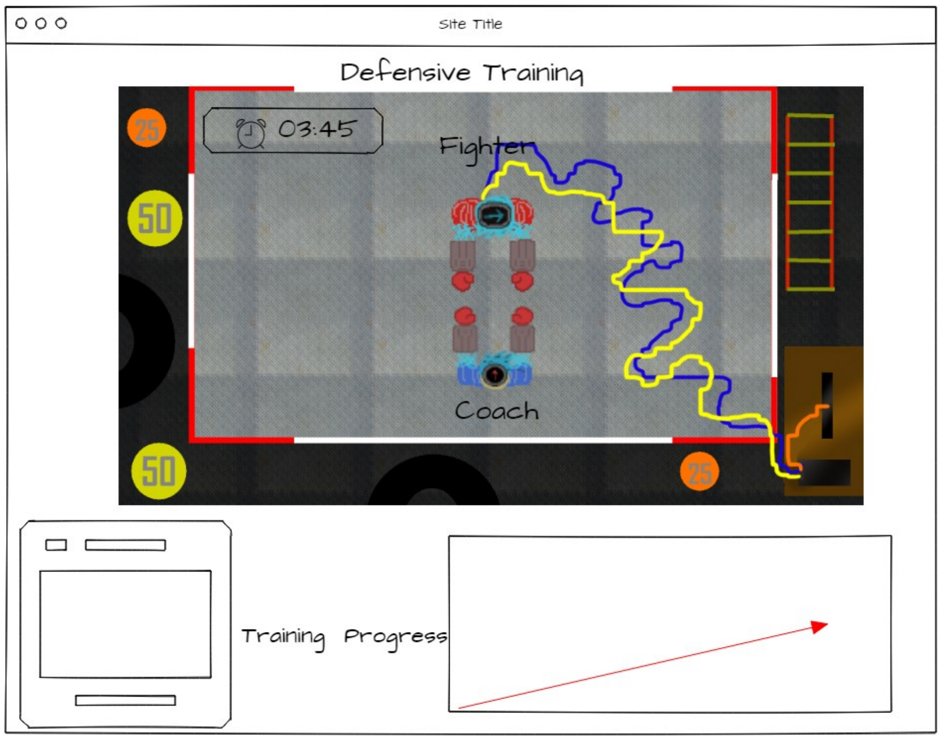


Figure 53: UI mockup survey defensive training screen

- Does training a fighter by sparring with it seem appealing?
- Will getting the instructions via phone be as effective in this stage vs the previous stage?
- Is giving the player training the AI the data still a useful idea in this instance?
- What design choices did you particularly like about the page?
- What design choices did you particularly dislike about the page?
- What pieces of information would you also like to see when training an AI to defend?

Tournament Phase

The final page we have a mock-up of is the tournament stage. In this stage, the player will take a back seat and let the AI take on the fight. The only interactions the player will have is to give some vague commands from the sidelines about general fighting style, which is the timer on the far right(a cool down for this ability). The AI will be at the bottom left of the screen with the enemy on the top left.

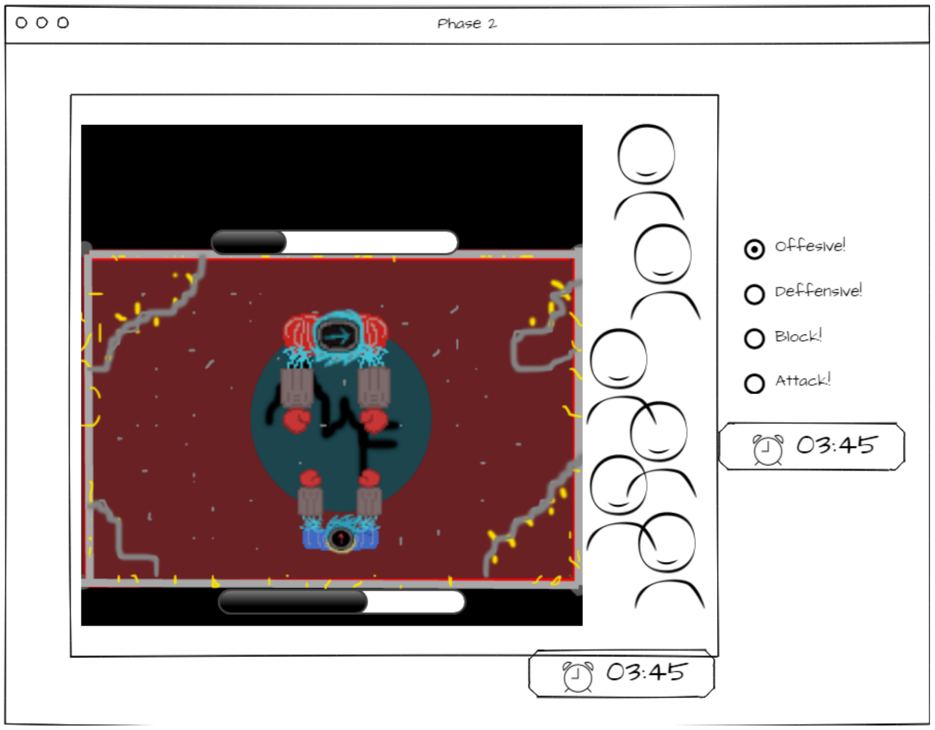


Figure 54: UI survey mockup tournament phase screen

- Will watching an AI that you trained to fight be a rewarding or boring experience?
- Is the shouting mechanic something that you think would be helpful to have some agency in the fight?
- What design choices did you particularly like about the page?
- What design choices did you particularly dislike?
- What pieces of information would you also like to see when watching an AI fight?

UI Testing Questions - MQP

After looking at the UI mockups provided please review and answer these questions.

- Have you played a fighting game or a game similar to "Punch Out", "Mortal Kombat", or "Street Fighter" before?
- Have you worked or interacted with any games that used a component where the computer learns either the players behaviors or the game took player input into some type of training?
- If you answered yes to the question above, what game was it?

Appendix H: Training Hyperparameters

Table 5: Final hyperparameters

Hyperparameter	**Value**
Batch size	16
Time horizon	32
Batches per epoch	10
Learning rate	0.001
Epochs	Determined by training duration
Max epochs	5000000