Five O'word

By Jefferson Koo Kurniadi (2902669816) L1AC

Overview

Five O'word is a game inspired from the game "Wordle" which is a game where the game program will pick a random word from a list of words that are 5 letters long, hiding it and letting the player guess the word in a limited number of tries given according to the difficulty chosen.

The main goal of Five O'word is for the player to guess the random 5 letter word taken from a list of words within the number of tries which depends on the difficulty selected by the player.

The game will respond accordingly to the inputs of the player where it will not allow any words less or more than 5 letters long and non-words, after a valid guess the game would respond the player by checking if the word typed in is correct with the random word, if the guess is not correct the program would check which letters are in the word and check whether they are in the same position of the random word where when it is done it will show the player all the letters not in the random word, letters in the random word, and letters in the random word in the correct position, this mechanic is what will help the player further refine their guess as to what the random word will be until either the player guessed correctly or run out of tries.

Introduction

The reason I made a Wordle-like game for my project is due to me asking my online friends for ideas on what I should make for my Final Project using Python, and a few months before that everyone in the server is playing Wordle almost daily, so one of my online friends responded with "hey why not make a Wordle game?" and that idea instantly clicked into me as a great idea for my first Final Project; a simple game that I know and familiar with.

After deciding on a Wordle like game, I made simple notes on how I should make the game mechanics and the ideas on how to make the game different from the original Wordle game. I made and figured out how I should make the bare game first and making sure it worked then started on how to make the game different to which I landed upon a difficulty selector for my version of the game.

Project Goal

The goal of this project is to make a Wordle-like with Python that is simple in nature where everyone can play Wordle without much hassle and the limit of once a day (without an account on the official "The New York Times").

Statement on the use of Artificial Intelligence

The AI tool used in this project is:

- ChatGPT 5.1 Auto (Plus)

AI tool is used in this project in different ways such as testing ideas if they work or not, debugging, word formatting (formatting words taken from a dictionary into a list), and in the specific def "definer" and "arranger" functions, which would be line 6 - 35 and line 92 - 93 of game_easy.py, game_medium.py, and game_hard.py files of the game.

The AI generated code for the "definer" and "arranger" functions had a few iterations where I tested them each for their complexity, usability, reliability, and how it is implemented, after a few trial and errors I pick the simpler one out of the few that could be integrated into the code that is now currently being used in the game.

One of the limitations or struggles is where the AI could not understand and overcomplicate the problem thus giving me a response that is not fit or not what I needed it to be in the game, this in turn required me to tweak and refine each output for a better result where it is now implemented as a simple solution put behind fail safes where any unintended output will not be processed by the "definer" and "arranger" functions.

The use of AI in this project is used more as a support tool for idea refinements and formatting, where things such as testing, code integration, debugging, and other QA related stuff is done by myself.

Examples of the prompts used are:

- "Can I make different files for each game mode to separate them for easier navigation and use the import module so that the game will run that file when selected"
- "Return the list of words in the format like
  ["WORDS",
  "WORDS",
  …
  ]

- “How to make the game know if the letters are correct and in the right spot”

Project Specification

Game Mechanics:

- The game will pick a random word from a list of predefined words given in Word_list.py
- The player will get a limited amount of tries to guess the random word
- The amount of tries the player gets is equivalent to the difficulty they choose (easy, medium, and hard)
- UPPERCASE and lowercase characters would be processed the same by the game

Game Rules:

- The player must input a 5 letter word; no more no less
- Any invalid inputs such as
    - Non-words (random string of letters)
    - Numbers
    - Words with more or less than 5 letters

  Will not count towards any of the tries given and the player would be prompted to try another guess
- The game will end either if the player guesses the word or when the player runs out of tries
- After a valid input the game will give feedback to the player based on:
    - Letters present in the word and in the correct position with “[]” (e.g., [A])
    - Letters present in the word but not in the correct position with “()” (e.g., (A))
    - Letters not present in the word will not be encased and shown as is (e.g., A)

Game Difficulty System

- The game has 3 difficulty levels:
    - Easy - providing 7 tries, good for beginners
    - Medium - providing 5 tries, a bit challenging, good for intermediate players
    - Hard - providing 3 tries, for ones looking for a challenge, hard even for the pros

Game Constraints

- Only 5 letter English words from a predefined dictionary from dictionary.com as of November 2025
- The list of random words are limited to the latest 12 December 2025 of rockpapershotgun.com

Game Technical Specifications

- Programming language: Python
- Code structure:
    - game_easy.py
    - game_hard.py
    - game_medium.py
    - main.py
    - word_check.py
    - word_list.py

  The “source.txt” file is not a file needed nor used in the game and is only there to serve the purpose of showing the user where the words for the dictionary and random words are taken from
- Running Environment
    - VS Code with the standard Python interpreter

Instructions:

1. Download and extract all the files from the Github Repository
2. Make sure all the files are in the same folder
3. The folder should have the files main.py, game_easy.py, game_medium.py, game_hard.py, word_check.py, and word_list.py. (reminder: the source.txt file is not part of the game and is there to serve as information as to where the words are sourced from, and could be deleted with no repercussions)
4. Run the game in the main.py file through the terminal, the game should now be able to be played

Note: When running the game for the first time of each session an “error” will show up as “KeyboardInterrupt” for a split second, this is within spec and will not affect your gameplay nor will it show up again during your session, the error message can be ignored.

How to Play:

After doing the initial set up run the game in terminal by running the game as any program in the main.py file, follow the instructions shown on screen.

Simply pick the level of difficulty and guess the word

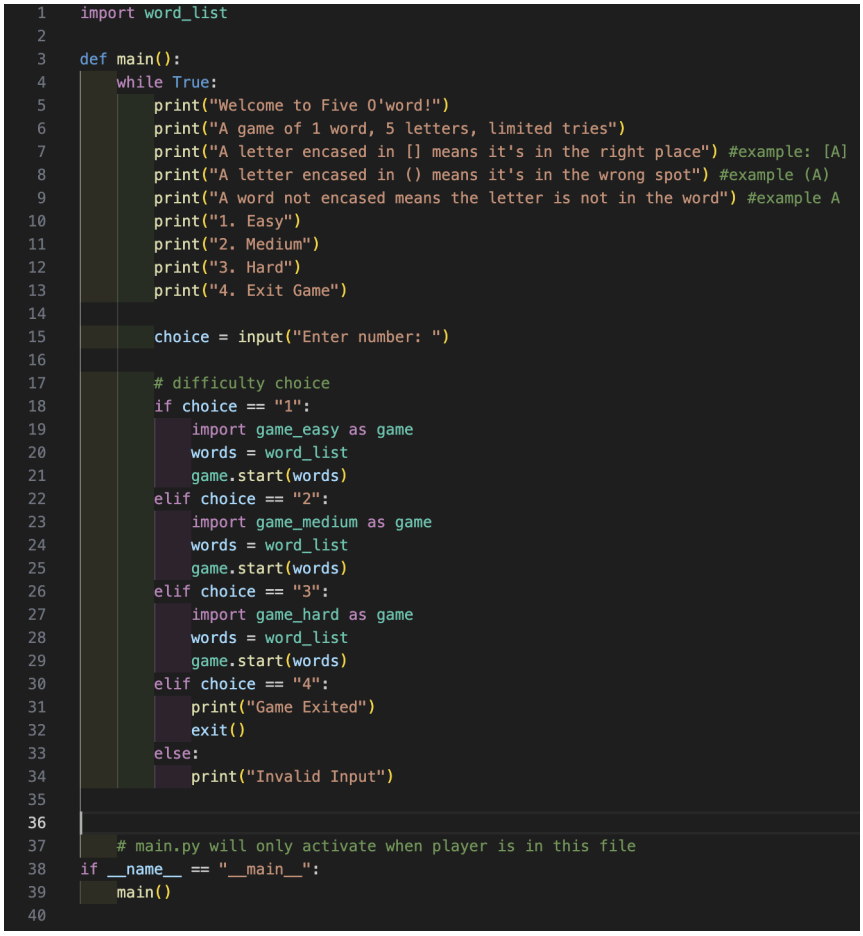
Game Architecture Design

This game is divided into 4 main components:

- main.py
- Game difficulty (game_easy.py, game_medium.py, game_hard.py)
- word_check.py
- word_list.py

main.py

This file is where the player will first interact with when starting the game.



```python
import word_list

def main():
    while True:
        print("Welcome to Five O'word!")
        print("A game of 1 word, 5 letters, limited tries")
        print("A letter encased in [] means it's in the right place") #example: [A]
        print("A letter encased in () means it's in the wrong spot") #example (A)
        print("A word not encased means the letter is not in the word") #example A
        print("1. Easy")
        print("2. Medium")
        print("3. Hard")
        print("4. Exit Game")

        choice = input("Enter number: ")

        # difficulty choice
        if choice == "1":
            import game_easy as game
            words = word_list
            game.start(words)
        elif choice == "2":
            import game_medium as game
            words = word_list
            game.start(words)
        elif choice == "3":
            import game_hard as game
            words = word_list
            game.start(words)
        elif choice == "4":
            print("Game Exited")
            exit()
        else:
            print("Invalid Input")


    # main.py will only activate when player is in this file
if __name__ == "__main__":
    main()

```

This file is where the main menu is located, it holds the main menu for the game where it shows the player how to play the game and the difficulty choices.

How it works:

```
1    import word_list
```

It first starts with the “import word_list” module, this module will load the word_list.py file which holds all the words that are possible to be randomly selected by the game.

Then the “def main” function would be defined.

```
3    def main():
```

This line of code by itself will not do anything and would require another line of code to run it, it would need the code in line 38 and 39.

```
38   if __name__ == "__main__":
39       main()
```

This code is responsible for making the game run.

This specific line of code is responsible for preventing this file to be run anywhere else that is not intended.

This two lines of code works by first having the user be in the main.py file, and if the player is in the main.py file it would run the file directly thus activating the “def main()” line prompting the program to continue the code to the “while True” code.

```
4        while True:
```

The “while true” code is essentially a loop that will turn on indefinitely until the player exits the game.

```
5       print("Welcome to Five O'word!")
6       print("A game of 1 word, 5 letters, limited tries")
7       print("A letter encased in [] means it's in the right place") #example: [A]
8       print("A letter encased in () means it's in the wrong spot") #example (A)
9       print("A word not encased means the letter is not in the word") #example A
10      print("1. Easy")
11      print("2. Medium")
12      print("3. Hard")
13      print("4. Exit Game")
14
15      choice = input("Enter number: ")
16
```

This line of code would be used to output the text such as game title, instructions, and options to the player so that the player knows what to do next. When the program goes into the "while True" and the loop runs these sentences would be printed again until the player exits the game.

The "choice" input in line 15 is for the user to respond to the program, given the on-screen instructions the player would type in the specific keywords 1, 2, 3, or 4 which will correspond to the next block of code.

```
17      # difficulty choice
18      if choice == "1":
19          import game_easy as game
20          words = word_list
21          game.start(words)
22      elif choice == "2":
23          import game_medium as game
24          words = word_list
25          game.start(words)
26      elif choice == "3":
27          import game_hard as game
28          words = word_list
29          game.start(words)
30      elif choice == "4":
31          print("Game Exited")
32          exit()
33      else:
34          print("Invalid Input")
35
```

This block of code works by checking for the player input, if the player types in “1” the first “if” will run, if the player types in “2” the “elif choice == “2”:” code will run, this will continue to the last option of “4” where the game would instantly exit. The last line “else” is there as a way to prevent the player from typing in anything else that is not intended (e.g., anything other than 1, 2, 3, or 4).

Typing in 1, 2, or 3 will make the program run the “import game_easy/medium/hard as game” line, that will load the corresponding difficulty file.
The “import game_easy/medium/hard as game” is a way to shorten the name into just game, instead of typing “game_easy.start” it can be typed as game.start, by using this it simplifies the code as the different difficulties are called using the same code.

Typing in 4 would make the game output “Game Exited” and stop the program from running thus “exiting” the game.

The line “words = word_list” is also another way to simplify the code, since all three difficulties uses the same set of words this would make it easier to use later on in the code.

Game Difficulty (game_easy, game_medium, game_hard)

In the Game Difficulty section all three difficulties would be counted as one, as all three of the difficulties uses the exact same code structure and logic, the main differences between the three would be the outputs such as “Easy difficulty started” being different in the medium difficulty, and the number of tries the players get.

Disclaimer: As a way of simplifying, from here on out game_easy, game_medium, and game_hard will be called as game_difficulty.

```
1    import time
2    import sys
3    from word_check import check_words
```

When each game file is started the first three lines of the code will start, “import time”, “import sys”, and “from word_check import check_words”. Each of these import modules are used for different things.

“Import time” would be used in the game as a way to track how long it takes the player to finish/lose the game, everytime the player gets the correct word or run out of tries the game will record how long it took the player to do it in minutes and seconds.

“import sys” would be used later in the game as a way for the player to exit the game once he finishes the game or run out of attempts.

“from word_check import check_words” is another module that would be used in the game, it is used for checking the validity of the word inputted by the player.

The exact use and how “import sys” and “from word_check import check_words” would be explained later on in this section of this report.

```
40    def start(words):
41        secret = words.random_word()
42        max_try = 0
43        start_time = time.time()
44        print("Easy difficulty started!")
45        print("You have 7 tries.")
46
```

After the “game.start(words)” line executes this section inside the game_difficulty, the game will output the difficulty mode selected and the amount of tries given once. During this the game will pick one random word from word_list which is now shortened to just “words” from the line “words = word_list” in the main.py file. The random word selected would be the secret word which the player must guess within the amount of tries given.

```
47
48    while max_try < 7:
49        player_guess = input("Input your word: ")
50        guess = player_guess.upper()
51
```

After the initial output the “while max_try < 7:” will start, it is a loop where as long as the max try is below 7 (this number depends on the difficulty) the game will keep on looping/running.

The “player_guess” is a way for the player to input their guesses into the game, and the “guess = player_guess.upper()” makes the player input into uppercase, this is intentional as for simplification every word in word_list.py and word_check.py is uppercase, so as to prevent any bugs, error, and confusion for the player all the words will be set to uppercase regardless.

```
51
52          # require 5 letter words no more no less, does not include into tries
53          if len(guess) != len(secret):
54              print(f"Please enter a {len(secret)}-letter word.")
55              continue
56
```

This block of code is responsible for filtering out words that are either too short or too long, the word length of the guess should always be the same as the word length of the secret word. Inputting a word that is either too long or too short does not count towards the tries given

The “len()” is used to see how many letters the word has, originally the idea for this game is to have different word lengths depending on the difficulty, this idea was scrapped and the code for it is not changed and remains, this opens the possibility to later add a longer or shorter length of word in the future.

If the word length of the player guess is equal to the secret word the game will continue on to the next line.

```
56
57          # make sure the player can't put in random words
58          if guess not in check_words:
59              print("Sorry, word is not in our dictionary")
60              print("Try again")
61              continue
62
```

This block of code is what handles the validity of the word, using the “from word_check import check_words” in line 3 of game_difficulty it works by loading in the list of words from the word_check dictionary and check if the inputted word by the user is in the dictionary. Without this block of code the player can type in any string of 5 letters like “aiueo” and it will be processed, in this game it would count towards cheating and making the game much easier.

Now with this implemented only words in the word_check dictionary can be used as the user guess and any word that is not in the dictionary would not be accepted by the game.

The “from word_check import check_words” is intentionally at the start of the program for the reason that with it being at the start, the whole dictionary is only needed to be loaded once instead of loading it again after every attempt, this saves time and computational power, making the game run faster.

Typing and inputting an invalid word does not count towards the max tries.

```
63          # win condition
64          if guess == secret:
65              end_time = time.time()
66              time_taken = end_time - start_time
67              total_seconds = int(time_taken)
68
69              minutes = total_seconds // 60
70              seconds = total_seconds % 60
71
72              print(f"Correct! time taken is {minutes} minutes and {seconds} seconds")
```

As for the win condition, it is processed by this block of code where after passing the word filters it will check whether or not the player’s guess is correct and match the random word, if so the game will stop and calculate using the “import time” module in line 1 of “game_difficulty”, it will count starting from when the player starts the difficulty level until the time the player gets the word correctly.

After the process is done the game will continue into the next block of code.

```
73          while True:
74              print(" ")
75              print("Play again? ")
76              print("1. Yes")
77              print("2. No")
78              print("3. Return to main menu")
79              decide = input()
80
81              if decide in ("1", "Yes", "yes"):
82                  return start(words)
83              elif decide in ("No", "no", "2"):
84                  sys.exit()
85              elif decide in ("Return", "Return to main", "Return to main menu", "3"):
86                  return
87              else:
88                  print("Invalid Input")
89
```

The game will go into another loop where the player can choose what to do next, play again, exit game, or go back to the main menu.

It works similarly to the main.py menu where the player must input accordingly to what is shown. 1 to play again, 2 to exit the game, and 3 to return to main menu, the game will then check if the player's input is valid or not and use the if to filter out what is typed, and if none matches the game will treat it as invalid; notifying the player and repeating the options again.

The way how the game exits in this menu uses "sys.exit()" which is different to how the main menu works as the main menu uses just "exit()" the reason for this is that when "exit()" is used in the game_difficulty a problem shows up where the game would go back to the main.py menu, this is unprecedented as it would nullify the third option of the menu, and "import sys" in line 2 of game_difficulty is needed to fix this problem, by using "sys.exit()" it would terminal the program which runs similarly to exiting the game.

```
89
90          else:
91              # passes a wrong guess to the top def definer and arranger
92              feedback = definer(secret, guess) # This line is part of def definer
93              arranger(guess, feedback) # This line is part of def arranger
94
95              max_try += 1
96              tries_left = 7 - max_try
97
98              if max_try < 7:
99                  print(f"Incorrect, you have {tries_left} tries left")
```

Now if the player's guess is not correct it would pass on into this block of code where the player's guess would be processed even further using def "definer" and def "arranger".

For every guess that passes the word filters that does not match the correct word a counter will count how many times the player has input a wrong answer, the game will later check if the player has surpassed the limit or not.

```
6   def definer(secret, guess):
7       result = ["absent"] * len(secret) # makes the guessed word into separate characters like ["absent", "absent"] depending on how long the word is
8       secret_list = list(secret)
9
10      # checks for characters that are correct place
11      for i, ch in enumerate(guess):
12          if ch == secret[i]:
13              result[i] = "correct"
14              secret_list[i] = None
15
16      # checks for character in word but not in place
17      for i, ch in enumerate(guess):
18          if result[i] == "correct":
19              continue
20          if ch in secret_list:
21              result[i] = "present"
22              secret_list[secret_list.index(ch)] = None
23
24      return result
25
```

The player's guess would then be passed on first into the def "definer" function, the use of this is for the player feedback on which letters are correct, which letters are in the wrong position, and which letters are not in the secret word.

How it works is by first using the "result = ["absent"] * len(secret)" it splits the letters from the secret word into something like ["absent", "absent", "absent", "absent", "absent"], it would go on for how long the word is, if the word is 5 letters long it would make the list 5 letters long.

After that the block of code in line 11-14 will process the letters from the player guess and matching it with the letters in the secret word, this part will only check for the letters in the correct place in the word by changing the "absent" to "correct" only if the letter of the player guess is in the specific location matches with the letter in the secret word.

Next it would go through the second part of def "definer", this part will check for the letters in the secret word but in the wrong position. It works by skipping the "correct" letters and then checking each letter if it is present in the secret word, if it is "absent" it would be changed into "present".

```
26  def arranger(guess, feedback):
27      out = []
28      for ch, status in zip(guess, feedback):
29          if status == "correct":
30              out.append(f"[{ch}]")   # for word sin correct spot in word
31          elif status == "present":
32              out.append(f"({ch})")   # for characters in word but in the wrong spot
33          else:
34              out.append(f" {ch} ")   # for characters not in word
35      print(" ".join(out))
```

After passing through def “definer”, def “arranger” will be the one that is responsible of formatting it in a way that can be read by the player, it works by changing out every “correct” with the letter encased in “[]”, any “present” to be changed with the letter encased in “()”, and lastly with else, it would only change the “absent” into the letter without it being encased by a bracket.

This process will repeat for how many times the word length is and after every loop it arranges the processed letters with “print(“ “.join(ou))” in something like [A] [B] C D (E) where each encasing would show what each letter means to the player

```
101         # losing condition
102         if max_try == 7 and guess != secret:
103             end_time = time.time()
104             time_taken = end_time - start_time
105             total_seconds = int(time_taken)
106 
107             minutes = total_seconds // 60
108             seconds = total_seconds % 60
109             print(f"Nice try. Time taken is {minutes} minutes and {seconds} seconds.")
110             print(f"The word is: {secret}")
```

If the player gets it wrong for 7 times (number changes depending on difficulty), the game would automatically detect it and prints out the time taken by the player similar to when the player wins, with the difference being it will show what the secret word is to the player.

```
111
112         while True:
113             print(" ")
114             print("Play again? ")
115             print("1. Yes")
116             print("2. No")
117             print("3. Return to main menu")
118             decide = input()
119 
120             if decide in ("1", "Yes", "yes"):
121                 return start(words)
122             elif decide in ("No", "no", "2"):
123                 sys.exit()
124             elif decide in ("Return", "Return to main", "Return to main menu", "3"):
125                 return
126             else:
127                 print("Invalid Input")
128 
```

Similarly to how the win condition works, the game will show the same menu where it gives the user the choice to either play again, exit the game, or return to the main menu.

This block of code is the exact same as the win condition and has no discrepancies.

word_check.py

```
    1 > DICTIONARY = [...
8660 
8661    check_words = set(DICTIONARY)
```

The word_check.py is a list of words named DICTIONARY where it holds all the valid words that are able to be processed by the game.

“check _words = set(DICTIONARY)” changes the dictionary into a set and make it able to be called using the import module, by defining the dictionary into a set and giving it another name it could be called using “from word_check import check_words” and be loaded into the game_difficulty file.

word_list.py

The function of word_list.py is to store all the words that can be used as the secret word, how it works is by putting the words into a list named WORDS .

By using the import random module the program simulates randomness by using the def “random_word” function it selects a random word from the list and forwarding it into the game_difficulty file where the selected word becomes the secret word.

References

Source for all the list of 5 letter words:
https://www.dictionary.com/games/word-finder/5-letter-words

Source for all the word used for the secret word:
https://www.rockpapershotgun.com/wordle-past-answers

AI tool used: OpenAI. (2025). ChatGPT (5.1 Version) [Large language model].https://chat.openai.com/.