

All Over The Wordle

Finding The Optimal Wordle Agent with Artificial Intelligence

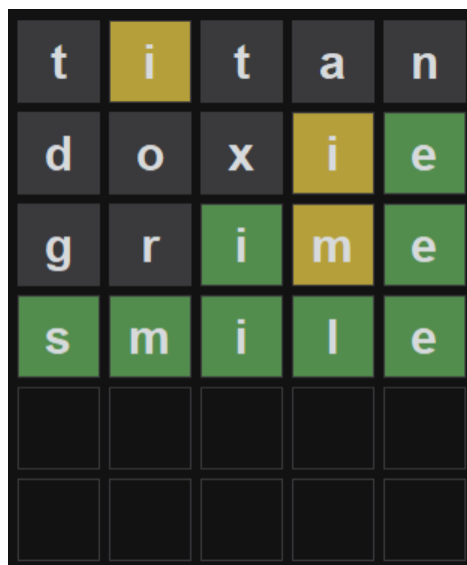
Yoav Orenbach
yoav.orenbach@mail.huji.ac.il

Adi Rabinovitz
adi.rabinovitz@mail.huji.ac.il

Dvir Amar
dvir.amar@mail.huji.ac.il

Amit Keinan
amit.keinan2@mail.huji.ac.il

67842 Introduction to Artificial Intelligence - Spring 2022



Contents

1	Introduction	3
2	Methods	4
2.1	The Games	4
2.1.1	Wordle	4
2.1.2	Absurdle	5
2.1.3	Noisy Wordle	5
2.1.4	Yellow Wordle	6
2.1.5	Vocabulary Wordle	6
2.2	The Algorithms	6
2.2.1	Algorithm’s Optimizations	6
2.2.2	Random Baselines	7
2.2.3	Adversarial Algorithms	7
2.2.4	Entropy	8
2.2.5	Reinforcement Learning	9
3	Analysis and Results	10
3.1	Evaluation Method	10
3.2	Wordle Results	11
3.3	Absurdle Results	13
3.4	Noisy Wordle Results	14
3.5	Yellow Wordle Results	15
3.6	Fake Vocabulary Wordle Results	16
3.7	Real Vocabulary Wordle results	17
4	Conclusions	19
A	Agents Optimization	20
A.1	Minimax	20
A.2	Entropy	20
A.3	Q-learning	22
A.3.1	Learning methods	22
A.3.2	Training method	23
B	Best Opener	24
C	How to Run the Code	24

Abstract

Wordle is a web-based word game invented in 2021. In this paper, we explore the performance of several artificial intelligence algorithms on several variations of Wordle. This comparison teaches us about the strengths and weaknesses of these algorithms and the nature of the different games. The algorithms that we explored are random baselines, adversarial algorithms - Minimax and Expctimax, Entropy, and Q-learning. We compared them on the following Wordle versions: Wordle, Absurdle, “Vocabulary Wordle”, “Noisy Wordle” and “Yellow Wordle”. The best performance we achieved on Wordle is 3.73 guesses on average using entropy with word frequencies.

1 Introduction

Wordle is a word game in which the player has to guess a five-letter English word using mastermind-style guesses and answers format. The goal is to guess the secret word with as few guesses as possible. If the player fails to guess within six attempts he loses the game. A more detailed explanation of the game rules is in Section 2.1.1.

We chose the game Wordle because it is interesting and not trivial to solve, and its popularity among the public and algorithm enthusiasts shows it. The states and actions spaces are large - there are 12,972 words that the player can choose to guess in each turn and $3^5 = 243$ five-length color patterns that he can get as feedback. Therefore, there are $12,972 \cdot 243 \approx 3.2 \cdot 10^6$ options for each turn, and $(3.2 \cdot 10^6)^6 \approx 1.1 \cdot 10^{39}$ possible states after six turns.

It is important to say that not all those states are possible. For example, we cannot guess the same word twice and get different results, so the actual state space is smaller. However, this is a simple way to model the game so these big numbers are relevant when it comes to run time. In our algorithms, we implemented a heuristic to reduce the state and action spaces.

We explored different variations of Wordle:

1. **Wordle** - the game in its original form.
2. **Absurdle** - an adversarial variant of the game.
3. **Vocabulary Wordle** - in this version we change the vocabulary of the game in different ways.
4. **Noisy Wordle** - in this version we add noise to the patterns that the player gets.
5. **Yellow Wordle** - in this version the player gets less information from the pattern.

Many properties of the game such as the state space size and the amount of information the user has changes between different variations of the game. That makes some game versions harder than others, requires some algorithm adaptations, and makes some algorithms more suitable than others.

We explored different algorithms for solving the games:

1. **Random Baselines** - for comparison with other algorithms.
 - (a) **Total random** - guesses a random legal word.

- (b) **Random** - guesses a random possible word, which means it uses a basic heuristic.
- 2. **Adversarial Algorithms**
 - (a) **Minimax** (with alpha-beta pruning)
 - (b) **Expectimax**
- 3. **Entropy** - an algorithm that uses information-theory-based heuristic.
- 4. **Reinforcement Learning** - Q-Learning

We wanted to explore how each algorithm works and how each game variation affects it. We optimized these algorithms on the different game versions, compared their performances, and learned about the advantages and disadvantages of each one.

2 Methods

We created a collection of games similar in nature to Wordle and several algorithms that solve them.

2.1 The Games

2.1.1 Wordle

This is **the game in its original form**. We will explain the game rules, which are the base for all the other games.

At the beginning of the game, a secret word is chosen randomly. Then, in each turn the player guesses a word, and gets a five-length color pattern - one color for each letter, according to the following rules:

- Green - the secret word contains the letter in the same position.
- Yellow - the secret word contains the letter in a different position.
- Gray - the secret word does not contain the letter.

Examples can be seen in Figure 1:



Figure 1: Wordle color description examples

There are two word lists in Wordle. We call the first one the "**legal words**" (12,972 words) - these are the words that the player can guess. However, not all legal words can be secret words. There is a subset of more common words, which we call the "**secret words**" (2,315 words), and only they can be chosen as secret words.

When you play the game online and try to guess a word that is not in the legal words dictionary you get a message that tells you to try a different word. Therefore, this list is effectively exposed to the human player, and we gave it to our algorithms. The secret words are not exposed to the human player, even though he has some prior knowledge about them because of their popularity, so we chose not to give our algorithms this list. It is important to note that some other Wordle solvers online use the secret words, so their results are not comparable to ours.

2.1.2 Absurdle

Absurdle is an **adversarial variant of Wordle**, in which "the game" plays against you. In this game, the player can take the same actions as in the original game, but the game works differently. The original Wordle picks a single secret word at the beginning of the game, and then the player has to guess it. Absurdle gives the impression of picking a single secret word, but instead, it considers the entire list of all possible words, which means secret words which also match the patterns and guesses so far. Each time the player guesses, Absurdle prunes its internal list as little as possible, attempting to intentionally prolong the game as much as possible. A casual Absurdle game can last for over a dozen turns, so it has no turn limit like Wordle.

This version is interesting because of its difficulty, and it gives us a sense of what happens to our algorithms in Wordle if they have bad luck.

2.1.3 Noisy Wordle

In this version, we add noise to the patterns that the player gets. Each turn, a random position in the pattern is randomly selected and its color becomes one of the three colors with equal chance. For example, a pattern might change the same way as shown in Figure 2 with a probability of $\frac{1}{5} \cdot \frac{1}{3} = \frac{1}{15} = 0.066$.



Figure 2: Noisy Wordle compared to Wordle example

In this case, since every position may change to two other colors and there are five such positions there are $1 + 2 \cdot 5 = 11$ different options for the real pattern, so agents better consider 11 patterns at each guess, making the game harder.

2.1.4 Yellow Wordle

In this version, for each guess, the game only reveals whether the letter is in the word and not whether it is in the exact position. That is, when Wordle marks a letter in green or yellow, Yellow Wordle will mark that letter in yellow. Such a change reveals significantly less information.



Figure 3: Yellow Wordle compared to Wordle example

2.1.5 Vocabulary Wordle

A Wordle variation in which we change the game vocabulary. We implemented two variations of the original game:

1. **"Real vocabulary"** - the vocabulary (the legal words and the secret words) is a random subset of the original vocabulary. The vocabulary size is configurable, and changing it changes the complexity of the game. It is interesting to explore how the vocabulary size affects the different algorithms, each one on its own, and compared to each other.
2. **"Fake vocabulary"** - The vocabulary size is the same as in Wordle but the words are random 5-length sequences of English letters. The English vocabulary has some properties that the algorithms can use, such as high variance in letter frequencies. Therefore we would like to see the capabilities of the algorithms with an arbitrary vocabulary that does not maintain any relationship between the words.

2.2 The Algorithms

We used several algorithms to solve Wordle and its variations and optimized each one for every game variation.

2.2.1 Algorithm's Optimizations

During the development of the algorithms, we examined different methods and compared them (full details in the appendices Section A). One of the most important methods used by the algorithms is filtering words from the vocabulary.

Vocabulary Filtering

When we want to guess a word a question arises - whether to choose a word from the wide list of legal words or only consider words that can be the actual secret word according to the

previous patterns. In the first approach, the algorithm considers all the legal words and the second one is a heuristic approach that tries to find a subset of good potential guesses. We chose to filter the legal words list after each turn, according to the pattern we got. We got a list of the words that match all the guesses and patterns, and called them the “possible words”. We let the AI algorithms guess words from this list because it improves the results of some of them, and significantly improves the run-time of all of the algorithms. This was tested and proven to work in [1]. In fact, there are algorithms like Entropy that only this filtering makes their run feasible. This filtering method will be further clarified with the following random algorithms, where one uses legal words and the other uses possible words.

2.2.2 Random Baselines

Total Random

In every turn, the algorithm guesses a **random legal word**. We used it as our simplest baseline algorithm.

Random

In every turn, this algorithm chooses a **random possible word**, which means a random word that can be the secret word according to the previous guesses and patterns. It is similar to the way that a naive human player would play the game. This simple variation of the total random algorithm reduces the search space significantly and therefore has the potential to significantly improve performance.

2.2.3 Adversarial Algorithms

Wordle is not a two-player game, and still, it can be modeled as one similar to [2, 3]. The first player is the “real” player, and his possible actions are the possible words that are the optional guesses. The second player is the “game”, and his possible actions are all the possible patterns - patterns that match the remaining possible words. This modeling makes sense because the player does not know the secret word, so the patterns he gets are not deterministic for him. This modeling enables us to use the adversarial algorithms - Minimax and Expectimax.

The evaluation function is:

$$-|possible\ words|$$

Namely, minus the number of possible words. We use the number of the remaining possible words as an indicator of the distance of the player from winning, and we add the minus because the player wants to minimize this distance. We tested several evaluation functions and found that this function works best, this exploration is described in Section A.1.

The action space is big so in most games, we ran the algorithm with *depth* = 1. In Vocabulary Wordle, the vocabulary was smaller so we were able to run the algorithms with *depth* = 2.

Minimax


When applying Minimax to this game tree the MAX player chooses a word that will minimize the number of remaining possible words, and the MIN player chooses a pattern that will maximize it. This algorithm is expected to have good worst-case results. For example, it is expected to work well with Absurdle because its assumption about the other player makes sense with an adversarial game like Absurdle. In Fact, the MIN player is an exact simulation of the Absurdle board. Additionally, we used alpha-beta pruning to improve the run time.

Expectimax

When applying Expectimax, the player tries to get the best results in the expected case. This assumption suits Wordle better because the secret word is chosen randomly. Therefore we expect this algorithm to solve Wordle in fewer guesses than Minimax on average.

2.2.4 Entropy

The goal of Wordle is to discover the secret words with as few guesses as possible. The player starts from a state with many possibilities and high uncertainty and tries to get to a state with fewer possibilities and less uncertainty. Therefore, reducing uncertainty in the game and gaining as much information as possible by cutting the possibilities space seems like a good heuristic. This algorithm uses entropy as a measure of this expected information gain.

For every guess, we can iterate over the possible words and compute the pattern that will be generated if it is the secret word. If we sum the number of times every pattern appeared and divide this sum by the number of possible words, we will get the probability of every pattern to appear for the given guess. For example, if we guess ‘tares’ and there are a hundred possible words, we compute the patterns generated from ‘tares’ and each one of the hundred possible words. If the pattern  was received 10 times, then the probability to get this pattern is $\frac{1}{10} = 0.1$.

we denote the probability of every pattern as p_i where $1 \leq i \leq 243$ (number of possible patterns), and the expected information we will get for the guess is:

$$\mathbb{E}[Information] = \sum_{i=1}^{243} p_i \cdot \log_2 \left(\frac{1}{p_i} \right)$$

The entropy measures what is the expected information gain from a guess, and the algorithm we use chooses the word with the highest entropy.

We can improve this algorithm by incorporating information about word frequencies in the English language, information that sheds light on the secret words list. Moreover, we can sometimes try to exploit instead of exploring, which means trying to guess a word we don’t think will have high information gain but has a high probability of being the secret word. We tried those improvements and got good results, more about it in the appendices Section A.2. However, we did not include the results of this improvement throughout the whole report because using word frequencies is a privilege that the other algorithms did not get. This method has the highest Wordle results, so we chose to mention them only for basic Wordle.

2.2.5 Reinforcement Learning

In Wordle, given the game state (guesses and their corresponding patterns) and the next action (a word to guess), we get a new pattern of colors. In other words, we get rewards for every letter in the word that we guessed. Therefore, it is possible to learn an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. Hence, Q-learning fits this game and its variations.

The main problem with using Q-learning is that the state space is too big. As explained in the introduction, there are about $3.2 \cdot 10^6$ different states in each turn. This space is too big to fit in the Q-table, even if we only consider the last guess with its pattern, instead of all six guesses.

For this reason, we tried to find the best way to use a Q-learning agent. First, Using deep Q-learning does not seem to do a good job of solving Wordle as explained in [4]. Second, we tried using approximate Q-learning where we take features of matches and differences between the current pattern and the next pattern that will be generated from the next action. However, it did not perform well with full analysis in Section A.3. Thus, we tried to see what happens if we do not consider the state of guess and pattern but only the actions, namely use a constant state so the size of the Q-table is only $1 \times 12,972$.

The logic is that if the algorithm is only considering actions from the possible words, then we already know that these actions conform to the guesses made so far. Thus, there is no merit in increasing the table with more states. The Q-learning algorithm associates every possible word with a Q-value and learns to guess words that give him higher rewards, i.e., words that will give him patterns with green colors. Therefore the algorithm can learn to guess words from the secret words list which other algorithms cannot do.

In addition, we found that the best way for Q-learning to deal with the variants of the Wordle game is to train on the basic Wordle game and test on the other games with the results shown in Section A.3.2. This training works much better than training on specific variants because the Q-learning agent does not care for the actual states on the game board, i.e., the guesses and their corresponding patterns, but only for the words he could guess. Therefore, we tested all variants of Wordle using the basic training of Wordle apart from the Fake vocabulary variant, in which there are different actions from the basic Wordle game.

It is important to note that this algorithm has an unfair advantage over the other algorithms. In the training, we ran simulation games with the list of secret words, and that way the algorithm was indirectly exposed to them. It is not fundamentally wrong, Wordle is an interesting game to solve even if the algorithm has access to the secret words list (the secret words are the more popular words so one can even argue that a human player has prior knowledge about this list). However, the other algorithms were not exposed to this list, and therefore the comparison is not totally fair. It is therefore recommended to take the comparison between this algorithm and the others with a grain of salt throughout this report.

3 Analysis and Results

3.1 Evaluation Method

Metrics

We evaluated our algorithms with two main metrics:

- Average number of guesses, and if the player does not succeed in six guesses we count it as six guesses.
- Win percentage, where winning is guessing the secret word in no more than six guesses.

These metrics are the most popular metrics for Wordle solving evaluation, and they complement each other.

Error Bars

We used the error bars to show the standard deviation of the number of guesses of each algorithm in each game. Therefore, it does not represent our measurement error, but rather another property of the algorithm. In addition, it is important to note that when an agent does not succeed within six guesses we count it as six guesses, and it affects the standard deviation. For example, an algorithm that never guesses the secret word will get an average of six guesses with zero standard deviation (and 0% win rate).

Evaluation Games

We ran each game multiple times for evaluation. The algorithms have different run times on different game variants, so we ran each variant for different amounts of games. We ran Wordle on all 2,315 secret words to simulate all possible Wordle games without repeating a secret word. This is the most comprehensive comparison between the algorithms because we check every possible secret word. In contrast, we ran Noisy, Yellow, and Fake Vocabulary on 100 random secret words since the algorithms’ run-time on these variations is much larger (some take over dozens of hours to run). Finally, Absurdle is a deterministic game where all AI agents return the same guesses, so we ran it for a single game with all algorithms except Random. The Random algorithm plays Absurdle differently each time so we ran it for 1000 games.

Total Random

The Total Random algorithm gets poor results, as expected. Therefore, we present its results only for basic Wordle, and use the Random algorithm as a baseline in the other games.

3.2 Wordle Results

In Figure 4, we can see the average number of guesses of each algorithm.

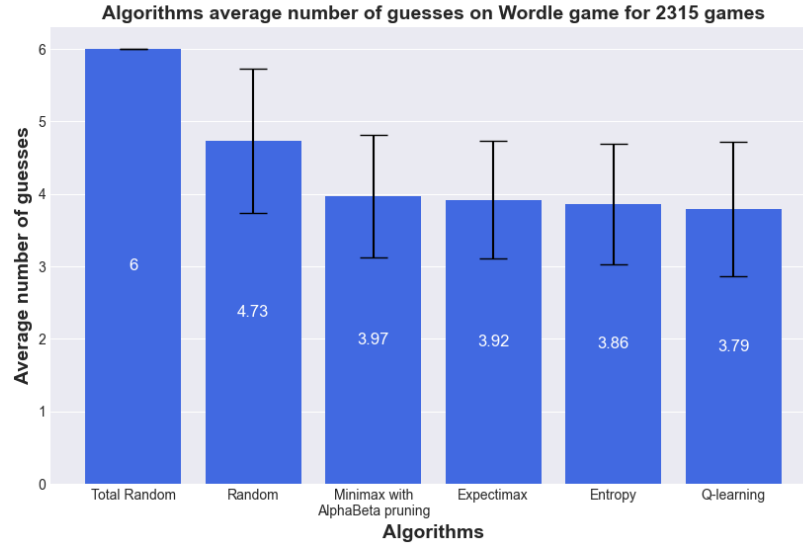


Figure 4: Average number of guesses of each algorithm in the basic Wordle game for all 2,315 secret words

We see that the order of the algorithms according to their success is: Q-Learning, Entropy, Expectimax, and then Minimax - all below four guesses on average, **far better than the random baselines**.

In Figure 5, we can see the win percentage of each algorithm.

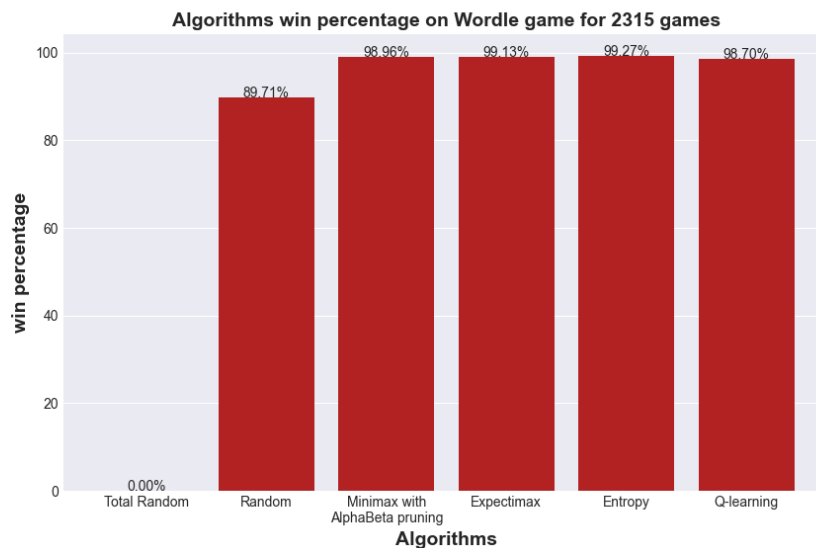


Figure 5: Success rate of each algorithm in the basic Wordle game for all 2,315 secret words

Here the order is different - Entropy, Expectimax, Minimax, Q-Learning, but also here all of the algorithms get pretty good results - 98.7% and above, **far better than the random baselines**.

As we see in the figures, the total random algorithm fails. That is because there are many options for the secret word, and when guessing a random legal word your chances to win “accidentally” are low. On the other hand, the random algorithm has 4.73 guesses on average and an almost 90% success rate. It is because it guesses only words that are compatible with the previous guesses and patterns. That way it reduces the number of words it guesses from in each turn until the list is small and its chances to win are high. **We conclude that filtering the words that do not match the previous guesses is enough to win Wordle in about 90% of the games, better than many human players.**

As expected, the Expectimax algorithm gets better results than the Minimax algorithm. Minimax succeeds in guessing the secret word in 3.97 guesses on average, while Expectimax succeeds in guessing in 3.92 guesses on average and wins more games. The advantage of Expectimax over Minimax was expected because the secret word is chosen uniformly, so maximizing the expected result works better in the basic game than maximizing the minimal result.

In addition, the Entropy algorithm reaches the highest winning percentage, and it manages to guess the secret word in 3.86 guesses while the Q-learning algorithm manages to guess the secret word in 3.79 guesses but with fewer wins.

Therefore, reducing the search space using entropy results in the highest win percentage in the game while learning the word that will yield the highest reward among all the possible words results in the lowest number of guesses.

In the search for the best possible performance in the basic Wordle game, we improved the Entropy algorithm and got our best Wordle results. We incorporated word frequencies into the entropy algorithm, as explained in Section 2.2.4. Using this method resulted in 3.73 guesses on average with a 99.13% win percentage, showing an encouraging performance enhancement.

Thus, we conclude that Entropy and Q-Learning learning are the best basic Wordle solvers, among the algorithms we tried.

3.3 Absurdle Results

All algorithms manage to win the Absurdle game given enough guesses, so we only look at the (average) number of guesses it took each one, as shown in Figure 6.

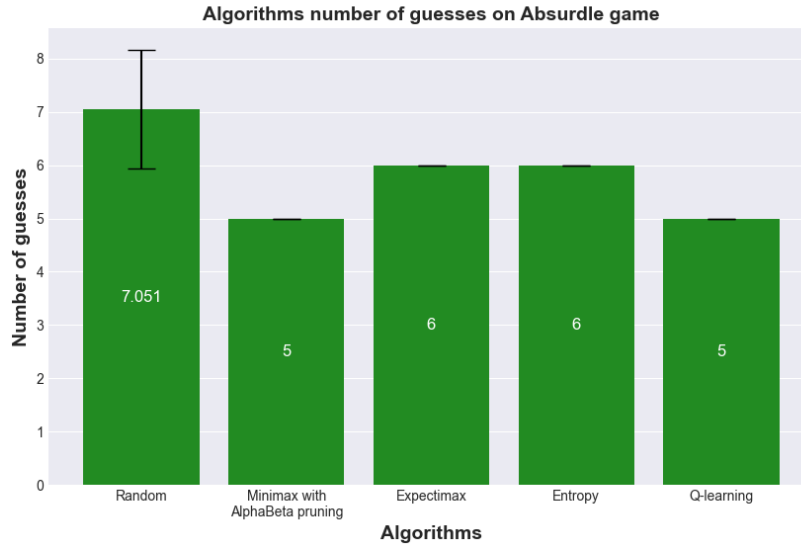


Figure 6: The average number of guesses of each algorithm in Absurdle

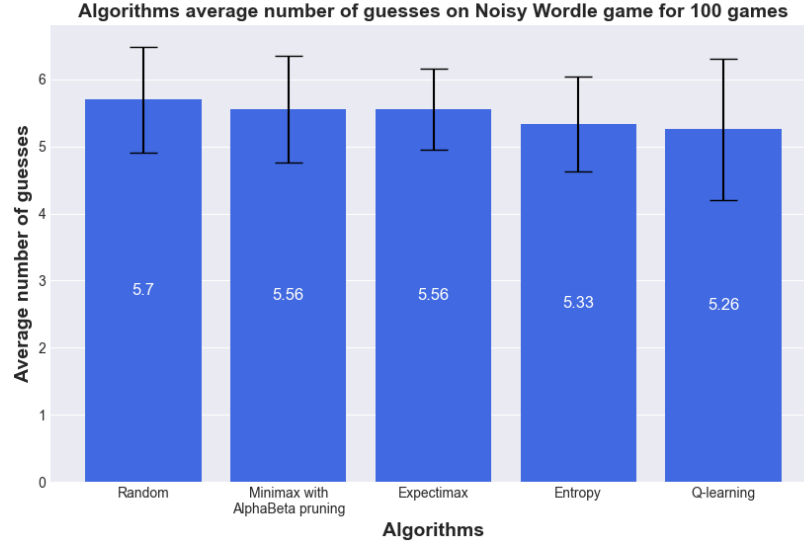
As expected, the Minimax algorithm gets better results than the Expectimax algorithm. Minimax wins in only five guesses while Expectimax needs six guesses. It is an interesting result because Expectimax was superior in the basic Wordle game. This makes sense since Expectimax isn't optimal for this problem. As we said earlier, Absurdle chooses the pattern that maximizes the number of words in the secret list, just like the assumption about the MIN player in Minimax. Therefore as a MAX player, we want to maximize this minimal score rather than the average score.

Similar to Expectimax, the Entropy algorithm beats the game in 6 guesses so it is worse than Minimax as well for this game. However, the Q-learning algorithm manages to win the game in 5 guesses.

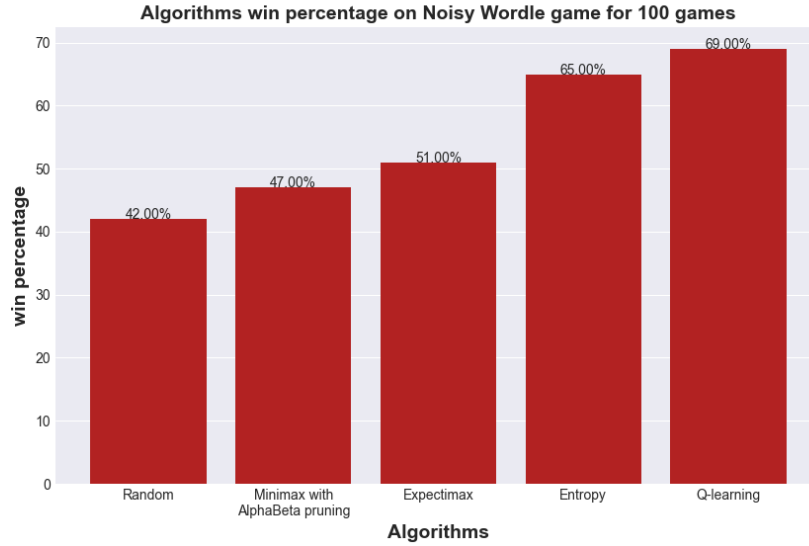
Thus, we conclude that Minimax and Q-learning are the best basic Absurdle solvers, among the algorithms we tried.

3.4 Noisy Wordle Results

In Figure 7, we can see the average number of guesses and the win percentage of each algorithm.



(a) Average number of guesses



(b) Win percentage

Figure 7: Algorithms results in Noisy Wordle over 100 Noisy Wordle games

As we can see in the figures, **the performances of all algorithms are much lower than in Wordle**. The Random algorithm achieves only 42% success because it takes more guesses to reduce the number of possible words.

The adversarial algorithms are not much better, Minimax and Expectimax achieve 47% and

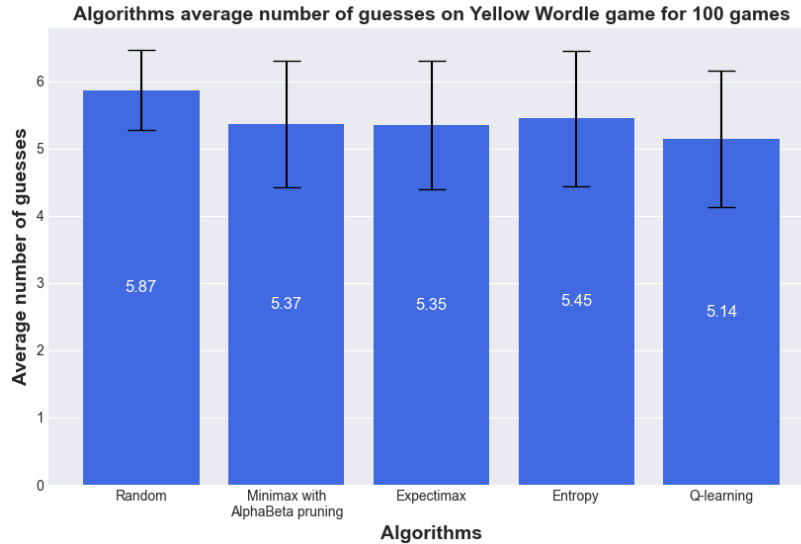
51% success rates respectively. These algorithms require much processing time, even at depth 1, since there are many more possible actions than in the basic game. We dealt with this situation with an approximation for the legal actions of the MIN player. In particular, we defined that the legal action of the MIN player is to return the pattern $\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare$ to discover as little as possible for the MAX player under the assumption that a color in the pattern can change back to gray. However, given the fact that it cannot run on all possible patterns that the MIN player can return, we conclude that this type of algorithm is not suitable for solving the noisy problem.

In contrast, **Entropy and Q-learning manage to achieve better results** with 65% and 69% success rates respectively. To reduce runtime, the Entropy algorithm calculated the best guess as in a normal Wordle game without considering noise.

Additionally, as described in the Section 2.2.5, we decided to use the trained Q-learning agent on the basic Wordle game as it produced better results. We can see that treating the state of the game as constant helps in dealing with potentially false patterns.

3.5 Yellow Wordle Results

In Figure 8, we can see the average number of guesses and the win percentage of each algorithm.



(a) Average number of guesses

Figure 8: Algorithms results in Yellow Wordle over 100 Yellow Wordle games

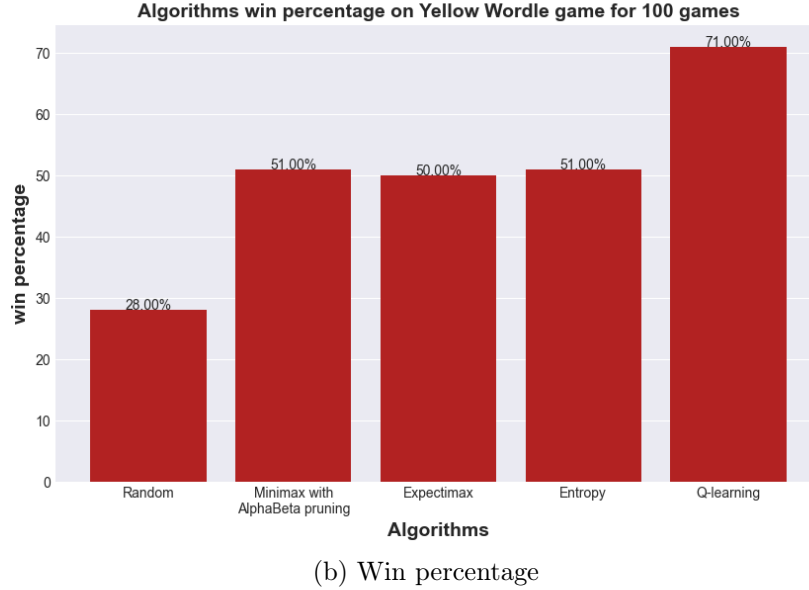


Figure 8: Algorithms results in Yellow Wordle over 100 Yellow Wordle games

This is the hardest game for the Random algorithm, even more than the Noisy game, it wins only 28% of the games. We also see that our other algorithms have difficulties - Minimax, Expectimax, and Entropy all win only about 50% of the games. The adversarial agents work a bit better than the Entropy agent since their average number of guesses is slightly lower. However, the Yellow game shows the capabilities of the Q-learning agent, which wins 71% of the games with the lowest average number of guesses. The Q-learning agent returns the optimal policy given the possible actions and does not directly use the pattern received. Thus, **it is more robust to such changes in the game logic and is the best agent for this game**.

3.6 Fake Vocabulary Wordle Results

As explained in Section 2.1.5, we created this game in order to see the algorithms' performance without the properties of the English vocabulary. For example, if we know that "E" is the most frequent letter in English and half of the words contain it, in basic Wordle we can use it to cut the possible words space by half only by looking at the feedback we get for this letter. However, with random vocabulary, the letters will be distributed more evenly, and the agents will not be able to exploit such phenomenons. Therefore, we compare the algorithms' scores with the real vocabulary and the fake vocabulary.

In Figure 9, we can see the average number of guesses of each algorithm in basic Wordle ("Real Vocabulary") and with fake vocabulary.

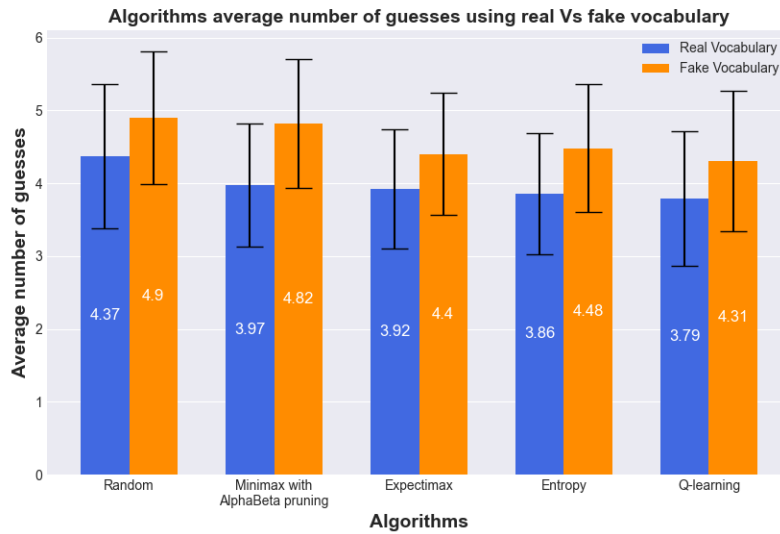


Figure 9: Algorithms average number of guesses on real and fake vocabularies (over 100 fake games)

First of all, we see that **all algorithms are better with real vocabulary than with fake vocabulary**. We expected this result because the structure of the English vocabulary helps to reduce the possible words space faster, as explained above.

In addition, it is interesting to see that **"smart" algorithms lose more from this change of vocabulary**. It makes sense because these algorithms exploit the vocabulary properties more wisely. It can be clearly demonstrated with Random and Minimax. With real vocabulary, Minimax is doing much better than random - 0.8 fewer guesses on average, but with fake vocabulary, the gap between them is almost closed.

We can conclude that the patterns in this game do not reduce many words because the vocabulary is uniformly distributed, and so it is difficult to find a guess that will cut the space effectively. Nevertheless, we can see that training the Q-learning algorithm on the fake words results in the lowest average guesses. It may do worse than its real vocabulary counterpart, but after training on the fake vocabulary words it manages to choose the word that will yield the best rewards with the highest Q-value, even if there are no relations between them.

3.7 Real Vocabulary Wordle results

For the last evaluation, we wanted to test the different algorithms on different vocabulary sizes. We checked a variety of vocabulary sizes - from 1000 words to 12972 legal words, and changed the size of the secret words list respectively. The goal was to see how the order of the algorithms changes with different vocabulary sizes.

In Figure 10, we can see the average number of guesses of each algorithm as a function of the vocabulary size.

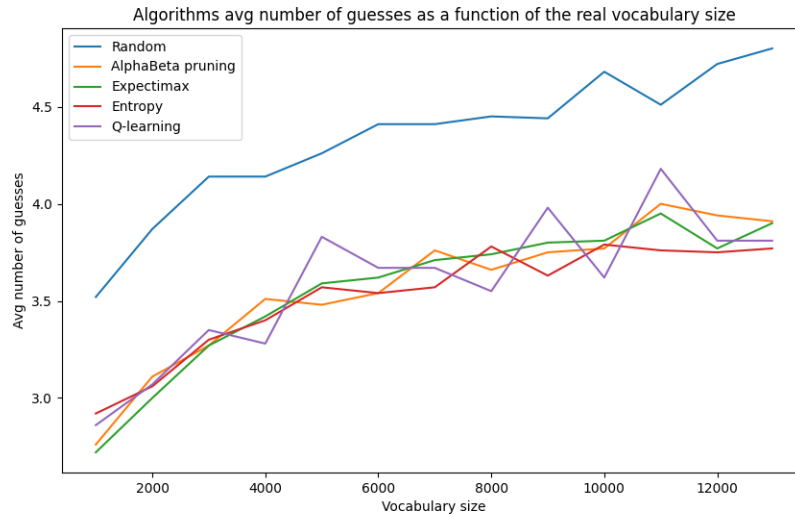


Figure 10: Algorithms average number of guesses on different vocabulary sizes over 100 games every vocabulary size

First, it is not surprising to see that **all algorithms improve when the size of the vocabulary decreases**.

Secondly, we can see **no matter what the vocabulary size is there is a significant gap between the random algorithm and the AI algorithms**. Even when the game is simpler, using AI methods has a significant impact.

We will explore how the order of the algorithms changes with the size of the vocabulary. For vocabulary sizes (1000 – 3000) the number of actions is small enough to run Minimax and Expectimax with $depth = 2$. Therefore, **the Expectimax algorithm is the preferred choice in this vocabulary range**. When we reach a vocabulary size of 4000 there is a shift - **Q-learning is the preferred algorithm**, though it has big spikes. These spikes are there because we trained the Q-learning algorithm on the full vocabulary size to avoid overfitting specific words with not a lot to choose from. From the vocabulary size of 6000 and onwards, **the Entropy algorithm usually does the best**. However, unlike Q-learning, the Entropy algorithm does not have big spikes. It is worth mentioning that Entropy is the worst AI agent with a small vocabulary (1K) and the best with a big one ($\sim 13K$). With a large search space, information-theory-based considerations seem more effective than other approaches.

4 Conclusions

In this project, we learned about the strengths and weaknesses of different artificial intelligence agents on Wordle and its variants. We discovered some interesting results:

- The random algorithm that filters the words that do not match the previous guesses wins Wordle in about 90% of the games, better than many human players.
- AI Wordle solvers can reach much better results than random baselines.
- Entropy and Q-learning are the best basic Wordle solvers, among the algorithms we tried.
- The English Vocabulary structure is a crucial part of Wordle. all algorithms are doing worse with fake vocabulary than with real vocabulary, and Smart algorithms lose even more from this change.
- Expectimax does better than Minimax on the basic game and most variants because these games are not adversarial so maximizing the expected score is a better choice.
- Minimax is best suited for solving the Absurdle game, as it upholds its adversarial assumption.
- For smaller vocabulary sizes, Expectimax with $depth = 2$ is better than the other algorithms.
- The Q-learning algorithm works quite well for all game variants, achieving the lowest number of average guesses on the basic Wordle game and the best one for dealing with fake vocabulary.

Thus, every Wordle game could benefit from a different algorithm and we hope that our work can act as a basis for future work on Wordle-type solvers using different artificial intelligence methods.

References

- [1] 3Blue1Brown, “Solving Wordle using information theory”, Feb 6, 2022.
<https://www.youtube.com/watch?v=v68zYyaEmEA> (accessed 27/8/2022).
- [2] Yotam Gafni, “Automatic Wordle Solving”, Jan 8, 2022.
<https://towardsdatascience.com/automatic-wordle-solving-a305954b746e> (accessed 27/8/2022).
- [3] Rebecca Ribas, “Wordle Solver - Using alpha-beta pruning to solve wordle”, Mar 1, 2022.
<https://levelup.gitconnected.com/wordle-solver-31601a5dbb42> (accessed 27/8/2022).
- [4] Andrew Ho, “Solving Wordle with Reinforcement Learning”, Mar 6, 2022.
<https://wandb.ai/andrewkho/wordle-solver/reports/Solving-Wordle-with-Reinforcement-Learning-VmlldzoxNTUzOTc4> (accessed 27/8/2022).
- [5] Daniel Nichols, “Wordle: Finding the Right Words to Say”, Jan 24, 2022.
<https://dando18.github.io/posts/2022/01/24/wordle-finding-the-right-words-to-say> (accessed 27/8/2022).

Appendices

A Agents Optimization

We optimized every algorithm we used on Wordle and its variations to achieve the best results on every game and examined the comparison between the best performing algorithms.

A.1 Minimax

We considered several heuristics as the evaluation function:

1. **Minus the number of remaining possible words** - the fewer possible words, the closer the win.
2. **Score by game turn**- by minimizing the number of guesses, the player will have to finish the game as quickly as possible.
3. **Score by pattern** - a weighted sum of the pattern colores:

$$10 \cdot |greenTiles| + 5 \cdot |yellowTiles| + 1 \cdot |grayTiles|$$

whern $|[color] Tiles|$ is the number of tiles from color $[color]$ in the pattern. We thought that maximizing this value will bring the agent’s guess closer to the secret word.

The results we got for 100 basic Wordle games are in Table 1:

	<i>Minimax</i>		<i>Expectimax</i>	
	Avg guesses number	Win percentage	Avg guesses number	Win percentage
Remiaing number of possible words	3.9	99%	3.86	100%
Game turn	3.99	97%	3.84	99%
Score by pattern	4.08	99%	3.93	100%

Table 1: Adversarial algorithms evaluation functions scores over 100 Wordle games

As can be seen, the first heuristic in the table - the remaining number of possible words - led to the lowest average number of guesses and the highest percentage of wins, and therefore we used it to evaluate the adversarial algorithms.

A.2 Entropy

As well as the method we discussed in Section 2.2.4, we also used an improved method. We briefly mentioned it in the report body and here we will elaborate on it.

In the basic algorithm, the agent measures the expected information received from each possible guess and returns the word with the maximum information. In the improved algorithm, the agent measures the remaining uncertainty among all of the possible words and returns the word with the minimal expected game score.

We will explain about the extra methods used in the improved algorithm:

Words Probabilities

To determine the probability that each legal word is indeed a secret word, we use the frequencies of the words in English. In the basic algorithm, it can be seen as we give each word the same probability $1/|legalWords|$.

For this method, We found an external source of the frequencies of the legal words in English. We could normalize these frequencies and use them as the probabilities of the words (we also tried it). However, we knew that there are only 2,315 secret words, and the probability of each word is 0 if it is not in this list and $1/|secretWords|$ if it is in this list. Therefore, we wanted to predict which words will be in this list. We used a “soft” prediction with a sigmoid function as described in [1]. That way we got a probability of each word being the secret word at the beginning of the game. During the game, we update these probabilities according to the list of possible words. We denote these probabilities $Pr[word]$.

Remaining Uncertainty

Next, we wanted to find the expected number of turns in the game by using the remaining uncertainty. Once we have a probability associated with each word we can compute the entropy of the entire distributions of possible words. We denote this entropy as h_0 and call it the remaining uncertainty. Then, we can compute the expected information for every guess similar to the first method. However, instead of counting each pattern based on a uniform probability, we will use the probabilities of the words that generated each pattern. We will denote this entropy as h_1 - this is the information gained from the guess. Now that we have both entropies h_0 and h_1 their difference is an estimate of the amount of information left in the game. Afterward, we can use a function, we call f , to turn this information into an expected number of turns in the game. We simply took f to be the identity function, as it performed better than the function suggested in [1].

Finally, if we treat the score of the game as a random variable with two outcomes - winning in the next turn and not winning in the next turn, then for every word we can compute the expected value of the game score. The expected value of the game score can be defined in the following manner:

$$\mathbb{E}[score] = Pr[word] \cdot 1 + (1 - Pr[word]) \cdot (1 + f(h_0 - h_1))$$

Namely, with probability $Pr[word]$ the game will end in the next turn, and with probability $1 - Pr[word]$ the game will end in $1 + f(h_0 - h_1)$ which is the expected number of turns in the game. Finally, our agent returns the word with the minimum expected game score.

The results of both methods on all 2315 secret words in basic Wordle are in Table 2:

	Avg guesses number	Win percentage
Basic Entropy	3.86	99.27%
Improved Method	3.73	99.13%

Table 2: Entropy based applications results over 2315 Wordle games

We can see that the second method wins with fewer guesses on average. However, we chose not to use it. In order to associate each word with a probability, we used prior information about the size of the secret words list. Hence, we don't believe that the comparison between this algorithm and the others is fair.

A.3 Q-learning

A.3.1 Learning methods

As described in Section 2.2.5, the state space in Wordle is too large to be kept in a Q-table. Therefore, we tried using different methods and chose the best-performing one as our agent.

Approximate Q-learning

First, we tried using approximate Q-learning. With this method, we do not need to hold values in our Q-table, but only learn the weights that are associated with the features of the game. The features we chose were the matches and differences between the last pattern of colors and the next pattern of colors. Specifically, given a state $(guess_1, pattern_1)$ and an action $guess_2$ we computed the pattern that will be received from $guess_1$ and $guess_2$ as if $guess_2$ was in fact the secret word. We'll denote this pattern as $pattern_2$ and our features were the number of matches and differences of colors between $pattern_1$ and $pattern_2$. Using features for specific colors or comparing $guess_2$ directly to $guess_1$ yielded similar results.

Constant State

Second, we tried using classic Q-learning by using different states than the ones shown on the game board. As detailed in Section 2.2.5, we used a constant state so our agent would give scores to the possible guesses, which are the remaining possible words.

Turn-Based State

However, a constant state was not our only option in decreasing the state space, as we could also use the state as the game turn number. Since there are 6 possible turns, the Q-table is of size $6 \times 12,972$ which is still possible to hold in memory, and this way we can give higher rewards for guesses that solve the game in fewer attempts, so our agent could win in even fewer guesses.

Results

After we saw the three methods were plausible for solving the game, we tuned the hyperparameters for each one. The results over 2315 basic Wordle games are in Table 3:

	Avg guesses number	Win percentage
Approximate Q-learning	4.73	89.84%
Classic Q-learning with state=turn	3.94	98.40%
Classic Q-learning with constant state	3.79	98.70%

Table 3: RL methods results over 2315 Wordle games

As we can see, using approximate Q-learning requires almost one more guess on average than using classic Q-learning for solving the game. We can conclude that the features are not meaningful enough to learn the Q-value from. However, using classic Q-learning with basic states leads to much more promising results. While both methods’ performance is close, we can see that using a constant state is slightly better than using the state as the turn number. Hence, we chose the constant state agent for our Wordle evaluations.

A.3.2 Training method

In addition, to improve the results of the Q-learning algorithm on the game variants of Wordle we tested the best way to train it. Specifically, we could train the agent on the basic Wordle game and test that agent on the other games, or train and test on the same game variant. Results can be seen in Figure 11.

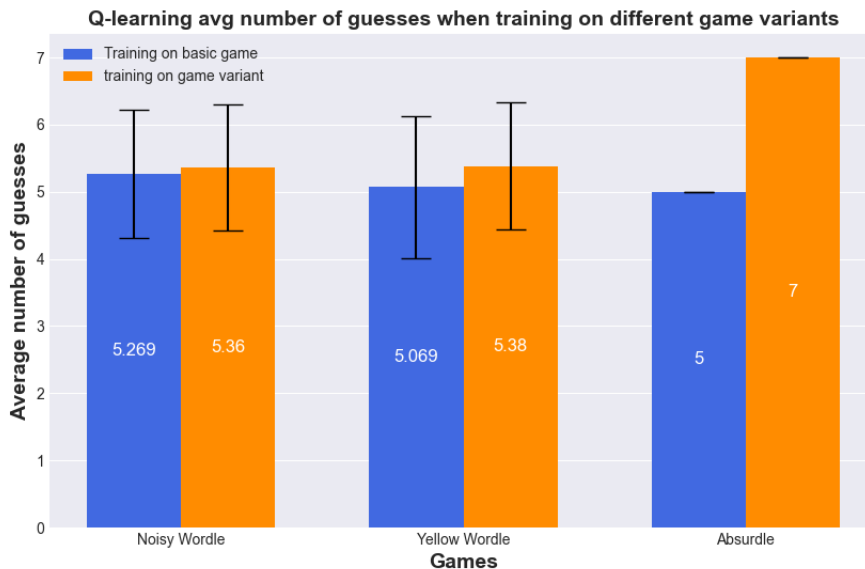


Figure 11: Q-learning average number of guesses when trained on different games

We can see that training the agent on the basic Wordle game results in a lower average number of guesses in every game variation. Therefore, apart from Fake Vocabulary where the actions are different, we train the agent on basic Wordle and test on the game variants.

B Best Opener

Many debate about what is the best word to start solving Wordle. We'd like to share our algorithms' answer to this question:

- Minimax's optimal opener is 'serai'.
- Expectimax's optimal opener is 'lares'.
- Entropy's optimal opener is 'tares'.
- Q-learning's optimal opener is 'leapt'.

All words play as a great first guess, and all algorithms use them to their advantage.

C How to Run the Code

- From the code's directory run the command:

```
pip3 install -r requirements.txt
```

- Run the command:

```
python3 run_with_gui.py
```

to run the game with GUI.

When running the GUI you can choose any Wordle game variation (apart from Fake vocabulary since it changes the words for all other games) and any algorithm we used. In addition, you can choose a secret word for the algorithms to guess or randomize one (except for Absurdle where there is no secret word).

- (Optional) run:

```
python3 simulate_games.py
```

to simulate multiple Wordle games with a pygame interface.

You can run `simulate_games.py` with different arguments to simulate different game variations and different algorithms. Specifically, you can use the flag `-n` to change the number of games, `-u` to use the pygame interface, `-g` to choose a game type, `-a` to choose an algorithm type.

For example, you can run:

```
python3 simulate_games.py -n 10 -u True -g Noisy-Wordle -a Q-learning
```

to simulate 10 Noisy Wordle games using the Q-learning algorithm.