Learning to Play Ms. Pac-Man with Reinforcement Learning

Zachary Esakof

Abstract—**Playing Ms. Pac-Man involves handling risk and reward trade-offs in simultaneously pursuing additional points while evading ghosts. To be successful, players must develop a strategy that incorporates both the ghosts' positions in 2D space as well as the proximity of rewards. I begin by training baseline reinforcement learning agents that follow random, proximal policy optimization (PPO), and Deep Q-Network (DQN) policies. Then I implement modifications to experiment with the impacts of gray-scaling, action trimming, reward hacking, and random starts in this arcade setting. Taken together, these modifications provide insights into strategies that succeed and fail when teaching an agent to play Ms. Pac-Man.**

I. Introduction

Created in 1982, Ms. Pac-Man is an iconic arcade game based on the original worldwide hit, Pac-Man. In this game, the player uses a joystick to navigate Ms. Pac-Man through a 2D maze. The goal is to score points by collecting pellets and fruits while avoiding the four ghosts that also move about the maze. If the player collects a power pellet (located in each corner of the maze), then the ghosts will turn a dark blue color and may be collected for additional points. The player has a top-down view of the maze, and can therefore incorporate ghost and pellet locations into their decision-making.

In recent years, reinforcement learning has emerged as a prominent approach to training agents to play games at expert levels of performance. Perhaps most famously, researchers at DeepMind trained AlphaGo in 2014, which was the first program capable of defeating the Go world champion. In this paper, I explore strategies for training a reinforcement learning agent to play Ms. Pac-man.

II. Overview

To train this agent, I leverage the Arcade Learning Environment (ALE). This section contains detail on Ms. Pac-Man's environment and reward function as provided by the classic arcade game's ALE emulation.

A. Observation Space

In Ms. Pac-Man, each observation is a 210x160x3 RGB color image of the game screen that a typical human player would see (Figure 1). That is, each image is comprised of 210x160 pixels ranging from 0 to 255 in value as well as three color channels. These images provide frame-by-frame information on the game's current state, and will ultimately be used to inform intelligent decision making.

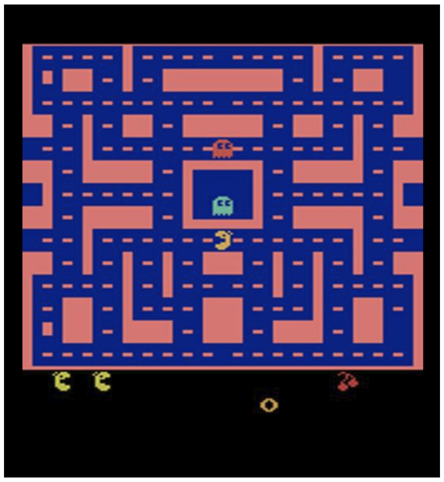

Figure 1. Initial state of the Ms. Pac-Man game environment as a human player would see it.

B. Action Space

The actions available to the agent include left, right, up, down, and diagonal movements that may be used to navigate the maze, pursue rewards, and evade ghosts. Additionally, there is a ninth movement, NOOP, which represents doing nothing (Table 1).

Action Number	Action
0	NOOP
1	UP
2	RIGHT
3	LEFT
4	DOWN
5	UPRIGHT
6	UPLEFT
7	DOWNRIGHT
8	DOWNLEFT

Table 1. Ms. Pac-Man action space. Available from https://www.gymlibrary.dev/environments/atari/ms_pacman/

C. Terminal State

The player starts with three lives and loses a life each time Ms. Pac-Man makes contact with one of the four ghosts roaming the maze. The game ends, and the ALE simulation reaches a terminal state, when the number of remaining lives reaches 0. There will also be a pause and level advancement if Ms. Pac-Man successfully retrieves all of the pellets scattered throughout the maze. Subsequent levels will feature some combination of faster moving ghosts, less potent power-ups, and various map configurations.

D. Reward Function

Collecting pellets is the primary way for an agent playing Ms. Pac-Man to score points, and collecting all of the pellets is required to advance to the next level. Each pellet is worth 10 points and there are enough to cover the entire maze. In addition to pellets, the agent can increase their reward by acquiring a power pellet, which is worth 50 points. The power pellet, however, also causes all ghosts to temporarily become slower and turn dark blue. In this state, Ms. Pac-Man can consume the ghosts for 200, 400, 800, and 1600 points for the first, second, third, and fourth ghost consumed, respectively. Finally, points can be scored by consuming fruit pieces that intermittently appear in the maze. The various fruits range from 100 points (cherries) to 5,000 points (bananas) (Table 2).

Fruit	Point Value
Cherry	100
Strawberry	200
Orange	500
Pretzel	700
Apple	1000
Pear	2000
Banana	5000

Table 2. Ms. Pac-Man fruit reward values. Available from https://pacman.fandom.com/wiki/Ms._Pac-Man_(game)#Scoring_System

III. Methods

I begin with a random agent, whose policy is to randomly sample from the nine actions in Ms. Pac-Man's action space. While this agent performs poorly, it acts as a lower benchmark against which I judge whether my learned policies are valid. That is, an effective policy should perform better, and ideally perform substantially better, than the random policy.

The two models presented here, Proximal Policy Optimization (PPO) and Deep Q-Network (DQN), are pulled from the StableBaselines Python library and trained using that library's default hyperparameters. These models and default hyperparameters are also used to train agents with the modifications detailed in this section.

A. Modification 1: Gray-Scaling

In its default state, each observation in Ms. Pac-Man manifests as a 3-dimensional array (width x height x channel). Besides for the ghosts, which occupy a small portion of the overall frame, there aren't many distinct colors in the Ms. Pac-Man game. Therefore, I theorize in this modification that gray-scaling the images entered into the neural network will reduce unnecessary information and allow the agent to learn a strategy more efficiently. To implement this change in the environment, I use the StableBaselines package's WarpFrame wrapper class. The first gray-scaling scenario uses the package's default, which shrinks the image down to 84x84 pixels in addition to gray-scaling. To isolate only the effect of gray-scaling, however, I conduct a second experiment where I preserve the original image's full resolution.

B. Modification 2: Action Trimming

In a typical Atari emulator game, there are at most 18 actions available to the user. As mentioned in Part II, Ms. Pac-Man's action space only consists of 9 actions. This modification alters the environment's action space to evaluate whether the diagonal movements impact the agent's performance. Reducing the number of action choices from 9 to 5 while maintaining the ability to move in all cardinal directions should allow the agent to more efficiently learn to navigate the maze. Reducing the number of possibly ineffective actions should further allow the agent to be more reactive and less likely to become trapped in a corner of the maze. That is, I theorize that actions such as DOWNRIGHT and DOWNLEFT are of minimal importance in Ms. Pac-Man.

C. Modification 3: Reward Hacking

For this modification, I experiment with new reward functions. In the first two scenarios, I increase the reward from bonus points that are awarded when Ms. Pac-Man collects a power pellet, ghost, or fruit piece. The first scenario multiplies bonus rewards by 10 and the second multiplies bonus rewards by 100. These adjustments should induce the agent to seek out bonuses and therefore tend to earn a higher score than the standard DQN approach. In the third reward hacking scenario, I create a reward function based on the time that Ms. Pac-Man survives in the maze instead of on point value from collecting pellets and fruit. Specifically, I assign each action at each timestep a value of 1, completely overriding all other rewards that the agent would normally consider. I expect this modification to drastically alter the agent's decision-making relative to the other modifications considered in this paper. Emphasizing longevity over a high score may also produce strategies that human players could use when they need to be evasive in actual games of Ms. Pac-Man.

D. Modification 4: Random Initialization

By default in the ROM emulation and in all physical arcade versions, Ms. Pac-Man and the ghosts begin in the same spot in the lower-middle and upper-middle parts of the screen, respectively. When training a reinforcement learning agent, this fixed initialization may limit the diversity of situations captured by trajectories in the training data. To simulate a random initialization in which the ghosts don't always begin in the same location, I force Ms. Pac-Man to execute a random number of NOOP actions at the beginning of each trajectory. While Ms. Pac-Man will still begin in the same location, the ghosts will have had between 0 and 300 frames to begin moving. Since each simulation of Ms. Pac-Man begins with a small jingle lasting 265 frames, during which time neither Ms. PacMan nor the ghosts can move, random initialization of less than 265 would have no effect. To solve for this, I had to modify the StableBaselines NoopResetEnv class to include a 265 frame minimum. Therefore, the agent-in-training actually started each trajectory with between 265 and 565 NOOP actions.

IV. Results

I have run 20 iterations of the Ms. Pac-Man simulation under random, PPO, DQN, and modification policies. Since trajectories produced from a policy will vary, these 20 iterations produce a distribution of the rewards that each policy can achieve. To visualize these distributions, I have produced box plots for each policy and placed them side-by-side (Figure 2). In a box plot, the line through the center of each box depicts that distribution's median. Based on this central point, the PPO policy significantly improved upon the random agent, while the DQN policy outperformed both the random and PPO agents.

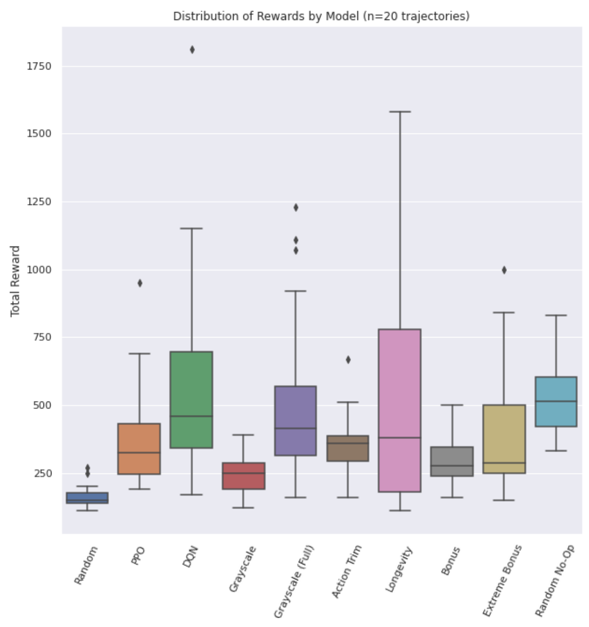

Figure 2. Trajectory total reward distributions for random, PPO, DQN, and modification policies.

Analyzing the modifications, most failed to improve upon the standard DQN with default parameters. Gray-scale (with downsizing) and the two bonus-boosted modifications in particular only had marginally better medians than the random policy, while the full-resolution gray-scale and random NOOP initialization modifications ranked very closely to the original DQN policy. Comparing the two grayscale scenarios, the full-resolution significantly outperforms the downsized gray-scale. Decreasing the image's size likely led to information loss than hindered the downsized gray-scale's performance. The similar medians between full-resolution gray-scale and the default DQN suggests that training on black-and-white images does not provide the agent with a significant advantage or disadvantage in the learning process.

Interestingly, varying the degree of reward hacking from 10x to 100x only slightly alters the median of the reward distribution. There was a substantial skew, however, towards larger rewards when providing the agent with a 100x boost on bonus rewards. This may indicate that the reward hacking had its intended effect and that an agent trained under this policy is more likely to engage in risky behavior to pursue bonus rewards.

Perhaps the most surprising result captured in Figure 2 is the wide range of total rewards that an agent trained under the longevity scenario achieved. While many outcomes in this distribution are rivaled only by the original DQN policy, the longevity scenario's worst outcome resembles a random policy. Watching the longevity policy's video at the link provided below is instructive here, as it shows Ms. Pac-Man hiding in the corners of the maze. Since this agent was trained to maximize its playtime, it will forgo pellets and position itself far from ghosts. This behavior, in turn, drives Ms. Pac-Man towards the corners, where the agent inadvertently picks-up a power pellet before being caught by a ghost and gaining the bonus.

In order to visualize how each policy performs relative to the others over the course of a simulation, I graph the worst and best performing trajectories from each policy's sample of 20 (Figures 3 and 4). These individual trajectories further illustrate the aggregate performance highlighted in Figure 2. That is, performance among these samples increases in going from random to PPO to DQN.

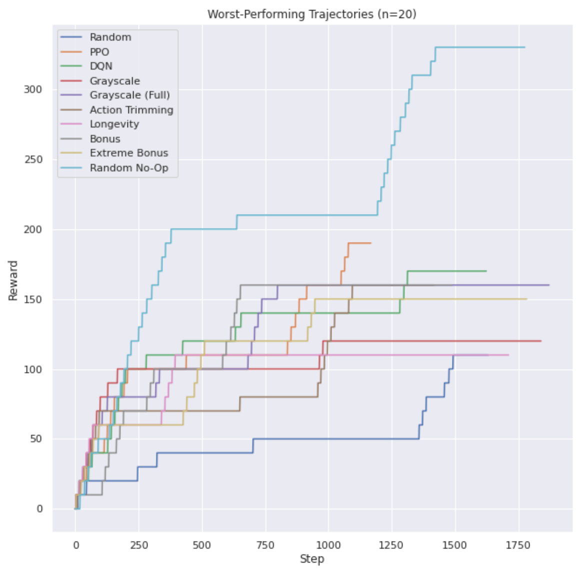

Figure 3. The worst trajectories produced by each of the random, PPO, DQN, and modification policies' 20 iterations.

I plot worst and best instead of the average trajectories for this analysis because each trajectory differs in length. If I were to use an average trajectory, then that trajectory's values would be comprised of a varying number of underlying trajectories. For example, the best-performing full-resolution gray-scale policy ran for more than 2,700 steps. Many full-resolution gray-scale iterations, however, ran for fewer than 2,200 steps (not shown). As a result, the average rewards that would be calculated from step 2,200 to step 2,700 would be based on fewer trajectories than the average rewards calculated from step 1,000 to step 1,500. Therefore, I use Figure 2 as the primary means of comparing reward distributions and Figures 3 and 4 as example trajectories of those distributions' observations.

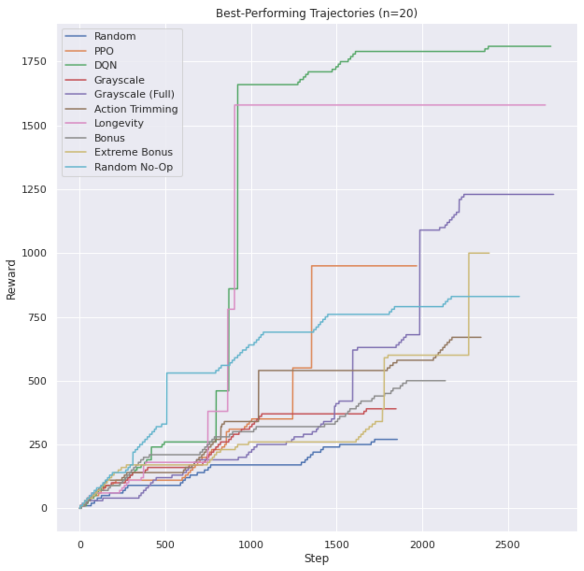

Figure 4. The best trajectories produced by each of the random, PPO, DQN, and modification policies' 20 iterations.

Video samples of the agents executing random, PPO, DQN, and modification policies to play Ms. Pac-Man are available on my YouTube channel (link: https://www.youtube.com/channel/UCn8eGj-48YBCl4eexOw30Qw). Note that there is an initial period where the simulation begins but Ms. Pac-Man doesn't move. This occurs because each simulation begins with a short tune that precedes any actions. The videos linked above, however, contain no audio. Furthermore, this period of no input has been removed from the trajectories in this report, which would otherwise show no rewards until step 270.

V. Conclusion

Ms. Pac-Man provides a rich environment for training a reinforcement learning agent. The variety of ways to score points and balance the risk of being caught by a ghost with the reward of consuming more pellets or fruit provide many strategies for the agent to learn. Comparing random, PPO, and DQN agents revealed that DQN performs best in the Ms. Pac-Man environment. Therefore, the subsequent scenarios in gray-scaling, action trimming, reward hacking, and random initialization focus exclusively on modifying Deep Q-Networks. Perhaps unsurprisingly, the results demonstrate that a DQN with default parameters is an effective learner. At the same time, the experiments also yield interesting insights. Gray-scaling at different resolutions (full and downsized) demonstrates how information can be lost at lower resolutions, while these experiments together show that training on black-and-white images is comparable to training on RGB images in Ms. Pac-Man. Stressing bonus rewards through a modified reward function precipitates riskier behavior while a reward function oriented toward survival rather than point accumulation causes Ms. Pac-Man to hide in corners to varying degrees of success. Finally, adding a randomized number of NOOP actions to the beginning of each training trajectory produces promising results and may warrant further exploration.

In exploring numerous way to improve a reinforcement learning agent's skill in Ms. Pac-Man, there were many promising experiments that I did not perform. When trimming actions, for example, one could also remove the NOOP action, leaving only UP, DOWN, LEFT, and RIGHT and creating a hasty or constant-motion agent. I added most of the modifications to the baseline DQN model, allowing me to easily attribute improvements to specific changes. However, stacking multiple modifications (i.e. full-resolution gray-scaling and extreme reward boosting) may induce the agent to learn more complex behaviors and ultimately increase performance. Blending reward functions with tune-able parameters for each component function is an interesting idea and may allow the agent to balance multiple incentives. For example, taking a weighted average of the extreme reward boosting and longevity reward functions to form a single function may allow the agent to both seek bonuses and avoid ghosts.

Acknowledgments

Thank you to Professor Held as well as to the 16831 TAs Shuli Jiang and Sam Triest

References

[1]https://www.gymlibrary.dev/environments/atari/#id1
[2]https://www.gymlibrary.dev/environments/atari/ms_pacman/
[3]https://pacman.fandom.com/wiki/Ms._Pac-Man_(game)#Scoring_System
[4]https://stable-baselines3.readthedocs.io/en/master/index.html