Reinforcement Learning for Pokemon Red

Alec Letsinger, Eli Weissler, and Josh Priest

***Abstract*—We demonstrate two Reinforcement Learning based approachs to playing Pokemon Red. The first approach, based on the DQN algorithm, was largely unsuccessful and was limited by the game's large state space. The second approach, which leverages the Proximal Policy Optimzation algorithm combined with Random Network Distillation, is capable of completing the game's first main task, which requires navigating to retrieve an item and backtracking.**

***Index Terms*—Reinforcement Learning, Pokemon, DQN, PPO, Fun Project, Random Network Distillation**

I. Introduction

Video games have long been used to benchmark reinforcement learning algorithms, as they provide predictable and repeatable environments to facilitate learning. However, not all games are equally easy to learn. Games that focus on quick reactions, such as breakout on the ATARI, are significantly easier to learn than those that require longer term strategy and exploration. Reaction games often have relatively small state and action spaces, with more frequent rewards, which make learning a policy easier. On the other hand, games such as Monetzuma's Revenge have notoriously been difficult to learn at a super-human level until recently [1]. The challenge stems from the large state space and requirement to recognize connections between far away events, such as a key in one room being used to open a door in another.

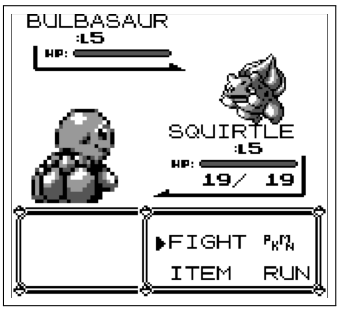
The game Pokemon Red, released on the Nintendo GameBoy in 1999, is similar to Montezuma's Revenge in that many state-of-the-art reinforcement learning algorithms fail to even make marginal progress in the game. It requires similar long term planning, including an early quest that requires the player to retrieve and return with a package from a nearby town. This paper explores the implementations of reinforcement learning algorithms based on both the Deep Q Network (DQN) and Proximal Policy Optimzation (PPO) algorithms.

II. Background and Related Work

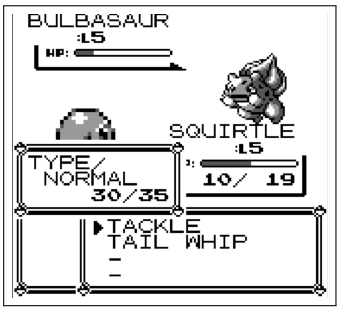
A. Description of Pokemon Red

Pokemon Red is the first installment of the Pokemon video game series, which imagines a world inhabited by monsters known as "Pokemon." The game follows a "Pokemon Trainer" traveling the world collecting and battling Pokemon.

The game has two main mechanics, open world exploration and battling. As seen in fig. 2, the world consists of a large "outdoor" area, with small "indoor" sub-areas that must be explored to progress through the game. The world is shown from an overhead view, with the player able to move up, left, right, or down. The player can also interact with other characters and items in the world. As the player progresses, they can enter battles with both other trainers and wild Pokemon. The opposing trainers must be defeated to progress in the game, while wild Pokemon may be fled, defeated, or caught.



(a) The player can either fight, switch Pokemon, use an item, or run (not available when battling other trainers)



(b) When fighting, Pokemon have up to four moves that they can use in battle, each with different effects. As Pokemon win battles, they increase in level and learn new moves.

Fig. 1: Example Pokemon battle. The battle continues until all of one trainer's Pokemon have an HP of 0. A Pokemon's HP is generally reduced after incurring an attack.

Pokemon Red presents a challenge for reinforcement learning algorithms from multiple angles. First, the game requires extensive exploration and long term planning. As seen in 2, one of the first tasks in the game involves navigating between towns to retrieve a package. This requires both finding the package and backtracking to deliver it. Second, the game includes two very different gameplay mechanics in exploration and battling. Although they should be distinguishable to the agent, battling is a both complex and somewhat rare event that will be difficult to experience enough to learn an optimal policy. That being said, even a random battling policy would likely be sufficient to progress through the game, as long as the player has stronger Pokemon than their opponents.

B. Project Inspiration

The inspiration for this work stems from a YouTube video posted by Peter Whidden on training agents to learn to play Pokemon Red using reinforcement learning algorithms with the state being imagery of the game similarly to how a human would interact with the game [2]. While this implementation made significant progress in the game, it eventually stalled progress upon entering the game location Mt. Moon, a dark cave where the game screen remains dark unless the agent does a very specific sequence of actions to teach one of the Pokemon the move Flash. Since the game screen is dark for every tile location within Mt. Moon, the reinforcement learning algorithm cannot distinguish between states in Mt. Moon leaving the agent stuck in the cave until all of its

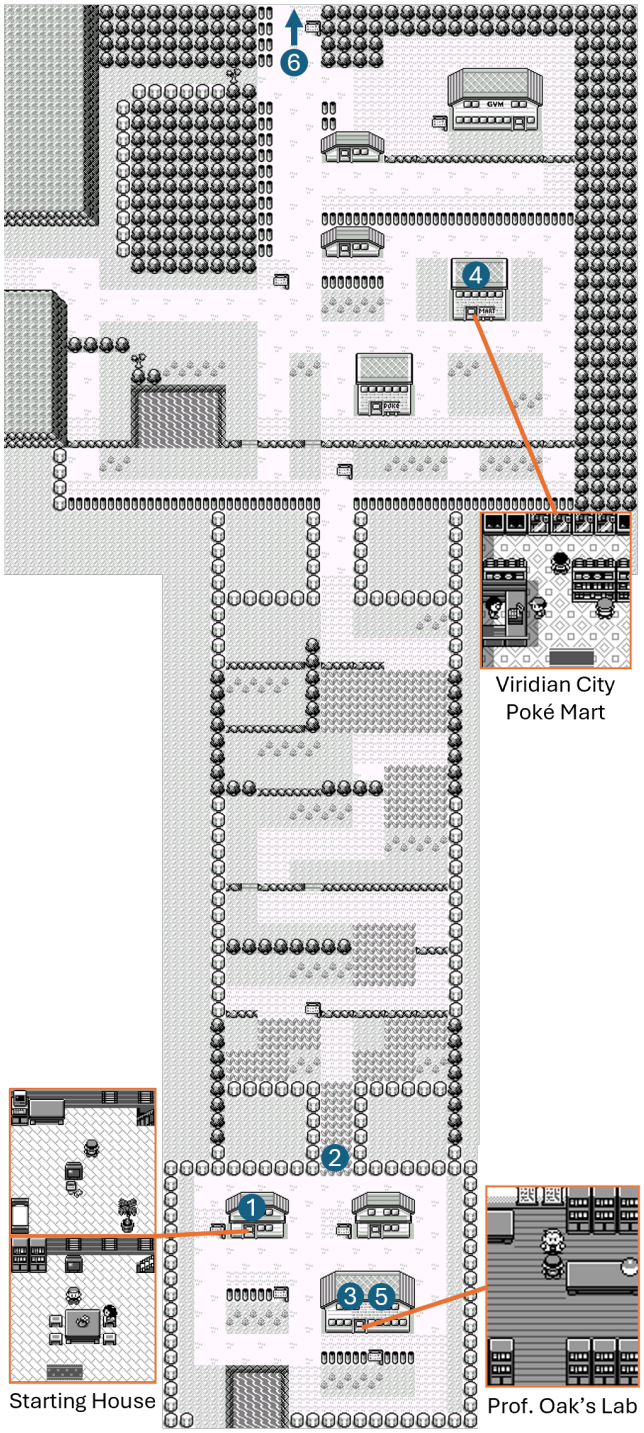


Fig. 2: Map detailing the beginning of Pokemon Red. The player begins in the starting house and must navigate to the numbered locations in order (interacting with other characters at each step) to progress in the game. Numbers 3, 4, and 5 represent a quest in which the player must go to the Viridian City Poke Mart and retrieve a package for Prof. Oak. Number 6 represents the direction of the next area in the game's progression.

Pokemon faint. This area of the game essentially becomes unobservable.

An alternative to using the gameplay imagery as the state data is to use the game's RAM memory to define the state. In theory this would allow the agent to distinguish between tiles in Mt. Moon as it has perfect observability to the game state (as the RAM by definition contains the fully defined game state). This has worked well in implementations on ATARI games such as Montezuma's Revenge where the RAM size is 128 bytes. However this quickly becomes less tractable on modern games where the RAM size is much larger. Even though Pokemon Red is by no means a large game, the allocated RAM on the Nintendo GameBoy is 8kB which is quite large for a reinforcement learning algorithm to consider.

To proceed past this point in the game, further iterations of Peter Whiddens implementation use both images of the game state as well as passing in hand-selected pieces of the game RAM to navigate through locations such as Mt. Moon.

Another problem with training a reinforcement algorithm to play Pokemon Red is the large state space and sparse reward structure. This problem is not unique to Pokemon Red as Montezuma's Revenge is another notable game with this problem. There have been various different successful approaches to incentivising exploration for the agent including Random Network Distillation[3], Go-Explore [4], and even reward shaping that simply rewards the agent for visiting states it has not seen before.

Random network distillation fundamentally works by implementing an intrinsic reward. This is done by initializing two neural networks to random weights, a predictor and target network. The predictor network takes in the game state and tries to predict the out of the target network which also takes in the game state. The predictor network is then trained to minimize the mean-squared-error loss of these two network outputs. This loss is then normalized by the running standard deviation of that loss and the added environment reward (known as the extrinsic reward). The idea is that the predictor network will better predict the target network if it has seen the current state before, thus resulting in a lower loss. States that have not been seen before will result in a higher loss thus a higher intrinsic reward [3].

The Go-Explore algorithm similarly incentives exploration but does so in phases. The agent maintains a list of high value states and the map of how to return to these states. It will then return to these high value states and explore starting from the promising states.[4] This assumes that high value states are near other high value states which is generally true for games such as Pokemon.

III. Problem Formulation

A. Levels of Success

This project is based on successive milestones that outline various levels of difficulty in order to either build upon previously implemented algorithms or experiment with new algorithms to improve performance.

1) Develop the environment in Python and learn a policy capable of getting out of the starting house within 100 steps: The first milestone is to implement a algorithm to develop a

policy that is capable of escaping the starting house in less than 100 steps. Fundamentally, the starting house is a series of grid worlds to be solved with a singular sparse reward at the exit. Potential suitable algorithms for this problem include value iteration, Monte-Carlo tree search, SARSA, and Q-Learning. Additionally a custom Gym environment will need to be developed in Python such that an agent can interact with it. This environment will largely be common amongst all algorithms implemented.

2) Progress further using random network distillation: Since Pokemon Red parallels Montezuma's Revenge in many ways, random network distillation is a promising way to encourage the exploration of large complex environments which is necessary to make significant progress in Pokemon Red. The goal here will be to complete the series of tasks defined in fig. 2 that define the beginning of Pokemon Red.

3) Further improve the algorithm to beat Brock and make it through Mt. Moon: Upon the beginning of this project, the record for AI game progression is Mt. Moon. Since then this record has been beaten, but this still proves to be a difficult benchmark to match.

B. Environment

The environment is configured as a Python AI Gym custom environment where the user can define state, action, and reward spaces as well as observation functions and transition functions. The state space is taken from game RAM at every step. The number and type of variables included in the state space is largely a tuning parameter as passing a smaller subset of the RAM will likely allow faster learning since there are less parameters to consider. If too few variables are passed the agent will be incapable of learning certain tasks. The largest state space the team considered is shown in Table 1 with a total dimensionality of 41.

The game is by definition Markov as the whole game RAM is passed as the state, but since only pieces of the RAM is being used, the Markov assumption likely does not hold for all states. Passing images of the game state also violates the Markov assumption as many previous frames often influence the current frame, resulting in an ill posed problem where one outcome can stem from countless unrelated scenarios. In the case where the state space consists of images, a common solution is overlaying multiple images to try to capture previous states in the current state. Using the game RAM as a basis for the state space typically will provide the agent with more complete information than just passing an image of the game state, in turn removing the ambiguity stemming from an image based state space.

Subsets of this state space were used to generate a simplified game model. For example, to traverse the starting house to meet milestone 1, the following subset of the state space was used:

$$S = \{X_{Location}, Y_{Location}\}$$

State Dimension	Description
1	Bias Term
X position	X position of player in current location
Y position	Y position of player in current location
Map Location	Integer representing map location ID
Type of Battle	Integer representing type of battle
Agent Pokemon 1	ID for first Pokemon in party
Agent Pokemon 1 HP	Current Health of first Pokemon in party
Enemy Pokemon	ID for Enemy Pokemon in battle
Enemy Pokemon HP	Current HP of Enemy Pokemon in battle
Party Levels	Array of current levels of party Pokemon
Party HP	Array of current health of each party Pokemon
Party Max HP	Array of maximum health of each party Pokemon
Gym Badges	Binary Array of Gym Badges Acquired
Flags	Summation of number of flags triggered

TABLE I: State Space Definition

The action space is relatively small and is defined with just 7 actions as shown below mapping to the possible inputs on a Nintendo GameBoy.

$$A = \{Up, Down, Left, Right, A, B, Start\}$$

The actions are then passed into an instance of Pokemon Red running using PyBoy, a GameBoy emulator package developed for Python development. The environment is configured such that once a button is pressed, the game is cycled for a frame to process the input. The button is then released and several frames are cycled to let any transience settle. Once an action is taken, the environment is then able to read the agents state directly from the emulator's memory. The environment assumes perfect observability.

The reward function is shaped primarily by using the total number of game flags that have been triggered. The game flags generally trigger once something has been done to progress the game such as getting your first Pokemon or defeating a trainer. Since the environment is entirely configurable, this reward function can be tuned to give the agent alternative incentives as needed.

IV. Solution Approach

Multiple reinforcement algorithms were implemented to try to teach an agent to play Pokemon Red including Q-Learning, Q-learning with Random Network Distillation, PPO, and PPO with Random Network Distillation.

A. Deep Q-Learning

Deep Q-Learning (DQN) is a variant of Q learning that uses a neural network to estimate the state-action pair value (Q-value) function of an the environment. [5]The agent then acts based on which action yields the highest Q-Value. This algorithm performs best on environments with low state and action space dimensionality. The agent is trained on interactions with the environment according to the loss function:

$$L_{DQN}(s, a, r, s') = (r + \gamma \text{max}'_a Q(s', a') - Q(s, a))^2$$

The core principle of the DQN algorithm is that the loss function goes to zero if Q is optimal, as it is a statement of Bellman's optimality condition. One of the main challenges

with Deep Q-learning is exploration, as it is only trained on state/action pairs that it encounters and may be unlikely to reach rewards based on the initial weights of the network.[5]

Our Deep Q-Learning algorithm was implemented from scratch using pyTorch. It was based on the version that was implemented during HW5.

B. Proximal Policy Optimization (PPO)

PPO is another reinforcement learning that follows the actor-critic model using a neural network to approximate the optimal policy function (Actor) and another neural network to estimate the value function for that policy function (Critic). The actor networks defines how the agent interacts with the environment while the critic function evaluates those actions. PPO diverges from traditional advantage actor critic (A2C) in that it uses clipping to ensure that the updates to the policy are not too large. A historical problem with policy networks is that the policy can change very dramatically within a training loop, which can lead to instabilities. By clipping the update to the policy network, the agent will learn a bit slower but will learn with much greater stability.[6]

Our PPO algorithm is based on a modified example of PPO for solving the cartpole problem [7].

C. Random Network Distillation (RND)

Both of these algorithms are capable of learning optimal policies but often struggle with the exploration of the environment. Easy traditional solutions to this are decaying epsilon greedy policies for taking actions in the environment which excel at exploring the environment early in training and then eventually exploiting the environment with the learned policy. In an effectively infinite horizon problem like Pokemon Red, this proves ineffective as the agent should be exploring both early in the game and later in the game.

Another solution to the exploration problem is using Random Network Distillation to generate intrinsic motivation to add to the reward function which promotes visiting previously unexplored states both early and later in the game. Historically, this approach has proved to be successful when implemented in Montezuma's Revenge, a reasonable comparison to Pokemon Red.[3]

Our RND algorithm was implemented from scratch as an addition to both the Q-learning and PPO algorithms.

D. Algorithm Selection

Deep Q-learning was the first algorithm tested due to its familiarity and ease of implementation. Although the team recognized that the large state space and need for exploration would cause a basic Deep Q-learning algorithm to stall out, it proved a valuable testbed for debugging the environment and learning about the problem. Some progress was achieved by reducing the size of the state space down to just X and Y positions. While Q-learning was eventually capable of getting out of the house (first location), it was not scalable to the remainder of the game. Random network distillation (RND) was added to encourage exploration, however it did not solve the fundamental limitations of Q-learning for our problem. We ultimately transitioned over to proximal policy optimization (PPO) as recommended by a group already working on a reinforcement algorithm to play Pokemon Red. PPO solves Q-learning's shortcomings with large state spaces and Q-learning's tendency to fixate on singular actions given a state. PPO still struggled with exploration as the entropy term found in vanilla PPO doesn't encourage the required early and late stage exploration enough. This was made evident by the agents typically acquiring their first Pokemon pretty quickly, but then quickly stalled progress when required to explore to find the next town. RND was integrated into the PPO algorithm to promote this necessary exploration for progress.

V. Results

A. Q Learning

Vanilla Deep Q-learning was initially implemented using the full 41 dimensional state vector and reward function:

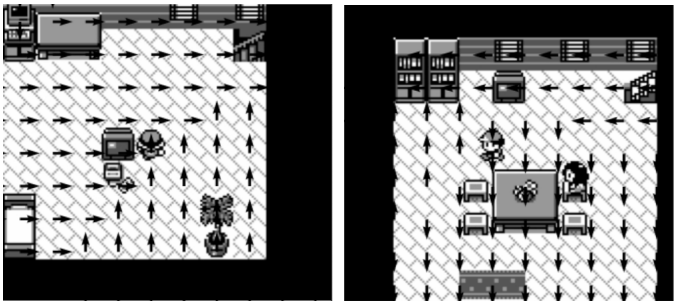
$$R(s,a) = N_{\text{Flags Triggered}} + 0.1 N_{\text{Locations Discovered}}$$

to incentivize both exploration and progression in the game. The agent used an epsilon greedy policy with a decaying epsilon to promote exploration early in the training process and exploitation later in training. The performance of this Deep Q-learning algorithm was *remarkably poor*, with the agent often requiring thousands of steps to even get out of the starting house. In the end, the performance appeared equivalent to that of a random walk.

Although this Q learning implementation did not prove fruitful, investigating its poor performance highlighted important issues in both the algorithm and the environment. First, the agent often got stuck in menus by hitting the START button. Second, the environment was initially set such that it did not hold down the directional inputs for long enough to move in a direction that it was not initially facing. As a result the state space was effectively not complete, as the players orientation also impacted the transition to a new state. As a result, the START button was removed as a possible action (it is necessary to complete the game, but only much further in), and the environment was modified to hold down the directional buttons for long enough to ensure one square of movement regardless of initial orientation.

The second attempt at Deep Q-learning was refocused around completing the first level of success: learning a policy capable of getting out of the starting house within 100 steps. This goal was achieved, although through the development process, it became clear that the method would not be suitable for general game play. To simplify the problem, the state space was simplified to include only the X and Y locations of the character. Multiple representations of the map location were tried (needed to distinguish between the floors of the house), such as including it in the state space as an integer and as a one-hot-encoded value. However, due to the X/Y coordinates being similar between the floors, the neural network model had great difficulty distinguishing between the two areas. Ultimately multiple networks were used, one for each floor.

This would be equivalent to treating both floors as independent grid-world type problems.



(a) first room (b) second room

Fig. 3: Optimal actions according to the Q learning network, taken from (Video). The player starts under the TV in the first room, and must navigate first down the stairs in the top right corner of the room, then out the door at the bottom of the second room (marked by the doormat).

Using a reward function of +10 for going downstairs or out the door, -10 for going back upstairs, and 0 otherwise, the agent was able to learn to usually get out of the house in under 100 steps (Video) using a softmax exploration policy (although it would occasionally get stuck in a corner). The softmax was a critical improvement over the epsilon greedy policy, as often two directions would be approximately equivalent (for example if moving diagonal is optimal). See fig. 3 for a visualization of the learned policy to get out of the house.

Ultimately, this level of complexity seemed infeasible for the larger game. With a single network, the agent struggled to distinguish between locations, while with multiple networks, the agent struggled to learn without an explicit notion of progress within each floor. Additionally, leaving the bottom floor in any manner (i.e., going back upstairs) had to be treated as a terminal state, with the episode terminating. Neither choice was a promising option to scale to the full game.

Random network distillation (RND) was additionally implemented in an attempt to encourage exploration within the limited (but already solved) Q-learning context. RND was successful in helping the agent learn how to get out of the house faster, especially if the random initial network weights resulted in a policy pointing away from the goals.

B. Proximal Policy Optimization

Upon encountering problems training an agent to effectively and efficiently learn to play Pokemon Red, we consulted experts in Pokemon reinforcement learning, including Peter Whidden, creator of the project which inspired this project. We explained to them the problems we encountered with deep Q-Learning detailed above. They suggested potentially transitioning to a more state-of-the-art algorithm in PPO to alleviate some of the drawbacks of deep Q-Learning. One advantage of PPO is that it used many agents to generate data to train the network at once and then uses a common network to select actions for the agents. Essentially creating a semi-cooperative multi-agent algorithm which has proven to work effectively in complex environments such as Pokemon Red.[8] It uses log probabilities to select actions for the agent so that they diverge in paths allowing for increased exploration and a reduction in redundant training data.

1) Initial PPO Experiments: One of the most important choices is defining the reward function to accurately reflect the goals of the game, while still providing enough feedback for the algorithm. The initial PPO experiments used a reward function of:

$$R(s,a) = N_{\text{Flags Triggered}} + 0.1 N_{\text{Locations Discovered}} + \sum \text{Pokemon Level}$$

This reward function aimed to encourage progression in the game, while rewarding agents for leveling up Pokemon (which will in turn make battling easier). Note the reward function here is defined such that the reward continues to be received every step after triggering a flag, discovering a location, or leveling up a Pokemon, not just the step in which the progression takes place.

The initial attempt at PPO implementation performed better than Deep Q-Learning had in making game progress, but still struggled to navigate efficiently including leaving the starting house within 100 steps and traversing Pallet town to get a starter Pokemon. The standard PPO implementation incorporated an entropy term in the loss function to maximize in order to promote diverse strategies and exploration. Additionally, the actor and critic networks were quite small in width and depth potentially limiting the learning capabilities of the networks.

In a second attempt at PPO, several changes were made to the hyperparameters and learning algorithm to improve performance. The first of these changes was changing the actor and critic network size to 3 layer with 128 neurons in each layer. Additionally, random network distillation was implemented because of its success in the deep Q-Learning implementation. These changes drastically improved the performance of the learning algorithm with an agent nearly reaching the level 2 goal of making it to Viridian Forest. During the course of training several agents were able to deliver Oak's package, a difficult benchmark due to having to make the long trek from Viridian City back to Pallet town, a place which had already been thoroughly explored, in order to progress the game and unlock Viridian Forest.

Various learning curves for the PPO algorithm with random network distillation are shown below in fig. 4 for a training session comprised of 100 agents training over 35 million total steps in the environment. Given the the positive trend in rewards at training termination, it can be assumed that further training would likely result in further improvements to the learned policy.

The maximum extrinsic reward subfigure tracks the best performing agent at any given step in the environment. Alternatively, the average extrinsic reward is the reward given from the environment averaged across all agents for a given step. The average intrinsic reward is the curiosity driven reward given to agents for visiting lesser visited states to promote exploration. The average reward is the summation of the average extrinsic reward and average intrinsic reward which most holistically illustrates the learning curve for the training session.

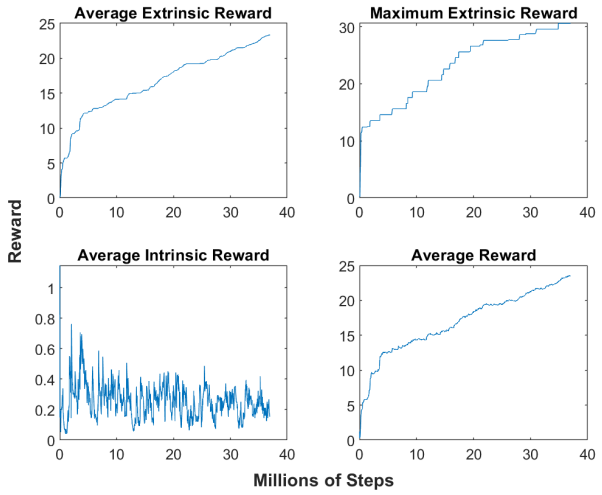


Fig. 4: Sample Learning Curve

One interesting thing to note about the PPO training is that typically once one of the agents figured out how to achieve a reward, shortly after, many of the other agents would then figure it out as well. This is due to a singular model controlling all of the agents, so they share experiences and learning to eventually move towards the desired outcome. This phenomenon can be seen when comparing the maximum flags triggered with the average number of flags triggered as shown in fig. 5 which depicts flags triggered in the first 1 million steps of training. Note how the maximum flags spikes from 0 around 100,000 total steps into the environment. Shortly thereafter, the average number of flags begins to rise in similar fashion as other agents learn where the reward is located to trigger these flags.

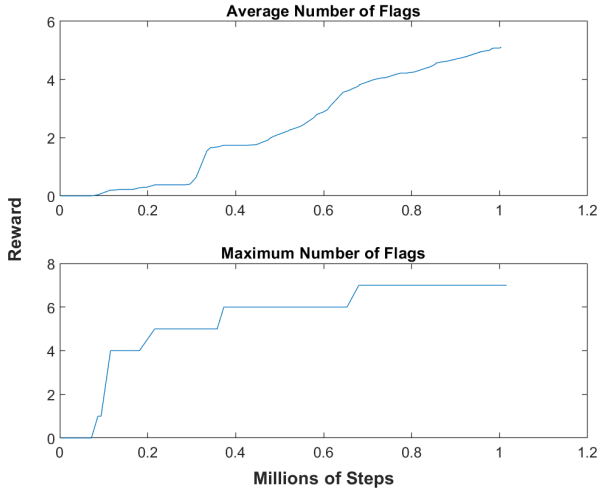


Fig. 5: Flags Reached

Based on the extrinsic reward structure, these flags are used to determine an agents progress in the game. The benefits of collaborative multi-agent learning greatly improved the performance of the policy learned compared to the previous efforts using Deep Q-Learning.

2) Benchmark Tests: To test the performance of the algorithm, a benchmark test was conducted. The progression of the PPO agent was compared using multiple reward structures with that of an agent following a random policy.

Using a reward of the form:

$$R(s,a) = A \cdot N_{\text{Flags Triggered}} + B \cdot N_{\text{Locations Discovered}} + C \cdot \sum \text{Pokemon Level}$$

the following three tests were conducted, each with at least 50 million total steps spread across 50 agents (for 1 million steps each). Note one trials was run for slightly more steps (as seen in the plots).

Trial	A	B	C
Flat	1	0.1	1
Sparse 1	100	0.1	1
Sparse 2	100	10	1

TABLE II: Reward Function Constants

fig. 6 shows how these different reward structures were able to progress through the game. The red curve represents agents acting under a random policy. Sparse 1 represents agents acting under the PPO algorithm with a sparse reward structure that more heavily incentivizes flag progression over Pokemon party levels and exploration. Sparse 2 is structured similarly to Sparse 1 without the emphasis on flag progression. Their reward structure is sparse because the agent only receives the flag reward once when they trigger the flag and then do not receive it again. Contrarily the flat curve rewards an agent at every step in the environment if they have a flag triggered. For example, if the agent has triggered 3 flags, at every subsequent step it will at least receive a reward of 3.

All of the PPO algorithms were able to acquire more flags than the random action algorithm. Interestingly, the random algorithm performed better initially triggering more flags earlier on average. Likely this is because random actions is purely exploration, but once the agent acquires their first Pokemon, it will likely struggle to reach the next town since very precise inputs are needed to navigate to Viridian City where multiple single tiles need to be hit and many ledges capable of falling off of setting back progress.

fig. 6 also shows that the random algorithm and reward structures that minimally emphasize flags largely begin to plateau progress after 40 million total steps in the environment. Sparse 1 does not experience this plateau suggesting that it may be the most optimal reward structure for training.

C. Agent Results and Progression

Throughout the course of training over 45 million total steps in the environment, agents began to make significant progress in Pokemon Red accomplishing difficult tasks such as navigating to Viridian City, evolving their starter Pokemon, delivering Oak's package, and acquiring the Town Map.

One crucial aspect of Pokemon is battling to level up party Pokemon to keep pace with the difficulty curve of the game. If the player fails to do this, they will likely start losing battles which stalls game progression. The only way to remedy this is to travel back to an area with lower level Pokemon and battle there to gain experience and thus levels. We perceived

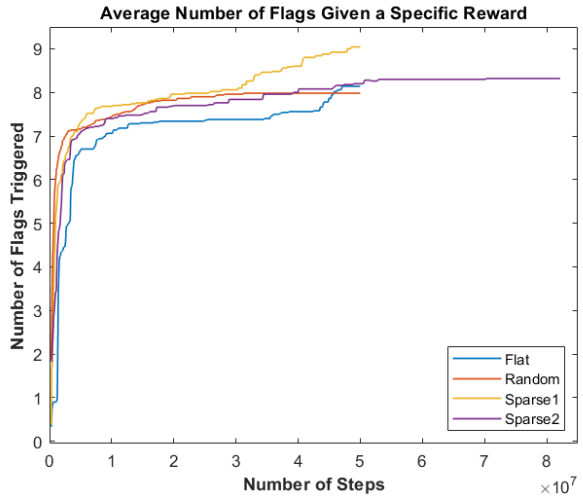


Fig. 6: Average Flag

that this would be difficult for the agents to learn if they had not sufficiently leveled, so the reward structure was altered to consider this. The agents were incentivized to level up their Pokemon given a reward for each level acquired. This structure was chosen so that if the agent was able to reach Brock, it would be more than equipped with the tools necessary to defeat Brock. On average by 40 million total steps, an agent would have a level 20 starter Pokemon which was usually evolved in the middle stages: Ivysaur, Wartortle or Charmeleon as shown in fig. 7.

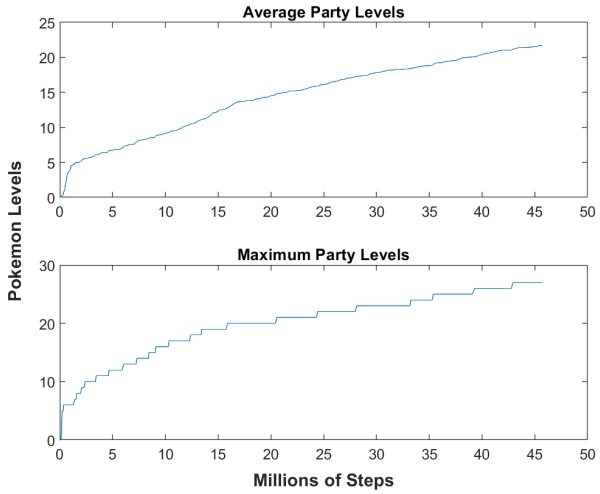
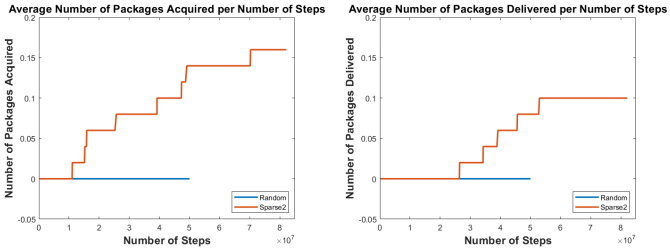


Fig. 7: Pokemon Levels

It is also worth noting that over the entire course of training, no agent ever caught an additional Pokemon despite having the ability to do. This is likely because the state representation has no way to distinguish between where the cursor is in the battle menu. The only information that the agent has access to is that it is in a battle or not. There is a parallel to human nature here in that generally when children first start playing Pokemon they often tend to over-level their starter Pokemon to breeze through the game instead of catching many Pokemon and leveling them all reasonably. Additionally, one agent never acquired their starter Pokemon at any point through the entire training session.

The furthest progress an agent made in the game was delivering Oak's package and then acquiring the town map. fig. 2 shows the starting point of Pallet town and the Pokemart in Viridian where the agent receives the package. The agent must then travel back to Pallet town to deliver the package in order to unlock the next new location, Viridian Forest. fig. 8 shows the average number of agents that acquired and delivered the package for both the PPO algorithm and a random action policy.

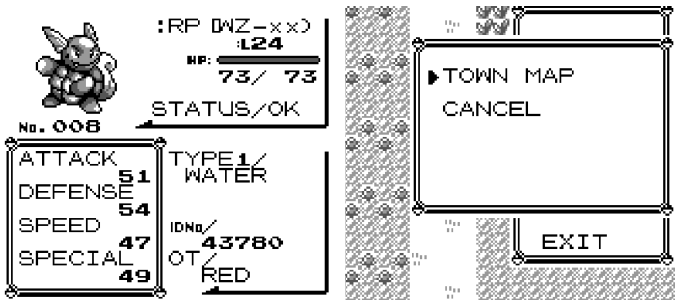
Importantly, no agent operating under a random action policy was ever able to pick up the package from the Virdian City Pokemart thus none were able to deliver it either. Contrarily, nearly 15 percent of agents operating under PPO were able to acquire the package with 10 percent of them delivering it to Professor Oak! Similarly to fig. 5, there is a flurry of activity following the first agent achieving either of these milestones. This is again likely a result of a single model controlling all 50 agents.



(a) package Acquired (b) package Delivered

Fig. 8: Average number of Packages Acquired and Delivered compared to the number of steps in the environment. Blue represents the PPO algorithm (Sparse 2), and Pink represents a random action policy. Note: this flag was unfortunately not saved for the Flat or Sparse 1 trials.

After 50 million steps, the furthest agent had leveled its starter, ":RP DWZ-xx" the Squirtle, high enough to evolve into a Wartortle which was then further further trained to level 24. Images depicting this agents progress are shown below in fig. 9



(a) Wartortle Stats (b) Town Map

Fig. 9: Furthest Progress made by an agent using a sparse reward structure trained for a total of 50 million steps in the environment.

VI. Conclusion

Two reinforcement learning approaches to playing Pokemon Red were presented. The DQN algorithm was able to successfully navigate out of the starting house, however was unable to cope with the game's large state space and need for exploration. A combination of PPO and RND was able to progress through the beginning of the game (as depicted in fig. 2). Over 1 million steps in the environment each, 10% of agents acting under the PPO policy were able to successfully deliver professor Oak's package, compared to none acting under a random policy.

If this work were to be continued, there are many possible avenues for improvement. A distinction between the sub-tasks of navigation and battling could be drawn. This may help focus the agents' exploration efforts. Second, in all the PPO trials, some agents did not manage to collect their starting Pokemon even after 1 million steps. It is likely that more get "stranded" at every progression (as evidenced by the plateau of delivered packages in fig. 8). This problem would need to be investigated and addressed either within the algorithm, or with the possibility of resetting stagnant agents.

Consulting with experts in the Pokemon reinforcement learning community, extensive reward shaping is another avenue for improvement. Using PPO and very specific reward shaping, their agents are able to regularly achieve the 3rd gym badge. Additionally, the results suggested that the Sparse 1 reward structure as show in fig. 6 was capable of continuing progress in the game given more training time. Allowing agents to train for more steps would give a better sense of the true limitations of the current PPO implementation.

The first level goal of getting out of the house in under 100 steps and most of the second goal of using RND to further progression in the game to Viridian Forest was achieved using Deep Q-Learning and PPO respectively. The third level goal seems to be reachable given the current implementation but will likely take a substantially large amount of training.

VII. Contributions and Release

A. Contributions

1) Alec: Developed the Pokemon Red Environment using game RAM, implemented first pass of DQN using full game state with poor results, implemented first pass of PPO with decent results, implemented RND on both DQN and PPO, aided in report writing.

2) Eli: Adapted the DQN algorithm to use different networks for different rooms. Developed the policy visualization and fine-tuned the DQN algorithm to achieve the initial goal of quickly leaving the house. Experimented with the different PPO reward structures. Helped with report.

3) Josh: Helped implement initial DQN. Helped implement preliminary PPO and subsequent application of RND on PPO. Helped with report.

B. Release

The authors grant permission for this report to be posted publicly.

References

[1] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, Feb. 2021. [Online]. Available: https://doi.org/10.1038/s41586-020-03157-9

[2] P. Whidden, "PWhiddy/PokemonRedExperiments," May 2024, original-date: 2019-12-25T07:52:16Z. [Online]. Available: https://github.com/PWhiddy/PokemonRedExperiments

[3] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov, "Exploration by random network distillation," *CoRR*, vol. abs/1810.12894, 2018. [Online]. Available: http://arxiv.org/abs/1810.12894

[4] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, pp. 580–586, 2 2021.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2 2015.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 7 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] "The 37 Implementation Details of Proximal Policy Optimization · The ICLR Blog Track." [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[8] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative a g e n ts."