Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

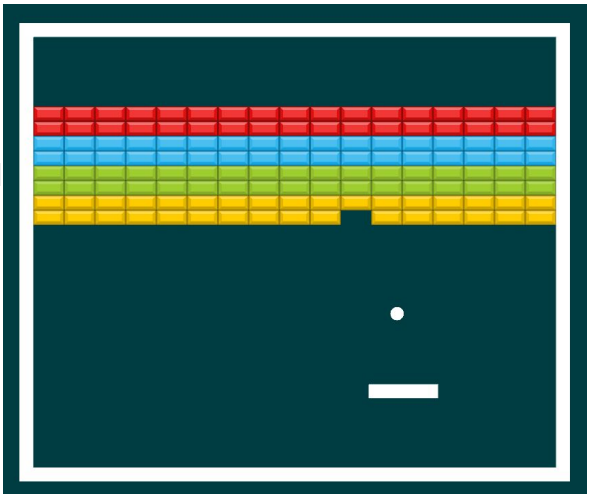
Intro:

Team Bigfoot set out this semester to create a version of the popular Atari game Breakout in Unity with the main focus being the development of a bot-controlled opponent trained using machine learning. The main tool for bot development is the Unity Machine Learning Agents package(ML Agents), a Unity-developed plugin for the popular game editing software. Starting off, neither group members had worked in Unity, Machine Learning, or C#, but we were able to effectively create a crisp and functional Breakout clone within the first half of the project. We spent the latter weeks of the semester creating ML agents and fine tuning our training methods so as to quickly produce a computer-controlled opponent of appreciable skill. We believe we have developed a clean game which showcases the rapidly growing technology surrounding machine learning and we hope you have as much fun playing it and observing the bots as we did building it.

User Perspective:

This project demonstrates machine learning by applying it to the classic arcade game Breakout. The objective of the game is to break all of the colored bricks on the screen. The bricks break when they are hit by a ball and it is the players job to keep the ball in play by moving a paddle left and right to keep it from hitting the bottom of the screen. Variations on the game exist in terms of brick layout, number of “lives”, and additional features such as powerups. Our version consists of a one life, one level model with the standard straight line layout of bricks.

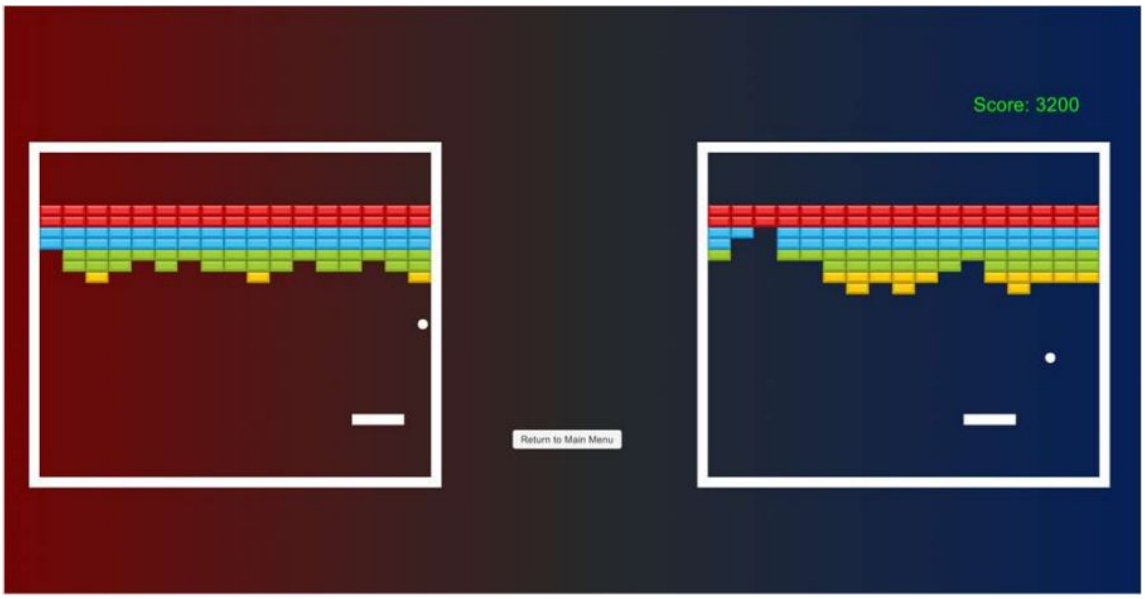
Shown: Game level with bricks, ball, and paddle



Players have two main interactions with the game, watching the AI and playing against it. Choosing to watch the AI will bring the player to the screen on the right. Here the paddle is connected to our trained model and will play the game on its own. The machine plays indefinitely with the game resetting on a dropped ball or a full clearing of the bricks. This option gives the player a no-stakes means of viewing the outcome of machine learning in an environment that clearly displays its results. Alternatively the player can choose to play against the AI where they will control their own paddle in a side by side competition. Here they can compare themselves with the computer and experience first hand that machine learning models can produce results equal to and potentially better than their human counterparts.

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

Shown: AI vs Player screen with the AI game on the left in red and the human screen on the right in blue



Usage Instructions:

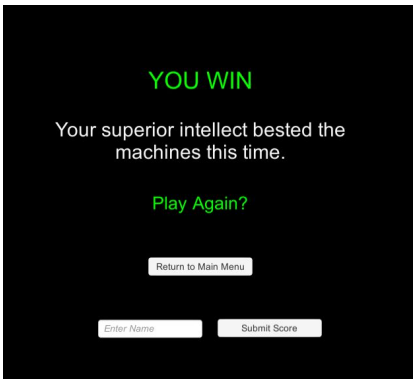
The zip file of the game contains an executable file named Breakout.exe. Open the file and after a brief Unity loading screen you will see the games main menu shown below. Watch AI will show just the trained model playing the game. The paddle will play indefinitely, resetting after the ball hits the bottom or when it breaks all of the bricks. To return to the main menu hit the “Return to Main Menu” button located below the game box.

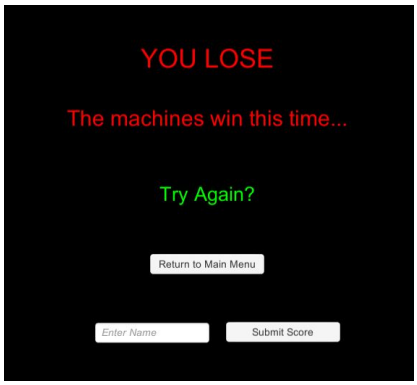
Shown: Main Menu screen

Play Vs AI will take you to the side by side game screen where you will play as the paddle on the right. Use the left and right arrow keys to control your paddle. The game ends when either ball hits the bottom edge of the arenas or when the player has broken all of their bricks. Note: if the test mode option was turned on from the Options page loss conditions are disabled. With this option turned on, return to the main menu by clicking the return button located between the two arenas that is only visible in this mode.

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

After winning or losing the game you will see a game over screen that looks like this:





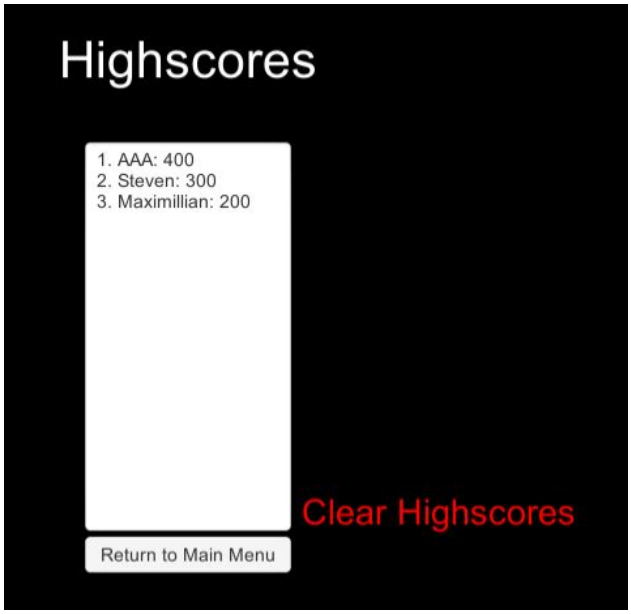
Shown Left: Game over screen when the Player wins

Shown Right: Game over screen when the player loses

From these game over screens you can start a new game by clicking the “Play Again? / Try Again?” buttons, return to the main menu by clicking the “Return to Main Menu” button, or submit your score before picking one of the previous two options. To save a high score simply click on the empty text box and type your name up to 11 characters and then click the submit score button. To view your high scores return to the main menu and click the “High Scores” button which will bring you to this screen:

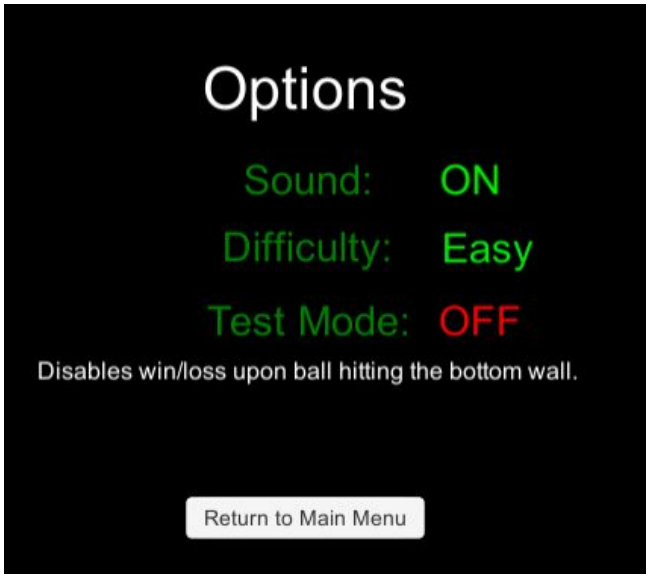
Scores are shown as Rank: Name: Score, from highest to lowest. Scores are maintained across play sessions but are local to your computer. Clicking the “Clear Highscores” button will erase all saved scores and is irreversible. Return to the main menu by clicking the button underneath the score window.

Shown: Highscore Screen



The Option menu, found by clicking on the “Options” button from the main menu, allows you to turn the player game sounds on and off, change the difficulty level of the AI, and turn test mode on and off. Options are changed by clicking on the On/Off or Easy/Medium/Hard text to the right of their listing. Sound is on by default and only affects the player arena on the Player Vs AI

screen. The difficulty level is Easy by default and affects the performance of the AI on both the Watch AI and Player Vs AI screens. To view/play against different difficulty settings you must return to this options menu first. As mentioned, test mode disables loss conditions for both player and AI and is turned off by default.



Shown: Options menu

The “About” Button on the main menu screen will take you to the About page which has written instructions for controlling the paddle in Player Vs AI and information about the project. A button at the bottom of the screen returns you to the main menu.

Lastly at the bottom of the main menu screen is the “Exit Program” button. When you are satisfied with your Breakout experience, click this to close the program.

Software and Systems Integration:

The team spent the first half of the semester developing the base Breakout game which we would then introduce our machine learning agents into.This process consisted entirely of work within Unity, creating game objects such as the walls, ball, and paddle within a project hierarchy. In order to create a game from this assortment of objects, we assigned various components to the objects to trigger default responses from Unity. Much of the game physics stems from built in Unity logic in the form of these components which contain many modifiable attributes. An example would be attaching a rigidbody2D component to the paddle objects such that the game’s physics engine is then able to apply forces and initialize the velocity of the object. On the other hand, a canvas component allows us to display our menus and score tracker.

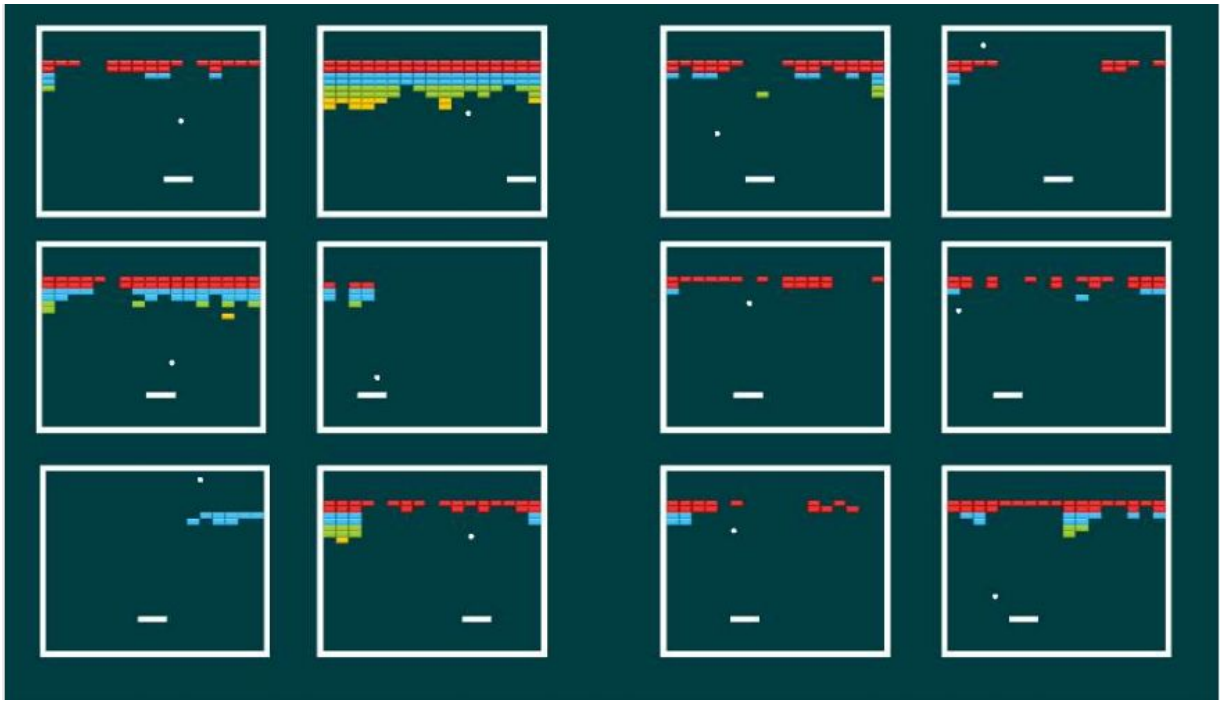
C# scripts were written to handle game logic not inherently available through Unity components. The interaction between the ball and paddle, where the ball reflects away at an angle that defies classical physics, required a script. In this script, the relative location of the ball in relation to the paddle midpoint is determined and then used to calculate a new velocity vector at an angle which mimics that of games like

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

Breakout and Pong. Scripts were also written to handle menu navigation logic, track game state, destroy bricks on collision, play sounds, and much more. Once we had a functioning skeleton of a game in place, we spent the remainder of the semester tweaking collision boxes, reflection angles, brick placement algorithms, and countless other bugs(balls flying outside the arena) until we felt we had a reliable game built which could function as a stable learning environment for our ML agents.

The latter half of the semester was spent integrating the Unity ML Agents toolkit into our project so we could train bots to control the AI paddles. A good amount of time was spent setting up an Anaconda environment to control version packages of Python, PyTorch, and various other dependencies of the ML Agents package. The general workflow to build an ML brain consisted of building the training arena, coding the agents, writing a configuration file which controls learning parameters, and then running the training environment to produce a model which could then be plugged in as an AI controller.

The first step was to split off a singular arena to act as a training environment in a separate scene, complete with logic to reset the arena upon a game ending condition. Later on, we duplicated this environment until we had a 4x3 grid of arenas, all controlled by separate instances of the ML agent trainer, but which were all running independent training episodes that contribute to the cumulative learning of the bot. This multiplication sped up training speed many times. We then progressed to writing scripts to initialize the agents and assign their observations, actions, and rewards. The paddle in the training environment was assigned several ML agent unique components that allow the agent to observe the surrounding environment and then periodically make a decision about the action they will take.



Shown: Training Environment Scene with Multiple Agents Running at Once

Lastly, we set the value of various parameters in a .config file which change the value of constants within the Python algorithm, thereby changing training quality and speed. Training can then start, and the agent is spawned into a fresh instance of the Unity training scene. The ball begins its motion and then feeds its observations and rewards over to the python ML training environment, which analyzes the info and spits out the bot's next action while iteratively improving the model it is forming decisions off of. The tensorboard utility was useful for graphing the learning rate of the agents so we can continue to manipulate training parameters and reward levels until a high quality bot is trained in a relatively quick time. After the number of training steps, or discrete decision events, specified within the configuration file, the training completes and the output model can be used as a brain in the actual implementation of the game.

Libraries, Languages, etc.:

Languages used: **C#**: This was a default language for the Unity platform that was used in all non-machine learning code. This includes ball and paddle movement, placement of the bricks in the arena, game flow logic, and menu operation.

Python: Unity ML-Agents is put together with the Python language and so is used in our model creation and training.

Unity: Unity is a real-time game development platform and was the core of this project. Every visual aspect of the project was constructed on the Unity platform from the sprites to the collision boxes that make the game work.

Unity Collaborate: Unity's built-in version control system for small groups working on the same project. We found early on that version control would be quite cumbersome with Unity, as there are many large meta files which are hidden behind the scenes. We were hesitant to fully rely on Unity Collaborate as the project grew more complex, but it was intuitive and easy to resolve small script conflicts and served us well till the project's conclusion.

Unity ML-Agents: An open source package for unity, this piece of the project deserves its own mention since it is the main driver of the project's machine learning capabilities. With the help of some additional tools this was the API that enabled the creation of our machine learning model.

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

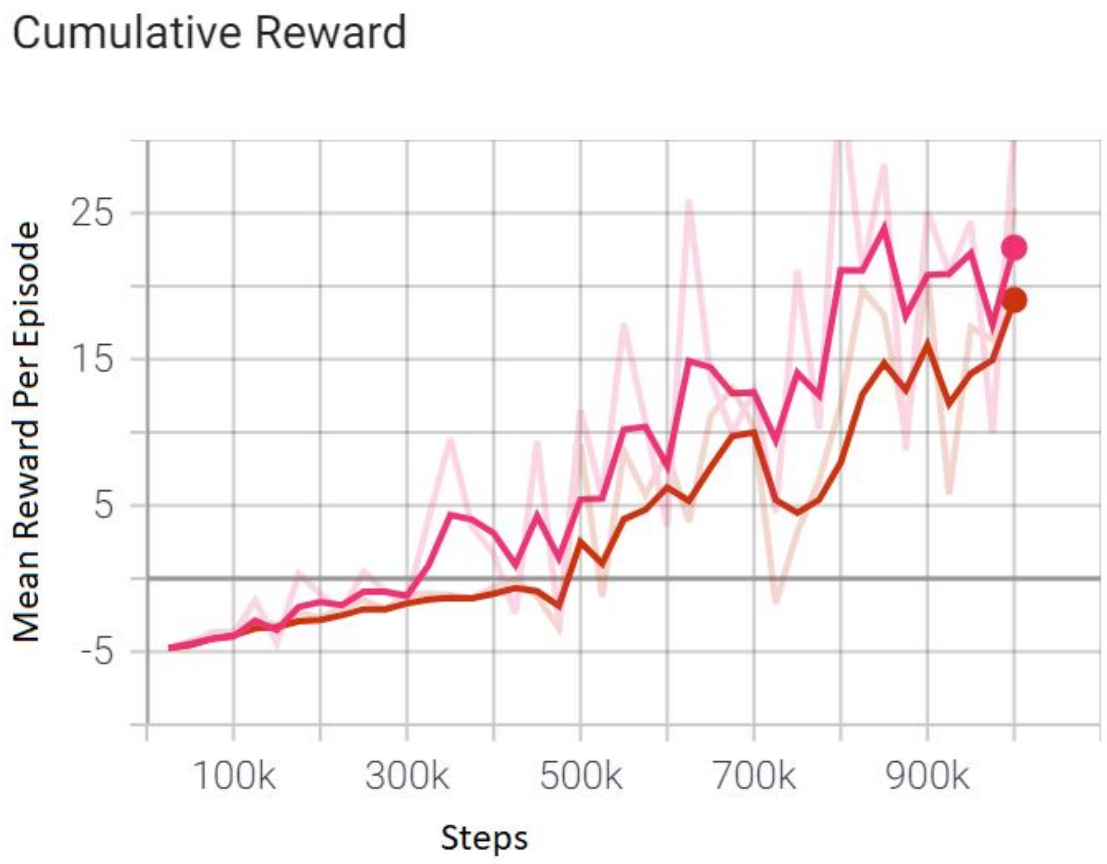
Anaconda: A Python based environment tailored for use in machine learning and data science projects. This is what we used to setup the model trainers. Activating a command prompt from within an initialized Anaconda environment allows us to select the specific package versions which play well with the ML Agents plugin.

PyTorch: A library needed by Unity ML-Agents for creation of the training model that is used by the AI paddle.

NumPy: This python library facilitates the computations that go into the model training to be done over large datasets.

TensorBoard: This visualization tool was used to make graphs of the training data allowing us to make sense of how the training parameters impacted the model and to more accurately make those changes. The graph below details the training progress of two of our most successful agents over the course of 1 million steps. These were produced with the same training parameters. The difference in reward between the graphs is due to the random nature of the PPO algorithm, where novel actions are taken based on random seeds in order to try and seek out rewards. The pink graph took a random action which led to a reward earlier than the brown graph, so it was able to expedite its learning. In this way, no two training sessions were ever the same, and convergence on a solution was not always achieved in the same number of steps.

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout



Shown: Tensorboard Depiction of Training Progress

Team Member Contributions:

Dean Branaman:

For the construction of the game Dean was responsible for the creation of the bricks, and later, creating the script that places them in the desired positions. The game score that updates in real time and the attached high score page was also his handiwork. Dean was the primary developer of the main menu screen from its visuals and layout to the scripts that connected it to the other scenes of the game. As the project developed he maintained the integrity of the menu scripts as more parts were added to it and did the same as new options were added to the options menu.

Like Chris he individually set up and explored the machine learning process. After the team's efforts were combined he assisted in experimenting with training parameters and finding a good model. Once the first iterations of the models were created, Dean enabled the option to select an AI difficulty. This involved finding and

plugging in a finished model via script, based on what the current difficulty selection was. He was instrumental in getting the various scenes that include agents to work alongside each other and in solving a late issue with building the game.

Chris Hauser:

Chris took on the initial development of the game arena utilizing the basic Unity components and writing the script for calculating the reflection angle off of the paddle. Once the game was in a reasonable state, he built the side by side arena format. He then aided with general fine tuning of the game to improve the reliability of the physics interactions and intuitiveness of collisions. He built the sound effect system, created and added in the game background, and added the about and options screen to the main menu. Lastly, he took care of the game over screens and the game state conditions to trigger a win or loss.

In the second half of the semester, Chris and Dean duplicated much of their work, because it was necessary for each of us to fully understand the ML agent tool because of its unintuitive nature and scale. In doing so, we were both able to develop unique intuitions about how to manipulate various training parameters and achieve a good learning outcome rather than teach each other bad habits or shallow understandings. In detail, Chris ended up splitting off the training scene environment, which was initially just one arena, and developing the scripts to initialize the observation, action and reward structure of the agent, and then set up the components to allow for the first batches of training. The remaining 3-4 weeks of the semester consisted of continuous iteration of the ML agents by tweaking training parameters. In addition, Chris duplicated the training arena and changed the brick laying and agent logic to allow for this simultaneous learning. He then had to update the original game stage to reflect the small changes made such that the agents performed in the same arena that they learned.

Task Breakdown:

Week	Team Member	Tasks
3	Dean	- Create Bricks: Make Prefabs, Create Bricks.cs to handle collisions and breaking

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

		- Start Menu screen: Implement play and options button, Create Menu.cs for switching between menu views
	Chris	- Make ball, paddle, and game arena. - Program basic physics interactions and collisions. - Investigated and implemented version control.
4	Dean	- Expanded menu with additional buttons - Visual improvements to menu - Implemented scoring system with GameManager.cs
	Chris	- Create game over screens and win/loss conditions - Create about page - Implement side by side mode(each controlled by different control scheme for now)
5	Dean	- Visual polish on menu - Created BrickLayer.cs to automate placing bricks in the game space - Worked on mid-point project report.
	Chris	- Added audio and background - Debugged collision physics(bounce out of arena, strange glancing blows)

		- Add options menu - Worked on mid-point project report.
6	Dean	- Installed necessary tools for ML training -Familiarized self with tools by creating a simple training environment
	Chris	-Installed ML-Agents plugin and set up environment dependencies. - Worked through Unity ML tutorials
7	Dean	- Tested training parameters - General bug fixes
	Chris	- Developed initial agents and iteratively manipulated training parameters.
8	Dean	- Created difficulty setting in options menu - Added difficulty selection functionality to training scripts
	Chris	- Incorporated trained agents into the side by side mode - Developed a training environment with 12 arenas to speed up learning.

9	Dean	- Created SAC model and attempted to find ideal training parameters - Finalized Watch AI scene - General bug fixes
	Chris	- Continued developing brains to finalize agents for use in difficulty settings - Fixed some bugs introduced during agent integration with base game
10	Dean	- Visual polishing - Solved build errors - Worked on Final Report
	Chris	- Polish Project - Worked on Final Report

Deviations from Plan

Initially we had planned to add in a second level with a different brick pattern if the AI/Player beat the first level. This was left out initially to simplify the training and ended up being beyond our scope for the time we had.

We started to pursue Soft Actor Critic algorithms and imitation learning methods as ways to produce other versions of the agents, but found that we still had more to fine tune on our Proximal Policy Optimization agents to fill the rest of the semester. Additionally, we used a constant velocity angle to initialize the ball each time the game starts, when we had noted that we would be randomizing the starting angle each game.

Team Bigfoot: Christopher Hauser, Dean Branaman
CS467 W21
Project: ML Breakout

We had initially hard coded the velocity angle as a way to reduce variables so that we could better see how the agents progressed in capability, and with other bug testing and fine tuning, it slipped through unnoticed. Running a training environment will immediately make it clear that the agents all act differently from one another despite the same initial ball trajectory, as they are all randomly seeded.

Conclusion:

The end product is a great tool for clearly showing the capabilities of ML agents in devising highly competent actors in video games capable of responding to a changing environment in a thoughtful way, something which has not been possible previously without the use of lengthy algorithms. The team worked well together and was able to meet all of the granular requirements as well as several stretch goals set forth in the initial prompt. This project was a great learning opportunity and chance to be exposed to new languages and technologies.