Learning a Squash RL Agent: Yasse-RL Halaby

Authors. Arin Mukherjee (arinm@princeton.edu), Saumya Malik (saumyam@princeton.edu)

1 Introduction and Motivation

The way sports are played is continuously evolving. Sometimes, new surprisingly effective strategies emerge and revolutionize the sport. In the NBA, teams averaged just a few 3-point attempts per game until the 1990s. The shot was seen as a last resort rather than an integral part of offensive strategy. However, over the next couple decades, its value as a high-reward option and a way to stretch the opposing team's defense began to be recognized and thus it gained popularity across the league. In the sport of high jump, the bar was traditionally cleared face-down. However, Dick Fosbury introduced a technique at the 1968 Olympics where he jumped over the bar back-first. This strategy allowed for higher clearances and soon became the standard technique in high jump. Sometimes, athletes/teams discover unorthodox strategies that are used more sparingly to stun opponents, such as Roger Federer's SABR technique in tennis and trick plays like the Philly Special in American football.

Figure 1: Dick Fosbury (top-left), Steph Curry (top-right), Roger Federer (bottom-left), Nick Foles (bottom-right)

Athletes are trained in line with contemporary strategic standards, leading to inherent bias in the way they learn to play their sport. For this reason, the discovery of these hidden strategies is gradual and infrequent. What if there was a way to learn a sport such that we could learn these hidden optimal strategies without being influenced by current strategic norms? This is the ultimate motivation for this project.

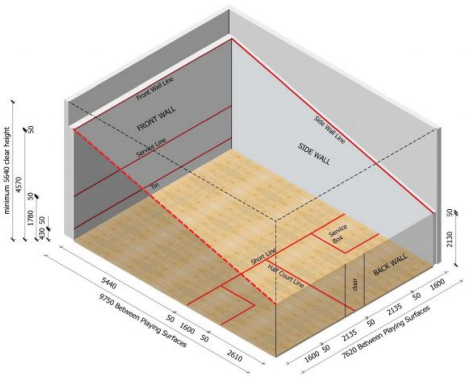

Figure 2: Diagram of squash court

Squash is a fast-paced racquet sport in which two players take turns hitting a small rubber ball against the front wall of a four-walled court. The goal is to hit a shot such that the ball bounces twice on the floor before your opponent is able to retrieve it. There is also a tin and out line on the front wall limiting how low and high you can hit the ball respectively. A typical rally between two right-handed players in the modern game will involve an exchange of deep shots (shots whose first bounce is a long distance from the front wall) along the backhand wall, with each player aiming to keep their shot as close ('tight') to the wall as possible. The idea is that hitting tighter shots limits the range of shots available to your opponent and also increases the probability of your opponent hitting a 'loose' shot. Upon receiving a loose shot, a player will typically hit lower shots that bounce on the floor quite close to the front wall

('going short'), thereby forcing the opponent to expend energy and put themselves in a bad position. This strategy (1. hit straight and deep, 2. wait for the loose ball, 3. go short) is taught universally to junior squash players around the world. Some basic principles implicit in this strategy include hitting the ball away from your opponent and staying in a central position on the court when it is your opponent's turn to hit a shot. In this project, we focus on setting up a reinforcement learning (RL) environment for squash and training a computer agent to learn such basic principles, setting the stage for discovery of more creative strategies in future work. We hope to train our own RL agent as capable as Yasser El Halaby ('06), a four time intercollegiate individual champion and alumnus of the Princeton Squash Team.

The state and action spaces in a full RL implementation are quite high-dimensional. A state would encode the dynamics of the ball, the two players, and their racquets as well as the players' energy levels. An action would encode at least one of the player's movements. To physically simulate a player accurately in a virtual environment, we would need to track each of their joints and limbs. Our reward structure would also be quite sparse. An agent would receive positive/negative rewards for winning/losing a match, which is best of 5 games, where each game is first to 11 points. We would need to consider rewards at the level of matches to fully account for strategies such as giving up an earlier game to save energy for a later, more crucial game. Obviously this full implementation is quite intractable. For this reason, in addition to squash being a niche application of RL, this problem has not been approached before to the best of our knowledge.

To make the squash problem approachable with the resources and time available for this project, we simplify the MDP as much as possible and encode the relevant dynamics in a custom OpenAI Gym environment [2]. The two RL algorithms we consider are PPO (Proximal Policy Optimization) and DQN (Deep Q-Learning). The specifics of the custom environment implementation and these algorithms are detailed in Section 3 - Methods.

2 Related Work

This project took inspiration from "Control Strategies for Physically Simulated Characters Performing Two-player Competitive Sports" [3], where an RL-agent is trained to learn boxing/fencing. In this work, the agent takes the form of a humanoid with many degrees of freedom. The characters are actuated by joint torques, allowing for realistic motion dynamics. The trained agents are ultimately able to perform responsive and natural-looking behaviors in these two-player competitive settings. For the scope of this project, these physically simulated characters are too complex, so we instead consider a much simpler version of the squash problem.

We map the 3D court onto a 2D Atari style screen. As shown in Figure 3, our initial setup is quite similar to that of Atari Pong, so we apply many of the same strategies to change our agent. Below we describe a method that has seen RL agents perform well at Atari Pong, detailed in Andrej Karpathy's blog post *Pong from Pixels*.

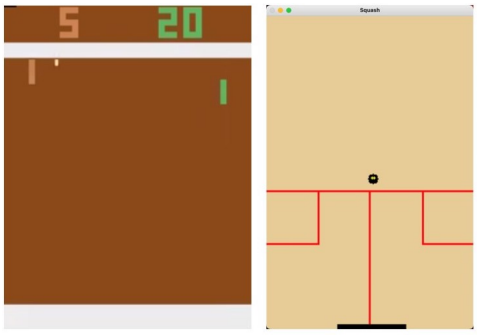

Figure 3: Visualizations of Atari Pong Gym Environment (left), simplest version of custom squash Gym Environment (right)

The agent is trained using a policy network that takes raw pixels as inputs. This method stands in contrast to defining the state as a vector that directly encodes information on ball position, ball velocity, both paddle positions, etc. This raw pixel approach is advantageous for a number of reasons. Firstly, it allows our learning algorithm to operate directly on the Gym environment's native representation. Additionally, working with pixels leaves room for the policy network to identify useful features that may not

have been considered in the manual feature engineering. This approach requires significantly more computational power in comparison so it is possible that in the case of this simple custom squash setting where we can easily manually encode the relevant features, that the pixel approach will require longer training times. However, we stick with this method as it will generalize better to more complex versions of the squash environment.

In the Atari Pong training, difference frames are fed to the policy network by subtracting the current frame from the previous frame. This preprocessing step helps the policy network focus on changes in the game state, thereby capturing a sense of ball velocity, which is crucial for making decisions. The network outputs a probability of moving the paddle UP or DOWN, and actions are sampled from this probability distribution. A policy gradient algorithm is used to adjust the network weights based on the outcomes that result from the sampled actions.

There are some key differences between the Atari Pong setup and our custom squash environment which we highlight in Section 3. Yet the setups are similar enough to justify using the approach detailed above.

In the end, we produced three novel custom environments for squash, including the first environment (to our knowledge) that captures some 3D dynamics

3 Method

In this section, we describe the two-prongs of our approach: implementing a custom squash environment, and applying RL methods (specifically, PPO and DQN) to train an agent to “play squash" in our custom environment.

These two prongs of our approach are related and together formed an iterative development approach in which we repeatedly did the following:

1. Constructed an environment
2. Trained an agent on the environment
3. Evaluated performance
4. Added complexity to environment dynamics
5. Repeat

In the subsections below, we provide an overview of the different versions of our Custom Squash environment (increasing in complexity), along the way highlighting different variations that we considered for the reward function. We then discuss how we implemented RL training. This iterative approach reflects a genuine substantial effort and the culmination of our many thoughtful re-considerations of our reward dynamics, how to capture complexities of the game, and how to go about implementing training. Finally, we include a short subsection at the end describing unsuccessful previous efforts, whose revisions ultimately culminated in our final product.

3.1 Custom Environment

In this subsection, we discuss the construction of our custom environments, describing how we formulate the squash problem as an MDP. We produced three novel custom squash environments, each capturing more complexities of the game. Broadly, we made a single player squash environment a multiplayer squash environment (with a hard-coded opponent paddle), both with two-dimensional ball movement and one-dimensional paddle movement as in Atari Pong, and we also made a multiplayer squash environment that allows for two-dimensional paddle movement anywhere on the court to capture freedom of movement in squash and allows for three-dimensional ball movement to more faithfully capture the notion of a second bounce in squash. (Implementing this last version was initially our stretch goal!) We now describe how we formulated each of these environments as an MDP. For dynamic visual references of each of these environments, we refer the reader to a folder of screen-recordings of our trained agents in each environment.

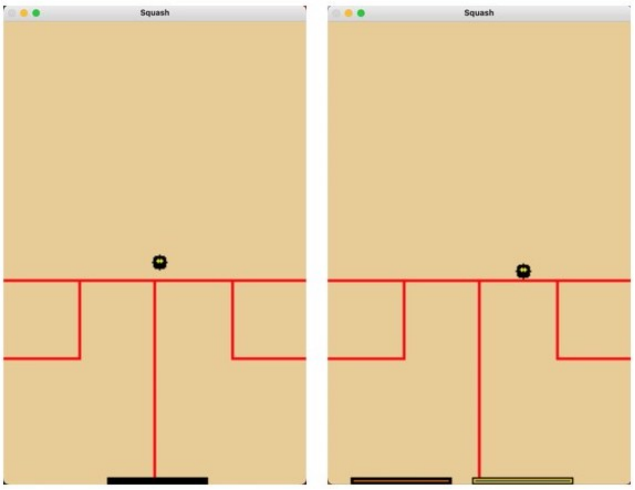

Figure 4: Version 1: Single Player Environment (Left); Version 2: Multi Player V1 Environment (Right)

3.1.1 Version 1: Single Player (2D ball movement, 1D Paddle Movement)

- High-Level Dynamics: There is a single paddle that is constrained to move along the back wall of the court. The ball is restricted to moving in two dimensions (within the plane of the game screen). Upon making contact with a wall or the paddle, the ball is redirected according to the angle of incidence with the plane of contact. Since the ball moves in 2D, we have no notion of height. In squash, a loss is defined as not reaching the ball before it bounces twice. However, as we have no concept of a bounce in this environment, we instead define a loss as the ball making contact with the back wall (without being blocked by the paddle).
- State/Observation Space: Gray scale pixel array of size 320×210. Screen in black. Ball in white. Paddle in white.
- Action Space: STAY, LEFT, RIGHT
- Rewards: -1 for the ball hitting the back wall, +0.5 for the paddle making contact with the ball.

3.1.2 Version 2: Multi Player v1 (2D ball movement, 1D Paddle Movement)

- High-Level Dynamics: We now introduce a second (opponent) paddle along the back wall. This paddle follows a fixed policy: Move the center of the paddle towards the ball's current position. Intuitively, this is a suboptimal policy. We would want to move to a future position of the ball by predicting its trajectory as opposed to always moving towards its current position. However, for the speed of the ball that we specify, this policy is effectively unbeatable, so at best our agent will match its performance. Training against a strong opponent is an intentional choice as we do not want to reward our agent for useless actions that were taken in a rollout in which the opponent was coincidentally too slow to reach the ball. We also introduce the notion of turns. This is not required in Atari Pong as the two paddles are on opposite sides of the screen. A loss is now defined as the ball making contact with the back wall when it is our turn to hit.
- State/Observation Space: Gray scale pixel array of size 320×210. Screen in black. Ball in white. Our paddle in white. Opponent's paddle in gray (to allow the policy net to distinguish between the two paddles). In Multi v1 Diff we feed the difference between two such frames.
- Action Space: STAY, LEFT, RIGHT

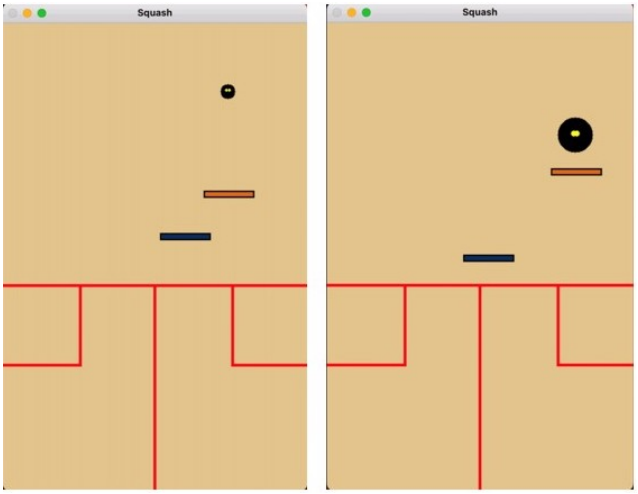

Figure 5: Version 3: Multi Player V2 setup with regular court size. The left image shows a ball close to the floor. The right image shows a ball at the peak of its bounce.

- Rewards: -1 for the ball hitting the back wall on our turn, +1 for the ball hitting the back wall on the opponent's turn, +0.5 for our paddle making contact with the ball on our turn.

3.1.3 Version 3: Multi Player v2 (3D ball movement, 2D Paddle Movement)

- High-Level Dynamics: We now allow the paddles to move in two dimensions. We redefine a loss as the ball bouncing twice. We track the height and z-velocity of the ball and update it according to simple kinematics equations. We encode the height of the ball into the pixel input to our policy net by having the size of the ball on the screen be proportional to the height. A ball close to the ground will be displayed as smaller, while a ball at the peak of its bounce will be displayed as larger. Upon making contact with the paddle, the ball's z-velocity is flipped. Hence, a player is allowed to hit the ball into the floor before it makes contact with the front wall (which is not allowed in squash) if the paddle makes contact with the ball when it is on the rise. Again, the opponent's paddle follows the fixed policy of moving towards the ball, which is still effectively unbeatable. As this environment is more complex than our previous versions, we tweak the court size to increase the likelihood of the event that our agent stumbles into the ball.
- State/Observation Space: Gray scale pixel array of size 160×210 (smaller to make it easier for the paddle to stumble across the ball in exploration now that the paddle can move in two dimensions). Screen in black. Ball in white. Our paddle in white. Opponent's paddle in gray. In Multi v2 Diff we feed the difference between two such frames.
- Action Space: STAY, UP, DOWN, LEFT, RIGHT, UP-LEFT, UP-RIGHT, DOWN-RIGHT, DOWN-LEFT
- Rewards: -1 for the ball bouncing twice after hitting the front wall on our turn, +5 for the ball bouncing twice after hitting the front wall on the opponent's turn, +2 for the paddle making contact with the ball on our turn. We define the reward structure in this way to highly encourage the paddle towards behavior that results in making contact with the ball.

3.2 Applying RL Methods

We applied two RL algorithms we learned in class, PPO and DQN, to each of our five squash environments described above. Before discussing details of training, let us quickly introduce these algorithms:

PPO, or Proximal Policy Optimization, is an advanced policy gradient method. PPO has a clipped objective function that limits the amount a policy can change on an update. This confers a great deal of stability and efficiency to PPO, among its primary advantages. Given how popular PPO is as policy-gradient option for many applications in Deep RL, we decided to apply it to our novel squash application.

DQN, or Deep Q-Learning, is an off-policy algorithm that leverages deep learning to train a neural network to serve as a function approximator for the Q function. DQN typically makes use of Experience Replay, a memory buffer that contributes better data efficiency (through data reuse) and better stability (through including uncorrelated transitions within each batch). DQN was first introduced in 2015 by DeepMind, and it is highly-performant on many Atari games. Indeed, most of the leaderboard on the Atari Pong Benchmark is comprised of implementations of (variants of) DQN. As such, since Atari Pong is the closest existing application to ours, we decided to apply DQN to our squash environments.

3.2.1 Trainer and Model Architecture

Off-the-Shelf Implementation We used off-the-shelf implementations for PPO and DQN training from RLLib [1], a popular RL library. Applying these off-the-shelf implementations required some work figuring out how to make them compatible with our custom environment. In particular, we had to fiddle around with the default model architecture a bit so that it would be compatible with our custom squash environment's observation space, but after this initial work, using off-the-shelf implementations proved very easy for running many training experiments with minor differences at once.

Model Architecture The underlying architecture for our adapted RLLib off-the-shelf trainers for both PPO and DQN consisted of a Vision Network with five convolutional layers and 4 fully connected layers, connected sequentially. The final fully connected layer would have dimension equivalent to that of the environment's action space and would be treated as a Categorical distribution from which actions are sampled.

Training Details We trained each model for 24 hours on a 10GB GPU (MIG partition) on the Della Cluster. Since each version of our environment has a slightly different state space, and the two algorithms have different training protocols, 24 hours amounted to a different number of steps for training run. Nonetheless, we observe that within 24 hours, most of our training configurations reach a peak and then either collapse or level off (as can be seen from the reward curves plotted in Figure 6), so we are optimistic that this was an appropriate training time for our model architectures. We saved intermediate checkpoints every ten steps of training, which proved very helpful for referring to time steps before training collapsed.

3.3 Previous Unsuccessful Efforts

In previous unsuccessful iterations, we tried to implement PPO and DQN ourselves (adapting homework code) before switching to the RLLib [1] implementations.

We also played around considerably with the reward function, at times experimenting with higher magnitude rewards (e.g., -50 for losing, +10 for hitting the ball) and also a reward penalty for being far from the ball (to encourage the paddle to be near the ball and also make rewards less sparse). However, our exploration of these alternate reward functions did not yield strong results, so we settled on the reward function described earlier in this section.

4 Results and Analysis

In this section, we discuss the results from our suite of 2×5 training experiments (2 algorithms over 5 environment variants) described in Section 3 above. We plot the reward

curves from each training run. We supplement these results with qualitative observations about the performance of each trained RL agent, accompanied by links to videos of each of the trained agents in action.

4.1 Training Reward Curves

We let each agent train for 24 hours. We show the reward curves in Figure 6. Recall that the agent is programmed to lose eventually and accrue a reward of -1.0 for losing. Furthermore, the reward for each hit on the ball is +0.5, so we can use these numbers to get a sense of the average number of times our trained agents successfully hit the ball (Number of hits $\approx$ (Mean reward + 1.0)/0.5) (which we later also visually evaluate and confirm).

Looking at the reward curves in Figure 6, we can observe that all of our implementations with 1D paddle movement— all of the PPO and DQN single player and multi player v1 plots— result in a trained agent that hits the ball on average 0-2 times. While not super successful, this indicates some amount of learning, with PPO Multi Player 1 (Diff) in particular even achieving a high positive reward at certain points in training. Furthermore, even qualitatively from the training curves, we can see that DQN training is relatively unstable [4], with its variants often achieving high rewards at some points of training but not consistently maintaining these rewards. In the next section, we look closer at the trained agents and evaluate their performance qualitatively.

4.2 Observing the Trained Agents' Performance

For each of our ten experiments, we evaluated the performance of the trained agent. Since RL training in general can be unstable and collapse, we thought it best to use the reward curves (and our intermediate checkpoints) to help us select the *best* performing checkpoint for each experiment. In Table 1, for each agent, we report the maximum mean reward achieved during training (averaged over 10 rollouts), corresponding to the peak of that agent's reward curve in Figure 6 and evaluate a rollout of that trained agent in its corresponding environment (by sampling actions from its trained policy). We report our qualitative observations on the performance and behavior of the agent, and the reader can follow along in the linked screen recordings of the rollout.

We encourage the reader to check out the videos of our trained Squash RL agents in action! All videos are contained in this linked folder (each video is labeled with the algorithm, version, and checkpoint's step number). Each rollout starts on the opponent's turn.

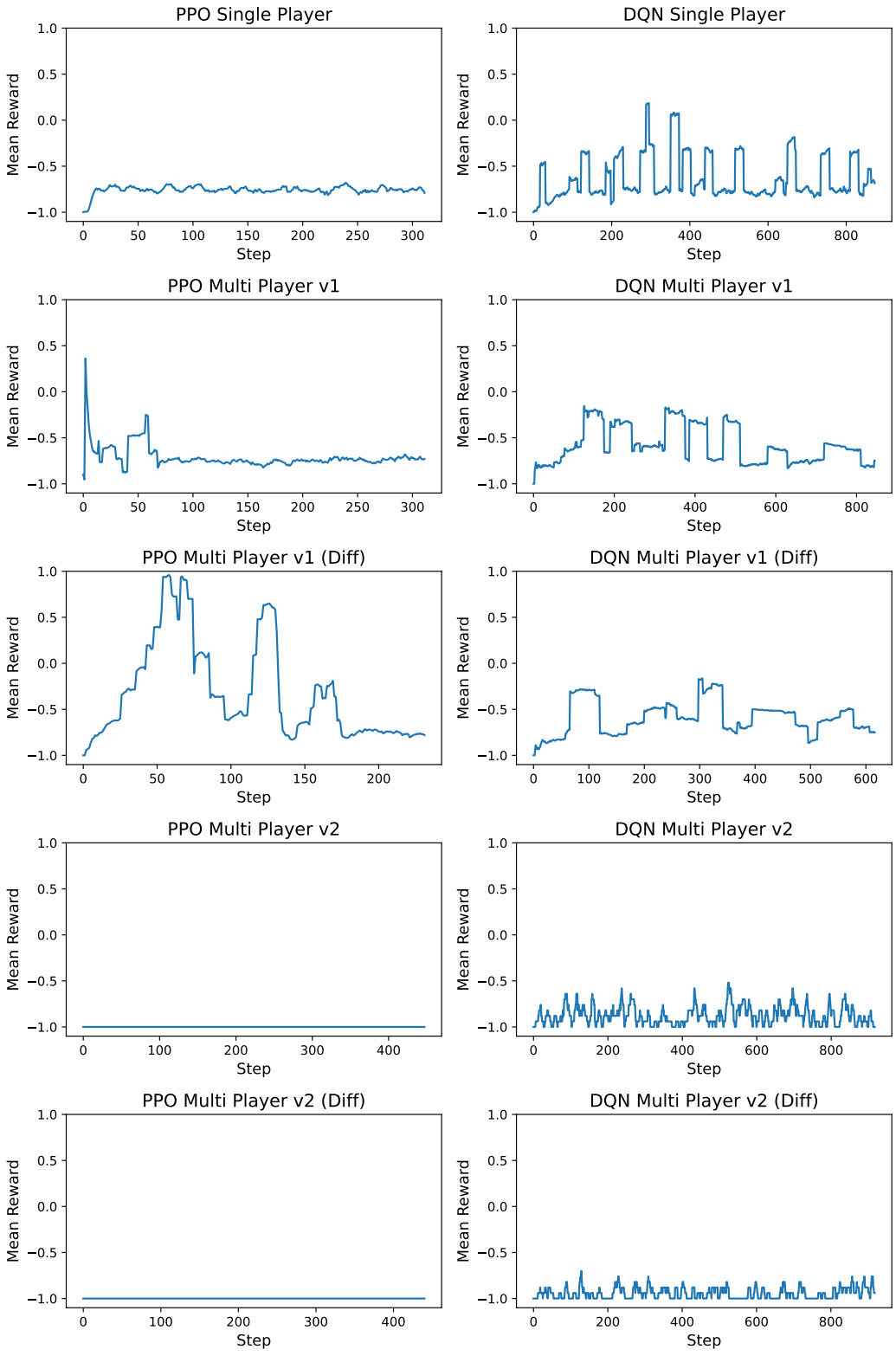

Figure 6: **Training Reward Curves.** The different models and algorithms varied considerably in their rewards encountered during training, with PPO failing to accrue any positive reward in the 2D movement Multi Player v2 setting, but excelling in the 1D movement Multi Player v1 setting. DQN generally achieves decent rewards, but does not always maintain them.

Table 1: Agent Performance in Various Experiments

Experiment	Max Reward Achieved	Observations of Agent's Performance
PPO Single	-0.68	Successfully hits the ball once; the ball bounces off and leftward. Correctly juts a little bit left (0:04) but then stops moving (gets stuck on the right wall) and fails to hit the ball the next time.
DQN Single	+0.185	Usually doesn't hit the ball, but learns to mostly stay central, and so it occasionally will hit the ball.
PPO Multi v1	-0.5	Gets one hit on its turn, but then misses on the next turn.
DQN Multi v1	-0.155	Hits the ball once on its turn, generally stays in center.
PPO Multi v1 (Diff)	**+0.96**	**Hits the ball 3 times (0:14, 0:36, 0:59)! Genuinely looks like a successful, strategic rally. When we miss, we are very close. Furthermore, we strategically don't move across the court when it is the opponent's turn (0:44).**
DQN Multi v1 (Diff)	-0.165	Is able to hit the ball anywhere from 1-4 times, but does particularly well when it gets lucky and the ball always lands relatively in the middle of the screen. If it ever has to leave the middle to chase the ball, it doesn't do so well.
PPO Multi v2	-1.0	Moves randomly, never finds the ball.
DQN Multi v2	-0.52	Correctly juts up when it is its turn to hit the ball and the ball is up, but doesn't move left and right enough to reach the ball. We didn't observe any hits, but it is possible (and evident in the reward output) that some hits were encountered at some point during training.
PPO Multi v2 (Diff)	-1.0	Moves randomly, never finds the ball.
DQN Multi v2 (Diff)	-0.70	Moves up and down erratically, we do not observe it finding the ball.

From Table 1, we can see that while most trained agents are not very good at the game (in particular the agents in the Multi v2 environments with 3D ball movement and 2D paddle movement, as was expected due to the complexity and difficulty in stumbling toward hitting the ball in exploration), even the agents that are pretty bad learn some interesting behaviors. Most of the agents trained in the Single Player or Multi v1 environment are able to occasionally hit the ball, and some (PPO Single, DQN Multi v2) make movements toward the ball even in instances where they miss. Interestingly, a few agents learn an emergent strategy to mostly stay in the middle (DQN Multi v1, DQN Multi v1 Diff), which outperforms learned strategies that get stuck to one wall (PPO Single).

Most impressively, though, is the performance of the agent trained with PPO in the Multi v1 environment, with the difference of frames fed as input (PPO Multi v1 Diff). As we can also see from its reward curve in Figure 6, this agent learned much better than the other agents. Indeed, qualitatively, watching the agent "play" in its environment, we see some impressive behaviors (we highly encourage the reader to check out the video!):

- The agent returns the ball 3 times! (These occur at timesteps 0:14, 0:36, and 0:59 in the video). Being able to return the ball 3 times on its turn is a non-trivial accomplishment for our agent!
- Even when the agent misses (as observed in other rollouts too), it is not far off from the ball. In general, on its turn, the agent always motions in the correct direction toward the ball. This signals a good degree of learning.
- Finally, the agent acts very strategically when it is *not* its turn to hit the ball, signaling its ability to learn from the "turn" information contained in its state as well as a relevant strategy. In particular, when the ball is going to the opposite end of the screen and it is not the agent's turn (0:44), the agent does *not* chase after it (in contrast to the agent chasing after balls on its own turn as discussed above). This is a good strategic move, as generally when the opponent returns the ball, it may return back to the agent's side of the screen, so it is generally (in the real world) strategically beneficial not to run across the court to chase a ball not on your own turn.

In general, we are quite impressed with the performance of the PPO Multi v1 (Diff) agent, and bestow upon it the title of "Yasse-RL Halaby"!

5 Limitations

One large limitation is that we were unable to train an agent that would perform well in our Multi v2 environment, where paddles can move in two-dimensions. This was initially our stretch goal, and we are excited to leave it to future work, but this setting was particularly difficult due to the improbability of stumbling upon contact with the ball in initial exploration (and thus increased sparsity of reward). The problem is also harder— it is practically harder (even for a human) to plan actions to hit the ball before its second bounce. Given more time, we look forward to exploring new ways to train on this environment.

Another general limitation is that models take a lot of time to train, and because of the time-bound nature of this assignment and the several iterations we took to finalize our MDP and training, we did not engage in as much exploration of model architectures or different RL libraries for exploring training. We are confident and excited for future work to build upon the progress we have already made, as well as our released environments, to train even more capable RL squash-playing agents!

Finally, there are of course limitations to our formulation as an MDP, as we highlighted throughout— this is somewhat inherent to formulating a very complicated problem as an MDP. There is room to further explore action spaces that allow for more aim of

6 Conclusion

The contributions of our project are two-fold: (1) the design and implementation of a suite of three novel custom RL environments for squash (including the first known gym environment that captures the 3D nature of a paddle-based game!) and (2) the application of PPO and DQN to our custom environments, culminating in our decently-capable RL squash player, Yasse-RL Halaby. Yasse-RL Halaby is able to return the ball multiple times and demonstrates strategic actions like following the ball when it is its turn to play but not dashing across the court on the opponent's turn.

This project represents a promising first step in applying RL to Squash, and we hope that releasing our custom environments will inspire further research into increasingly capable Squash RL agents! Our code, including all environments and training implementations, can be found in our Github repository.

References

[1] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. (2018). RLlib: Abstractions for distributed reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR.

[2] Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.

[3] Won, J., Gopinath, D., and Hodgins, J. (2021). Control strategies for physically simulated characters performing two-player competitive sports. *ACM Trans. Graph.*, 40(4).

[4] Xu, Y. (2023). Deep reinforcement learning and imitation learning based on vizdoom. In *Proceedings of the 2022 6th International Conference on Electronic Information Technology and Computer Engineering*, EITCE '22, page 1700–1706, New York, NY, USA. Association for Computing Machinery.