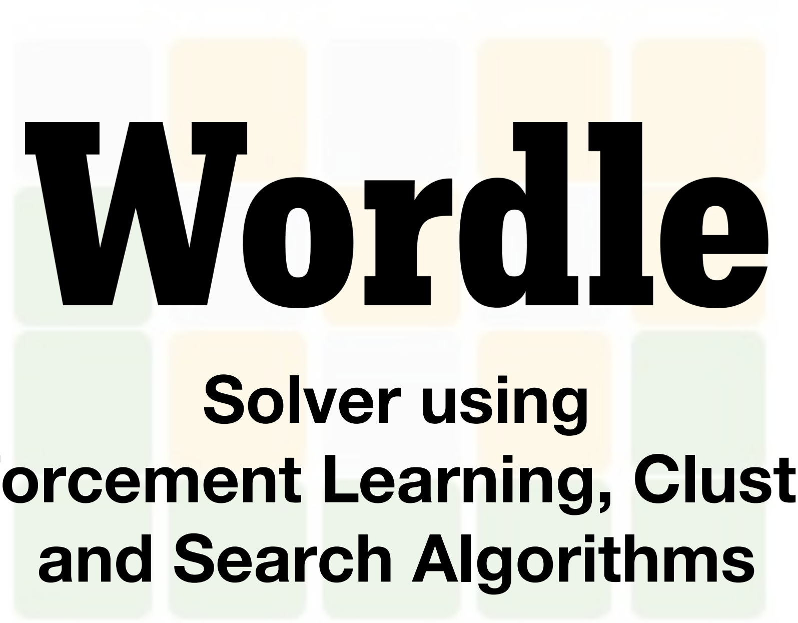


Wordle

Solver using Reinforcement Learning, Clustering and Search Algorithms

50.021 Artificial Intelligence

Group 3

Lee Min Shuen

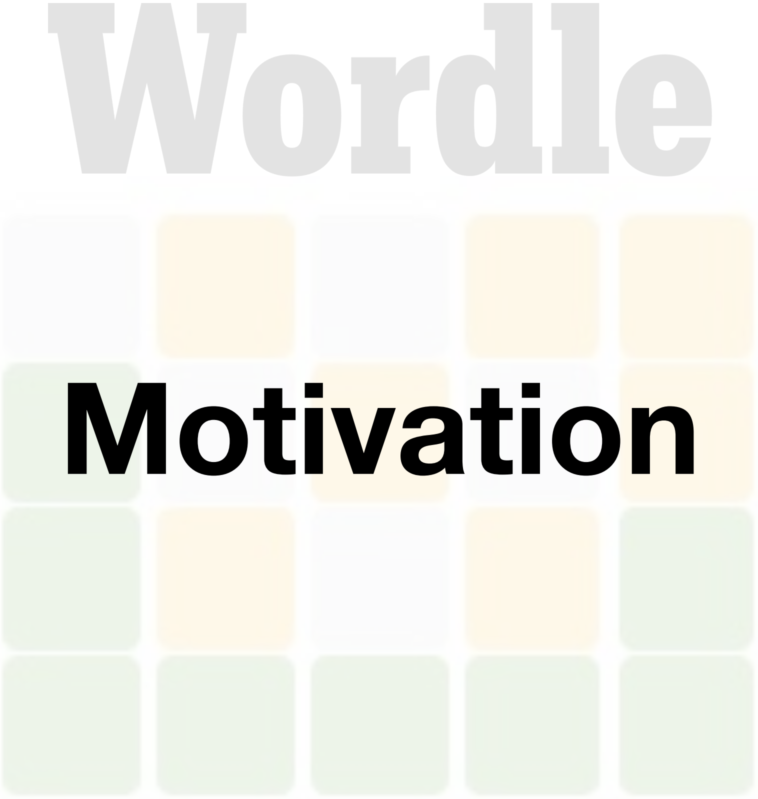
Samuel Sim Wei Xuan

Riley Riemann Chin

Tan Hong Yew

Agenda

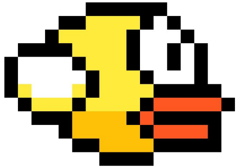
- Motivation
- Task Description
- Dataset
- Models
- Evaluations
- Graphical User Interfaces
- Improvements



Motivation

Motivation

- Addictive games (e.g. Flappy bird in 2014)

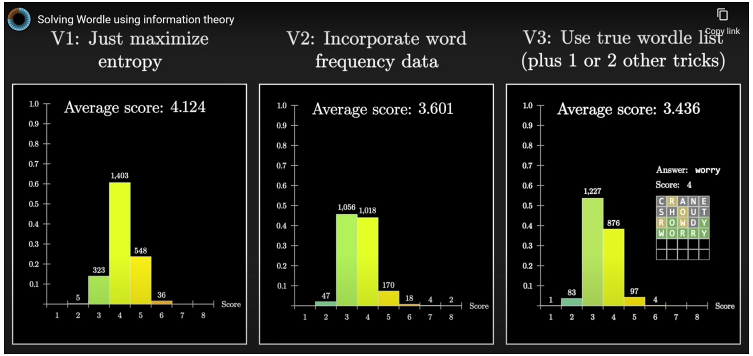


- Most recent craze is Wordle

Wordle

Motivation

- Famous math YouTube channel, 3Blue1Brown aimed to find the best starting word using information theory

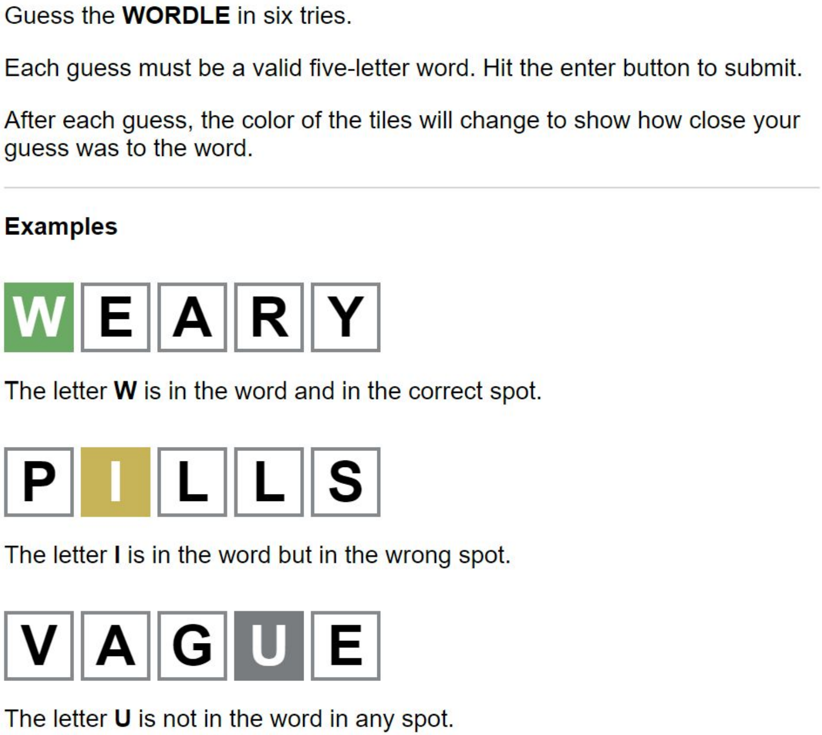


Screenshot of Solving Wordle using information theory by 3Blue1Brown

- How might we apply AI techniques to solve Wordle?

Task Description

How to play Wordle



Guess the **WORDLE** in six tries.

Each guess must be a valid five-letter word. Hit the enter button to submit.

After each guess, the color of the tiles will change to show how close your guess was to the word.

Examples

W E A R Y

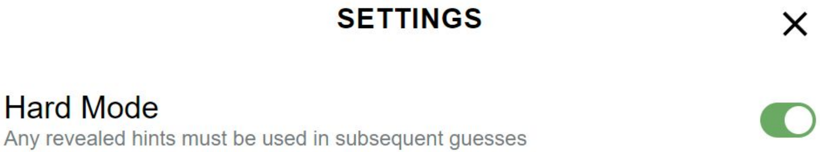
The letter **W** is in the word and in the correct spot.

P I L L S

The letter **I** is in the word but in the wrong spot.

V A G U E

The letter **U** is not in the word in any spot.



SETTINGS ✕

Hard Mode ☑

Any revealed hints must be used in subsequent guesses

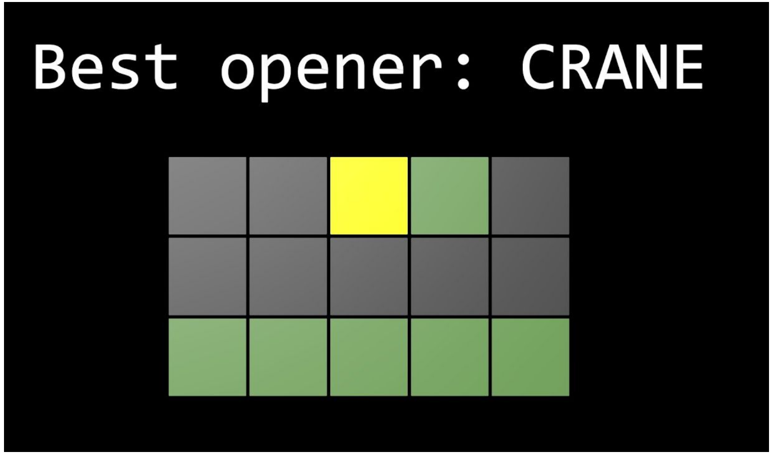
Task Description

- To use AI techniques to help solve the daily goal word in hard mode
- **Proposed algorithms:**
  - Reinforcement learning
  - Clustering algorithms
  - Search algorithms

General Strategy

Same starting word for all algorithms: CRANE

- Best starting word using information theory
- According to 3Blue1Brown



Best opener: CRANE

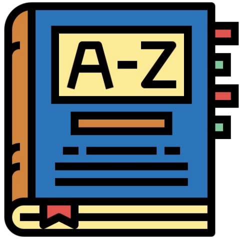
Screenshot of best opener for Wordle using information theory by 3Blue1Brown

Wordle

Dataset

Considerations

- 5-letter English words
- The Free Dictionary lists more than 150,000 5-letter English words

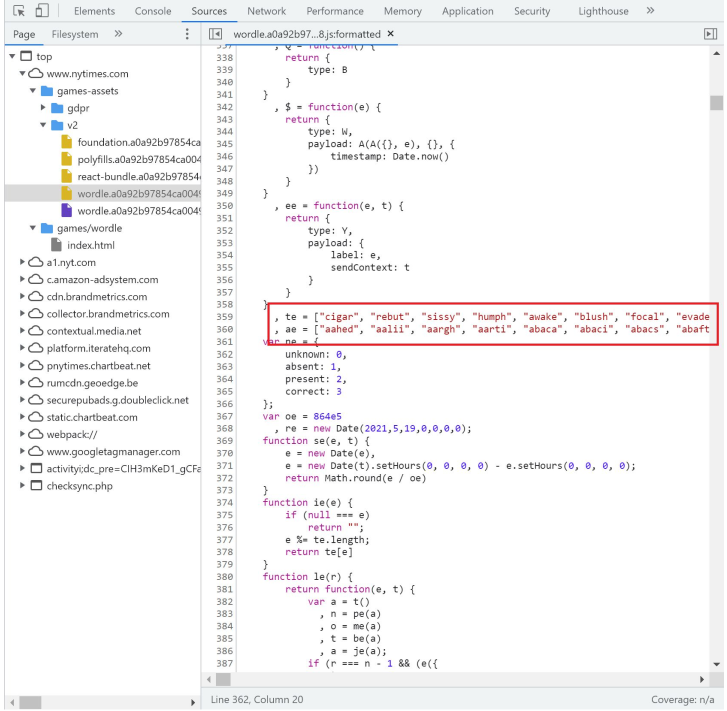


https://www.thefreedictionary.com/5-letter-words.html

Data Extraction

- Extracted 2 list of 5-letter words from Wordle javascript code
    - List 1: 2309 goal words
    - List 2: 12974 accepted words

- Training data: List 1 + List 2
    - 15283 words
- Testing data: List 1
    - 2309 goal words



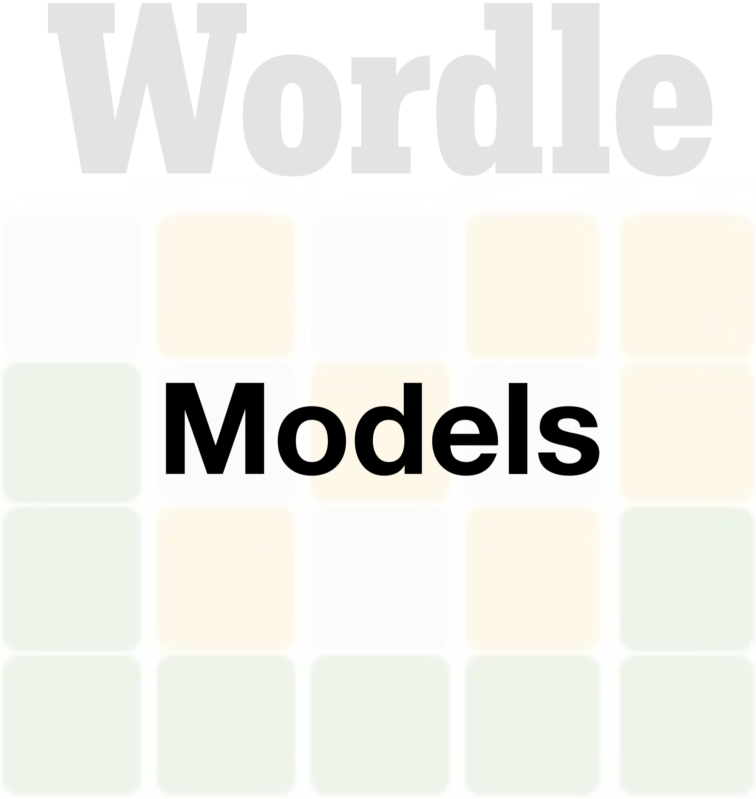
```
        return {
            type: B
        }
    }
      , $ = function(e) {
        return {
            type: W,
            payload: A(A({}, e), {}, {
                timestamp: Date.now()
            })
        }
    }
      , ee = function(e, t) {
        return {
            type: Y,
            payload: {
                label: e,
                sendContext: t
            }
        }
    }
      , te = ["cigar", "rebut", "sissy", "humph", "awake", "blush", "focal", "evade
      , ae = ["aahed", "aalii", "aargh", "aarti", "abaca", "abaci", "abacs", "abaft
    var ne = {
        unknown: 0,
        absent: 1,
        present: 2,
        correct: 3
    };
    var oe = 864e5
      , re = new Date(2021,5,19,0,0,0,0);
    function se(e, t) {
        e = new Date(e),
        e = new Date(t).setHours(0, 0, 0, 0) - e.setHours(0, 0, 0, 0);
        return Math.round(e / oe)
    }
    function ie(e) {
        if (null === e)
            return "";
        e %= te.length;
        return te[e]
    }
    function le(r) {
        return function(e, t) {
            var a = t()
              , n = pe(a)
              , o = me(a)
              , t = be(a)
              , a = je(a);
            if (r === n - 1 && (e({
```

Javascript code of Wordle from https://www.nytimes.com/games/wordle/index.html

Some samples

- aahed
- aalii
- aargh
- aarti

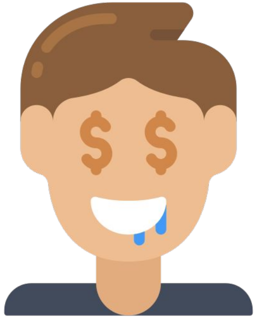
- cigar
- rebut
- sissy
- humph



Models

Quick Overview

Reinforcement Learning

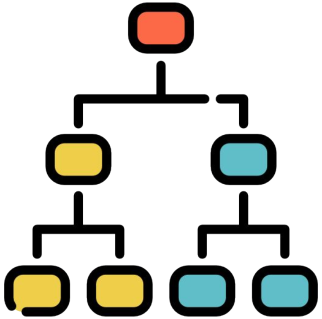


Search Algorithm

Reinforcement Learning

3 Variations

- Q-Learning
- Q-Learning with Hierarchical Clustering
- Modified Q-Learning with Hierarchical Clustering



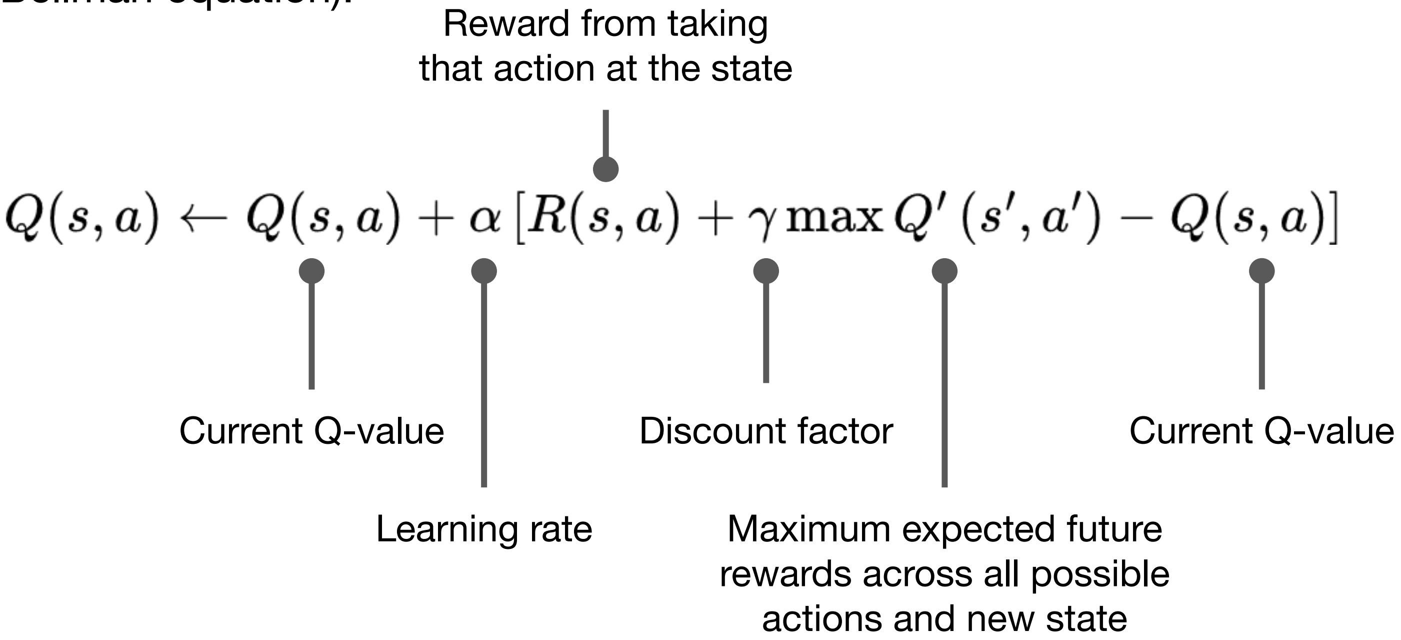
1) Q-learning: Q-table

Q-table: $Q(s,a)$ consists of

- Actions, a : Next word guess from the corpus
- States, s : List of words from the entire corpus of 15283 words

1) **Q-learning:** Q-function

Q-learning (Bellman equation):



Reward from taking that action at the state

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max Q'\left(s',a'\right) - Q(s,a)\right]$$

Current Q-value

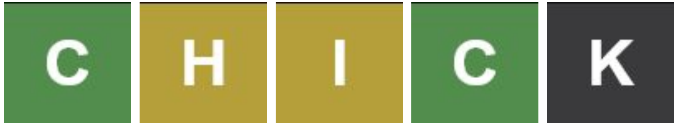
Learning rate

Discount factor

Maximum expected future rewards across all possible actions and new state

Current Q-value

1) Q-learning: Reward System



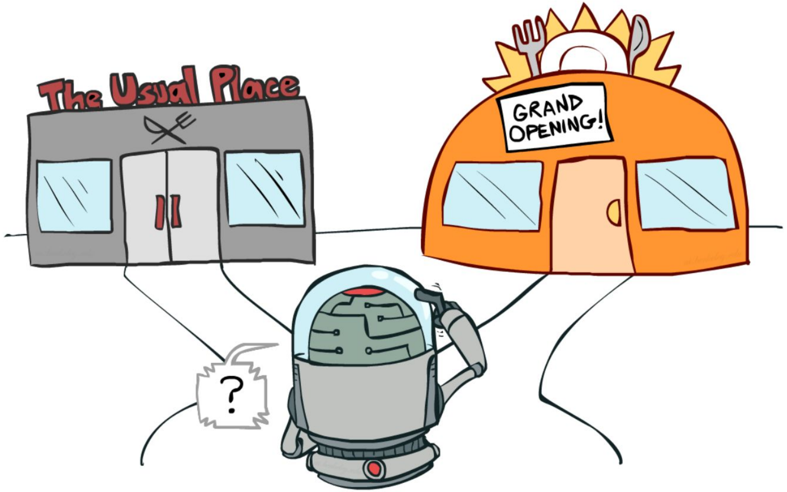
Action:	C	H	I	C	K		
Reward:	+10	+5	+5	+10	-1	=	+24

- Higher score assigned to green letters than yellow letters
- Negative score assigned to black letters as it wastes a guess
- Reward for playing this action is updated and recorded in Q-table

1) Q-learning: Epsilon-Greedy

Choose an action at random instead of exploiting previous knowledge through the Q-table

- ϵ : probability of selecting a random action and exploring



1) Q-learning: Decaying Epsilon

Given that for each iteration, the goal is to guess the goal word within 6 tries

- Want to explore in first few tries
- Start to exploit only in later tries

Solution: Decaying epsilon using an inverse square function

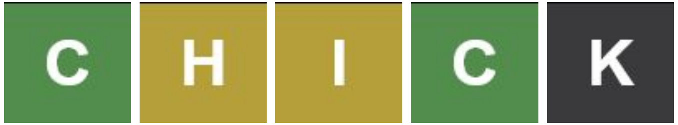
$$\varepsilon \leftarrow \frac{\varepsilon}{(\,stepnumber)^{\,2}}$$

1) Q-learning: Filtering

Hard-mode: Next guess has to use the color hints from the previous guess

Solution: Words no longer feasible are removed from the search space

- Green letters not at current position
- Yellow letters in current position
- Black letters present



1) **Q-learning:** Limitations

- Huge state space/Q-table: 15283 x 15283
- After each iteration of Wordle, Q-table is re-initialized to 0
- No carrying over of Q-table values, due to the filtering of state space

Solution: 2nd Variation - Addition of clustering algorithm

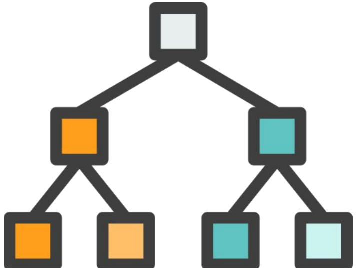
2) Q-learning with Hierarchical Clustering

Q-table: $Q(s,a)$ consists of

- Actions, a : Next cluster to select next word from
- States, s : List of cluster number

Clustering

- Reduces the size of state space
- Able to learn from previous iterations Q-tables
- Constant Q-table size, not affected by filtering



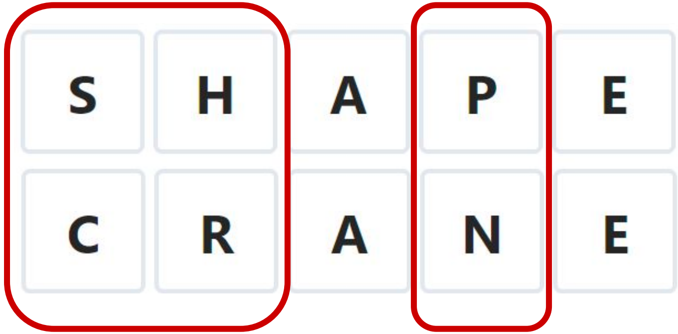
Why Hierarchical Clustering

- Sentiment meaning of words have 0 effect on guesses
- Tree structure more logical than a word vector space
- Bottom-up approach of grouping similar words together

Solution: Agglomerative hierarchical clustering + Levenshtein distance measure

Levenshtein distance

- Minimum number of single-character edits required to change one word to another



S	H	A	P	E
C	R	A	N	E

Example: Levenshtein distance = 3

2) Q-learning with Hierarchical Clustering

Everything else remains the same

- Reward system
- Epsilon-greedy policy
- Decaying epsilon
- Filtering function on corpus

3) Modified Q-learning with Hierarchical Clustering

- Since the daily wordle only comes from the 2309 goal words
- Solving this specific game only, and not a general 5-letter guesser

Modification:

- Training data: List 1 + ~~List 2~~
    - Only train on the 2309 goal words
- Testing data: List 1
    - 2309 goal words

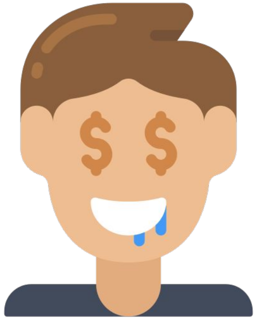
Reinforcement Learning Strategy

1. Initialize a *Wordle* class which creates the Q-table and randomly selects a daily wordle from the list of goal words
    a. Initialize a *Clustering* class to cluster on the corpus (for clustering version)
2. Uses the Q-learner to select the next guess
3. Passes the guess to an *Evaluation* class which gets the scoring and does the filtering
4. *Evaluation* class then passes the reward to update the Q-table for the next guess
5. Each run terminates when the goal word is reached
6. Reinitialize Q-table to 0 (for non-clustering version) or pass the Q-table to next run (for clustering version)

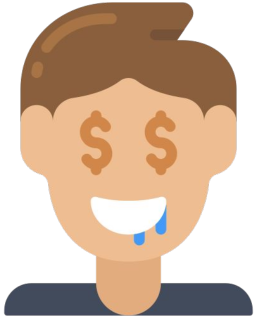
Search Algorithm

2 Variations

- Greedy-search
- Modified Greedy-search



1) Greedy Search



- Actions:
  - List of words from the entire corpus of 15283 words
- States:
  - Next guess from the same corpus
- Scoring:

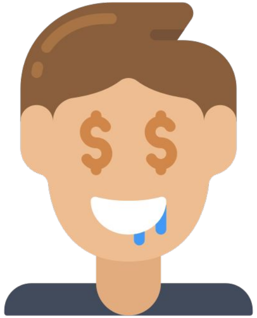
Score each letter of the alphabet by frequency of occurrence in the corpus

→

Score each word by summing up the scores of each of its letters

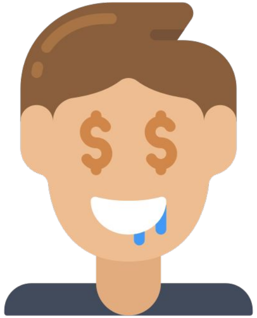
1) Greedy Search

- Evaluation:
  - Evaluate green → yellow → grey letters to choose
    - Node with the lowest heuristic
    - Closest to the presumed goal state



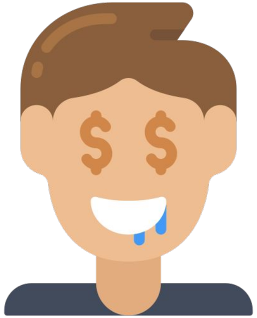
Common limitations of Greedy Search

- Not optimal
  - Looks for the optimal local solution at every step
  - Cannot guarantee finding an optimal global solution
- Not complete
  - Might get stuck in a loop

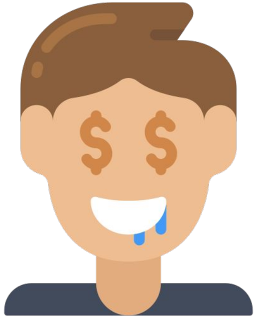


1) **Greedy Search:** Limitations

- Not optimal
  - Cannot guarantee finding the goal word within the 6 tries
- ~~Not complete~~
  - Due to the nature of Wordle, the algorithm will always run to completion and will not get stuck in any loops.



2) Modified Greedy Search



Modification:

Similar to the reinforcement learning case

- Training data: List 1 + ~~List 2~~
  - Only train on the 2309 goal words
- Testing data: List 1
  - 2309 goal words

Search Algorithm Strategy

1. Randomly selects a daily wordle from the list of goal words
2. Initialize our word dictionary based on occurrence of words
3. Evaluates guess result and selects best option
4. Repeat step 3 until goal word is reached for termination

Summary of our 5 Models

Reinforcement Learning

- Q-learning
- Q-learning + hierarchical clustering
- Modified Q-learning + hierarchical clustering

Search Algorithm

- Greedy search
- Modified Greedy search

Evaluations

Overview of Evaluations

1. **Grid Search**
   - To find optimal hyper-parameters
2. **Simulation analysis**
   - To analyze performance of models

Grid Search (Reinforcement Learning)

Different combinations of hyper-parameters

- Learning rate: [0.1, 0.01, 0.001]
  - Typical learning rates used
- Discounting factor: [0.5, 0.6, 0.7, 0.8, 0.9]
  - Our RL model is more focused on earlier rewards (from earlier guesses)
  - Instead of typical values of 0.99 or 0.9, lower discount factors are better
- Exploration rate: [0.5, 0.6, 0.7, 0.8, 0.9]
  - Due to decaying exploration rate, the lower bound cannot be too low
- Number of clusters: [6, 7, 8, 9, 10]
  - Arbitrary choice

Grid Search (Reinforcement Learning)

- For each of the RL variations:
  - Ran 100 simulations (runs of Wordle) for each hyperparameter combination
- Evaluate on 3 performance metrics
  - Win rate (Percentage of attempts with < 7 guesses)
  - Average number of guesses
  - Time taken for each Wordle run

Grid Search (Reinforcement Learning)

- Results of grid search, best combinations of hyperparameters

	Q-learning	**Q-learning + Clustering**	**Modified Q-learning + Clustering**
Learning rate	0.1	0.1	0.001
Discounting factor	0.8	0.9	0.9
Exploration rate	0.8	0.5	0.9
Number of clusters	-	6	9

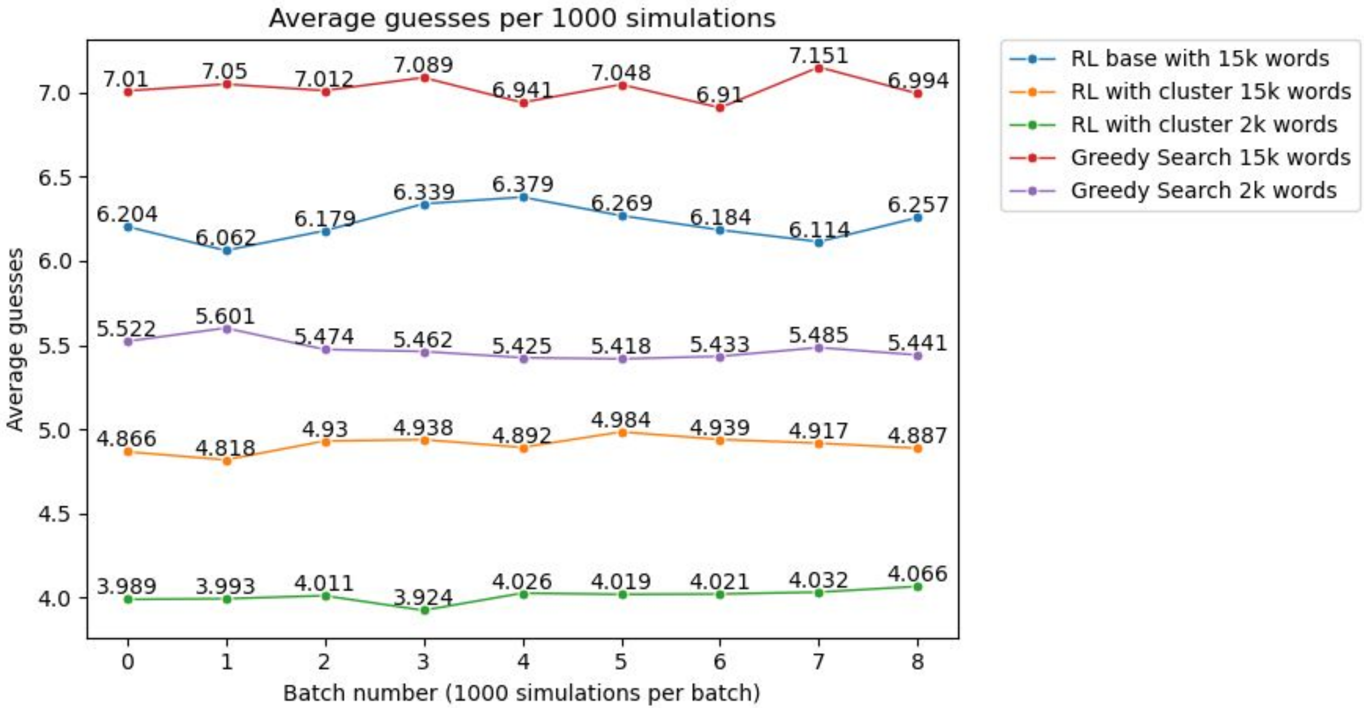
Simulation Analysis

With the found hyperparameters, to compare model performances

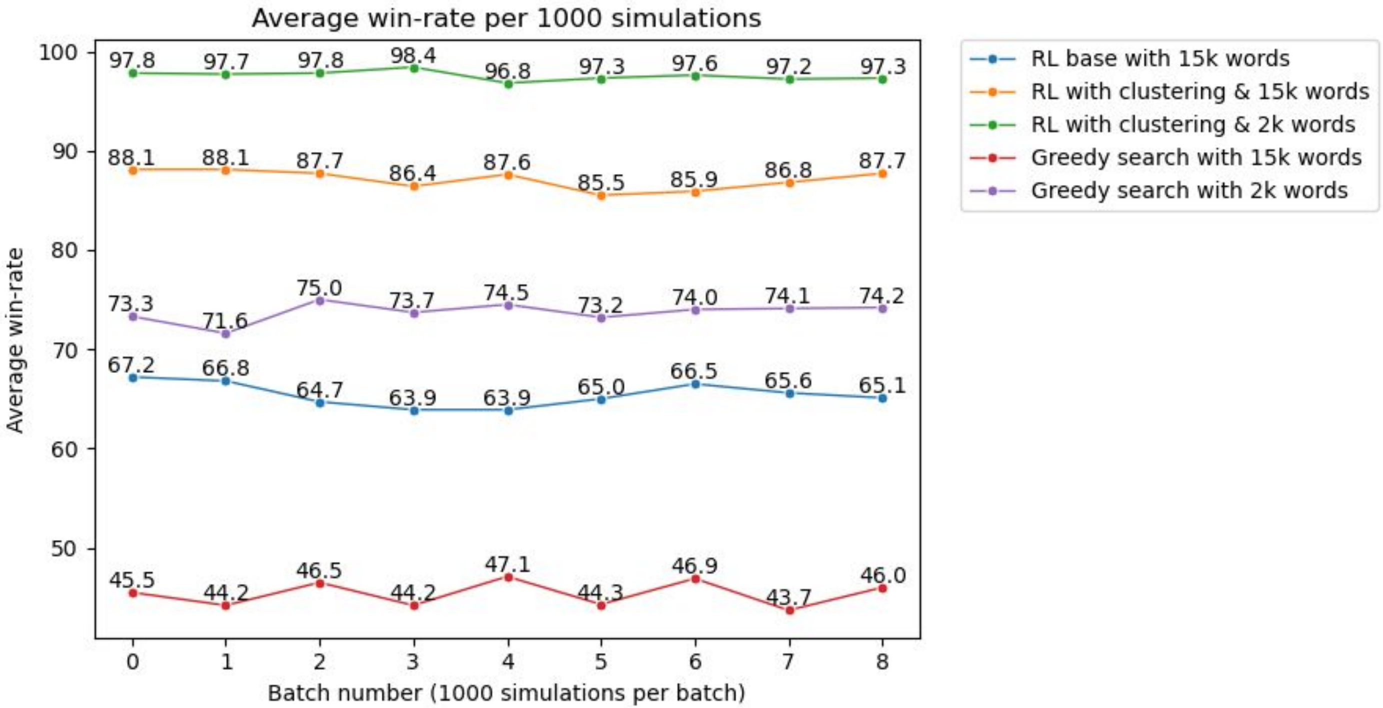
For each of the 5 models:

- Run 10,000 simulations
- Did a batch-means (size of 1000) of the number of guesses and win-rate
- Discard the first batch, to remove any initialisation bias
- Estimate mean and variance of each performance metric
- Compute the 95% confidence intervals

Results: Average number of guesses



Results: Average win-rate



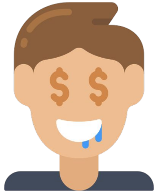
Results: Confidence Intervals

	95% CI of average win-rate/%	**95% CI of average number of guesses**
Q-learning	(64.4, 66.5)	(6.213, 6.230)
Q-learning + Clustering	(86.4, 87.8)	(4.906, 4.910)
Modified Q-learning + Clustering	**(97.4, 97.7)**	**(4.008, 4.010)**
Greedy Search	(44.1, 46.6)	(7.019, 7.027)
Modified Greedy Search	(73.0, 74.4)	(5.471, 5.476)

Summary of Findings

Reinforcement Learning

- Requires less guesses
- Higher win-rate
- Clustering improves its performance



Search Algorithm

- Takes less time to run

Training on only the goal words gives consistently better performance for both

Best Model

Reinforcement Learning

- Q-learning
- Q-learning + hierarchical clustering
- **Modified Q-learning + hierarchical clustering**

Search Algorithm

- Greedy search
- Modified Greedy search

To be used for our daily wordle solver GUI

Graphical User Interfaces

Overview of GUIs

1. **Kivy GUI**
   a. Visualize differences in performance of different models
2. **PyGame GUI**
   a. Observe an agent solving the daily Wordle

1) Kivy GUI

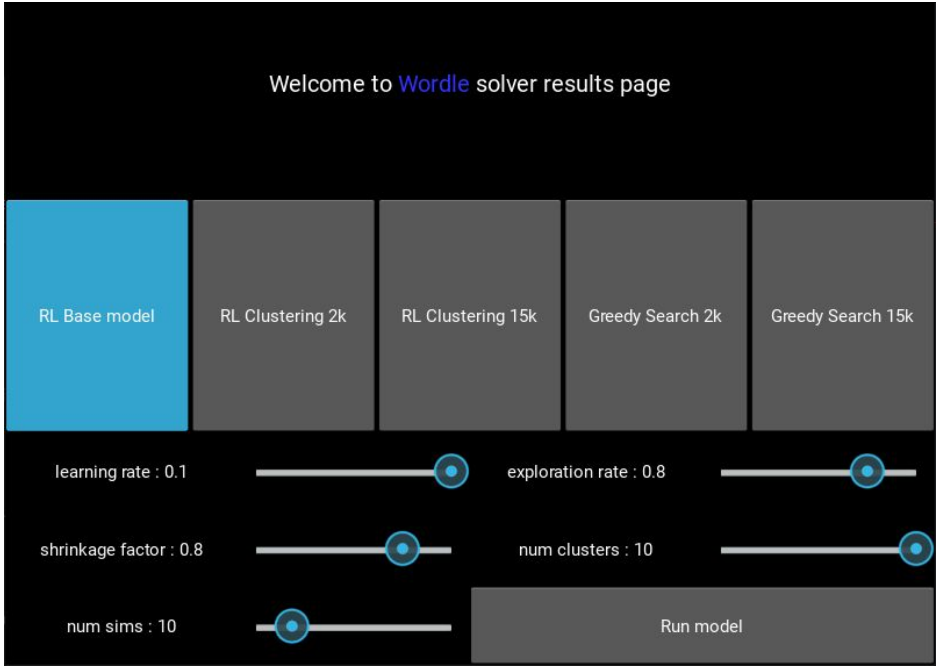
Visualize the difference in performances of different models

- Model selection
- Custom parameter values
- Graphical visualisation

1) **Kivy GUI:** Model Selection

Select from the five models:

- Q-Learning
- Q-Learning + Clustering
- Modified Q-Learning + Clustering
- Greedy search
- Modified Greedy search



Welcome to Wordle solver results page

learning rate : 0.1

exploration rate : 0.8

shrinkage factor : 0.8

num clusters : 10

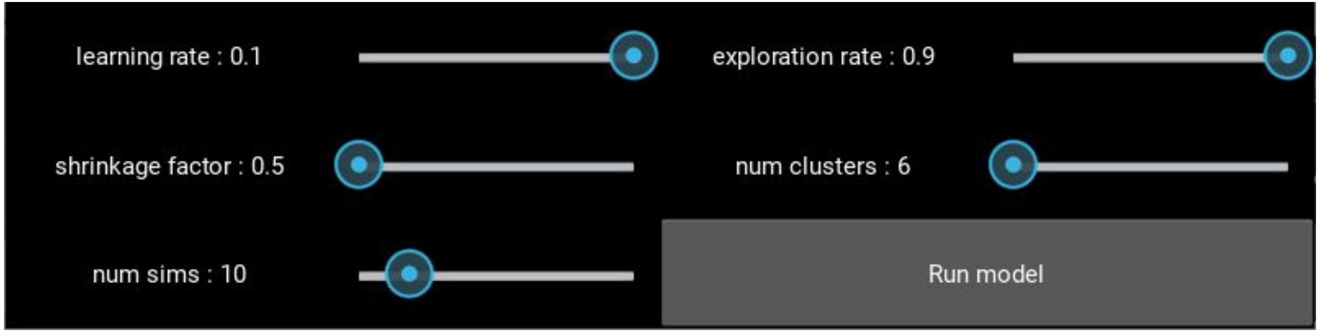
num sims : 10

Screenshot of loading page of Kivy GUI

1) **Kivy GUI:** Customer Parameter Values

Results from grid search used as default settings in our GUI

- User can try out different settings and observe the performance



learning rate : 0.1

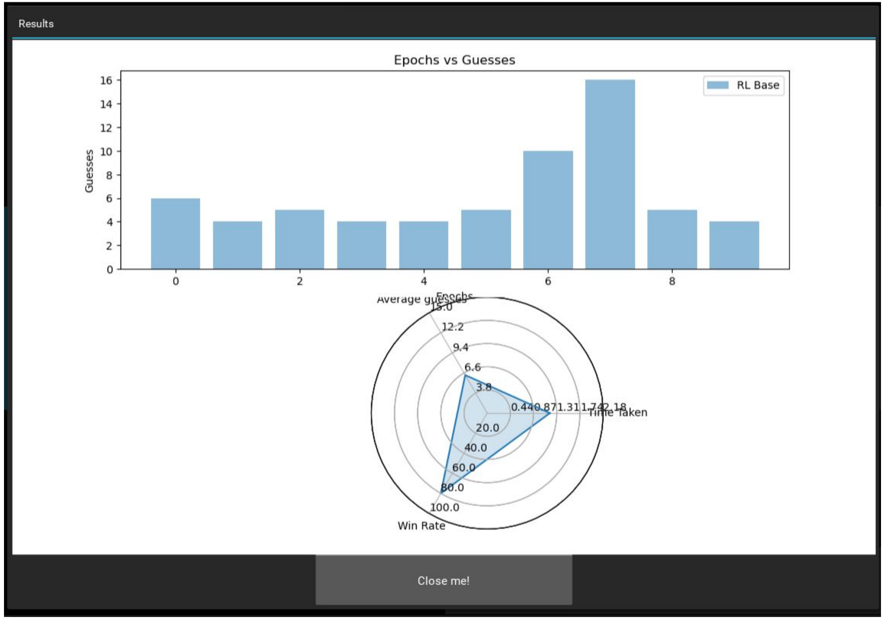
exploration rate : 0.9

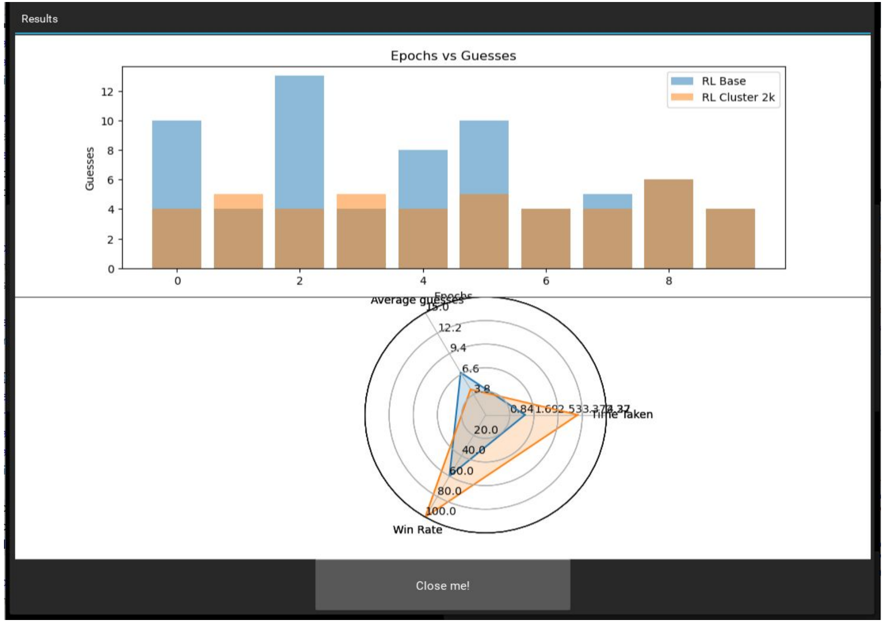
shrinkage factor : 0.5

num clusters : 6

num sims : 10

1) **Kivy GUI:** Visualisation & Comparison

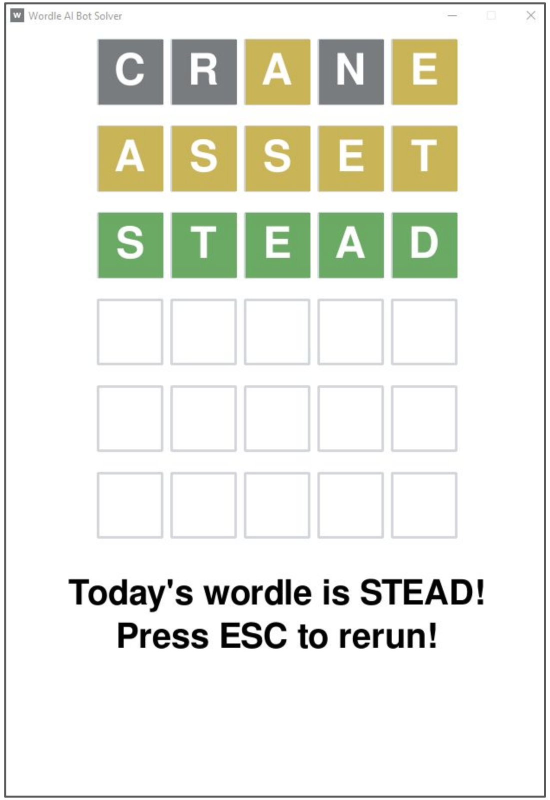




Two screenshot of results page, with 1 run and 2 runs (for comparison) respectively

2) PyGame GUI

- Observe an agent solving the daily Wordle in hard-mode
- Keep pressing “enter” to play
- Press “esc” to rerun the solver



C R A N E

A S S E T

S T E A D

Today's wordle is STEAD!
Press ESC to rerun!

Screenshot of PyGame GUI Daily Wordle Solver

Improvements

Possible Improvements

- Deep Q-learning
  - Might be an overkill given already how well our model performs, ~97% win-rate
- Discretization of words within clusters
  - To form a probability distribution within each cluster to select the next guess from
- Generalize for other wordle variations
  - E.g. 6 letters

Demonstration

https://github.com/Samthesimpsons/AI-Reinforcement-Learning-Project

Wordle

Thank You!

QnA