Reward and Exploration Strategies for Solving Wordle with Deep Reinforcement Learning

December 14, 2022

Bennett Cohen[1a], Sean Tsung[1b], and Isadora White[1c]

[1a,1b] Department of Industrial Engineering & Operations Research (IEOR)

[1c] Department of Computer Science

University of California, Berkeley

https://github.com/icwhite/cs285_final_project

Abstract

We present methods to solve the New York Times' popular word-guessing game *Wordle* using Deep Reinforcement Learning by framing the problem as a Partially Observable Markov Decision Process. We create a scalable and fully customizable Wordle environment and train a Proximal Policy Optimization (PPO) agent from Stable Baselines to solve it. We motivate basic Wordle principles and strategies, before analyzing the impacts of (1) Reward Function Design, (2) Exploration Bonuses, and (3) Environmental Reshaping on a baseline solver.

We begin by analyzing the effects of reward function specification on the performance of a deep learning agent in Wordle. We implement three reward function classes: a "sparse" reward that assigns +1 for a winning guess, and -1 for all other guesses; an "information gain" reward based on the number of new green and yellow letters in a guess; and a "word elimination" function that gives higher reward to words that reduce the number of words that much the pattern of information provided by observations (we call these "possible words"). After comparing these different reward functions on a smaller Wordle environment with 100 valid words (Wordle-100), we find that the elimination reward achieves the best results. We also incorporate a new winning condition that requires an agent to reduce the number of possible words to 1, which drastically improved performance.

After building baseline models in Wordle-100 and full Wordle, we experiment with exploration bonuses to encourage agents to learn the dynamics of the large game better. We implement exploration bonuses and tune the entropy regularization of our model. The exploration bonuses are inspired by count-based bonuses and model-based bonuses on Atari and Gridworld environments. However, we determine that the entropy regularization approach performs better on both Wordle-100 and Wordle.

A critical issue that we experienced while training agents to solve Wordle was that the agent would frequently not guess the words in the set of remaining possible words. To fix this,

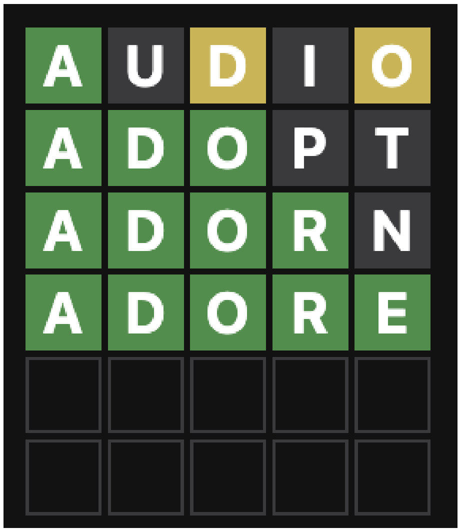
we reshape the environment such that the observation an agent makes is a binary array with the same length as the number of valid words, where 1 represents if a word is possible, and 0 otherwise. Using this reshaping, we achieve 100% win rate on Wordle-100 and improved performance on the environment with Wordle with 500 words. Lastly, we create a two-phase training process that first trains an agent to only guess possible words, before training it to solve the puzzles better. However, on the Wordle environment with 500 words we still only achieve 60% of the words guessed inside the possible words, despite the higher win percentage.

The main contributions of this report are insights into the impact that reward and environment design can have on performance in the game of Wordle and an examination of exploration techniques for Wordle. We conclude our report with a proposal to use Supervised Behavioral Cloning as a pre-training step to ensure an agent only will guess possible words before fine-tuning a model for optimal performance.

Contents

Introduction to Wordle

Wordle is an easy-to-learn, but hard-to-master word-guessing game with a simple premise: players have six attempts to guess a random five-letter word of the day. For each letter in each guess, players are told whether the letter is (1) Not in the word (gray square), (2) In the word but in the wrong spot (yellow square), or (3) In the word and in the correct spot (green square). Consider the sample game below.



A U D I O
A D O P T
A D O R N
A D O R E

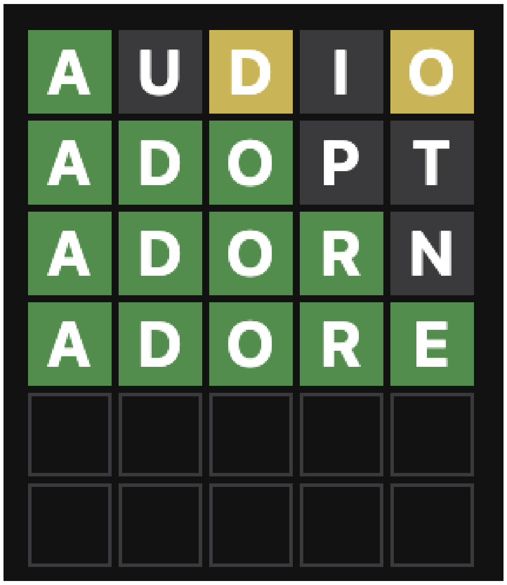
Figure 2.1: An example Wordle game solved in 5 guesses.

From the first guess, we can deduce the answer begins with `A`, contains a `D` but not in the third spot, contains an `O` but not in the last spot, and does not contain `U` or `I`. The next guess gets the correct spots for the `D, O`, and also tells us there is no `P` or `T` in the word. The third guess is almost correct as we get the 4th letter `R`, and we finally win on the 4th guess with `ADORE`.

The English dictionary contains 3,194 5-letter words, however, the original Wordle game uses 2,310 words, excluding uncommon words like DRUIT. Variants of the game change the word length, the number of possible guesses, or the number of boards being played at the same time.

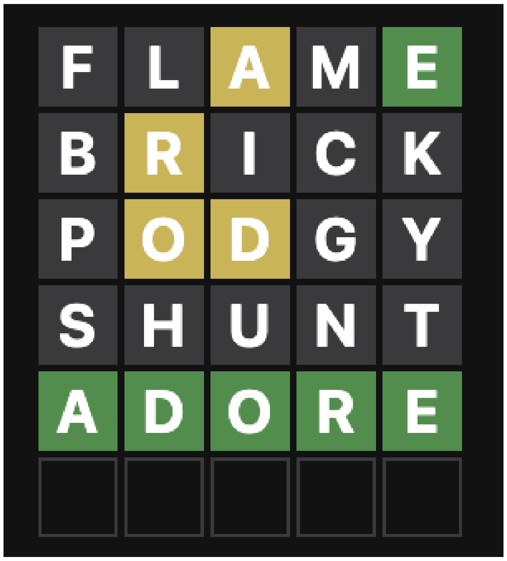
i Wordle Strategies

There are two main classes of Wordle strategies that humans use. Strategy #1 is to guess a word with common letters, and incrementally guess words that follow the pattern of information provided by the grid. This is what was followed above, and is the most common way humans play. The benefit of a solution like this is that when our early guesses are strong, we'll generally get the right answer faster. But, if our first 1–2 guesses are poor, our score will be very high, or we might not get the solution at all. Strategy #2 is to try to eliminate as many letters as possible by making a series of guesses with unique letters to reduce the number of possible words remaining. The benefit is that we can end up with more stable long-term results, but in the case of a common word, we have fewer guesses to actually get it right. Our best-case performance will be lower. We can see the performance of the two approaches below:



A U D I O
A D O P T
A D O R N
A D O R E

Figure 2.2: Strategy #1



F L A M E
B R I C K
P O D G Y
S H U N T
A D O R E

Figure 2.3: Strategy #2

ii Wordle Environment for Deep RL

In order to implement any Deep RL algorithms on this problem, we first must build an environment with which an agent can interact. We build an environment `Wordle` that is compliant with OpenAI Gym for usability and reproducibility. Environments for reinforcement learning are defined by three things: states, actions, and rewards.

The **state** of the environment must contain all the information about the board. Note that each cell of the board actually stores two pieces of information: the guess letter and guess color. To capture this, we have two separate boards of size 6×5 for letters and colors, respectively. We flatten the two boards into a single array of length $2 \times 6 \times 5 = 60$. The letters are encoded by their index in the alphabet $\in \{0, 1, ..., 25\}$, and the colors are encoded such that `green` $\rightarrow 2$, `yellow` $\rightarrow 1$, and `gray` $\rightarrow 0$. Empty cells in both grids are denoted by -1.

The **actions** an agent can take are guessing words. Note that only valid words can be guessed, so the action is the index of the array of $2,310$ valid Wordle words.

The **reward** an agent receives for taking actions must capture information about the board state and guesses remaining. This will be discussed in the next section. An agent has won the game if a guess matches the answer word in six guesses or fewer. A game terminates if an agent has won or if it reaches six guesses without winning.

We added greater flexibility to this environment to allow agents to play games with different word lengths, a different number of guesses, and more boards played at the same time.

iii Performance Metrics

Wordle has a very short episode length, as most games end between three and six time steps (guesses). No game can exceed six guesses. There are a few metrics that we look at when evaluating models.

- **Win Rate**: How many games does it win on average?
- **Average Episode Length**: How many guesses does it need on average to win?

- **Guess Ceiling**: What is the maximum number of guesses it needs to win? For a given set of valid answers and guesses, we can compute a best-case lower bound on this at around five guesses [4].

There is some sense that if the average episode length (guess count) is very low, it will solve more games, but that isn't guaranteed. We prioritize these two metrics over the guess ceiling metric.

Related Works

i Wordle Solvers

Ko [8] employs a deep RL approach to solving Wordle. Ko achieves a 99% win rate with an A2C (Advantage Actor Critic) approach. Ko notes that attempts to use DQN to solve the full-sized Wordle problem were not successful. Ko also observes that for the full Wordle problem, there are 13,000 possible guess words. Instead of directly using an action space consisting of 13,000 possible actions, Ko implemented a neural network approach to handling the state and action representations that means the model only needs to learn 130 outputs instead of learning the full space of 13,000 actions.

Bhambri et al. [2] formulate the Wordle problem as a Partially-Observable Markov Decision Process (POMDP) and solve it with an information gain approach. They start with a sequential maximum information gain (entropy reduction) policy wherein at each guess a sequential attempt is made to maximize the information gain associated with that guess. They augment this heuristic policy by employing a rollout approach which uses maximum information gain as the base heuristic. They were able to achieve close-to-optimal performance that substantially exceeded the base heuristic. They evaluated the performance of their method by comparing the average number of attempts taken to solve the Wordle problem with a given starting word, and compared it to the results from established optimal strategies for playing Wordle.

ii Exploration

Agents can be encouraged to explore a wider variety of actions through the use of an exploration bonus. Simple approaches include count-based methods, wherein an exploration bonus based on $n_t(\mathbf{s})$ (the number of times state $\mathbf{s}$ has been visited at/by time t), such as $1/\sqrt{n_t(\mathbf{s})}$, is added to the original reward [1, 3].

More complex exploration bonuses include the Random Network Distillation (RND) bonus, which takes the error of a neural network which predicts the features of observations from a fixed randomly initialized neural network and uses it as an exploration bonus. In their paper proposing RND, Burda et al. [3] were able to able to build a system with their novel exploration bonus that could play the Atari game Montezuma's Revenge better than the average human. This is significant since Montezuma's Revenge is notorious for being a difficult problem for RL agents where a good/suitable exploration method can make a big difference.

Another strategy is to encourage "intrinsic curiosity" in cases where extrinsic rewards are sparse. Pathak et al. [5] formulate "curiosity" as the error in an agent's ability to predict the consequences of its actions. They formulate an inverse dynamics model which predicts the action taken to get from one observation to another, and jointly optimize it with a forward dynamics model which takes in the current action and state and predicts the next state. The prediction error in the feature space outputted by the forward dynamics model is then used as the intrinsic reward signal.

Building a Baseline

i On Q-Learning versus Policy Gradients

We first confirm and formalize the prior anecdotes about the failures of Deep Q-Learning (DQN) in solving Wordle. Upon each reset of the environment, a new answer is selected, and the dynamics of the game are chosen. For two identical observation vectors, the true answer may be different. In DQN, an agent samples transitions at random from a Replay Buffer, which stores its past experience. This buffer contains transition tuples containing information about the observation, action, reward, and the next observation. However, Wordle is modeled as a Partially Observable Markov Decision Process (POMDP), in that the entire state is not captured by the observation. Specifically, the agent does not have access to the true answer, which is fundamental to understanding the dynamics of any single game. When the agent sample transitions from the replay buffer at random, it is sampling specific *guesses* from completely different games. Any correlation found between observation and reward is dependent on the game being played, due to the game dynamics. Thus, a DQN agent would overwrite what it has learned over and over again as the Q-values would change too much with each sampling.

ii Reward Function Design

Wordle has many strategies, many variants, and many ways to gauge performance. As such, we have worked through multiple types of reward signals to see what yields the best performance overall. Because the dynamics of the game change with each reset, it can be difficult to grasp how reward functions change behavior. To illustrate this, we show how each reward varies for three different strategies in the same exact game. The answer for this game is `ADORE`.

2.1 Sparse -1/1 Reward

Our first reward function specification is a sparse -1/1 reward. For each guess, the agent receives a reward of $+1$ if the guess is correct, and a reward of -1 if it is incorrect. The goal is to encourage an agent to win games quickly. We define won_t as a binary indicator variable for if the game has been won with some guess t. Formally, the -1/1 reward is written as follows:

$$R(t) = \begin{cases} +1 & \text{if} \quad \text{won}_t \\ -1 & \text{if} \quad \text{otherwise} \end{cases} \tag{4.1}$$

In games with $\text{n\_words} \ll 2,310$, this reward does work, mostly because, given enough iterations, there is a learnable pattern between the board and the possible words within the allotted guesses. However, in non-trivial examples, this fails to work. Intuitively, it rewards a guess with four green letters the same as all gray letters. If an agent hopes to converge towards all five green letters, this is not a strong formulation.

The plot below shows two different action trajectories for the strategies outlined earlier, along with a trajectory for random action sampling. We see the non-cumulative reward at each guess/timestep. Recall that Strategy #1 follows the pattern of information provided by the environment, and Strategy #2 seeks to eliminate unique letters. This is the same game as shown in Figures 2.2 and 2.3.

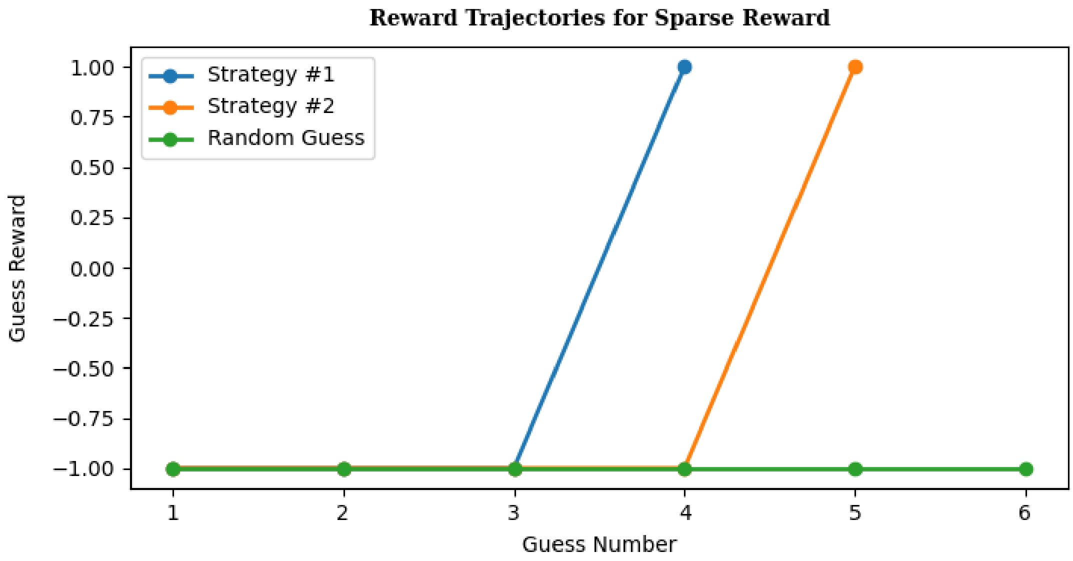


Figure 4.1: Reward trajectories for a sparse reward function.

The issue of sparsity described above is underscored here. Both of the two strategies are clearly better than randomly guessing words in hopes of getting the answer. However, until it gets the correct answer at guesses three and four respectively, the reward signal deems the agent's actions no better than random guesses.

2.2 Information Gain Reward

The shortcomings of a sparse reward function lend themselves to using a more dense, information-driven reward function. We define $N_{c,t}$ to be the number of letters of color c guessed at guess t, $c \in C = \{\text{green, yellow, gray}\}$, and $n_{c,t}$ to be the number of *new* letters of color c guessed at guess t (i.e. have not been discovered to be of that color yet). We apply a 10-point penalty if the game is not won at guess t. We can define our quasi-information gain reward function now:

$$R(t) = 2 \times n_{\text{green},t} + n_{\text{yellow},t} - (\sum_{c \in C} N_{c,t}) - 10 \times (1 - \text{won}_t) \tag{4.2}$$

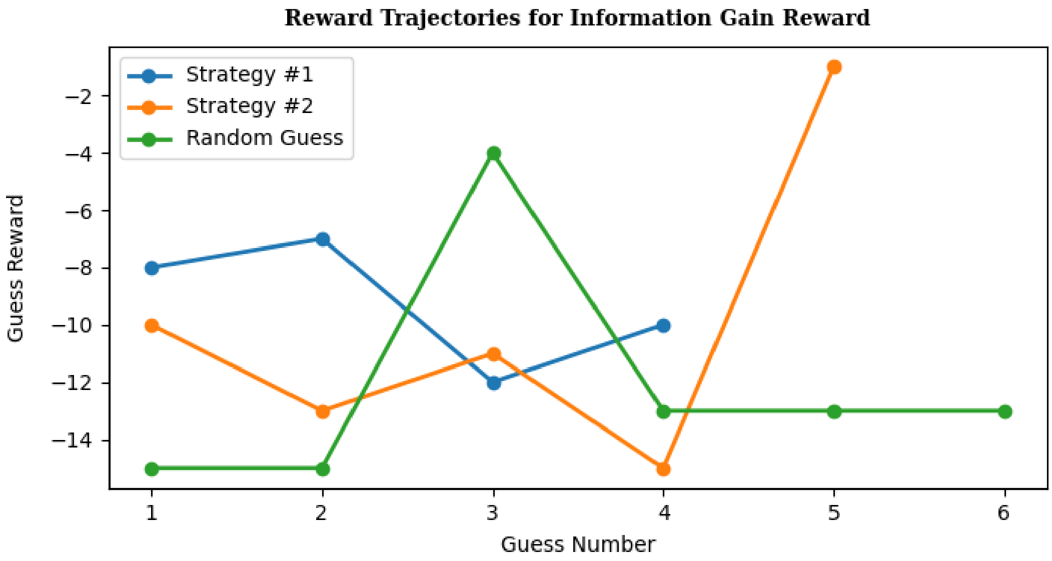


Figure 4.2: Reward trajectories for an information gain reward function.

The results above are quite intuitive. Each guess of Strategy #1 is highly correlated to the previous guess, which means the new information provided by each new guess must decrease. Strategy #2 has uncorrelated guesses, so there is more opportunity for new information, although it doesn't always get it (because it is closer to a random guess than #1). At its last guess, Strategy #2 skyrockets as it puts all the information it has learned together to get a single guess.

2.3 Word Elimination Reward

Let's return to the two human strategies from earlier. At their core, they are approaching the problem from opposite sides. The first tries to add letters to construct a pattern, whereas the second tries to check as many letters as possible. In the end, they both can be seen as constructing patterns that a good guess must follow. They each systematically shrink the list of guesses that follow a pattern. In the earlier stages of a game, there are many words that satisfy the pattern, so a human picks one at random from the words that come to mind. However, in the end, there are likely only a few valid words. Similarly, a good agent would reduce this number of *possible words* (i.e. words that follow the pattern of information from the colors) as fast as possible. We should reward guesses that sufficiently reduce this search space, but don't want to enforce a method under which this reduction should be done. We still also include a penalty for not winning to encourage shorter games. This reward function is defined as follows with P_t representing the set of possible words after a guess t.

$$R(t) = \frac{\text{size}(P_{t-1}) - \text{size}(P_t)}{\text{size}(P_{t-1})} + \text{win}_t - (1 - \text{win}_t) \tag{4.3}$$

The plot below shows the reward trajectory for the strategies and random guessing. We see that Strategy #1 makes a much larger reduction of possible words earlier in the game but slows down after. Strategy #2 takes a different approach and slowly improves its reductions.

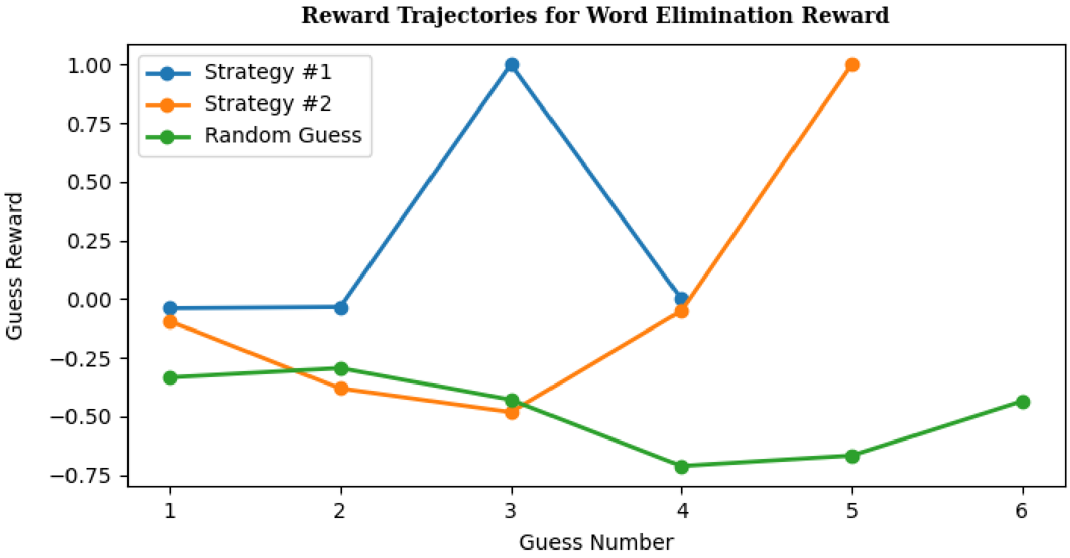


Figure 4.3: Reward trajectories for a word elimination reward function.

We can see these trajectories more below, where we track the number of possible words remaining after each guess in this easy game. Strategy #2 takes a slightly slower approach to convergence (defined as 1 possible word remaining), with more stable step sizes.

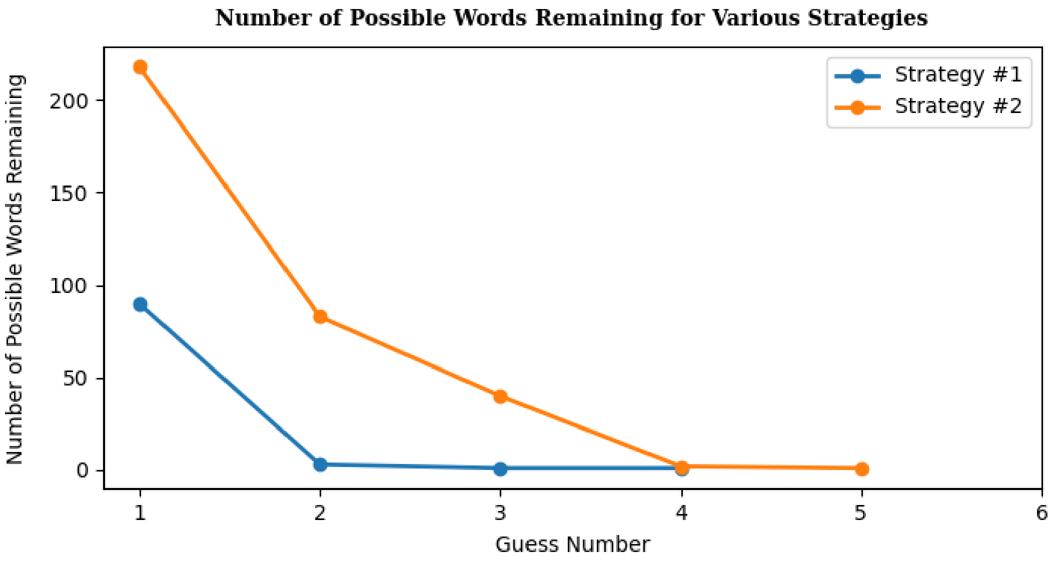


Figure 4.4: Number of words remaining with basic Wordle strategies.

iii Baseline Model Results

Let's first consider how Strategy #2 performs as it is a non-RL solution. Each game will begin with the same four guesses: `FLAME, BRICK, PODGY, SHUNT`. For guesses #5 and #6, we'll select a word from P_t at random.

3.1 Wordle-100 Results

We first start on a smaller game that has 100 possible guess words and answers, Wordle-100. In this game, we investigated baseline model performance when using different reward functions. Our initial word elimination reward function includes a penalty for each incorrect guess. We tried using penalties of size -1 and -10 and found that -10 is too large, severely worsening performance. We also introduce a new bonus if the agent wins and a penalty if they lose (at the end of the episode). We tested the same size bonuses and penalties as with the individual guesses and found +1 and -1 for both led to better performance. The model with +1 for wins and -1 for incorrect guesses achieved the highest win rate after 1 million timesteps.

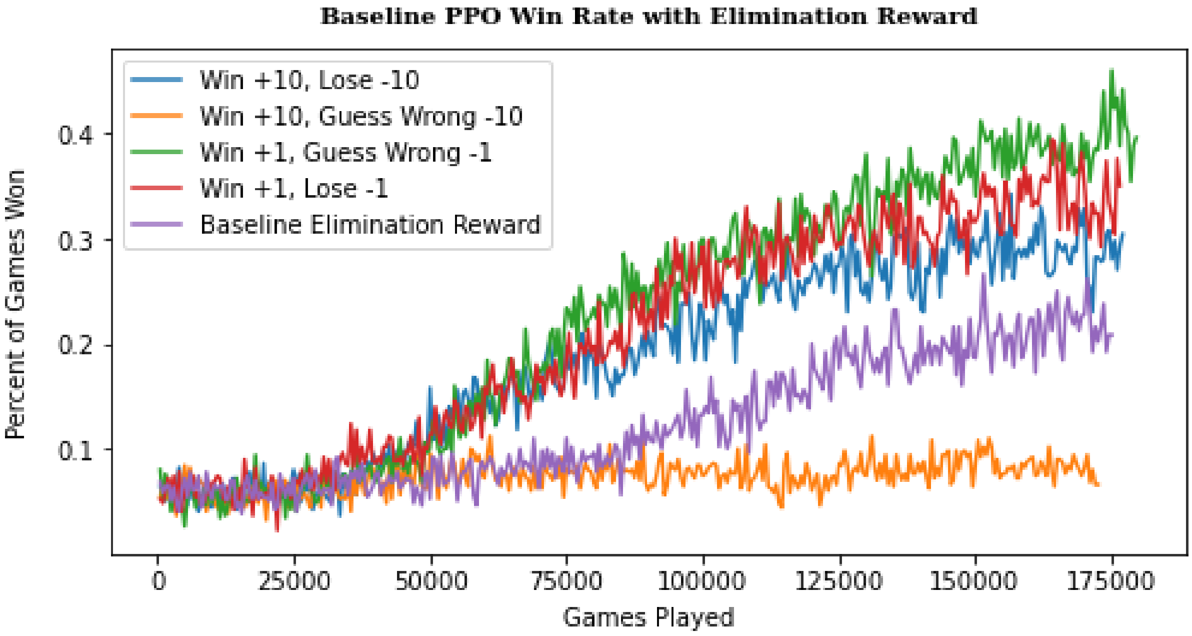


Figure 4.5: Baseline PPO Win Rate with Elimination Reward

Taking a closer look at the baseline PPO results with +1 for wins and -1 for incorrect guesses, we see that the average reward is increasing, whilst the average number of guesses is decreasing. The relevant graphs are below:

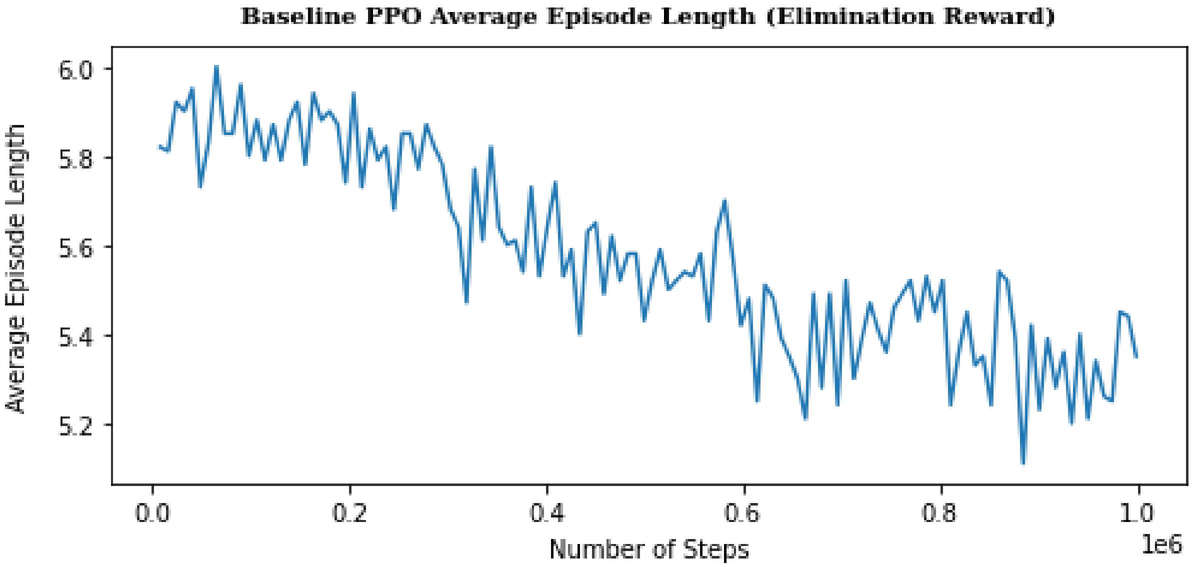


Figure 4.6: Baseline PPO Episode Length with Elimination Reward

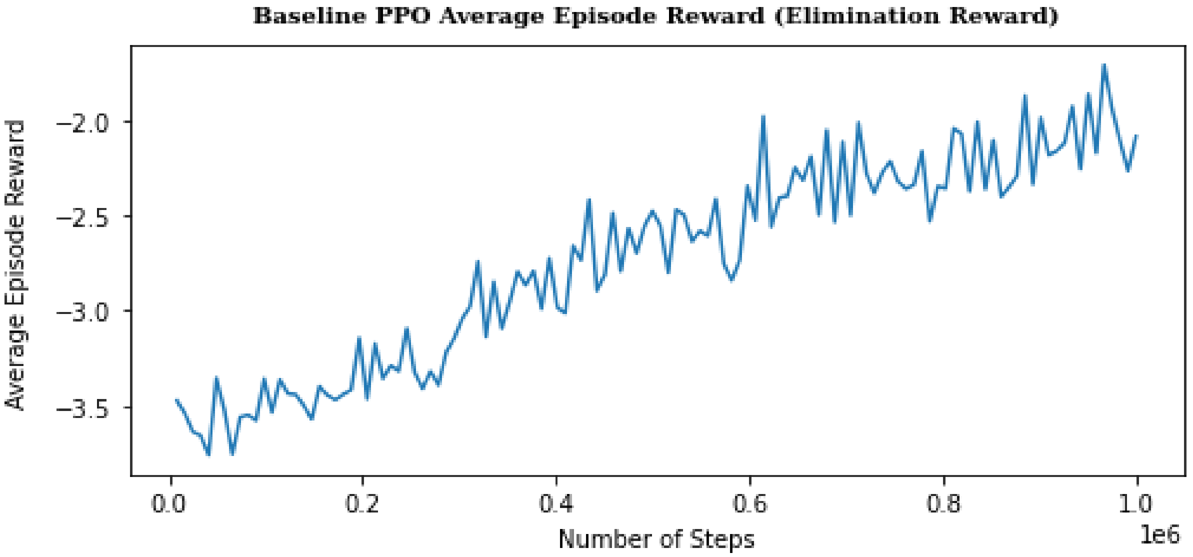


Figure 4.7: Baseline PPO Mean Reward with Elimination Reward

We also tested the model with sparse rewards and information gain rewards, where the information gain reward also had a -1 penalty for incorrect guesses. We find that the elimination reward with the aforementioned bonus and penalty (+1, -1) for wins and incorrect guesses respectively outperforms both sparse and info gain rewards for Wordle-100.

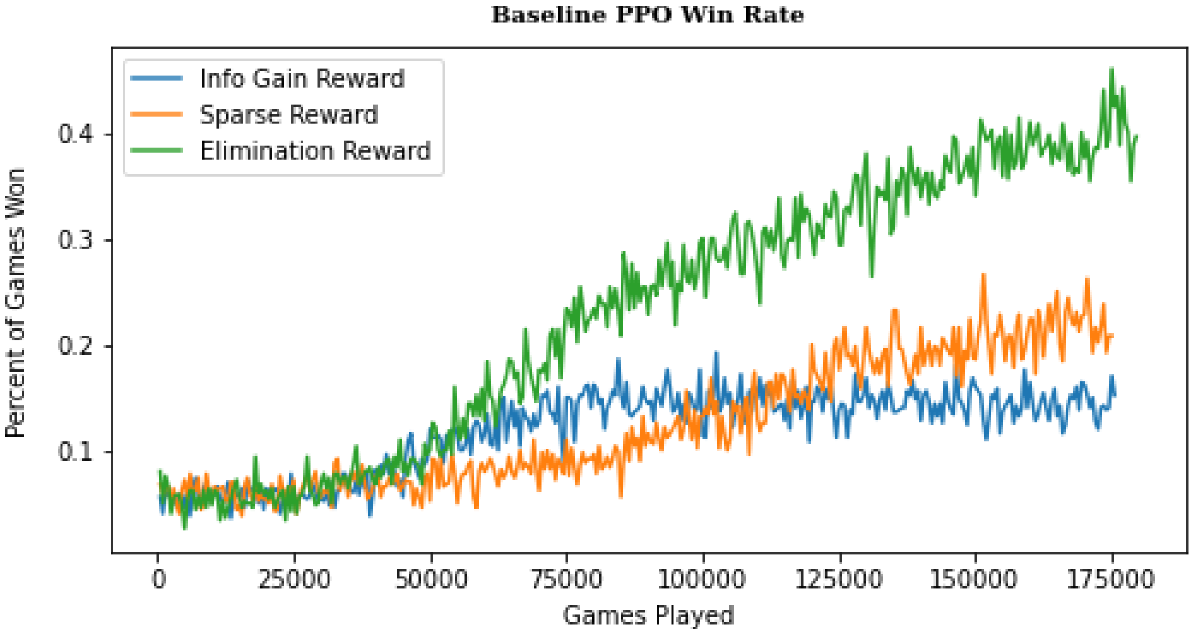


Figure 4.8: Baseline PPO Win Rate for Different Rewards

iv Full Wordle Results

At first, we tried to make the problem of Wordle simpler by shrinking the environment. Another way to simplify the problem is to change the win condition for Wordle. Instead of guessing the word explicitly, we set the win condition to be if the agent successfully eliminates all possible words, such that size$(P_t) = 1$. Reformulating the problem in this way makes winning at full Wordle much more achievable. Even a random policy will have a reasonable win rate of 70% with this win condition. This modifies the problem paradigm to be much more similar to Strategy #2 described in the introduction section. We refer to this as the **elimination win condition** for the rest of the report, as we win by eliminating words instead of explicitly guessing the word.

We apply the same structure of rewards, win bonuses and incorrect guess penalties that were found optimal in Wordle-100. After 1 million time steps, our agent had an average episode length of 3.3. This means the agent could reduce size(P_t) = 1. in 3.3 guesses on average. Because the agent has to take the final guess, our average guess count is 4.3 guesses.

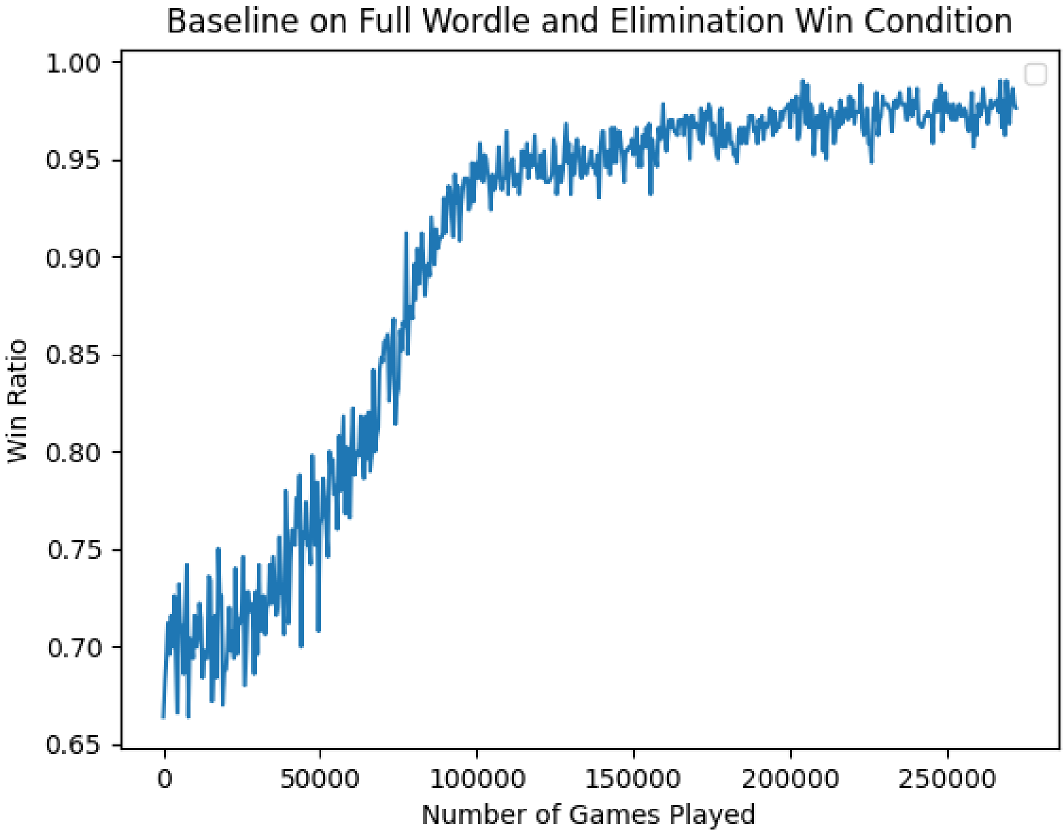


Figure 4.9: PPO and Elimination Reward Win Rate for Full Wordle

Exploration Methods for Wordle

When analyzing the baseline models in various games to understand its learned strategy, we notice the agent begins with the word `SPAWN`. This is not considered one of the best starting words for Wordle, as it contains letters that are not likely to be included in the answer word, such as `W`. Most common starting words contain many different vowels, such as `AUDIO`. We hypothesize that the model is converging too quickly to the choice of `SPAWN` as the first word and may achieve better performance if exploration is introduced.

We introduce Wordle-specific exploration bonuses inspired by the ideas from count-based exploration [1], model-prediction-based exploration [5] and RND [3]. In addition, we experiment with an entropy regularizer to increase exploration.

i Exploration Bonuses

1.1 Word Count Based Approach

This approach is inspired by the count-based methods discussed in [1]. Throughout the training process, we track how many times the agent has guessed a certain word, and store that count as n. For each guess of that word, we add a Bonus $= \frac{1}{\sqrt{n}}$ to the reward.

This bonus was created to address the concern that the agent might never guess certain words. This would make it unable to learn which letters are in those words and how that word could play a crucial role in winning the game of Wordle.

1.2 Dynamics Model Prediction Error

We construct a dynamics model of Wordle that will predict whether each of the letters of a guess will turn yellow, green, or gray. We pass in the board state containing all of the letters guessed so far, and the colors of all the previous guesses. As the agent makes guesses, we train the dynamics model. The error in the model's prediction becomes the exploration bonus for the model.

The motivation for the exploration follows that of RND [3]. For board states or trajectories that an agent has experienced less often, there is higher uncertainty in the dynamics model, which leads to higher prediction error, thereby giving a higher exploration bonus. The hope is that this bonus will guide the agent to take a large variety of trajectories and try out a variety of strategies.

Additionally, for words like `LEAST`, there is higher entropy for each of the letters because each of the letters occurs more frequently in the set of possible Wordle answers. In theory, the higher entropy for the letters in the world will lead to a higher exploration bonus for high-frequency letters. We contrast this with words like `WRYLY`, which contain many uncommon letters and are more likely to be gray on average. This would mean that the model should have low prediction errors on words like these.

However, due to the stochasticity and the black-box nature of the dynamics model, it is hard to be certain which exploration bonuses are given in which states. It is possible that although the model's loss decreases through training, there are outliers and many state trajectories that confuse the agent.

1.3 Difference of Words

Another aspect of exploration in Wordle is to encourage the agent to take actions within a game that differ from each other. For Wordle, this is simply guessing words with different, unique letters. This is precisely what Strategy #2 explicitly does, as it guesses 20 of the 26 letters of the English alphabet. We formulate this exploration bonus as

$$\text{Bonus} = \frac{\text{unique letters guessed}}{\text{total letters guessed}}$$

1.4 Exploration Weight

As in the RND paper[3], we want to control the amount to which the exploration bonus is utilized during training by taking advantage of exploration weights. We use the following formulation

$$\text{reward} = w_{\text{explore}} \times [\text{exploration bonus}] + (1 - w_{\text{explore}}) \times [\text{environment reward}]$$

1.5 Normalization of Bonuses

So far we have described three exploration bonus strategies. Firstly, a strategy based on the number of times a word has been guessed, a model prediction error approach, and an approach based on the difference of the words within a rollout. The word count approach will decrease in scale at a rate of $\frac{1}{\sqrt{n}}$. The model prediction error approach has an unknown scale and also decreases at an unknown rate, while the difference of words method will decrease within each game, but not for each rollout.

Therefore it is necessary to create normalization for each of these bonuses. To implement this we keep track of the most recent 500 bonuses computed and use the mean and standard deviation from this buffer to normalize.

ii Entropy Regularization as Exploration

For this project, we use the Stable Baselines implementation of Proximal Policy Optimization [7] [6], which is a variant of the actor-critic algorithm utilizing multiple workers and a trust region. A hyperparameter included in the PPO implementation we used is the entropy coefficient. The entropy coefficient works as a regularizer on the cost function encouraging a probability distribution with greater entropy. The loss term of PPO is formulated as

$$\text{Loss} = [\text{policy gradient loss}] - [\text{entropy}] \times [\text{entropy coefficient}] + [\text{value coefficient}] \times [\text{value loss}]$$

Entropy is a measure of randomness or uncertainty over a probability distribution. The closer a distribution is to uniform, the higher the entropy. Therefore, this loss regularization incentivizes the actor to output a probability distribution that is closer to uniform. A probability distribution that is more uniform is more likely to try out a wider variety of actions, instead of only having one or two actions taken with high probability. In this sense, increasing the entropy of the output probability distribution encourages exploration, because the agent is more likely to visit a large variety of states.

The recommended range for the entropy coefficient of PPO is in the range of $[0, 0.01]$ and we chose a value of 0.001 because this seemed to lead to better results. In addition to the entropy coefficient, it is also possible to introduce an entropy bonus to the PPO reward. However, we did not find the work involving exploration bonuses and Wordle promising enough to consider this.

iii Results and Analysis

3.1 Exploration Bonus on Wordle-100

We run these bonuses using unsupervised exploration, such that for the first 20k iterations, the reward was simply the exploration bonus (exploration weight = 1, exploitation weight = 0). Over the next 20k iterations, we applied a linear interpolation of the exploration weight to 0. Training resumed as normal after 40k iterations.

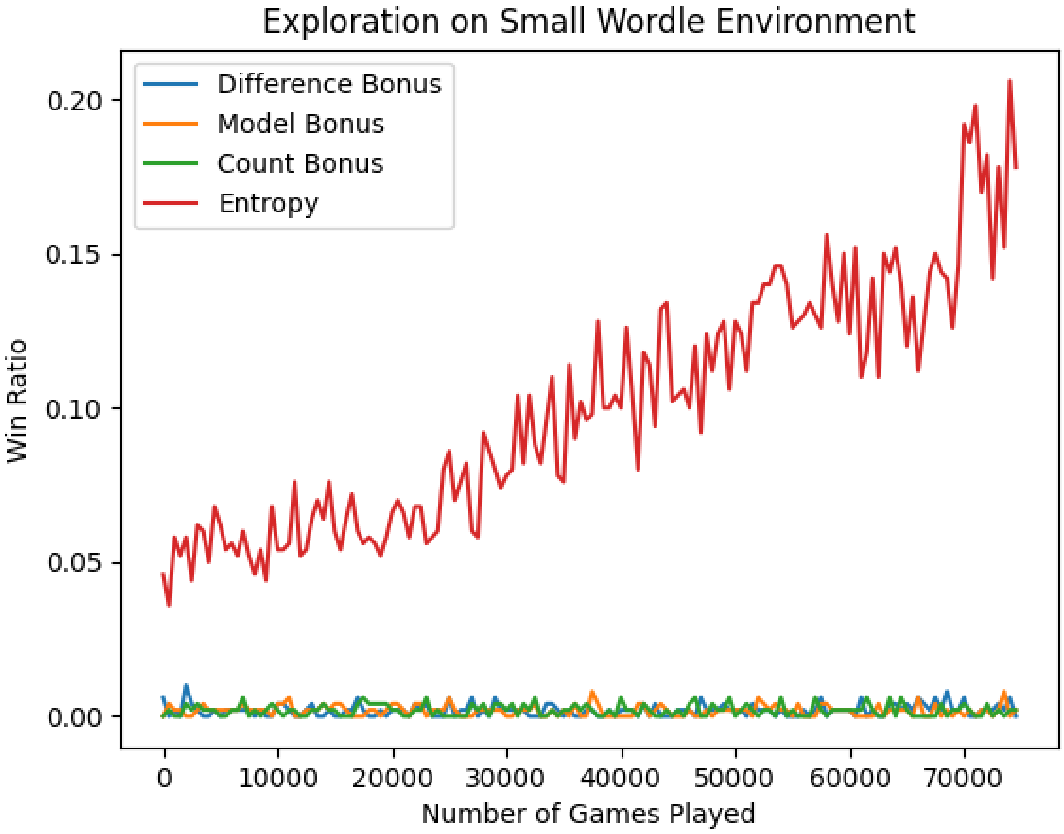


Figure 5.1: Exploration on Wordle-100 with true win condition.

On Wordle-100, the entropy regularization performed much better than the exploration bonuses. However, entropy regularization did not perform as well as the baseline.

On the other hand, the exploration bonuses performed significantly worse. It is possible that the agent converged to a suboptimal policy during the unsupervised learning exploration phase causing the catastrophic performance in these runs. It could be that the Wordle-100 environment is too small to benefit from exploration and increased randomness in the policy.

3.2 Exploration Bonus on Full Wordle

As in Wordle-100, we trained using the same unsupervised exploration, followed by linear interpolation until we reached a final exploration weight of 0.1. We hypothesize that maintaining the exploration bonus throughout the training process would be beneficial because there are many more words to choose from, leading to an exponentially larger state space to explore.

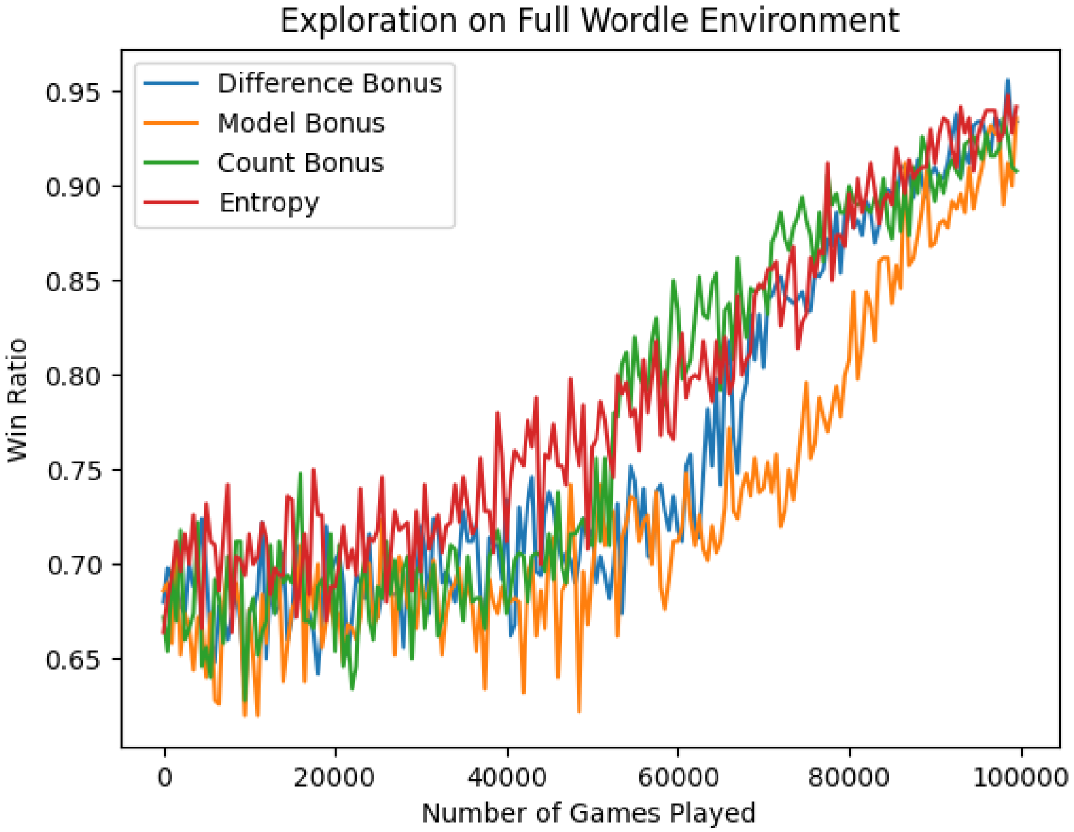


Figure 5.2: Exploration on Full Wordle with elimination win condition.

After ≈100k games played by all the agents, we see that the bonuses and entropy-based approaches have about the same win ratio. However, the entropy approach and the count-based bonus began to see more improvements more quickly at around 50k games played. The model-based prediction error lagged behind the other agents until about 80k games were played, and then rapidly improved.

It is possible that the exploration bonuses were confusing the agent on the reward signal. In the construction of the model-based and difference-based bonuses, we tried to create bonuses that would encourage simultaneously exploring the state space of Wordle, while also providing some incentive that could help win the game. However, in whatever respect these bonuses encouraged Wordle victories, it was not as good of a signal as the elimination reward constructed in the previous section. The gains based on trying out a larger variety of strategies were limited.

We ran the best exploration method for 500k more iterations to create a more complete comparison with the baseline. The entropy regularization achieved a 1% increase in win ratio (up to 98%) and a 0.1 episode length reduction (down to 3.3). However, the win rates still fluctuate wildly, so this 1% difference is almost irrelevant.

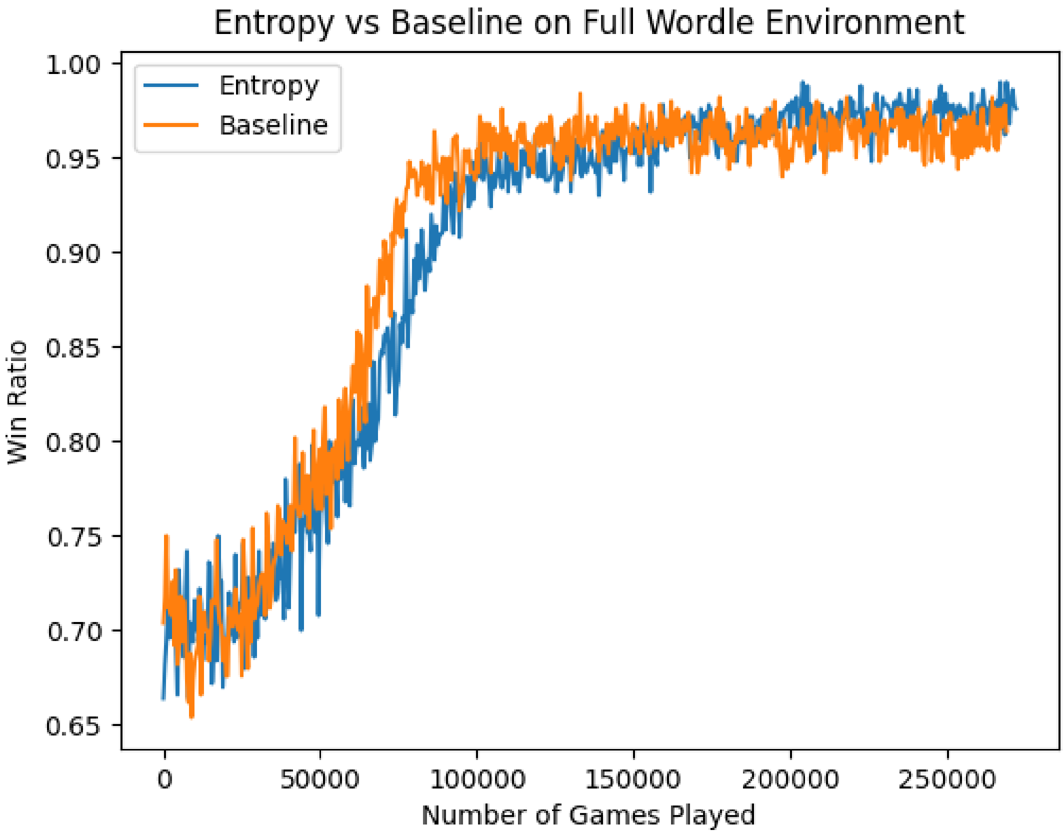


Figure 5.3: Comparison of entropy regularization exploration approach to the baseline on full Wordle with the elimination win condition.

Based on the results, we conclude that the entropy regularization strategy is the best for the Wordle problem. We see that entropy regularization did provide a small performance boost to the Full Wordle environment with elimination win condition, however, the improvement might not be statistically significant.

Performance Improvement via Environment Reshaping

Our best experiments include a reward function that penalizes more guesses and rewards reducing the number of words that match a given pattern. However, an agent still must learn the mapping between states and possible words P_t. In a classic Wordle game, each row has $2,310$ valid words × 3^5 color possibilities $= 561,330$ total possibilities. In order to fill in a row, all previous rows must be filled in (i.e. cannot make guess #3 until guess #2 and guess #1 have been made). A rough count of the total possible states is then $\sum_{i=1}^{6} 561,330^i = 3.13 \times 10^{34}$ states, excluding the overlaps of repeat colors and such.

Now, we introduce a more explicit state representation as a binary array corresponding to each of the valid words. Each element takes on a value of 1 if that word $\in P_t$, and 0 otherwise. For each guess, we see that the number of nonzero elements of P_t must be $\leq$ the number of nonzero elements in P_{t-1} (i.e. we can never increase the number of possible words). While this doesn't explicitly decrease the total state space possibilities because 2^{2310} is much larger than the old size, we hypothesize that this drastically reduces the complexity of the mapping between states and high-reward actions.

i Restricting Valid Guesses via Reward

In an ideal scenario, we could explicitly restrict an agent from guessing words $\notin P_t$. While it likely is possible to manufacture an environment to do that, it would defeat many exploration attempts. Instead, we opt to take a two-phase training approach, similar to unsupervised exploration: In Phase I, we train an agent with a reward function designed to encourage an agent to guess words $\in P_t$.

$$R(t) = \begin{cases} +1 & \text{if word } \in P_t \\ -1 & \text{otherwise} \end{cases} \tag{6.1}$$

Once the agent has learned a policy that guesses a sufficient proportion of words in P_t—which is chosen such that it knows the importance of this while still leaving room for exploration—we train a Phase II agent with the same reward structure as earlier. **Note:** this new structure actually enforces a true win instead of the word elimination win condition, so the episode length = average guess count.

First, we try this new structure on Wordle-100. It learned the correct behavior incredibly quickly and had a 100% win rate just after Phase I. After Phase II, the agent had an average episode length of 3.05, which is incredibly strong. While these results were exciting, they likely just highlighted issues of scalability. With only 100 randomly selected words, the set of possible words after just one guess, P_1 becomes incredibly small, so it is much easier for an agent to win.

We move to Wordle-500, which yields predictably less hopeful results. After 1 million timesteps of training in Phase I, the agent fluctuates around 60% of words within P_t, which is an improvement from earlier, but not so high that it was winning sufficient games. After Phase I, the agent was winning around 30% of games, and after Phase II, it could win around 65% of games. Again, these are true wins—not the word elimination condition wins as earlier.

The sample game below shows a real game this agent played. With its first guess, it guesses three vowels, which is a very good strategy for word elimination. However, with its second guess, it doesn't actually guess a valid word because a valid word cannot contain `V` or `E`. It finally gets to the answer `SAUNA` within four guesses—not bad!

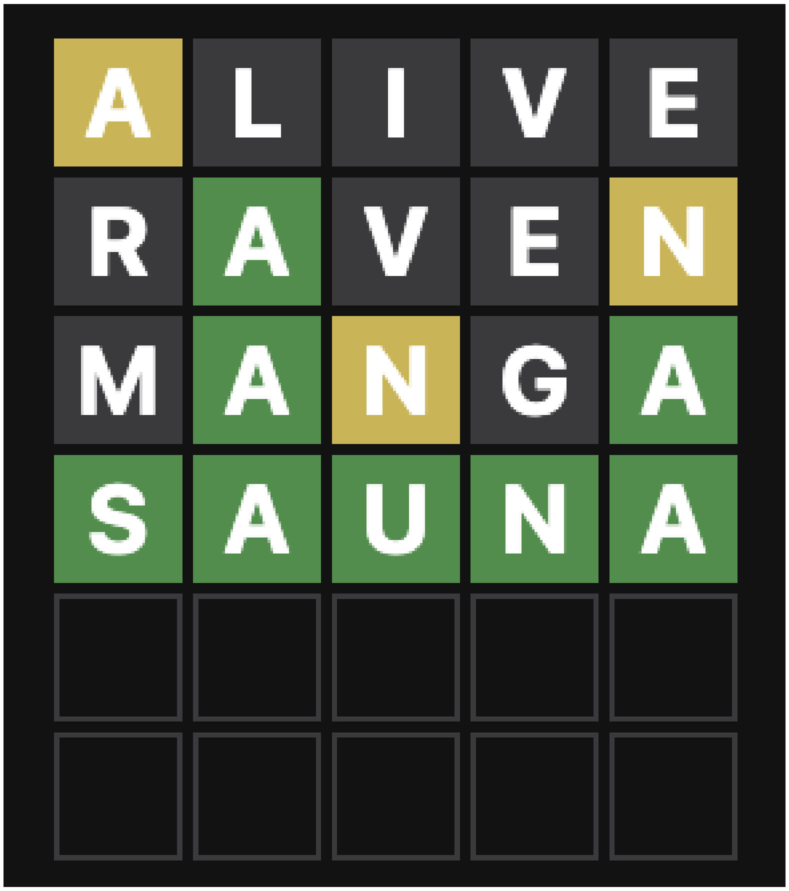


Figure 6.1: Sample gameplay from a two-phase training process

With this new state representation, we were able to encourage guessing within P_t, but could not yield the same performance in the real game as the word elimination win condition. However, knowing that we can use RL in a larger system, the modified condition can still be useful so that we can play until size$(P_t) = 1$. If we have guesses remaining, we will just play the remaining word in P_t.

ii Extension: Back to Basics with Behavior Cloning

Throughout this process of changing environments, states, and rewards, we've learned the importance of enforcing that guesses fall within the set of the possible words, P_t. However, we haven't answered the question of just how important it is. To illustrate this, we build a simple heuristic agent will the following policy π at some timestep t:

Algorithm 1 Randomized Greedy Search Heuristic

```
$t = 0$
$P_t$ = all valid words
while not DONE do
    if $t = 0$ then
        $\pi(t) =$ AUDIO
    else
        $\pi(t) = x \in_R P_t$ (randomly sample from $P_t$)
    end if
    $t = t + 1$, Compute new $P_t$
    if $t = 6$ or guess = answer then
        DONE = True
    end if
end while
```

This agent always starts with the word `AUDIO` and then selects a random word from the set of possible words until it wins or the game is over. An agent with this simple policy can solve 99.8% of games, with an average episode length of 4.04 guesses. Does this mean reinforcement learning is not needed for the problem? Well, not quite.

We propose a different approach to the task. Instead of using reward penalties to encourage an agent to guess words within P_t, we instead can quickly generate an *expert policy* from our greedy heuristic algorithm for all possible game states. We then can train a Behavioral Cloning (BC) model to learn this expert policy in place of our Phase I training. The key opportunity area lies in that the agent greedily selects an action at random. However, given that an agent knows the value of each action, we can tune the discounting factor in a model to implicitly analyze next-step rewards on each possible guess such that it can make more informed guesses. This change reduces the size of the task the agent has to learn, and we believe will allow it to converge closer to the theoretical minimum of 3.0 guesses per game.

Given the timeframe of this project and GPU usage, we weren't able to perform pre-training using BC to fine-tune a model for optimizing guesses within P_t. With future work, we hope it can help become a state-of-the-art Wordle solver.

Results & Discussion

To summarize, upon recognizing and verifying that DQN is not the best method for the Wordle problem setup, we began by building a baseline based on Proximal Policy Optimization (PPO) and testing different reward function designs. We found that our baseline PPO with a reward function based on word elimination, with bonuses and penalties for wins and incorrect guesses respectively, yielded the best baseline true win rate on Wordle-100, with a 40% true win rate after roughly 175k games. After recording these preliminary results, we shifted to full Wordle and introduced the elimination win condition, and achieved a win rate of ≈97% after around 100,000 games.

We then tested different exploration methods; a difference bonus, a dynamics model bonus, a count bonus, and an entropy-based method. We found that the exploration methods did not perform very well on Wordle-100, with only the exploration bonus showing an appreciable increase in true win rate with the number of games played, but still performing worse than the baseline policy. We surmised that the Wordle-100 problem might have been too small to benefit from our exploration methods. Indeed, we saw much more promising results on the full Wordle environment, with all the exploration methods tested tested reaching a 95% elimination win rate by around 100,000 games played, but with the entropy and count-based approach starting to improve earlier. With an additional 500K iterations, we found that we could reach a 98% win rate with the entropy method with average episode length of 3.3, as compared to our baseline results of 97% and 3.3, respectively. Taking into account the degree of fluctuations we see in the win rate even towards the end of the run, we do not believe this to be a significant difference. It is interesting to note that though it might appear at first glance that our exploration and baseline win rates are comparable to the 99% win rate achieved by Ko [8], in truth these figures are not directly comparable since Ko considered a much larger set of possible guess words (13,000) and used the true win condition. In any case, our results are quite promising on the full Wordle environment.

Finally, we attempted performance improvement via environment reshaping, with the idea being that we might be able to reduce the size of the state space and better encourage the agent to choose words that are in the set of possible words remaining by using a state representation that explicitly encodes this set at each step. We paired our new state space with a two-phase training approach with a different reward function in each phase, the first phase being designed to encourage guesses within the set of possible words. This new approach yielded a true win rate of close to 100% just after phase I for Wordle-100, and 30% after Phase I and around 65% after both phases for Wordle-500.

Conclusion and Extensions

In conclusion, through this project we have learned that the choices involved in designing an environment are critical for the performance of the agent. The choice of the elimination reward function over the original sparse reward function merited an increase in win rate of over 20% on Wordle-100. In addition, the choice to reshape the observation space resulted in a 100% win rate on the same environment during the unsupervised pre-training phase. With this new observation space, the Wordle-500 environment was tractable to approach. It is possible that there are even better ways to reshape our environment which could lead to better results. For example, we could combine the reshaped observation space with our original observation space containing the letters and colors to create an observation space with even more information.

Furthermore, through the course of our experiments, we have learned how important it is to guess words within the set of possible words. In our experiments, we attempted to pre-train the agent to guess words only in the possible words. However, on Wordle-500 the pre-trained agent only guessed around 60% of the words within the possible words. We hypothesize that we could use behavior cloning as a pre-training step so that the agent could learn to guess words only in the possible words.

Bibliography

[1] Marc Bellemare et al. “Unifying count-based exploration and intrinsic motivation”. In: *Advances in neural information processing systems* 29 (2016).

[2] Siddhant Bhambri, Amrita Bhattacharjee, and Dimitri Bertsekas. “Reinforcement Learning Methods for Wordle: A POMDP/Adaptive Control Approach”. In: *arXiv preprint arXiv:2211.10298* (2022).

[3] Yuri Burda et al. “Exploration by random network distillation”. In: *arXiv preprint arXiv:1810.12894* (2018).

[4] *Establishing the minimum number of guesses needed to (always) win Wordle*. `https://alexpeattie.com/blog/establishing-minimum-guesses-wordle/`. 2022.

[5] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.

[6] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[7] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[8] *Wordle Solving with Deep Reinforcement Learning*. `https://andrewkho.github.io/wordle-solver/`. 2022.