Pinbot – Applying Reinforcement Learning to Pinball Machines

Sahil T Chaudhary
Department of Mech. Eng.
Carnegie Mellon University
stchaudh@andrew.cmu.edu

Richa Mohta
Department of Mech. Eng.
Carnegie Mellon University
rmohta@andrew.cmu.edu

Albert Xiao
Robotics Institute
Carnegie Mellon University
anx@andrew.cmu.edu

Luis F. Cuenca
Department of Mech. Eng.
Carnegie Mellon University
lcuencac@andrew.cmu.edu

Ethan Holand
Robotics Institute
Carnegie Mellon University
eholand@andrew.cmu.edu

Abstract: This paper explores the application of reinforcement learning to play the classic arcade game of pinball, considering two settings: simulation and physical hardware. In the simulation setting, the Unity ML-Agents framework is utilized to train an agent to play a virtual version of the game on a computer. Transfer learning is proposed to use the agent's weights from the simulation as pre-training for a physical system equipped with multiple cameras to provide inputs analogous to those in the simulation; however, this process remains a work in progress. The physical system is further designed for fine-tuning by interfacing with its electrical components. This approach aims to develop two comparable agents: one trained to play the simulation and another capable of playing the physical game. The performance of these agents is evaluated against human players, and preliminary insights from this process are presented.

Keywords: CoRL, Robots, Learning, Pinball, PPO

1 Introduction

Reinforcement learning has been successfully applied to games like Go, Pong, and Dota 2. Games serve as ideal learning environments due to their defined boundaries, clear objectives, and opportunities for strategic decision making. Pinball is a classic arcade game in which a player uses two flippers to keep a ball on the playfield while attempting to hit various targets. Pinball is dynamic and requires highly reactive game play and control. The objective is to maximize the score of the game over the course of three balls (turns). Hitting a particular target increases this score by some base amount; completing multiple shots in specific sequences can award far greater points. These rules and layout vary drastically across more than 3000 unique pinball designs produced since the 1930s.

Pinball has recently experienced massive growth in popularity [1, 2]. Capitalizing on the recent resurgence in this classic arcade game, we believe that producing an AI agent to play pinball could generate more buzz around the game, especially for those interested in technology and engineering. There has been minimal prior work conducted in this space, with ample opportunity for further development in new directions.

2 Related Work

2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy gradient method for reinforcement learning [3] that uses a clipped surrogate objective to maintain a balance between policy updates and stability by constraining the change in action probabilities. Its ability to handle continuous state spaces and discrete action spaces makes it a strong choice for tasks like pinball machine control, where it can effectively learn complex game dynamics and optimize precise flipper movements and trajectory planning in real-time scenarios..

2.2 Firepower vs A.I

In 2019, the user `3rdaxis` released a custom virtual pinball that recreates the 1980 ”Firepower” pinball machine, but with an added AI opponent [4]. The AI featured a complex hand-crafted algorithm that flips when the ball enters the detected zones. This system was never deployed to a physical machine.

2.3 AutoPinball - Kennesaw State University

In 2020, four students at Kennesaw State University developed a fully custom physical pinball machine with simplified self-playing software [5]. The system used a hand-crafted policy that activates the flippers when a ball is detected in a specified ”flip zone” above the flippers. No learning was used; in fact, the game did not have a scoring system implemented.

2.4 3DPinballAI - Microsoft Azure

In 2020, Microsoft’s Azure team in Australia trained a reinforcement learning model to control a pinball machine using Unity ML-Agents framework and Proximal Policy Optimization (PPO) [6]. The model was first developed in a simulator [7], then re-trained on a completely different physical machine. Two cameras were used to observe the score and playfield, and a Raspberry Pi was used to control the flippers. The project was intended for display at Microsoft Build 2020 but faced technical issues due to differences in convention lighting compared to the training environment. Public showcases ceased due to the pandemic, and no publications resulted from the work.

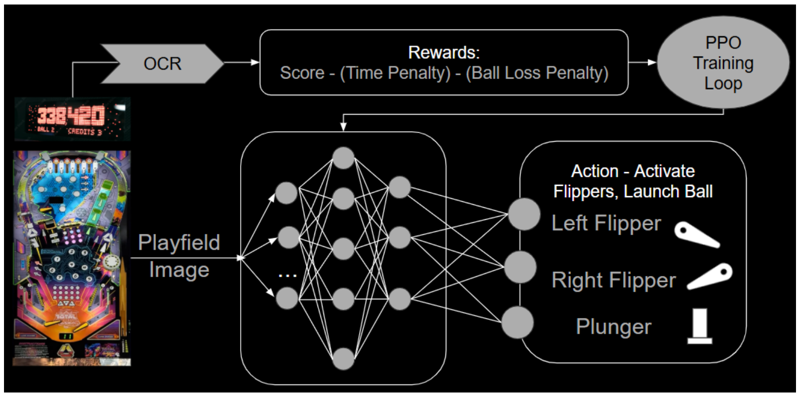
3 Methodology

To expand upon previous works, our team sought to be the first to implement transfer learning for pinball. A simulated version of the machine will first learn the policy, and those weights will be deployed the model onto real hardware. This should substantially accelerate real-world training time, compared to the baseline of four days by Microsoft [6].

Total Nuclear Annihilation (TNA), a pinball machine released in 2017 by Spooky Pinball, was selected as the candidate platform. It features a single-level layout with no ramps, meaning nothing is obscured from an overhead view; it features an autoplunger, allowing the ball to be launched into play via an electrical pulse; and the scores are persistently shown via a set of 7-segment displays, which is critical for the reward signal. Furthermore, a digital recreation of this game was available open-source online.

For the hardware environment, we acquired a physical TNA pinball machine, on loan by local operator PGHPinball. To enable reinforcement learning on the physical system, multiple cameras and electronic components were deployed to stream the playfield to a computer, detect the score and game-over states, and remotely control the flippers and plunger, as seen in Figure 1. This is fed into a PPO model, as seen in figure 2.

Figure 1: Team with the physical system.


OCR
Rewards:
Score - (Time Penalty) - (Ball Loss Penalty)
PPO Training Loop
338420
Playfield Image
Action - Activate Flippers, Launch Ball
Left Flipper
Right Flipper
Plunger
Figure 2: Diagram of our training framework.

By combining simulation-based training with hardware fine-tuning, we aim to develop a robust agent capable of playing on the real-life TNA machine.

3.1 Problem Formulation

The PPO agent's states, actions, and rewards are defined as follows. The agent has five state observations: a downsampled image of the playfield; the ball's x and y position; and the ball's x and y velocity. Note that these values are normalized to remain within the region $[0, 1]$, to prevent saturating the network.

The action space is represented as a vector of four discrete actions. The first action, idle, releases the flippers. The second and third actions activate the left and right flippers, respectively. The fourth action activates both flippers simultaneously. The agent samples one action 20 times per second.

The reward function is calculated by factoring in the score, play time, ball position, and ball loss. The score is the raw game score tracked by the machine, and it is the direct metric we seek to increase. A large penalty when a ball is lost to encourage the agent from prematurely ending a game. Furthermore, to encourage activity in the typically sparse environment, a small reward is added whenever the ball is on the field above the flippers, and detracted when at or below. This position-based reward is necessary, as a simple time-based award would reach a local minima where the agent simply traps the ball on the flipper. We sum these three components to get the reward function as follows.

$$r_t = R_{score}(t) + R_{location}(t) + R_{ball}(t)$$

Where

$$R_{score}(t) = 0.00001 \cdot (\text{Score}(t) - \text{Score}(t-1))$$

$$R_{location}(t) = \begin{cases} 0.001, & \text{if PoseY}(t) \text{ is above flippers} \\ -0.0001, & \text{otherwise} \end{cases}$$

$$R_{ball}(t) = \begin{cases} -0.3, & \text{if Ball just drained} \\ 0, & \text{otherwise} \end{cases}$$

The game ends when three balls have been lost (not counting ball saves). At this time, the reward is reset to zero, and a new game is initialized.

3.2 PPO Model Setup

Additional settings are used to configure the PPO model. The current model is set relatively small, featuring 1 layer and 128 hidden units. Experimentation with more layers and units has begun, but a compact network was initially favored to prevent overfitting and reduce training time.

A recurrent neural network (RNN) is used to give the agent a short-term memory of 35 frames (about 1.5 seconds). This allows the agent to better factor in the game dynamics, as a single game frame does not convey ball velocity or acceleration. For the game frame image, a simple encoder with two convolutional layers is used to transform frames to the agent's space.

The agent's reward signals are influenced by a gamma γ of 0.99, encouraging the agent to care about long-term rewards.

To update the model, an epsilon ϵ of 0.2 was used, which will keep the updates more stable, but slow the training process slightly. A learning rate of $3e-4$ is implemented, with a linear schedule.

3.3 Simulation

We use the open-source Visual Pinball X project as the pinball simulation engine, with a digital recreation of TNA, developed by VPinWorkshop. Unity ML-Agents was used to handle the logistics of training the reinforcement learning agent. To communicate the score, ball number, position, and velocity from the simulation to the Unity agent, data is written directly to disk and then read. A virtual keypress simulator (WindowsInput) was used to send actions from the agent to the game.

3.4 Physical Control Implementation

Physical implementation consisted mainly of interfacing with the game input signals (flippers, plunger, and start button) to enable training and testing of the learning algorithm.

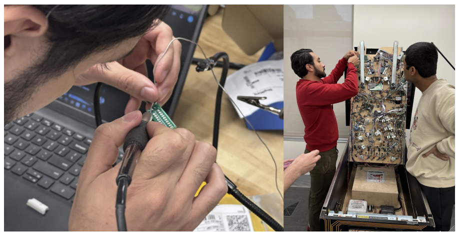
3.4.1 Input Signals

To access the game flippers, start button, and launcher signals, we utilized a Teensy microcontroller and a custom PCB board, connected via a serial interface. This setup enabled remote actuation of these components without relying on the physical elements. With the assistance of Spooky Pinball, we identified the correct control board, 'SW-16,' for handling input signals. Once identified, the custom PCB was connected to the system as illustrated in Fig (2) and Fig (3).

To facilitate external control, we developed an interface program consisting of two interlinked codes: one running on Arduino for direct hardware interactions and the other in Python for UI and high-level logic. The two codes communicated via a serial connection, enabling Python to send commands and receive updates from the Arduino. This program allowed the computer keyboard to remotely actuate the flippers, plunger, and start button.

3.4.2 Score Detection

Since directly interfacing with the score calculation was challenging, we used Optical Character Recognition (OCR) on the seven-segment display (SSD) on the back panel. Noise from animations caused blurring, so we adjusted camera settings (exposure, gain, brightness, contrast, and saturation) using OpenCV to produce a nearly binary image highlighting the SSD. We then used PyTesseract


(a) Custom wiring into the machine

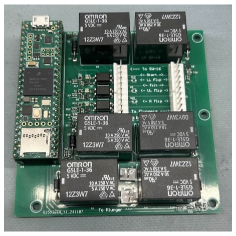
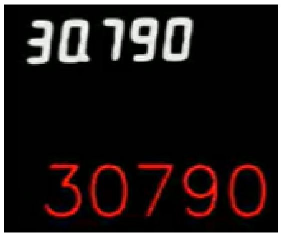
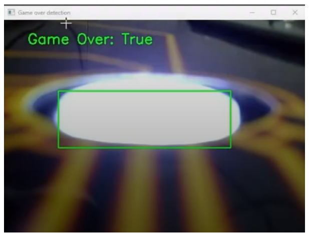

(b) Custom PCB designed to control game flippers, plunger, and start button over USB.

Figure 3: Custom electronics setup



(a) Example frame of OCR score recognition from camera. Actual score in white, detected in red.



(b) Example frame of game-over detection. Detects when start button is lit, indicating game over.

Figure 4: Vision-based detection of game states

with fine-tuned weights for the seven-segment font, along with additional preprocessing like thresholds and binary dilation, to improve recognition accuracy. To handle blinking during gameplay, we cached the last 10 scores and used the mode as the current score, introducing minimal latency. This process allowed us to reliably detect the score.

3.4.3 Playfield Streaming

We use a camera to capture a bird's-eye view of the playing field. Aruco tags are employed to crop and correct for skewed perspectives by identifying parallel lines on the rectangular field and performing a perspective warp for a corrected view.

3.4.4 Game Over State Detection

In order for our learning algorithm to automatically start a new game, a game over detection system was developed using a computer vision approach. On TNA, once the game has ended, the start button

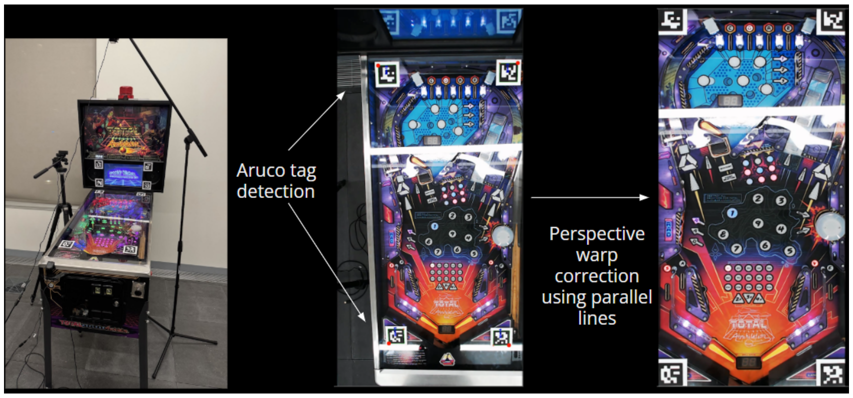


Figure 5: Playfield perspective correction using aruco tags and perspective warp.

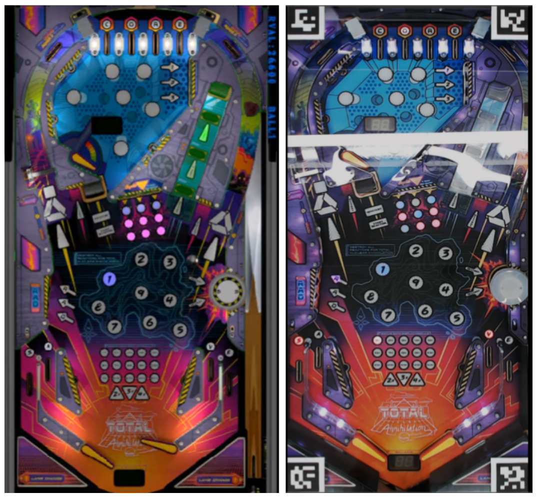


Figure 6: Comparison of simulated game (left) and physical game (right). Note that glare on playfield was greatly reduced in future testing.

begins to flash. Utilizing one camera, the system captures a video feed using OpenCV focusing on a defined region of interest (ROI) where the LED is located. By analyzing the brightness values, the system determines the LED state. A low brightness value indicates OFF, and periodic fluctuations indicate FLASHING thus, indicating a game over condition.

3.5 Physical Training

A machine malfunction halted deployment onto the physical system. With the guidance of the manufacturer's support, we were able corrected the error and returned the game to fully operational state. However, further physical deployment has remained paused until a model has been tuned in simulation, to prevent unnecessary wear and tear on the machine.

Once a model is pre-trained in simulation, transfer learning onto the physical machine will be attempted. Care has been taken to match the camera perspective and action space across both machines, as seen in Figure 6. Additional tuning is likely to be needed to match both systems, such as the game's pitch and the simulation's physics settings.

4 Experimental Results

After training for 90k steps, the model was evaluated against three baselines: a human player, a randomized agent, and no agent. 10 cases were run for each scenario, with aggregated results below.

	Human Player	Trained Agent	Untrained Agent	No Agent
Score Mean	38,952.86	33,952.00	14,420.00	8,358.00
Score Std Dev	26,546.47	35,787.92	9,981.91	376.43
Time Mean (s)	69.29	57.46	37.68	36.00
Time Std Dev	15.18	25.77	7.88	11.81

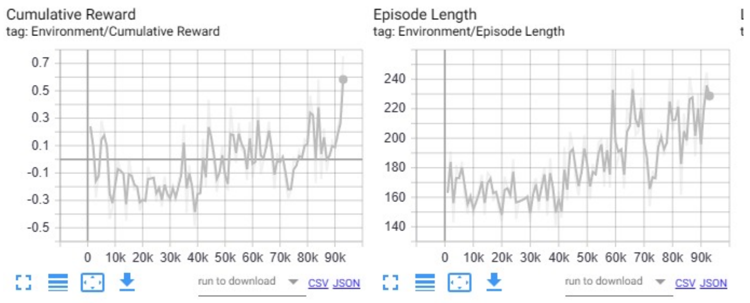


Figure 7: Cumulative Reward and Episode Length

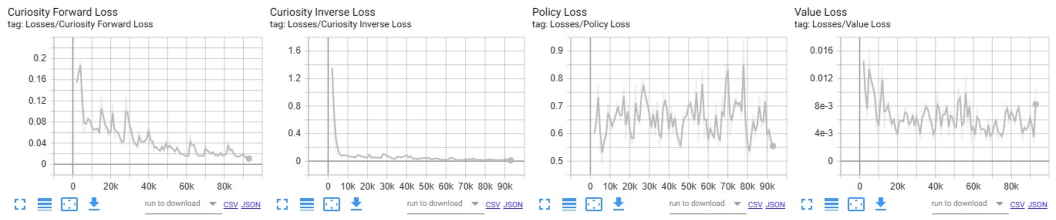


Figure 8: Losses

Rewards and losses are displayed in Figures 7 and 8. The final cumulative reward and episode length show notable growth, suggesting that the agent improved and learnt by the end of the training. The losses also reduce and stabilize as the model learns, suggesting that the training converged.

4.1 Observed Behaviors

Several notable behaviors were qualitatively observed during the model tuning process. Initially, the ability to launch a ball was provided as an agent action. The agent eventually learned to just idle without launching a ball, afraid of the future penalty when the ball drained. To eliminate this case, the ball launcher was put on an automatic timer and removed from the action space.

Within TNA, one of the highest scoring actions is to lock a ball in the upper right section of a playfield. If three balls are locked this way, a multiball will be unleashed, scoring major points. The strong reward from this action is visibly reflected in the model. During gameplay, the agent is frequently seen slowing down the ball until it is captured on a flipper. It then takes a precise shot at the lock zone. This behavior has been demonstrated consistently, multiple times, and is very tricky to pull off accidentally. On at least one occasion, the team witnessed the agent complete a set of three locks and begin a multiball, a challenge for even a human player.

5 Conclusion

In this work, we developed a reinforcement learning model to play pinball in simulation, showing heightened performance to random actions, and near-comparable performance to a human player. We laid the groundwork to transfer this system onto a real, physical system using cameras, electronics, and computer vision to detect game state, rewards, and termination states, manifesting agent actions in the real world.

5.1 Limitations

Although there is ample opportunity to improve the current algorithm with further model tuning and reward shaping, performance exceeding an expert player may be impossible. This is due to the skill of nudging, in which the player subtly pushes and shoves a table to influence the trajectory of the ball. On an expert level, nudging is critical to escape situations where the flippers cannot reach the ball. Since both our physical and simulated systems have no way to perform this motion, there are frequent unwinnable scenarios, in which the ball drains no matter what.

5.2 Future Work

Continued work will be necessary to match and exceed human ability. More training, reward function tuning, and other hyperparameter tuning will be conducted before we begin the transfer learning process. We aim to continue working on this project in the coming months, and the first point of focus will be achieving strong results on the simulation. Additionally, with the physical system fixed, we are eager to perform transfer learning to the real game after strong results on the simulation. Once these checkpoints are achieved, we can then explore other models and methods, such as using imitation learning to provide an initial set of weights. Furthermore, we would be interested in exploring a foundational pinball model, generalized for performance on any machine layout.

Acknowledgments

This project would not have been possible without the immeasurable support from the Robotics Institute and the greater pinball community.

We sincerely appreciate Doug Polka and PGH Pinball for making this opportunity possible by lending us their own rare machine. We would like to acknowledge AJ and Tory from Spooky Pinball, as well as Gerry Stellenberg from Multimorphic, for their guidance wiring into the machine.

We would love to thank Jarrod Homer, from Northeastern University, for his time and effort designing the custom control board.

We would like to thank Elliot Wood and his team at Microsoft Azure for their efforts in open-sourcing their previous work with impeccable documentation, and for providing initial pointers to the team.

Finally, from the RI, our sincere thanks to Jess Butterbaugh for securing us a labspace, and Kaitlyn Buss for her incredible help and responsiveness in procuring all the hardware we needed. Lastly, we would love to thank Professor Guanya Shi for creating the opportunity for us to embark on this project, and for providing the funding to enable us see it through.

References

[1] Pinball is booming in America, thanks to nostalgia and canny marketing. *The Economist*, May 2023. URL https://www.economist.com/united-states/2023/05/14/pinball-is-booming-in-america-thanks-to-nostalgia-and-canny-marketing.

[2] P. Funt. Welcome to the pinball renaissance. *The Wall Street Jorunal*, May 2024. URL https://www.wsj.com/sports/welcome-to-the-pinball-renaissance-2b83895b.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[4] 3rdaxis. Firepower (vs a.i.), Feb 2019. URL https://www.vpforums.org/index.php?app=downloads&showfile=14051.

[5] T. Gragg. Autopinball - spring 2020, 2020. URL https://tylergragg.com/autopinball-spring-2020/.

[6] B. Collins. The ai pinball player that could beat humans within 4 days. *Forbes*, 2020. URL https://www.forbes.com/sites/barrycollins/2020/05/20/the-ai-pinball-player/.

[7] E. Wood. Azure 3dppinballai. https://github.com/Azure/3DPinballAI, 2020.