ESC101: Introduction to Computing **Jan-Apr 2019**

Project Report: Pong − With a Dash of AI

Ashwin Shenai
180156
ashwins@iitk.ac.in

Project Mentor: Madhukant
Course Instructor: Swaprava Nath

Abstract

This project aims to develop a python-based application of the game Pong from scratch.

1 Introduction to the Problem

Pong is one of the earliest arcade video games. It is a table-tennis inspired game featuring simple two-dimensional graphics. In it, the player controls the paddle by moving it vertically across the left or the right side of the screen. The objective is to reach 21 points before the opponent; each player earns points when the opponent fails to return the ball.

The aim of this project is to create a python-based application for Pong using the **Pygame** library. It also involves creating a few AI models which the user can play against.

2 Project Goals

The project was executed with the following goals in mind:

1. Creation of a GUI so that the user can select options and navigate through the application.
2. Development of the 2 player interface, that allows 2 players to play the game using the keyboard.
3. Modifying the 2 player interface to build a player vs AI interface.
4. Creating AI models for the player vs AI interface using simple algorithms.
5. Creating more AI models based on Reinforcement Learning [1]
    - Selecting the parameters and policies for the models.
    - Creating an interface for training the models.
    - Integrating the models with the player vs AI interface.
6. Training the RL-based models to improve their accuracy.

[1] henceforth abbreviated as RL.

3 The Implementation

3.1 GUI

The GUI for the application was written using **Thorpy**. Thorpy provides key press reactions, clickable buttons and text objects that facilitate the smooth working of a basic GUI. The events are handled by an event queue which loads the elements from the application screen and regularly checks them for events, and also executes the event callbacks.

3.2 Game Interface

The game interface for the application was built using **Pygame**. The player pads and balls are classes derived from the base *pygame.sprite.Sprite* class. In the 2 player game, the speed of the ball increases with successive collisions so as to make the game more challenging. When a point is scored, the ball automatically resets to a random direction and location in the central portion of the game area.

The game runs in a control loop that breaks when the score condition is achieved. The keyboard events are handled using an event loop nested inside the control loop. The keyboard events are called from the pygame event queue using the *pygame.event* module. A refresh rate of 100 ms is set for the keys so that the pads move uniformly when a key is kept pressed for a long time.

The ball's collisions in the game area are detected using the *pygame.sprite.collide_rect* function. These collisions are handled by calling the ball's update function in every iteration of the control loop, which checks for collisions and updates the scores and object positions. After the update function is called the objects are blitted onto the background and the display is updated by calling the *pygame.display.flip* function.

To create the player vs AI interface, a new AI class is derived from the Pad class. The class constructor requires two arguments, these specify which model to load and the side on which to load the pad. The event queue for the second player controlled pad is then replaced by the AI class' *predict* function, whose output is then used to move the pad. The **importlib** module is used to load the model using the specified argument.

In the trainer interface, one of the pads is an AI pad running the model that is to be trained. The other pad is either a player-controlled pad or another AI pad running an algorithm-based model depending on the user's choice.

3.3 AI Models

Each model has a *get_prediction* function, that is called by the AI class' *predict* function. This function outputs either -1, 0 or 1, which make the pad start moving up, stop moving, or start moving down respectively.

3.3.1 Algorithm Based

The most basic model implemented in this project is the RANDOM model. This model uses the **random** module to get a number between 1 and 1000 and returns the number modulo 3.

Another model uses the ball's position data and the pad's position data, both continuously written to .txt files during runtime, to just follow the ball. The accuracy of this model can be varied by changing the allowed difference between the ball and pad's positions.

To increase the variety of models available, another model was implemented by combining the above two. The random module is used to randomly flip the output in order to decrease the model's accuracy.

3.3.2 Reinforcement learning Based

Reinforcement learning involves training the AI by allowing to take actions and then reward it positively if it takes the right action, and penalize it if it takes the wrong action. The model consists of a neural network that takes in the game screen as input. The network has one fully connected hidden layer consisting of 200 units. The connections are stored in the form of a weight matrix, passing the input layer to the hidden layer is equivalent to multiplying it with the weight matrix. After the hidden layer, another weight matrix is multiplied to the output of the first layer in order to obtain one single number. The network is trained i.e. the weight matrices are calculated in such a way that this number represents the log probability of going upwards. By taking the sigmoid of this number, we get the probability of moving upwards, and if it is less than a certain value, the pad goes down, otherwise it goes up. In this project the Policy Gradients algorithm has been implemented. It involves the following steps:

1. Start with randomized weight matrices. The game screen is processed using **pillow** and input into the network, and the probability of going up is calculated. This leads to execution of random actions.

2. Based on the reward policy, each action gives a +1 or -1 reward depending on whether the action taken was correct. The reward and action are stacked over the whole game (episode).

3. After one episode is over, the discounted reward is calculated. The discounted reward adds a discount factor, to each reward so that actions taken long ago do not greatly influence current actions.

4. Using the discounted rewards, gradients for the weight matrices are calculated. These are then fed into the optimizer, which moves in the direction of the positive gradient. Since wrong actions give negative rewards, the weights are optimized so that those actions are avoided.

5. The above process is repeated during training and the weights are optimized till the model achieves sufficient accuracy.

Using the above algorithm, two models were created with different reward policies.
One model was written using **numpy**, with the following reward policy:

+1: the AI pad's score increases

-1: the opponent pad's score increases

0: neither player's score changes

The other model was written using **tensorflow**, with the following reward policy:

+1: the ball is near the right edge of the game area and the pad successfully hits it

-1: the ball is near the right edge of the game area but the pad does not hit it

0: in all other cases

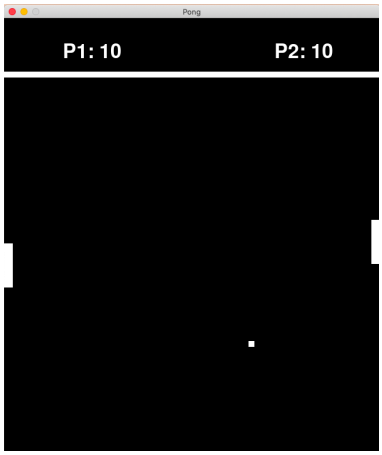

Figure 1: Game Interface

Figure 2: Main Screen

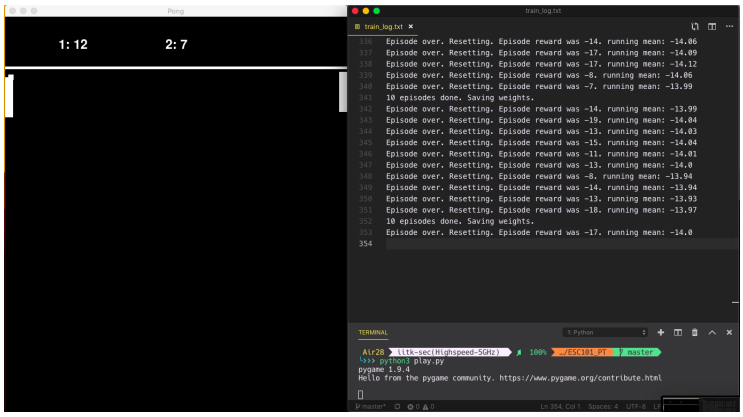

Figure 3: Model Training

4 Results

The application runs as expected. In the case of the RL-based AI models, the numpy model does not perform well. This is due to a faulty reward policy, as the pad's score can increase due to the opponent's mistake, so it does not reinforce successfully hitting the ball which is our desired objective. With sufficient training and proper tuning of parameter, the RL models achieve near perfect accuracy.

5 Summary

The objective of this project was to create a python-based Pong application. To achieve this, various Python based libraries were used. Most part of the application was built using Pygame and Thorpy. A 2 player interface was implemented. This interface was then tweaked to create a Player vs AI interface. Few models were created for the AI pad. In one model a simple ball following algorithm was implemented and two models were based on Reinforcement Learning. These models were trained by the policy gradients using numpy and tensorflow with different reward policies. The code for the project is hosted at this link.