COLLEGE OF ENGINEERING

Electrical Engineering and Computer Science

ML-BREAKOUT

Jonathan Reuter
Joel Strong
https://github.com/ReuterJo/ML-Breakout

Development of an Atari Breakout game clone and train a neural network with reinforcement learning using Unity ML-Agents to play the game.

MOTIVATION

We were both excited about this project and the ability to create a functional game and learn how to implement machine learning agent to play the game. This project has given us the opportunity to:

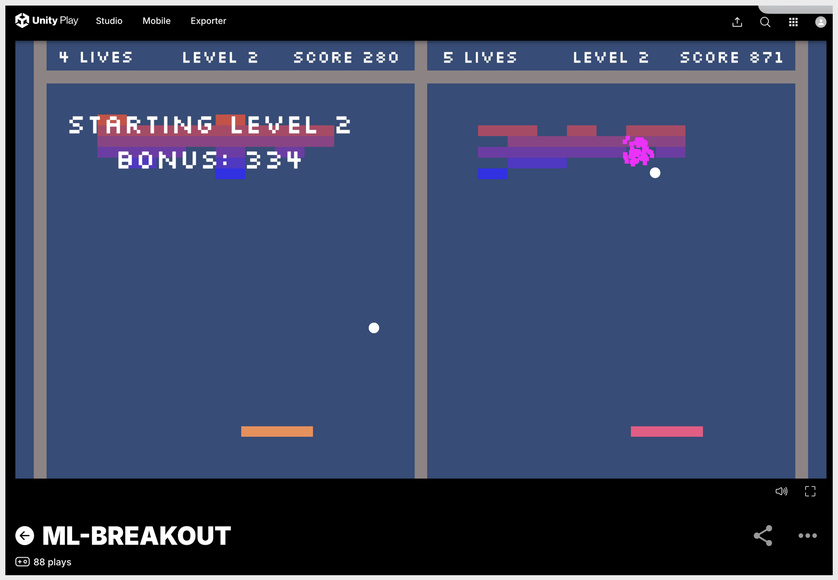
- Learn the Unity game framework and game development in a real-time environment.
- Learn the ML-Agents framework and how to train and implement a ML model.
- Recreate a classic and instantly recognizable arcade game.
- Use a Trello Kanban board to track and assign tasks, issues, and timelines.
- Use a shared GitHub repository with CI/CD to asynchronously develop the program.
- Learn to deploy a finished product to both WebGL and Desktop environments.

RETROSPECITIVE

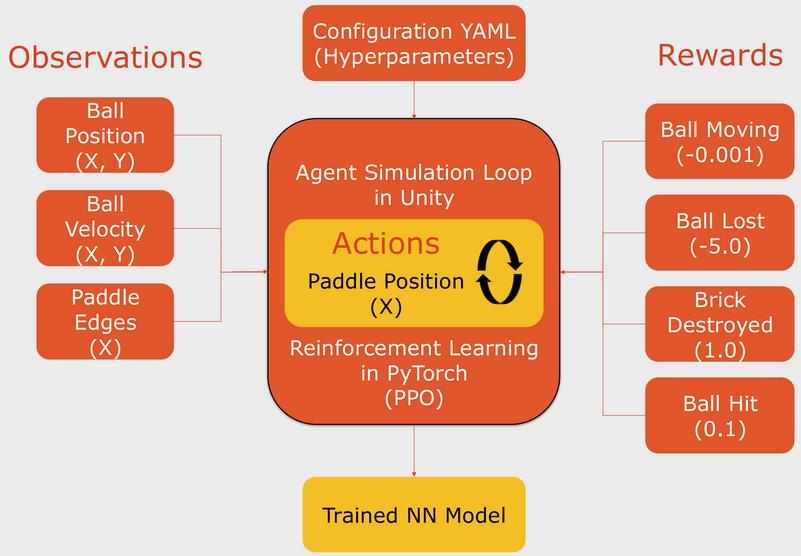
We had a number of key successes and lessons learned during this project:

- GameGenerator – We both wanted to simplify the project by using reusable elements. By defining a single Game object and using Direct Injection to pass the configuraiton data to successively lower levels, we were able to generate any game type (single/double player, single/multi-level and playing/training/debug modes) on demand.
- WebGL version – Once our desktop program was feature complete, we both wanted to investigate deploying an online version of the game for our portfolios. Due to the single-threaded nature of the WebGL environment, we found we could not use the GameGenerator, async, delay, and await features. We were able to branch from prior code versions and refactor the program for WebGL deployment. Both versions have a similar look and feel but have different implementations to meet the differences of their respective deployment environments.

Oregon State University



WebGL deployment hosted on Unity Play (Player Left and Agent Right)



Training Process for ML-Agent

GAME FEATURES

- Ability for a user to compete against ML-Agents trained with increasing levels of ability.
- Defined and implemented the game dynamics for an Atari Breakout clone including the ball velocity, a paddle collider shape, game walls, lost lives, and detecting brick collisions.
- Designed a purposeful game UI including retro 8-bit font and soundtrack, real-time updates to game fields and pop-up text for game messages. Included visual and sound effects to improve the user experience.
- Created a progressive multi-level game that dynamically shrinks the paddle width, increases ball velocity, and generates increasingly complicated brick patterns by level.
- Implemented a persistent game Leaderboard that saves and loads high scores (name, score and date) using a JSON file in the desktop version of the game.
- Control via Main Menu buttons and Pause menus with the ability to Resume, Restart, and Return to the Menu during the game.
- Used functional testing ensure the game dynamics, gameplay logic, and user interface elements met the defined characteristics. We also used non-functional testing for the game performance, usability and compatibility with the desktop and WebGL deployment versions.
- Implemented debugging mode that displays real-time game statistics for game development debugging.
- Deployed a hosted WebGL and MacOS desktop version of the game.

AGENT TRAINING PROCESS

- The ML-Agents toolkit allows for Unity games to serve as an environment for reinforcement learning with ML algorithms implement in PyTorch.
- The agent learns about its environment through a vector of observations about ball position, velocity, and paddle position.

```
/// <summary>
/// Collect observations for ML model to process
/// </summary>
/// <param name="sensor">The vector sensor output</param>
0 references
public override void CollectObservations(VectorSensor sensor)
{
    // Observations of ball x and y position
    sensor.AddObservation(ballBehavior.transform.position.x);
    sensor.AddObservation(ballBehavior.transform.position.y);

    // Observations of ball x and y velocity
    sensor.AddObservation(ballRd.velocity.x);
    sensor.AddObservation(ballRd.velocity.y);

    // Paddle observations for x position of left and right edge
    paddleCollider = this.GetComponent<PolygonCollider2D>();
    colliderWidth = paddleCollider.bounds.size.x;
    sensor.AddObservation(this.transform.position.x + colliderWidth / 2);
    sensor.AddObservation(this.transform.position.x - colliderWidth / 2);
}
```

ML-Agent CollectObservations Method

- The agent interacts with its environment through actions to move the paddle and is rewarded when completing specific game relevant events such as destroying bricks.
- The configuration hyperparameters determine the characteristics of the training model, such as how many units per neural network layer, and how many layers.
- During training, the weights of the neural network are iteratively updated until a maximum reward is achieved. The result is a neural network model capable of playing the game on par with a skilled human.