Autonomous Tennis

AI tennis game

Diogo Cabral
Artificial Intelligence
Instituto Superior Técnico
Oeiras Lisboa Portugal
diogocabral2000@tecnico.ulisboa.pt

Rafael Candeias
Artificial Intelligence
Instituto Superior Técnico
Oeiras Lisboa Portugal
rafaelmcandeias@tecnico.ulisboa.pt

Vasco Piussa
Artificial Intelligence
Instituto Superior Técnico
Oeiras Lisboa Portugal
vascopiussa6@tecnico.ulisboa.pt

INTRODUCTION

In this paper, we seek to find which playing strategy is the best to follow in a game of tennis.

To do so, we developed and tested four agents, each with a different strategy, played them against each other several times, and discovered which performs the best according to our evaluation metrics.

We came up with this idea because it's a topic that hasn't been deeply explored in artificial intelligence. Moreover, one of our colleagues plays tennis.

ENVIRONMENT

The game was developed using the pygame engine, so the environment consists of a screen with a tennis court and some extra space for the players and ball to move in. The screen is oriented from its Top-Left corner, i.e. it's position is (0,0). As the objects go right, their x grows higher, and as they go down, their y increases. Even though the game is in two dimensions, we still computed a third vector z, to simulate the ball touching the ground or the net. Only the ball is aware of the z axis.

The agents can be **top players** or **bottom players**, with each being restricted to its corresponding half of the court. The ball is the only one that can move on both halves. Both players have **speed,** how fast they can move on the court, **force**, how fast does the ball travel when they hit it, **position**, their location on the environment, **name**, for identification purposes, **energy**, which indicates the initial amount of energy the agent has, and **stamina**, to know the current amount of energy left. The ball object has fewer attributes. It also contains a **position** and **speed** attribute, along with **z**, to know its current height, and **ground**, to tell how many times the ball has touched the floor. Both players and ball **speed** and **position** attributes are splitted into vectors for each axis they recognize. All agents have a speedx and speedy attribute to tell how much they move each step of the computation for the given axis. Along with this, the ball has a speedx, speedy and speedz to inform how much it is moving at each time frame in the three axis. Additionally, we included **gravity** and **air resistance** forces that are constantly reducing speedz, speedx and speedy. Gravity only affects speedz and air resistance the other two speeds.

1 Action and observation space

Both agents have the same **action space**: (*Move Up*, M*ove Down*, *Move Left*, *Move Right*, *Stay*, *Hit Straight*, *Hit Left*, *Hit Right* and *Serve*), which is discrete. They also have the same **observation space**, where they are capable of observing the ball's position, their own position and the opponent's position. Since the ball and agent's position is in a continuous space, the observations are infinite.

2 Environment properties

2.1 Accessibility

Concerning the environment's characteristics, one is **accessible** when all its agents can obtain complete, accurate and up to date data about its state. In our case, this is true, since all agents can access the other player's position and ball position precisely and in real time by reading the object's position attribute.

2.2 Determinism

Regarding determinism, an environment is deterministic if all actions have a guaranteed effect. However, even though moving actions are certain to behave always the same way (An agent choosing to move left undoubtedly will reduce its position.x attribute given its speedx), hitting the ball is not. Hitting the tennis ball changes its three different speed vectors, speedx, speedy and speedz. The speed vector for axis x is obtained with a uniform distribution between an interval of values. This interval is determined by the action the agent intends to use and where on the field it is located. Subsequently, we give the speedy of the ball 70% of the agent's chosen force minus the speedx from the uniform distribution. The last vector, speedz, would be the remaining value of the chosen force. For example, the same agent, located in the exact same position and deciding the action left could get a speedx value of -0.5 once and -0.75 other time. Therefore, it is a **non-deterministic** environment.

2.3 Static or dynamic

The environment is considered to be **static** when nothing is changing while the agents are deliberating. Our code is typed in as a sequence of steps. First the bottom agent acts, following the top agent, and lastly we update the ball's position. So, nothing is happening while the agents are making a decision.

2.4 Discrete or continuous

Moreover, the environment is **continuous**, since the agents do not have a finite observation space, as it was explained above.

2.5 Episodic or sequential

Lastly, an environment is characterized as episodic when it can be divided into a series of episodes independent of each other. Our environment state depends on how the agents played previously, since it repositions and starts a new round after every point. When a point is scored, we start a new round repositioning the agents into starting points and the agent that did not serve the previous round will now serve. However, if no point was scored, the round continues. Despite this, the state of the environment in the current episode does not require information from the previous one, since after an episode finishes (a match finishes), both players have their stamina restored and return to their starting positions. So it is considered to be **episodic**.

METHODOLOGY

1 Agents

1.1 Agents type

As expressed before, we came up with four different agents that differ in their strategies, which we called modes. However, despite their divergences, all these agents compute their decisions based on real time observations, such as the opponent's position, its position and the ball's position and speed. Therefore, our smartest agents have more intelligent reactions, but reactions still. They do not have an internal state, and compute decisions entirely based on the present with no reference to their history. Thus, they are all **purely reactive** agents. Moreover, all agents follow the same algorithm for striking the ball. Whenever they do so, they compute a speedx bearing in mind their position on the court. I.e. all agents, when playing on the bottom side of the field and on the right side of the court, if they decide the action '*Hit left*', they will come up with a speedx value from the uniform distribution between the intervals -1.75 and -2.

1.2 Random agent

The four agent types we created are called random, beginner, expert and pro. The **random** agent is aimless and chooses blindly all its actions. At every time step, if it is its turn to play and touches the ball, it arbitrarily chooses 'Hit left', *Hit right* or *Hit straight*, but if it is not it's turn to play, or it did not hit the ball, then it moves to a random position. All randomness associated with its decisions are computed with the uniform distribution available in the numpy python library. Because it is such a simple strategy, we used it as a baseline for evaluating the other three modes.

1.3 Beginner agent

Following it, we developed the **beginner** agent. It is much smarter than the random agent, since it tries to predict which x value the ball will have when it reaches the agent's current y value. However, it entirely focuses on the ball's x and never computes the actions *Move up* nor *Move Down*. We perform this estimation by calculating how much time it would take for the ball current y value to reach the agent's y value, given its speedy vector and y position. Subsequently, we use this time value and the ball's speedx in a common physics movement formula to predict what x the ball will have. These are the equations used:

$$Y_{agent} = Y_{ball\ curr} + (speedy_{ball\ curr} \times t) + (0.5 \times air \times t^2)$$

$$x_{final} = x_{ball\ curr} + (speedx_{ball\ curr} \times t) + (0.5 \times air \times t^2)$$

Despite this, we do not expect it to have the best performances, since slow balls might hit the ground twice before reaching the agent's y value resulting in a point loss. Also, choosing randomly a direction to send the ball is unintelligent.

In a game of tennis, the ball's first bounce must be on the opposite side of the player that hitted it, but always inside of the court lines. After a few tests, we concluded that the beginner agent lost several points due to hitting the ball when it would clearly fall outside of the field limits. This situation revealed to be a dull decision as the agent loses a point that could easily be won simply by doing nothing when the ball has not touched the ground and is outside of the field limits. Additionally, as the game proceeds, the agents lose their stamina, which affects the force they choose to strike the ball. So, by the end of the match, most balls hit the net or hit the ground twice before reaching the agent's y value.

1.4 Expert agent

Thus, for our third agent, **expert**, we decided to implement a few more conditions to avoid these simple but punishable mistakes. Firstly, we removed the x value prediction that the beginner agent had for a better divergency of testing, and also since it is heavier to compute than having the agent move towards the x and y ball value. At every step, the

agent sees which of its position value is further away from the ball's position with an absolute function. If the agent's position x value is further away from the ball's x value than its position y value is from the ball's y value, then it will perform the actions *Move right* or *Move left*. If not, it computes *Move up* or *Move down*. Secondly, to address the poor decisions referenced above, the agent does not move when the ball is outside of the court and its ground attribute is equal to 0. This way we save stamina. Moreover, this agent also decides where to send the ball by observing the opponent's position on the court and its own position. If the other agent is on the opposite side of the field of the current agent, then it would decide the action *Hit straight*. Otherwise, it computes the action contrary to the opponent's location. As a means to save stamina, it also does not move whenever it isn't its turn to hit the ball.

Despite these implementations, we theorized that an agent not doing anything to save stamina while it isn't its turn to play can lead to a point loss due to bad position. As we performed tests, we verified that most points were lost since the agent ended up in the field limits, making the opponent hit the ball to the opposite side, where the agent could not reach it in time. Additionally, some points were lost as a result of using the same force to hit the ball independently of the agent's y position value. I.e. as the agent hits the ball closer to the net, the probability of it landing out of bounds increases.

1.5 Pro agent

Therefore, we came up with our fourth and last agent, **pro**. Whenever it isn't its turn to play, the agent moves to the middle of the field for a better chance at reaching incoming responses. This is a simple but effective algorithm and was implemented by observing where the player is, calculating where the middle x of the field is, and moving towards that position. It first centers its x position value, and then it's y position value, since an agent's side of the field is larger on the x value than on the y value. Moreover, to implement a force reduction system whenever the agent is near the net, it is constantly checking if it is 150 lesser or above the net y position. With this information and its own position, it can precisely tell if it is in the zone closer to the net, or not. If it is, it reduces the speedz that it had chosen to hit the ball by half.

The attributes used for the agents had to respect an interval of values. Energy must be between]0, 1], force between]4, 7] and speed between]2, 7]. The random agent was created with a speed 7, force 5.5 and 0.8 energy and all other agents have a speed of 3, force of 5.5 and an energy of 0.7. We came up with these similar values, since we focused more on testing the different algorithms implemented. However, we had to give a smaller speed to beginner, expert and pro agent, as they move towards one or both ball's position values. If they had bigger speeds than 4, they would reach the net or the ball's x value before the ball had passed the middle of the field, which is completely unrealistic. Consequently, having such a big energy value when their speeds are much slower is unappropriate, since they would never feel exhausted.

We also add the ability for the pro agent to be able to adjust its force depending on its position on the field to reduce the number of missed points caused by too much strength used.

2 Evaluation

2.1 The games

Before diving into the evaluation metrics, it is preferable to explain how a match proceeds. There is a top player and a bottom player. The first one is located on the top half of the field, whereas the second one on the bottom half. The players do not switch halves during the match, nor do they regain energy. Also, the score and rule system followed is exactly like a real game of tennis. However, a match ends when a player reaches a score of 15 and each point adds plus one to their score.

To evaluate our agents, we decided to play them all against each other as bottom and top players. Therefore, our main file began to have two loops that iterated through all existing agents. The outer loop chose an agent to play as a top player and the inner loop iterated through all remaining agents to play on the bottom half. So, with all four agents competing against the other three, it took $4 \times 3 = 12$ games. However, this was insufficient data to have solid arguments for evaluation. Thus we created a third outer loop that does 24 iterations. All in all, our code when ran computes $24 \times 4 \times 3 = 288$ games.

2.2 Metrics

- **Number of wins**
- **Average score per game**

To evaluate all agents fairly, we keep track of their scores for each game. With this information, when a match ends, we can tell which player won and how many points did it score. Later, for each agent, we plot graphs with their number of wins and their average score in each game. We took this approach to emphasize agents that lost with high scoring points. This way, when studying the plots, if the agent that won most games has a similar amount of scoring average than another, then it isn't preposterous to state that they are not so different.

Every agent has a .csv file where we put information related to every game it participated in. These files are created to draw plots that lead to visual conclusions. By the end of the 384 games, we place the information that we tracked in the .csv file of the corresponding agent. It holds a sequence of 192 entries. Each entry represents a game in two numbers,

a win_flag and a score. The first number can be 1, the agent won, or -1, the agent lost. The score is simply the number of points the agent scored in that match.

3 Results

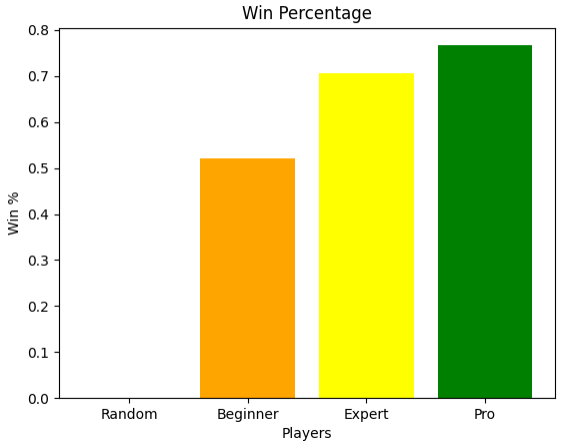

Image 1 - Win percentage graph

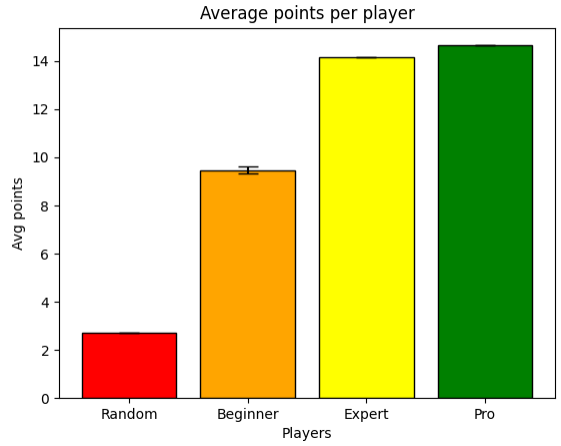

Image 2 - Average points per player with confidence interval graph

3.1 Results analysis

As shown in both plots before, our theories were accurate. The random agent revealed to be the worst, the beginner was extremely better and the expert was astronomically different from the baseline. Between the expert and pro agent, we can detect some difference, but not too much.

Regarding the win percentage plot, the random agent did not get a single win, the beginner had a win percentage between 50% - 55%, the expert had a 70% win percentage and the pro agent had one between 75% - 80%. These are understandable, but unexpected. In fact, the random agent was extremely unintelligent as it would not do anything rationaly, making it plausible to lose every single game. However, the proximity from the expert to the pro agent came also as a surprise, since the pro agent had a repositioning and force reduction system. One to avoid missing the ball, and another to not send the ball outside of the field limits. Therefore, we can conclude that these optimizations aren't as important as we thought them to be. After some deliberation, we came to the following theory: since the players do not compute the precise amount of force they need to use to send the ball inside the field limits, sometimes they might send it incorrectly. Therefore, hitting the ball might harm the agent's score. However, without them, the pro agent wouldn't have won.

Looking at Image 2, we can see similar results to Image 1, Random was considerably worse than the other models having an average of 3 points per game, the Beginner was quite the improvment with around 9 points per game and then the Expert and Pro really close with 14 and around 14.5 points respectively. Concerning the confidence intervals, unlike the others, the Beginner showed an interval difference bigger than 0, this higher variability would be caused by having some high point games and then some low point games, showing its still an unpredictable model unlike the Expert or Pro.

FUTURE WORK

In the future we would like to create more agents with the same algorithm(beginner,expert,...) but with different characteristics. Doing this we would be able to see which are the agents characteristics that determine the most how well the player plays tennis.

We think it would be a good idea to add an option for the tennis games to be played in doubles with teams of two so that we can add teamwork to the agents and what differences we can observe from the regular single matches. Do the best characteristics of the agent and the most effective algorithm change when we switch to doubles matches.

We want to solve a problem that sometimes happens in which the players repeatedly strike the ball close to the net

between each other. This makes the agents lose stamina quicker than expected.

If we had more time, we would have liked to implement Reinforcement learning models into our project, such as Q-learning and SARSA. It was our initial approach but we chose against it because it would take too much time to get the right hyperparameters.

REFERENCES

Link to the original pygame used as a base to develop this project:https://github.com/roberto257/pygame-tennis