COMP 4106 - Final Report

Students:

Joshua Harris - 101091864

Margaret Venes - 101069737

Michael Ting - 101068936

April 12, 2021

Introduction of The Topic

Our final project for COMP4106 is a pong, goal-based agent. It uses a modified environment of one of the original video games, Pong. Pong was originally a two-player game where a ball bounces off the top and bottom of the screen, as well as the player's paddles. As the ball bounces, it continually gets faster until one player misses the ball and the other player scores. This goes on until the win score is reached where a winner is declared, and a new game can be started. There are two main modifications made to the game, the first is there can be more than one ball active at the start of the game, and the game can be played Player vs. Player, Player vs. AI, or AI vs. AI. Another important aspect to consider is we recognize that we mentioned using a finite state machine and adversarial search as our main methods for artificial intelligence but decided to use a goal-based agent instead as it suited the problem better upon further inspection. In relation to the different ways the game can be played, there are two AI types; a good AI which has a much more robust goal function examining multiple balls at once, and a bad AI which prioritizes 1 ball based on time.

Motivation of The Topic

When exploring projects, we came to a consensus that games are a key part of our childhood and offer infinite complexity for AI. We chose a novel idea to take the simplest of games ever made, Pong, and add an AI element to it. After exploring a few avenues, we landed on a design change that could add infinite complexity to the game matching the capabilities of AI today.

This resulted in the easy decision to make an AI based on one of the earliest widespread arcade games ever made, Pong. This allows us to have fun playing pong as well as incorporate fundamentals of Artificial Intelligence as well.

Description of Method(s) from Artificial Intelligence Used

Original project proposal

We originally proposed developing an artificial intelligence agent primarily through the use of a finite state machine and adversarial search. We proposed an agent that would have had the goal to return the ball at 2 distinct areas on the opponent's side. The complexity steamed from the skill levels implemented. As the skill increases, the bot would better optimize its aim targeting areas that would maximize its likelihood of scoring. But, we decided to implement a "Good AI" or a "Bad AI". Also, we changed our AI method to a goal-based agent as we ultimately had one goal: minimizing the other players score by allowing the least amount of balls in.

As the project progressed, we changed the complexity of the game from aiming at the optimal areas to score to minimizing the number of balls scored on the AI's side. We increased the agent's difficulty by adjusting how many balls ahead the agent would consider when deciding which balls to move the paddle to and prevent the opponent from scoring.

Current method

Goal based agent

We implemented a goal-based agent for our project. We ended up utilizing this because we decided that our AI bots will have a goal of not letting the balls move past their respective paddles. It selects actions that will maximize our goal state. We built a goal-based agent because

it best fits our new game dynamics of having multiple balls. The agent's primary goal is to deflect as many balls as possible. Also, it will have the compound effect of maximizing its points gained. Our agent maximizes its goal state when using the function movePaddle(self,balls).

Bad AI

Our bad AI implementation only checks for the position of the top ball in the given to it. The priority is a simple calculation of the approximate time the ball would score, the sooner the ball scores, the higher the priority. If 2 balls have the same arrival time, it then prioritizes their speed, followed by ball ID. It does not take into account other balls that are in the list and does not choose which ball is the most reachable and optimal for success.

```
def prioList(side, ballsPlaying):  # priority list for balls
        priotemp=[]
        for pongballs in ballsPlaying:
            if pongballs.arrivalT[1] == side:
                heapq.heappush(priotemp,(pongballs.arrivalT[0],
                               pongballs.ball_vel[0], pongballs.id, pongballs))

        return priotemp
```

Figure 1

Good AI

On the other hand, our good AI implementation looks at the next 3 balls that are going to arrive based on time and calculates which ball(s) in our list are the most reachable by distance for our paddle. It uses the canGetToBall(self, bally, startPos) function to check if the ball in our list is reachable from the paddle.

There are three main cases in which the AI looks for. First of all, if the first ball is reachable, then that is our starting target position. Second, if there are two balls, and the first is not reachable, but the second is then the second ball is our target. Lastly, the last step is the most complicated which is for three or more balls. If we can get to the second balls position, from the

first ball's end position in time, then we continue aiming for the first ball position; if we cannot, we check to see if the second and third ball can be blocked if we skip out on the first. If this is the case, we set the second ball as the target and save two balls rather than one. After the target is determined, the paddles move toward the target and re-updates the target every frame.

```
if (len(orderedBalls) > 1 and not reachBall1):
   reachBall2 = self.canGetToBall(orderedBalls[1], self.y)
   if (reachBall2):
       target = orderedBalls[1][3].final_pos[1]

if (len(orderedBalls) > 2 and reachBall1):
    reachBall2 = self.canGetToBall(orderedBalls[1],
                                   orderedBalls[0][3].final_pos[1])

    if (not reachBall2):
        reachBall2 = self.canGetToBall(orderedBalls[1], self.y)
        if (reachBall2):
            reachBall3 = self.canGetToBall(orderedBalls[2],
                                           orderedBalls[1][3].final_pos[1])
            if (reachBall3):
                target = orderedBalls[1][3].final_pos[1]
```

Figure 2

The Result(s) or Outcome(s) Achieved Both Positive and Negative

Positive

Overall, the good AI performs very well under most situations. Against a human player, none of our group members were able to defeat the good AI. However, when faced with the good AI against the bad AI agent, the good AI will always win over time. Moreover, it might not win immediately due to the random RNG nature of the multi-ball spawning; however, the good AI's ability to look 3 balls ahead on average makes it perform 80% better than the bad AI.

Negative

Due to time constraints such as balancing other academic requirements, it is not a perfect AI and could still be improved. The main areas are considering more than three balls and having robust strategies cases like these, more solutions are discussed in the direction for future work. The other negative of the good AI is the processing time it takes to compute. Due to running more loop iterations, the good AI is significantly less efficient than the bad AI, although this was never an issue on our tests, running on a slower machine, or many more balls might make the calculations too slow to run on common hardware.

Discussion of the implications of the work

The implementations for the project are mainly entertainment purposes, however there are definitely real-life implementations that could be possible. A good example is a trash collector robot, if this were in a river, our robot would optimally catch as much trash from flowing down the river as possible (analogous to our AI stopping the pong balls from scoring).

Directions for future work

Future work would be dedicated towards optimizing the algorithm, there aren't any performance issues at the moment. However, trying to do more iterations with the method currently in use would create issues. So, the next step would be modifying the algorithm. At the moment, all the potential options such as going for a second ball if the first is unreachable or going for a second and third ball are hard coded. We need to optimize this where it recursively, or sequentially searches without hard coding the search values and assigning priority based on

these results. This would help in the case where giving up the first two balls, could lead to getting the next three for example.

Another case would be to integrate a neural network that would use paddle movements, ball direction with velocity, and opponents paddle position as inputs to better choose which balls to go for. Currently, the agent does not predict if the opponent is in a position to hit a ball back. In slow speeds it doesn't matter but as the balls increase in speed, the limited movement velocity of the paddle can make it impossible to return fast balls aimed in positions far from the paddles current position if it does not predict it will be returned by the opponent.

User Manual

The application can be run using Python3 with no additional arguments. The application is simple to run, by default it runs on a new AI vs. old AI basis. However, this can be changed to be Player vs. AI (or player vs player if you really desired, this doesn't show the project however). The change is done through code only for the time being, the line changes needed are on line 256 and 257, with paddle one being the left, and paddle two being the right. "badAI" represents the old AI, while "goodAI" represents the new AI, players can be playable with 'Player' instead. Additionally, all import arguments must be pip installed and available for use, this includes pygame, and heapq.

```
paddles.append(Paddle(1, GREEN, round(HALF_PAD_WIDTH - 1),
               round(SCREEN_HEIGHT/2), "goodAI"))   #Paddle 1
paddles.append(Paddle(2, RED, round(SCREEN_WIDTH + 1 - HALF_PAD_WIDTH),
round(SCREEN_HEIGHT/2), "badAI"))    #Paddle 2
```

Figure 3

Statement of Contributions

Joshua Harris: AI helper functions such as approximate time calculations and estimates, first iteration of AI's decision making (Bad AI), report editing, and implementation of multiple balls.

Margaret Venes: Code cleanup and revamp for the final project, First implementation of paddle AI's paddle movement, and most of the report.

Michael Ting: Final iteration of AI complex decision making and implementation (Good AI), optimizing and adjusting many of the parameters and AI settings, report editing.