LAPORAN PROYEK

Permainan Wordle dengan Greedy Best First Search

Anggota:

11S22010 – William Julio Napitupulu

11S22021 – Rachel Putri Nababan

11S22024 – Gracesia Romauli Marbun

11S22034 – Joel Chandio Aritonang

11S22041 – Viktris Maria K Lubis

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO

INSTITUT TEKNOLOGI DEL

Role Anggota:

Nama	Peran
11S22010 – William Julio Napitupulu	● Mencari referensi ● Diskusi terkait topik dan merancang kode program
11S22021 – Rachel Putri Nababan	● Mencari referensi ● Diskusi terkait topik ● Membuat laporan akhir
11S22024 – Gracesia Romauli Marbun	● Mencari referensi ● Diskusi terkait topik dan membantu kode program ● Membuat laporan ToR dan laporan akhir
11S22034 – Joel Chandio Aritonang	● Mencari referensi ● Diskusi terkait topik dan membantu kode program
11S22041 – Viktris Maria K Lubis	● Mencari referensi ● Diskusi terkait topik dan membantu kode program ● Membuat laporan ToR dan laporan akhir

I. Deskripsi Singkat

Pengembangan aplikasi Wordle menggunakan algoritma Greedy Best First Search bertujuan untuk menemukan kata yang tepat dengan cara yang cepat dan efisien. Dalam permainan Wordle, pemain harus menebak kata rahasia berdasarkan umpan balik yang diberikan setelah setiap tebakan. Algoritma ini berperan penting dalam proses pencarian solusi secara sistematis.

Proses dimulai dengan menginisialisasi daftar kata-kata yang mungkin (Open List) dan menetapkan kata target yang ingin dicari. Pada setiap langkah, algoritma memilih kata dari Open List yang memiliki nilai heuristik terbaik, yang dapat dihitung berdasarkan dua faktor: jumlah huruf yang cocok dan posisi huruf yang benar. Nilai heuristik ini dapat dirumuskan dengan:

$$\mathbf{H = C + P}$$

di mana H adalah nilai heuristik, C adalah jumlah huruf yang cocok, dan P adalah jumlah huruf yang berada di posisi yang tepat.

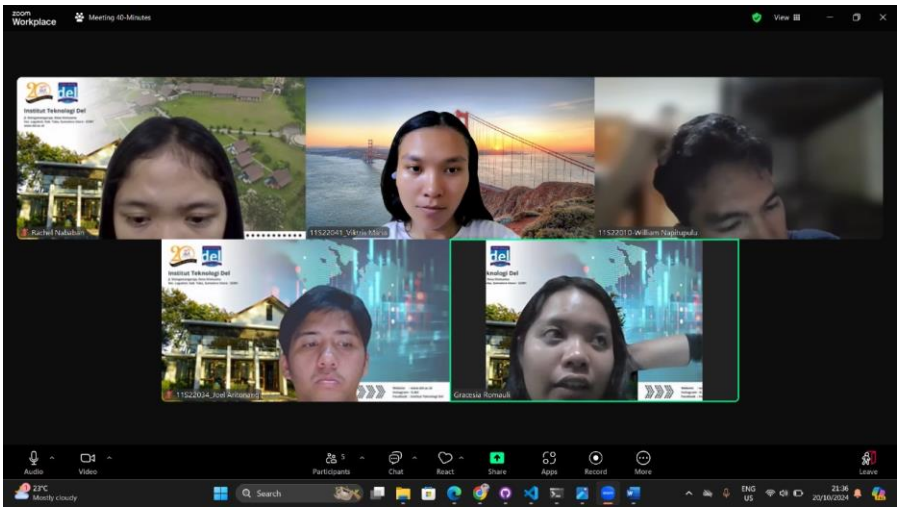
Setelah memilih kata, algoritma akan menganalisis umpan balik yang diperoleh dari tebakan tersebut. Jika huruf tertentu tidak ada dalam kata target, kata-kata yang mengandung huruf tersebut akan dihapus dari Open List. Selain itu, kata-kata lain yang tidak memenuhi kriteria juga akan dikeluarkan. Proses ini diulang hingga kata yang tepat ditemukan atau tidak ada kata lain yang bisa dievaluasi.

Setiap langkah dalam algoritma ini bertujuan untuk mempersempit pilihan kata berdasarkan umpan balik yang diterima. Sebagai contoh, jika tebakan awal adalah "TABLE" dan umpan balik menunjukkan bahwa huruf 'T' dan 'A' tidak ada, huruf 'B' ada tetapi di posisi yang salah, serta huruf 'L' dan 'E' berada di posisi yang benar, maka semua kata yang mengandung huruf 'T' dan 'A' akan dihapus. Algoritma kemudian akan mencari kata lain yang mengandung 'B' tetapi tidak di posisi ketiga, dengan 'L' dan 'E' di posisi keempat dan kelima.

Dengan pendekatan ini, algoritma Greedy Best First Search dapat meminimalkan jumlah tebakan yang diperlukan untuk mencapai solusi, sehingga meningkatkan efisiensi dalam permainan Wordle. Hal ini tidak hanya membuat permainan menjadi lebih menantang, tetapi juga meningkatkan pengalaman bermain secara keseluruhan.

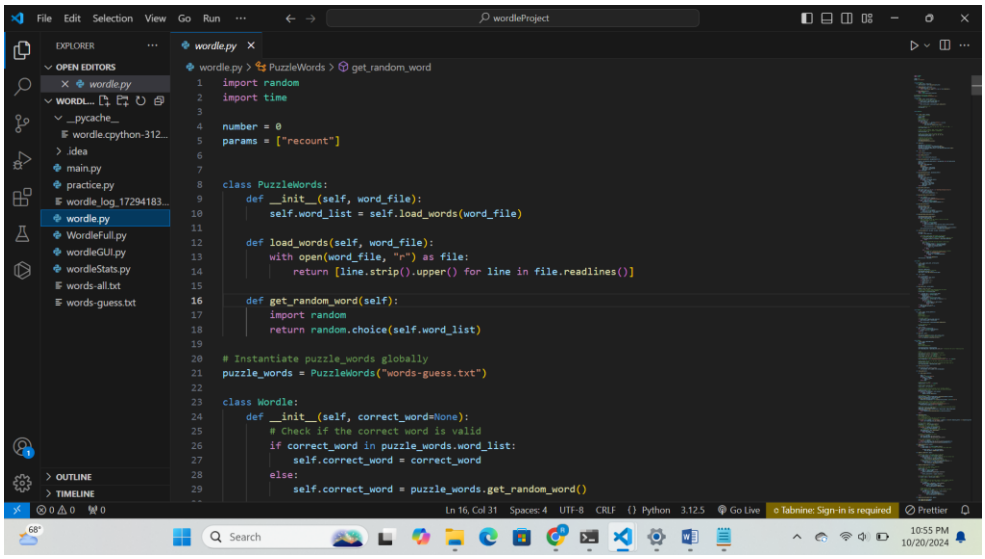
II. LAMPIRAN

Dokumentasi Bukti Kerja kelompok

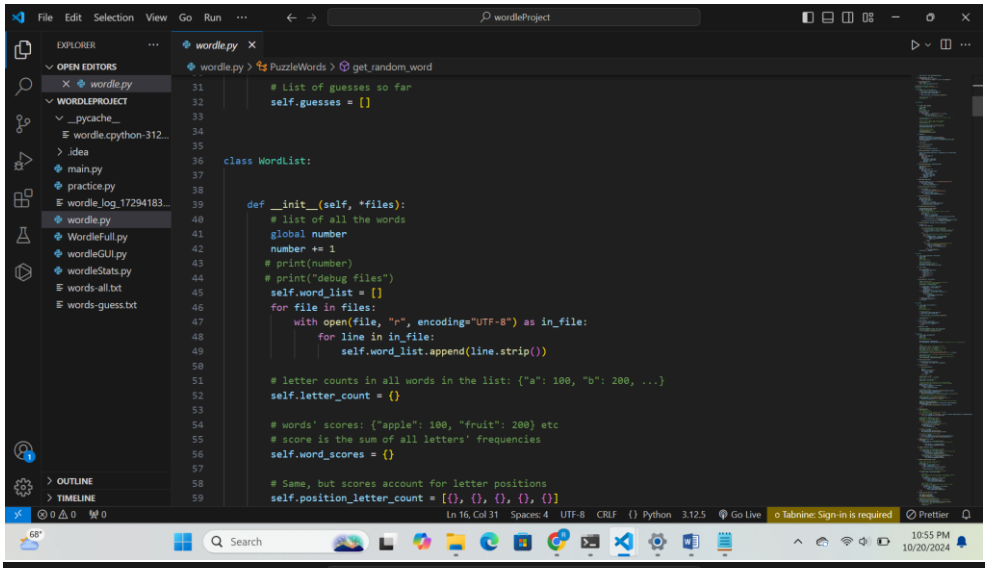


Semua Codenya:

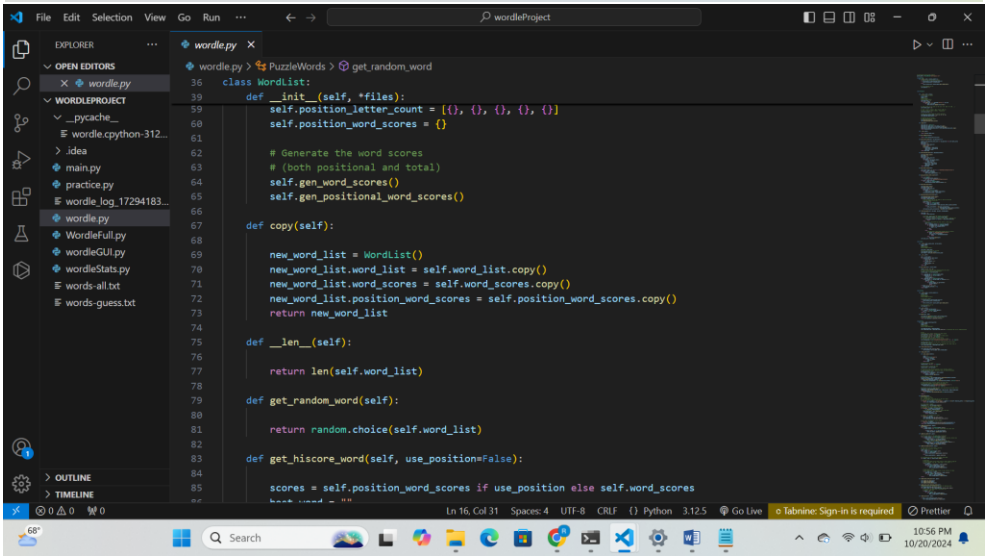
- **Wordle.py**



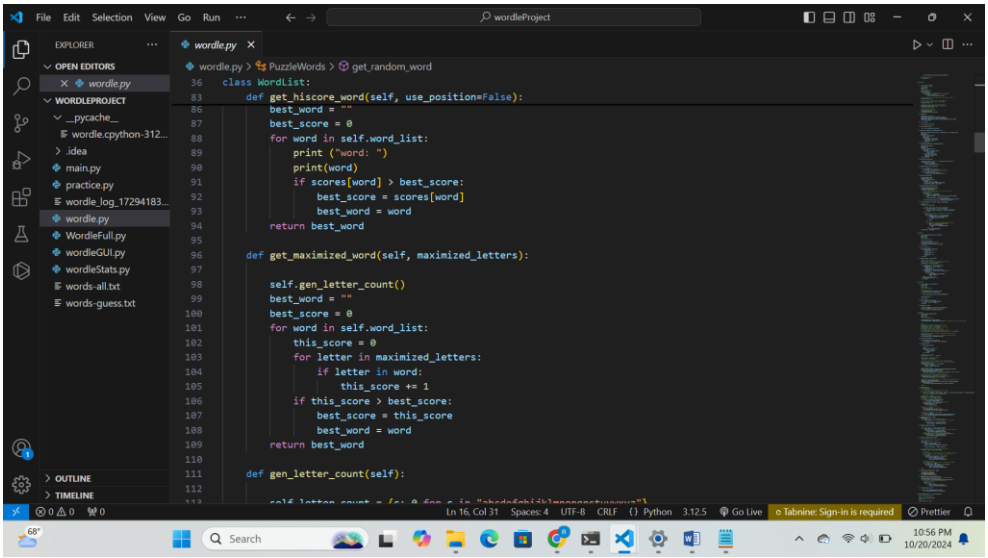
```
import random
import time

number = 0
params = ["recount"]


class PuzzleWords:
    def __init__(self, word_file):
        self.word_list = self.load_words(word_file)

    def load_words(self, word_file):
        with open(word_file, "r") as file:
            return [line.strip().upper() for line in file.readlines()]

    def get_random_word(self):
        import random
        return random.choice(self.word_list)

# Instantiate puzzle_words globally
puzzle_words = PuzzleWords("words-guess.txt")

class Wordle:
    def __init__(self, correct_word=None):
        # Check if the correct word is valid
        if correct_word in puzzle_words.word_list:
            self.correct_word = correct_word
        else:
            self.correct_word = puzzle_words.get_random_word()
```



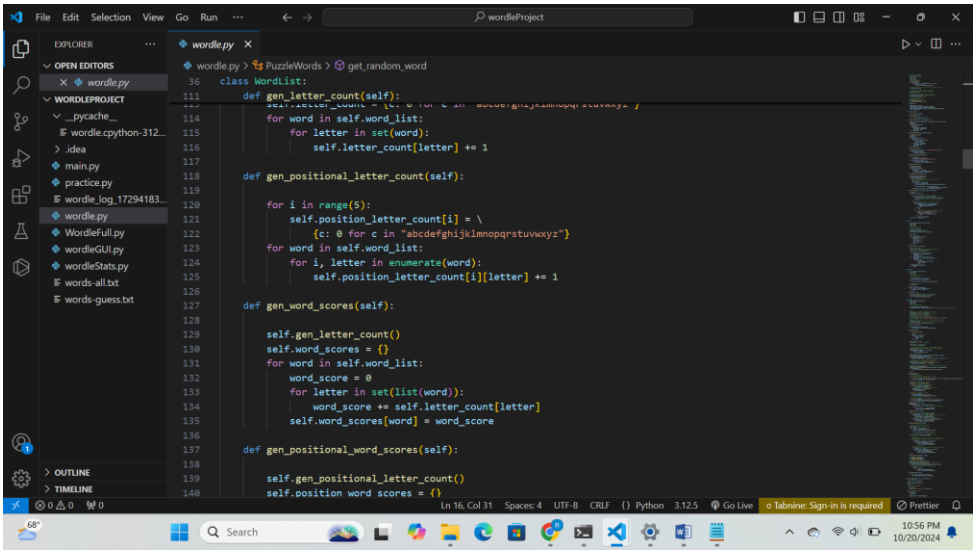
```
        # List of guesses so far
        self.guesses = []


class WordList:


    def __init__(self, *files):
        # list of all the words
        global number
        number += 1
       # print(number)
       # print("debug files")
        self.word_list = []
        for file in files:
            with open(file, "r", encoding="UTF-8") as in_file:
                for line in in_file:
                    self.word_list.append(line.strip())

        # letter counts in all words in the list: {"a": 100, "b": 200, ...}
        self.letter_count = {}

        # words' scores: {"apple": 100, "fruit": 200} etc
        # score is the sum of all letters' frequencies
        self.word_scores = {}

        # Same, but scores account for letter positions
        self.position_letter_count = [{}, {}, {}, {}, {}]
```



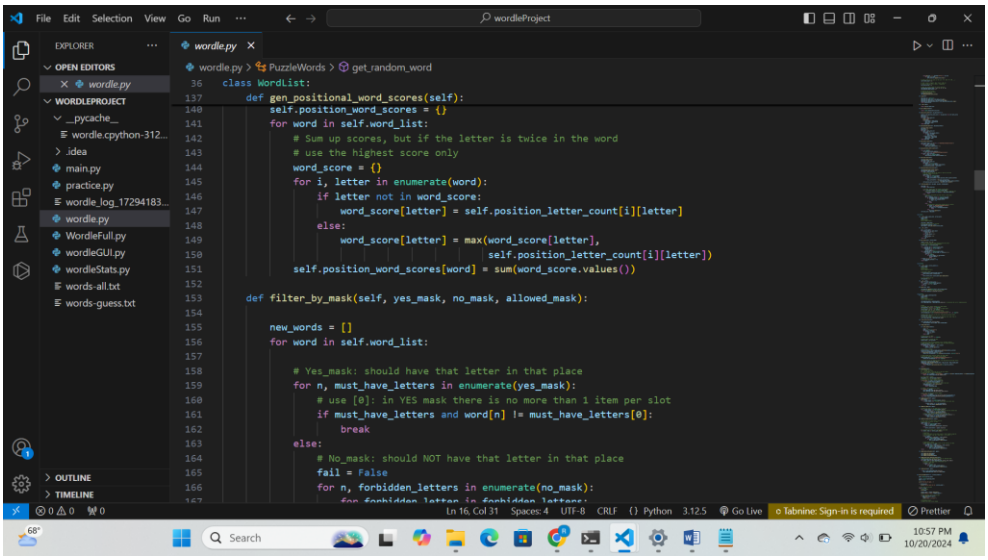
```
class WordList:
    def __init__(self, *files):
        self.position_letter_count = [{}, {}, {}, {}, {}]
        self.position_word_scores = {}

        # Generate the word scores
        # (both positional and total)
        self.gen_word_scores()
        self.gen_positional_word_scores()

    def copy(self):

        new_word_list = WordList()
        new_word_list.word_list = self.word_list.copy()
        new_word_list.word_scores = self.word_scores.copy()
        new_word_list.position_word_scores = self.position_word_scores.copy()
        return new_word_list

    def __len__(self):

        return len(self.word_list)

    def get_random_word(self):

        return random.choice(self.word_list)

    def get_hiscore_word(self, use_position=False):

        scores = self.position_word_scores if use_position else self.word_scores
```



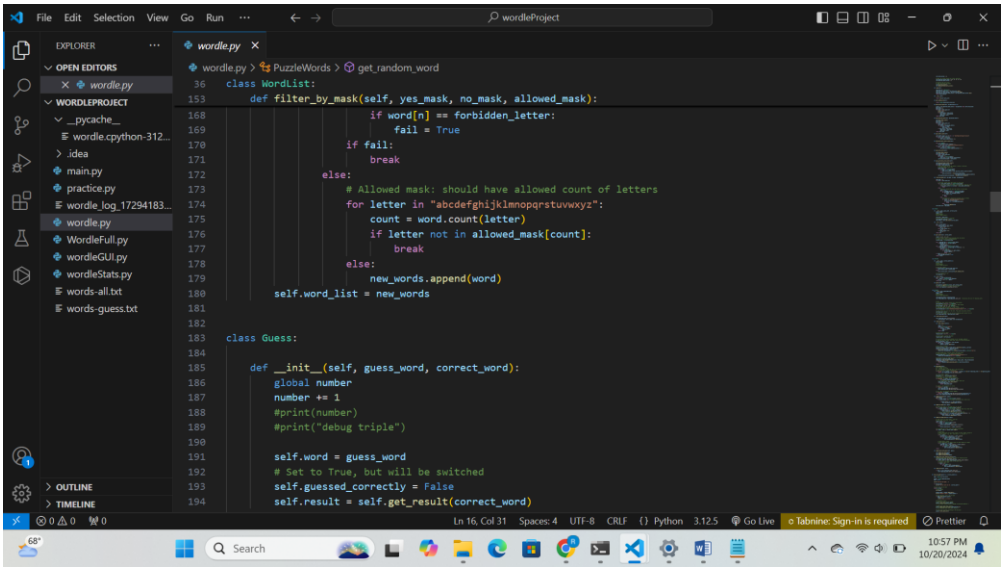
```
class WordList:
    def get_hiscore_word(self, use_position=False):
        best_word = ""
        best_score = 0
        for word in self.word_list:
            print ("word: ")
            print(word)
            if scores[word] > best_score:
                best_score = scores[word]
                best_word = word
        return best_word

    def get_maximized_word(self, maximized_letters):

        self.gen_letter_count()
        best_word = ""
        best_score = 0
        for word in self.word_list:
            this_score = 0
            for letter in maximized_letters:
                if letter in word:
                    this_score += 1
            if this_score > best_score:
                best_score = this_score
                best_word = word
        return best_word

    def gen_letter_count(self):
```



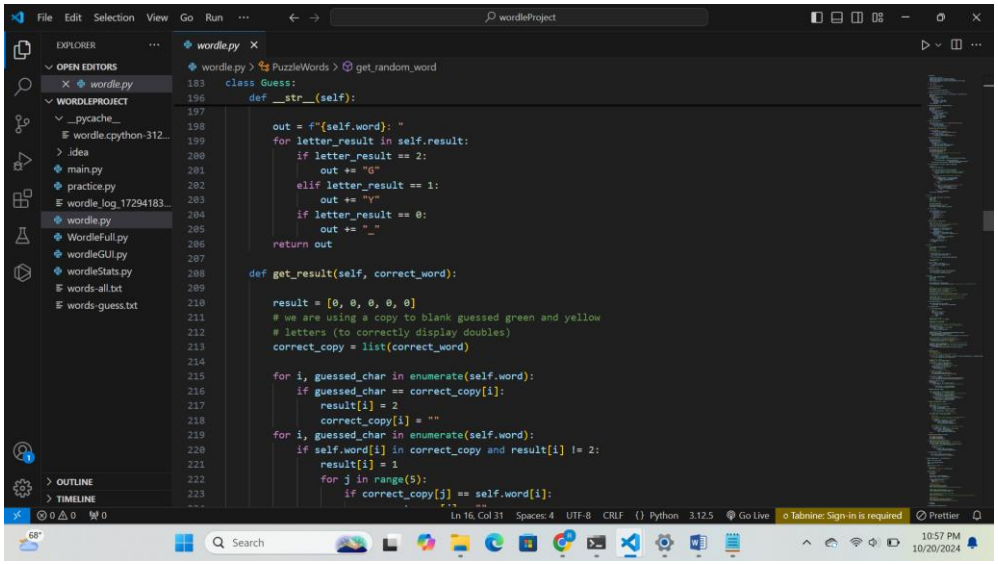
```
class WordList:
    def gen_letter_count(self):
        for word in self.word_list:
            for letter in set(word):
                self.letter_count[letter] += 1

    def gen_positional_letter_count(self):

        for i in range(5):
            self.position_letter_count[i] = \
                {c: 0 for c in "abcdefghijklmnopqrstuvwxyz"}
        for word in self.word_list:
            for i, letter in enumerate(word):
                self.position_letter_count[i][letter] += 1

    def gen_word_scores(self):

        self.gen_letter_count()
        self.word_scores = {}
        for word in self.word_list:
            word_score = 0
            for letter in set(list(word)):
                word_score += self.letter_count[letter]
            self.word_scores[word] = word_score

    def gen_positional_word_scores(self):

        self.gen_positional_letter_count()
        self.position_word_scores = {}
```



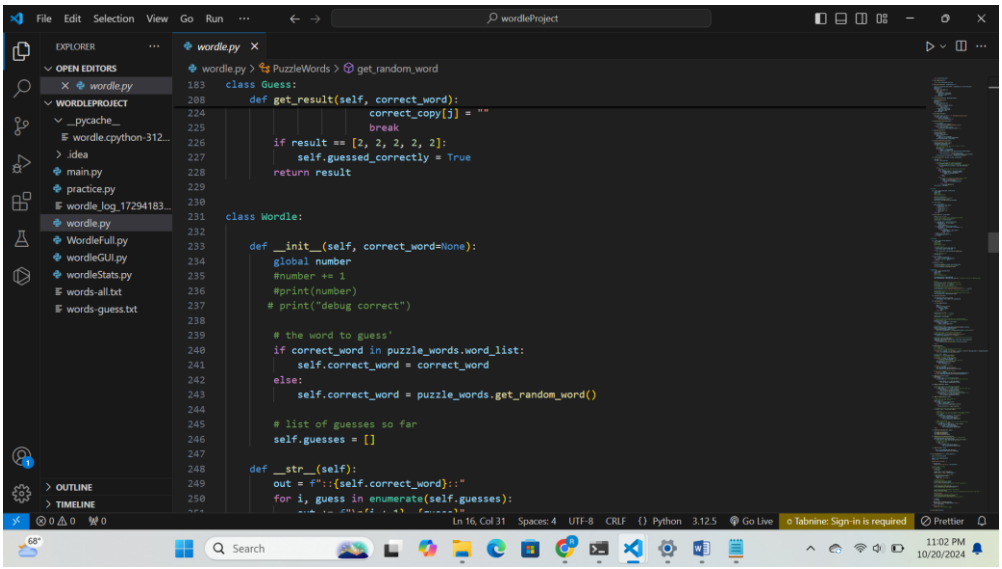
```
    def gen_positional_word_scores(self):
        self.position_word_scores = {}
        for word in self.word_list:
            # Sum up scores, but if the letter is twice in the word
            # use the highest score only
            word_score = {}
            for i, letter in enumerate(word):
                if letter not in word_score:
                    word_score[letter] = self.position_letter_count[i][letter]
                else:
                    word_score[letter] = max(word_score[letter],
                                             self.position_letter_count[i][letter])
            self.position_word_scores[word] = sum(word_score.values())

    def filter_by_mask(self, yes_mask, no_mask, allowed_mask):

        new_words = []
        for word in self.word_list:

            # Yes_mask: should have that letter in that place
            for n, must_have_letters in enumerate(yes_mask):
                # use [0]: in YES mask there is no more than 1 item per slot
                if must_have_letters and word[n] != must_have_letters[0]:
                    break
            else:
                # No_mask: should NOT have that letter in that place
                fail = False
                for n, forbidden_letters in enumerate(no_mask):
```



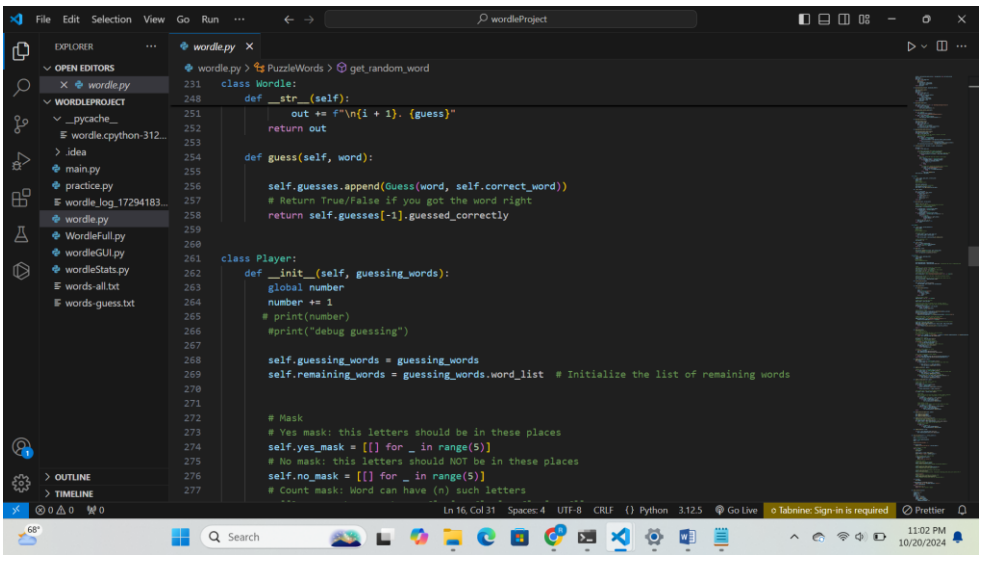
```
class WordList:
    def filter_by_mask(self, yes_mask, no_mask, allowed_mask):
                    if word[n] == forbidden_letter:
                        fail = True
                if fail:
                    break
            else:
                # Allowed mask: should have allowed count of letters
                for letter in "abcdefghijklmnopqrstuvwxyz":
                    count = word.count(letter)
                    if letter not in allowed_mask[count]:
                        break
                else:
                    new_words.append(word)
        self.word_list = new_words


class Guess:

    def __init__(self, guess_word, correct_word):
        global number
        number += 1
        #print(number)
        #print("debug triple")

        self.word = guess_word
        # Set to True, but will be switched
        self.guessed_correctly = False
        self.result = self.get_result(correct_word)
```



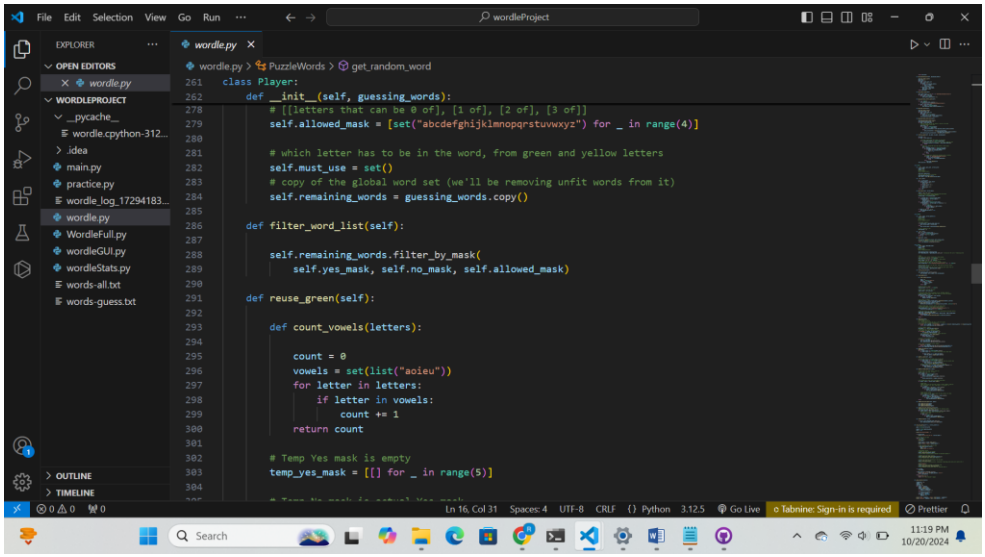
```
class Guess:
    def __str__(self):

        out = f"{self.word}: "
        for letter_result in self.result:
            if letter_result == 2:
                out += "G"
            elif letter_result == 1:
                out += "Y"
            if letter_result == 0:
                out += "_"
        return out

    def get_result(self, correct_word):

        result = [0, 0, 0, 0, 0]
        # we are using a copy to blank guessed green and yellow
        # letters (to correctly display doubles)
        correct_copy = list(correct_word)

        for i, guessed_char in enumerate(self.word):
            if guessed_char == correct_copy[i]:
                result[i] = 2
                correct_copy[i] = ""
        for i, guessed_char in enumerate(self.word):
            if self.word[i] in correct_copy and result[i] != 2:
                result[i] = 1
                for j in range(5):
                    if correct_copy[j] == self.word[i]:
```



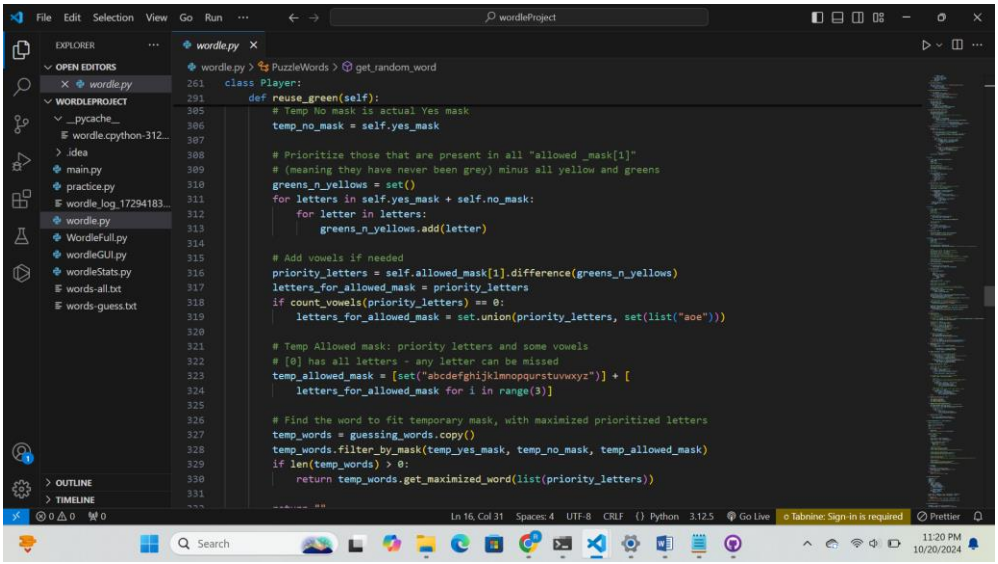
```
                        correct_copy[j] = ""
                        break
        if result == [2, 2, 2, 2, 2]:
            self.guessed_correctly = True
        return result


class Wordle:

    def __init__(self, correct_word=None):
        global number
        #number += 1
        #print(number)
       # print("debug correct")

        # the word to guess'
        if correct_word in puzzle_words.word_list:
            self.correct_word = correct_word
        else:
            self.correct_word = puzzle_words.get_random_word()

        # list of guesses so far
        self.guesses = []

    def __str__(self):
        out = f"::{self.correct_word}::"
        for i, guess in enumerate(self.guesses):
```



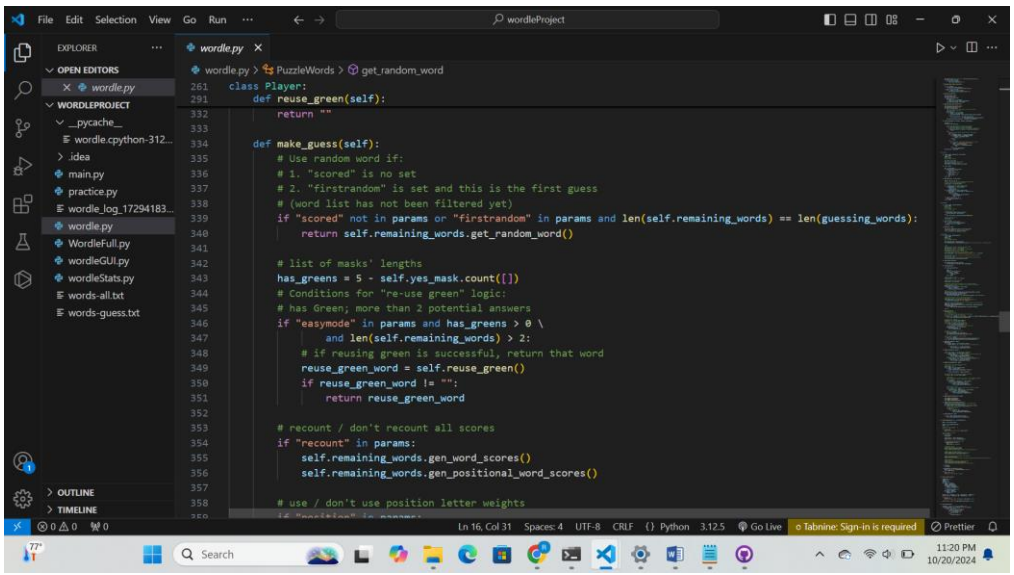
```
class Wordle:
    def __str__(self):
            out += f"\n{i + 1}. {guess}"
        return out

    def guess(self, word):

        self.guesses.append(Guess(word, self.correct_word))
        # Return True/False if you got the word right
        return self.guesses[-1].guessed_correctly


class Player:
    def __init__(self, guessing_words):
        global number
        number += 1
       # print(number)
        #print("debug guessing")

        self.guessing_words = guessing_words
        self.remaining_words = guessing_words.word_list  # Initialize the list of remaining words


        # Mask
        # Yes mask: this letters should be in these places
        self.yes_mask = [[] for _ in range(5)]
        # No mask: this letters should NOT be in these places
        self.no_mask = [[] for _ in range(5)]
        # Count mask: Word can have (n) such letters
```



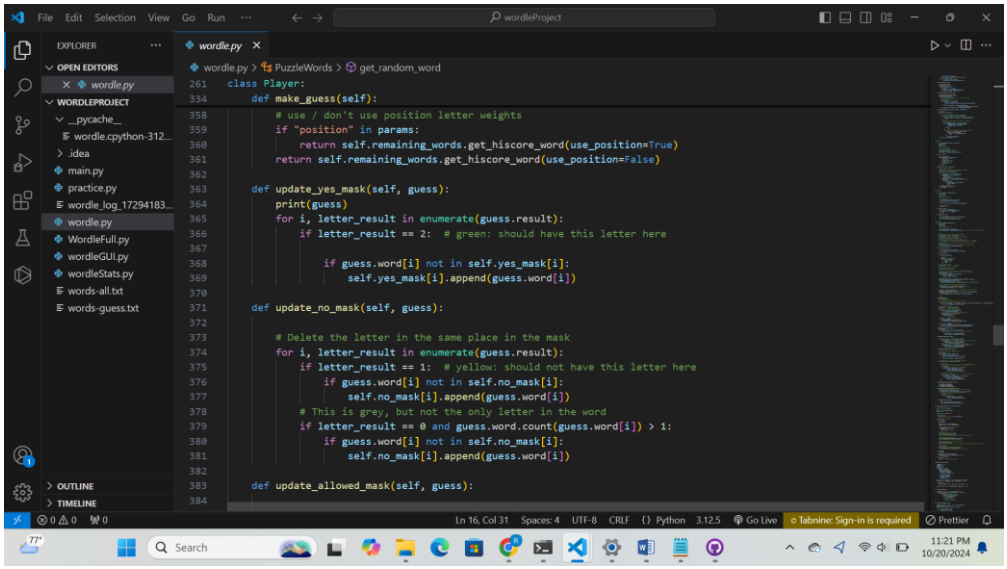
```python
class Player:
    def __init__(self, guessing_words):
        # [[letters that can be 0 of], [1 of], [2 of], [3 of]]
        self.allowed_mask = [set("abcdefghijklmnopqrstuvwxyz") for _ in range(4)]

        # which letter has to be in the word, from green and yellow letters
        self.must_use = set()
        # copy of the global word set (we'll be removing unfit words from it)
        self.remaining_words = guessing_words.copy()

    def filter_word_list(self):

        self.remaining_words.filter_by_mask(
            self.yes_mask, self.no_mask, self.allowed_mask)

    def reuse_green(self):

        def count_vowels(letters):

            count = 0
            vowels = set(list("aoieu"))
            for letter in letters:
                if letter in vowels:
                    count += 1
            return count

        # Temp Yes mask is empty
        temp_yes_mask = [[] for _ in range(5)]
```



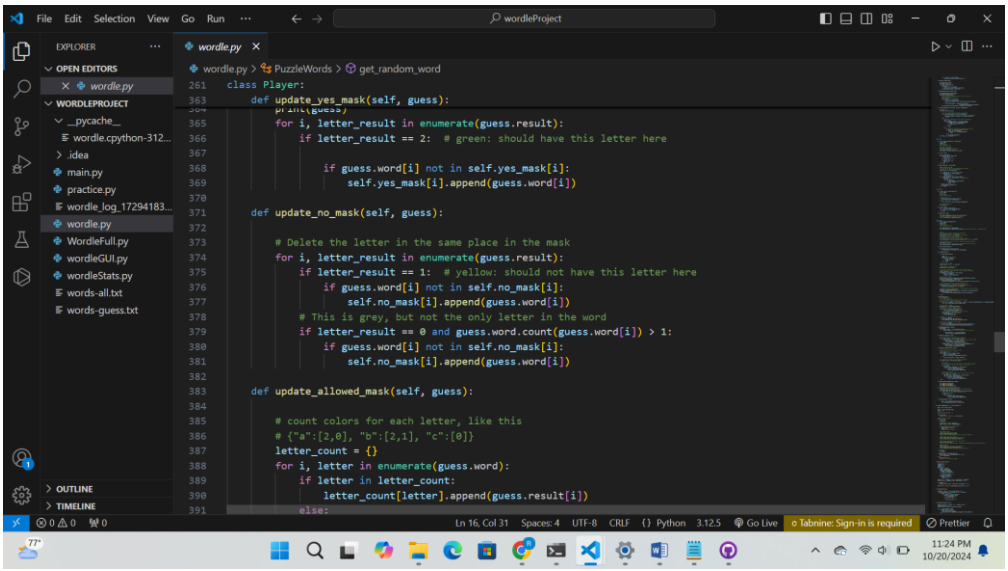
```python
class Player:
    def reuse_green(self):
        # Temp No mask is actual Yes mask
        temp_no_mask = self.yes_mask

        # Prioritize those that are present in all "allowed _mask[1]"
        # (meaning they have never been grey) minus all yellow and greens
        greens_n_yellows = set()
        for letters in self.yes_mask + self.no_mask:
            for letter in letters:
                greens_n_yellows.add(letter)

        # Add vowels if needed
        priority_letters = self.allowed_mask[1].difference(greens_n_yellows)
        letters_for_allowed_mask = priority_letters
        if count_vowels(priority_letters) == 0:
            letters_for_allowed_mask = set.union(priority_letters, set(list("aoe")))

        # Temp Allowed mask: priority letters and some vowels
        # [0] has all letters - any letter can be missed
        temp_allowed_mask = [set("abcdefghijklmnopqurstuvwxyz")] + [
            letters_for_allowed_mask for i in range(3)]

        # Find the word to fit temporary mask, with maximized prioritized letters
        temp_words = guessing_words.copy()
        temp_words.filter_by_mask(temp_yes_mask, temp_no_mask, temp_allowed_mask)
        if len(temp_words) > 0:
            return temp_words.get_maximized_word(list(priority_letters))
```



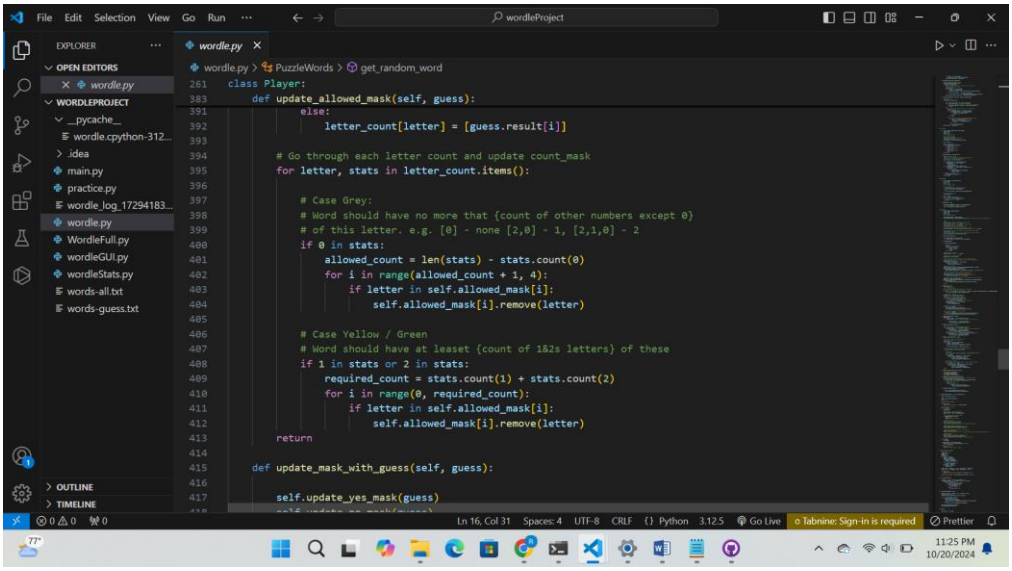
```python
class Player:
    def reuse_green(self):
        return ""

    def make_guess(self):
        # Use random word if:
        # 1. "scored" is no set
        # 2. "firstrandom" is set and this is the first guess
        # (word list has not been filtered yet)
        if "scored" not in params or "firstrandom" in params and len(self.remaining_words) == len(guessing_words):
            return self.remaining_words.get_random_word()

        # list of masks' lengths
        has_greens = 5 - self.yes_mask.count([])
        # Conditions for "re-use green" logic:
        # has Green; more than 2 potential answers
        if "easymode" in params and has_greens > 0 \
                and len(self.remaining_words) > 2:
            # if reusing green is successful, return that word
            reuse_green_word = self.reuse_green()
            if reuse_green_word != "":
                return reuse_green_word

        # recount / don't recount all scores
        if "recount" in params:
            self.remaining_words.gen_word_scores()
            self.remaining_words.gen_positional_word_scores()

        # use / don't use position letter weights
```



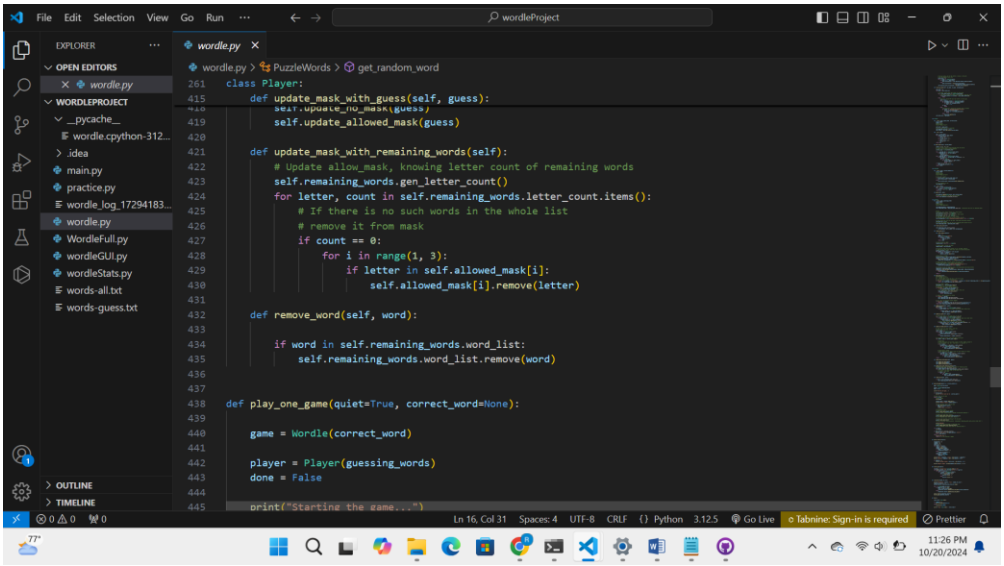
```
class Player:
    def make_guess(self):
        # use / don't use position letter weights
        if "position" in params:
            return self.remaining_words.get_hiscore_word(use_position=True)
        return self.remaining_words.get_hiscore_word(use_position=False)

    def update_yes_mask(self, guess):
        print(guess)
        for i, letter_result in enumerate(guess.result):
            if letter_result == 2:  # green: should have this letter here

                if guess.word[i] not in self.yes_mask[i]:
                    self.yes_mask[i].append(guess.word[i])

    def update_no_mask(self, guess):

        # Delete the letter in the same place in the mask
        for i, letter_result in enumerate(guess.result):
            if letter_result == 1:  # yellow: should not have this letter here
                if guess.word[i] not in self.no_mask[i]:
                    self.no_mask[i].append(guess.word[i])
            # This is grey, but not the only letter in the word
            if letter_result == 0 and guess.word.count(guess.word[i]) > 1:
                if guess.word[i] not in self.no_mask[i]:
                    self.no_mask[i].append(guess.word[i])

    def update_allowed_mask(self, guess):
```

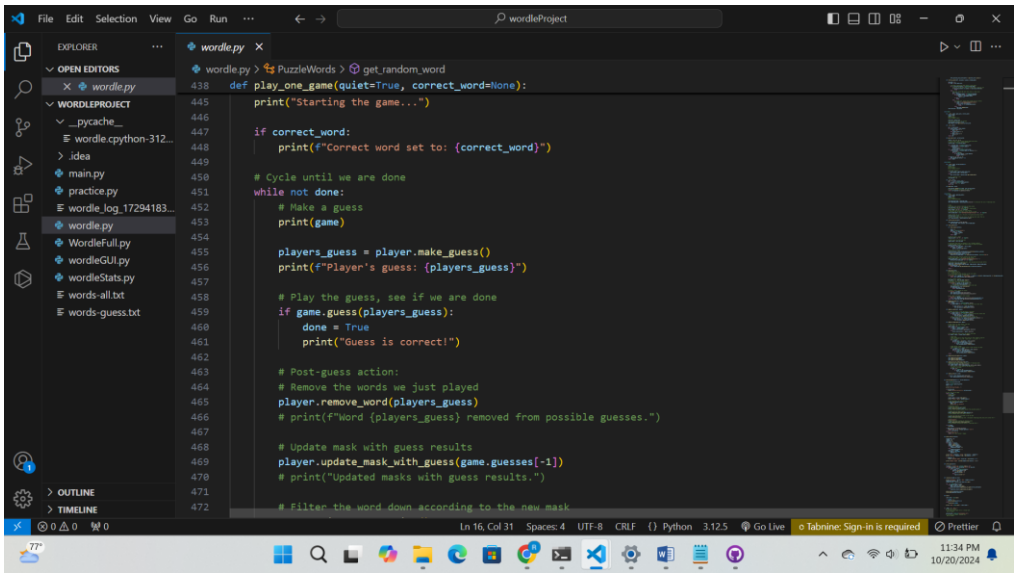


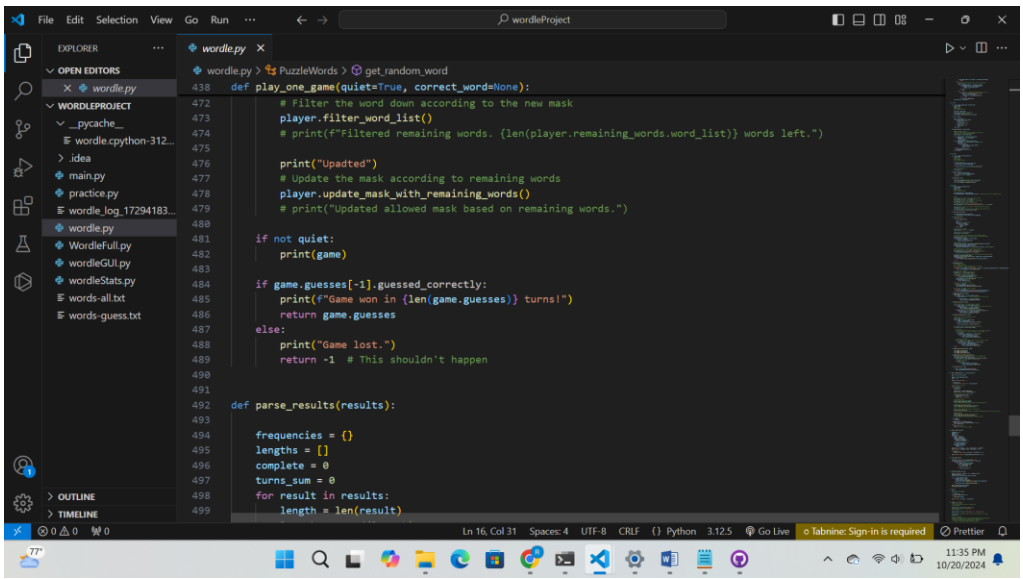
```
    def update_yes_mask(self, guess):
        print(guess)
        for i, letter_result in enumerate(guess.result):
            if letter_result == 2:  # green: should have this letter here

                if guess.word[i] not in self.yes_mask[i]:
                    self.yes_mask[i].append(guess.word[i])

    def update_no_mask(self, guess):

        # Delete the letter in the same place in the mask
        for i, letter_result in enumerate(guess.result):
            if letter_result == 1:  # yellow: should not have this letter here
                if guess.word[i] not in self.no_mask[i]:
                    self.no_mask[i].append(guess.word[i])
            # This is grey, but not the only letter in the word
            if letter_result == 0 and guess.word.count(guess.word[i]) > 1:
                if guess.word[i] not in self.no_mask[i]:
                    self.no_mask[i].append(guess.word[i])

    def update_allowed_mask(self, guess):

        # count colors for each letter, like this
        # {"a":[2,0], "b":[2,1], "c":[0]}
        letter_count = {}
        for i, letter in enumerate(guess.word):
            if letter in letter_count:
                letter_count[letter].append(guess.result[i])
            else:
```

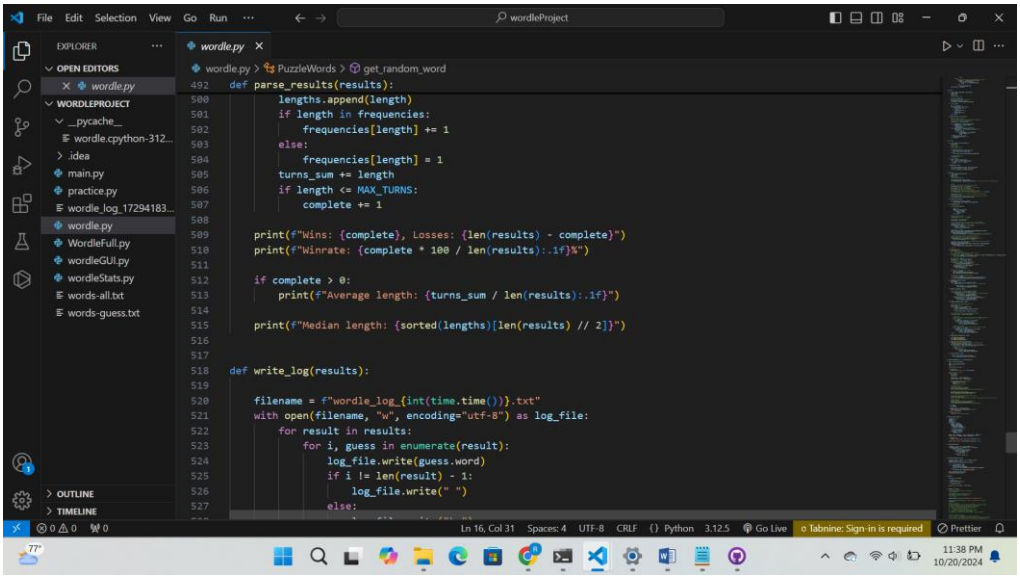


```
            else:
                letter_count[letter] = [guess.result[i]]

        # Go through each letter count and update count_mask
        for letter, stats in letter_count.items():

            # Case Grey:
            # Word should have no more that {count of other numbers except 0}
            # of this letter. e.g. [0] - none [2,0] - 1, [2,1,0] - 2
            if 0 in stats:
                allowed_count = len(stats) - stats.count(0)
                for i in range(allowed_count + 1, 4):
                    if letter in self.allowed_mask[i]:
                        self.allowed_mask[i].remove(letter)

            # Case Yellow / Green
            # Word should have at leaset {count of 1&2s letters} of these
            if 1 in stats or 2 in stats:
                required_count = stats.count(1) + stats.count(2)
                for i in range(0, required_count):
                    if letter in self.allowed_mask[i]:
                        self.allowed_mask[i].remove(letter)
        return

    def update_mask_with_guess(self, guess):

        self.update_yes_mask(guess)
```

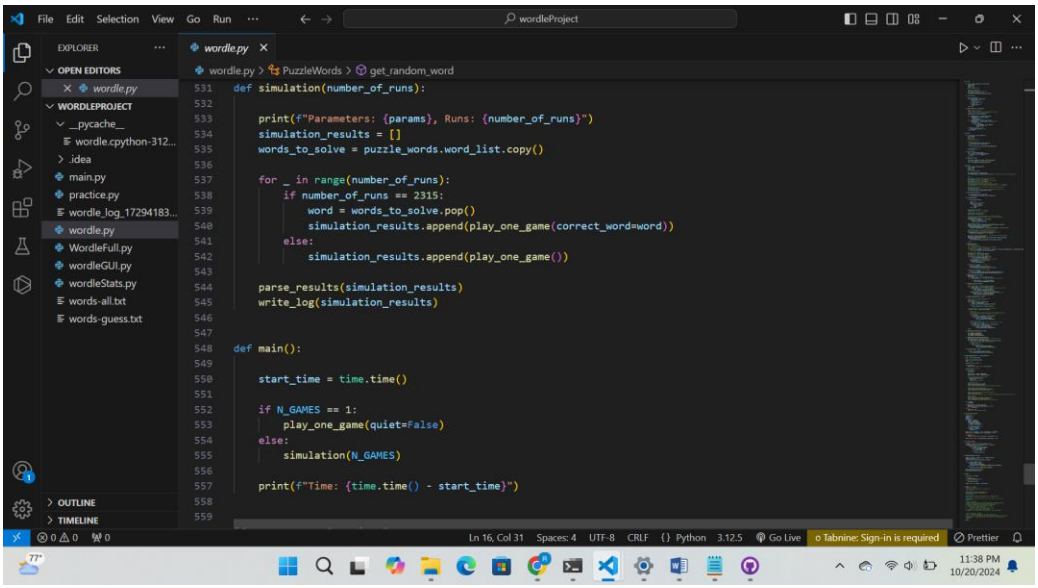

wordle.py
PuzzleWords > get_random_word
class Player:
def update_mask_with_guess(self, guess):
self.update_allowed_mask(guess)
def update_mask_with_remaining_words(self):
# Update allow_mask, knowing letter count of remaining words
self.remaining_words.gen_letter_count()
for letter, count in self.remaining_words.letter_count.items():
# If there is no such words in the whole list
# remove it from mask
if count == 0:
for i in range(1, 3):
if letter in self.allowed_mask[i]:
self.allowed_mask[i].remove(letter)
def remove_word(self, word):
if word in self.remaining_words.word_list:
self.remaining_words.word_list.remove(word)
def play_one_game(quiet=True, correct_word=None):
game = Wordle(correct_word)
player = Player(guessing_words)
done = False
print("Starting the game...")


wordle.py
PuzzleWords > get_random_word
def play_one_game(quiet=True, correct_word=None):
print("Starting the game...")
if correct_word:
print(f"Correct word set to: {correct_word}")
# Cycle until we are done
while not done:
# Make a guess
print(game)
players_guess = player.make_guess()
print(f"Player's guess: {players_guess}")
# Play the guess, see if we are done
if game.guess(players_guess):
done = True
print("Guess is correct!")
# Post-guess action:
# Remove the words we just played
player.remove_word(players_guess)
# print(f"Word {players_guess} removed from possible guesses.")
# Update mask with guess results
player.update_mask_with_guess(game.guesses[-1])
# print("Updated masks with guess results.")
# Filter the word down according to the new mask

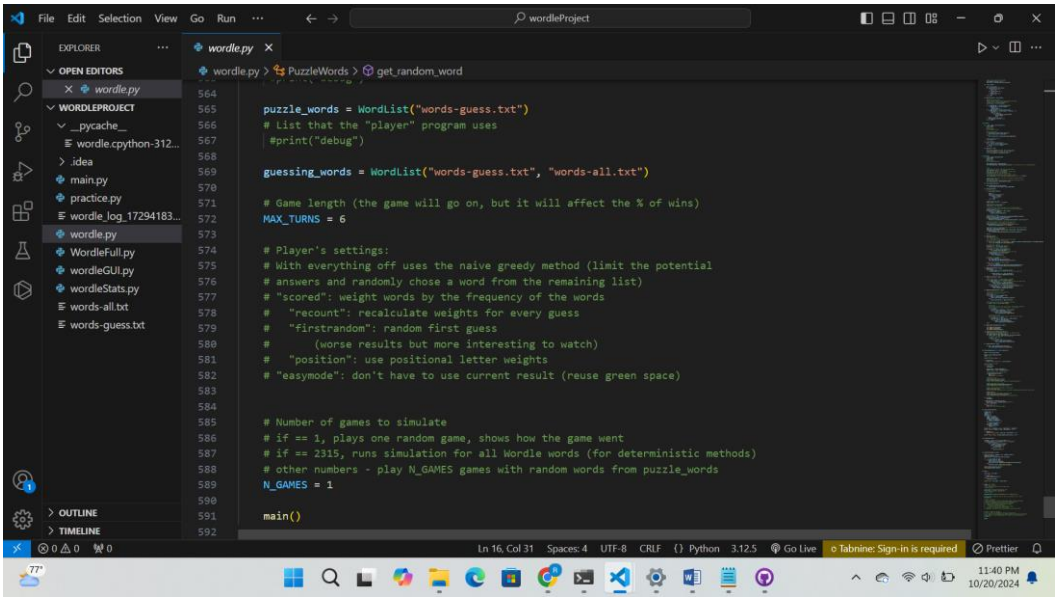

wordle.py
PuzzleWords > get_random_word
def play_one_game(quiet=True, correct_word=None):
# Filter the word down according to the new mask
player.filter_word_list()
# print(f"Filtered remaining words. {len(player.remaining_words.word_list)} words left.")
print("Upadted")
# Update the mask according to remaining words
player.update_mask_with_remaining_words()
# print("Updated allowed mask based on remaining words.")
if not quiet:
print(game)
if game.guesses[-1].guessed_correctly:
print(f"Game won in {len(game.guesses)} turns!")
return game.guesses
else:
print("Game lost.")
return -1 # This shouldn't happen
def parse_results(results):
frequencies = {}
lengths = []
complete = 0
turns_sum = 0
for result in results:
length = len(result)



```
def parse_results(results):
        lengths.append(length)
        if length in frequencies:
            frequencies[length] += 1
        else:
            frequencies[length] = 1
        turns_sum += length
        if length <= MAX_TURNS:
            complete += 1

    print(f"Wins: {complete}, Losses: {len(results) - complete}")
    print(f"Winrate: {complete * 100 / len(results):.1f}%")

    if complete > 0:
        print(f"Average length: {turns_sum / len(results):.1f}")

    print(f"Median length: {sorted(lengths)[len(results) // 2]}")


def write_log(results):

    filename = f"wordle_log_{int(time.time())}.txt"
    with open(filename, "w", encoding="utf-8") as log_file:
        for result in results:
            for i, guess in enumerate(result):
                log_file.write(guess.word)
                if i != len(result) - 1:
                    log_file.write(" ")
                else:
```

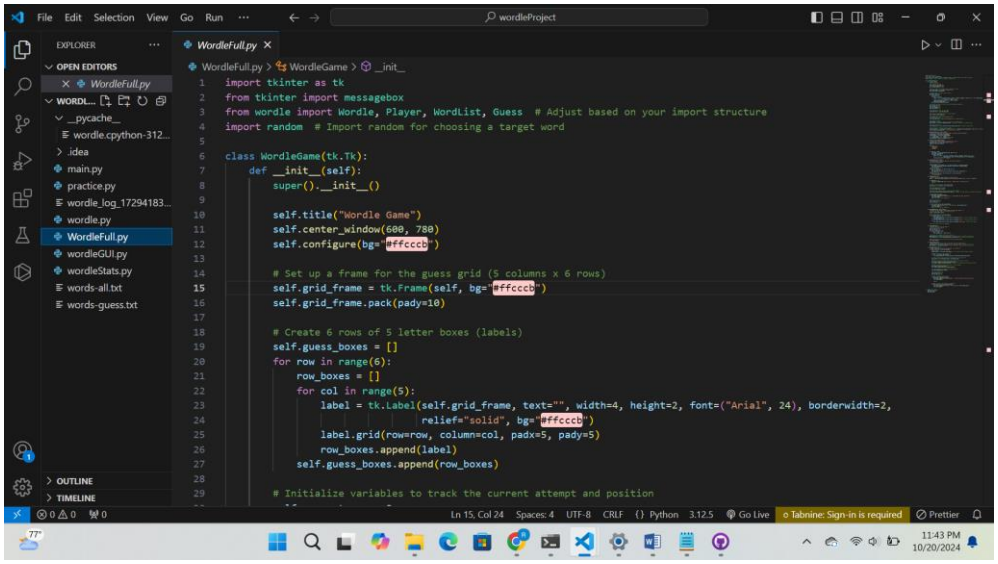


```
def simulation(number_of_runs):

    print(f"Parameters: {params}, Runs: {number_of_runs}")
    simulation_results = []
    words_to_solve = puzzle_words.word_list.copy()

    for _ in range(number_of_runs):
        if number_of_runs == 2315:
            word = words_to_solve.pop()
            simulation_results.append(play_one_game(correct_word=word))
        else:
            simulation_results.append(play_one_game())

    parse_results(simulation_results)
    write_log(simulation_results)


def main():

    start_time = time.time()

    if N_GAMES == 1:
        play_one_game(quiet=False)
    else:
        simulation(N_GAMES)

    print(f"Time: {time.time() - start_time}")
```

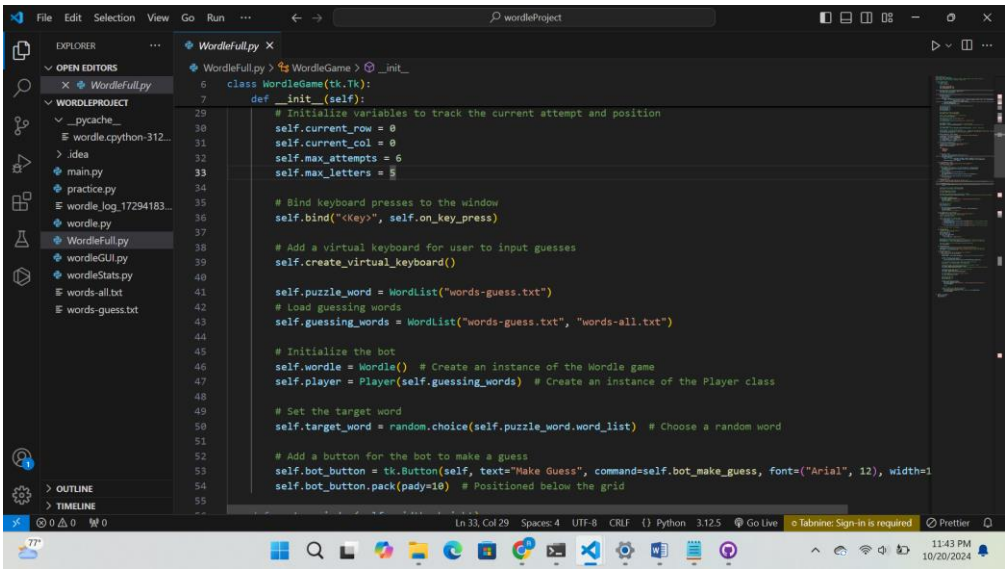


```
    puzzle_words = WordList("words-guess.txt")
    # List that the "player" program uses
     #print("debug")

    guessing_words = WordList("words-guess.txt", "words-all.txt")

    # Game length (the game will go on, but it will affect the % of wins)
    MAX_TURNS = 6

    # Player's settings:
    # With everything off uses the naive greedy method (limit the potential
    # answers and randomly chose a word from the remaining list)
    # "scored": weight words by the frequency of the words
    #   "recount": recalculate weights for every guess
    #   "firstrandom": random first guess
    #       (worse results but more interesting to watch)
    #   "position": use positional letter weights
    # "easymode": don't have to use current result (reuse green space)


    # Number of games to simulate
    # if == 1, plays one random game, shows how the game went
    # if == 2315, runs simulation for all Wordle words (for deterministic methods)
    # other numbers - play N_GAMES games with random words from puzzle_words
    N_GAMES = 1

    main()
```

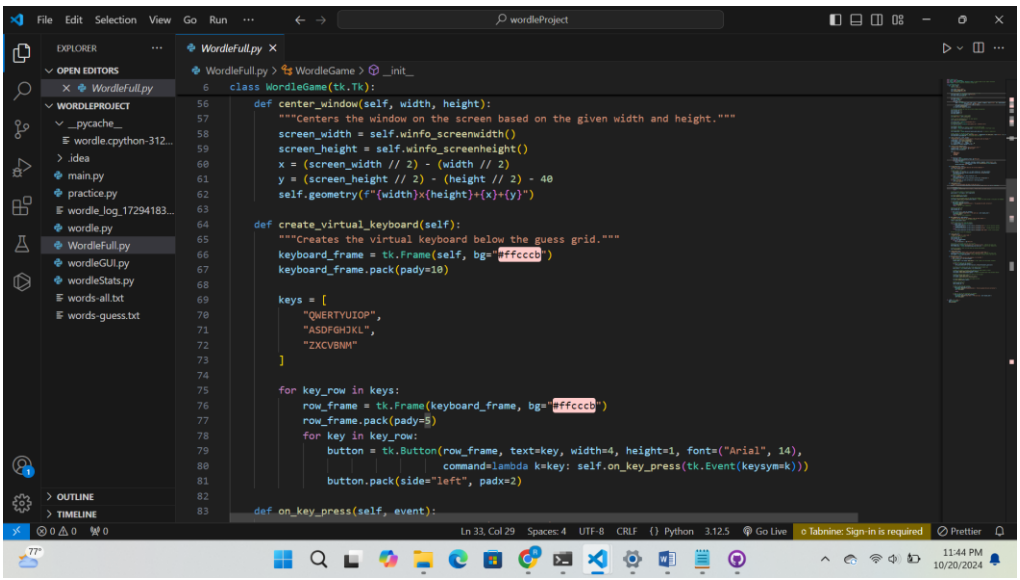
- **WordleFull.py**



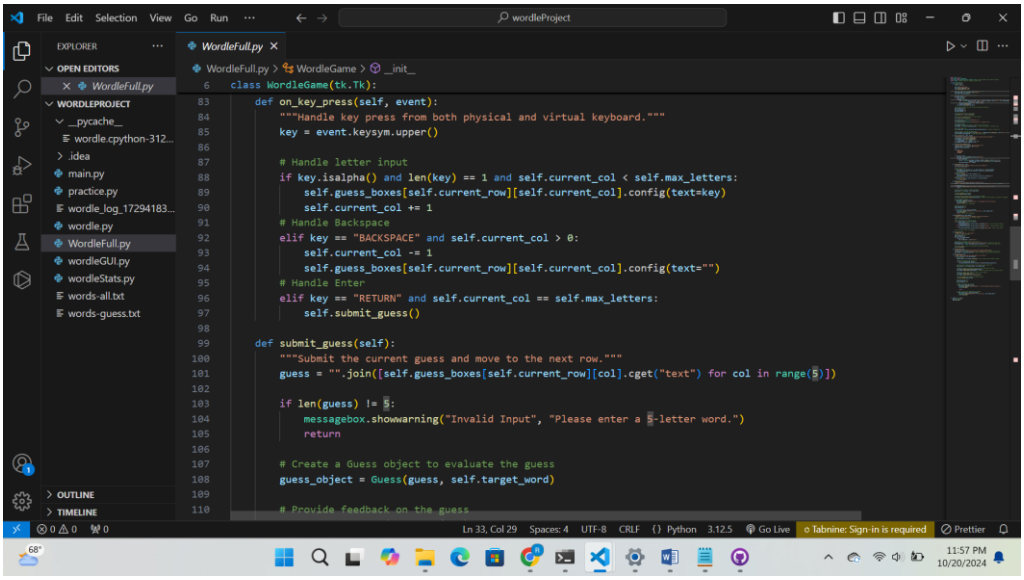
```python
import tkinter as tk
from tkinter import messagebox
from wordle import Wordle, Player, WordList, Guess  # Adjust based on your import structure
import random  # Import random for choosing a target word

class WordleGame(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("Wordle Game")
        self.center_window(600, 780)
        self.configure(bg="#ffcccb")

        # Set up a frame for the guess grid (5 columns x 6 rows)
        self.grid_frame = tk.Frame(self, bg="#ffcccb")
        self.grid_frame.pack(pady=10)

        # Create 6 rows of 5 letter boxes (labels)
        self.guess_boxes = []
        for row in range(6):
            row_boxes = []
            for col in range(5):
                label = tk.Label(self.grid_frame, text="", width=4, height=2, font=("Arial", 24), borderwidth=2,
                                 relief="solid", bg="#ffcccb")
                label.grid(row=row, column=col, padx=5, pady=5)
                row_boxes.append(label)
            self.guess_boxes.append(row_boxes)

        # Initialize variables to track the current attempt and position
```



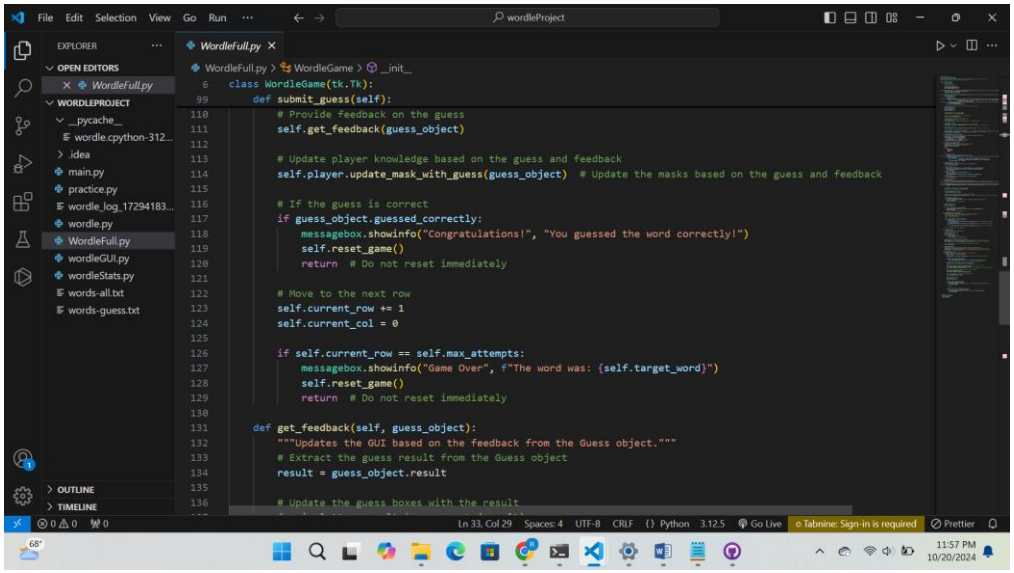
```python
class WordleGame(tk.Tk):
    def __init__(self):
        # Initialize variables to track the current attempt and position
        self.current_row = 0
        self.current_col = 0
        self.max_attempts = 6
        self.max_letters = 5

        # Bind keyboard presses to the window
        self.bind("<Key>", self.on_key_press)

        # Add a virtual keyboard for user to input guesses
        self.create_virtual_keyboard()

        self.puzzle_word = WordList("words-guess.txt")
        # Load guessing words
        self.guessing_words = WordList("words-guess.txt", "words-all.txt")

        # Initialize the bot
        self.wordle = Wordle()  # Create an instance of the Wordle game
        self.player = Player(self.guessing_words)  # Create an instance of the Player class

        # Set the target word
        self.target_word = random.choice(self.puzzle_word.word_list)  # Choose a random word

        # Add a button for the bot to make a guess
        self.bot_button = tk.Button(self, text="Make Guess", command=self.bot_make_guess, font=("Arial", 12), width=1
        self.bot_button.pack(pady=10)  # Positioned below the grid
```



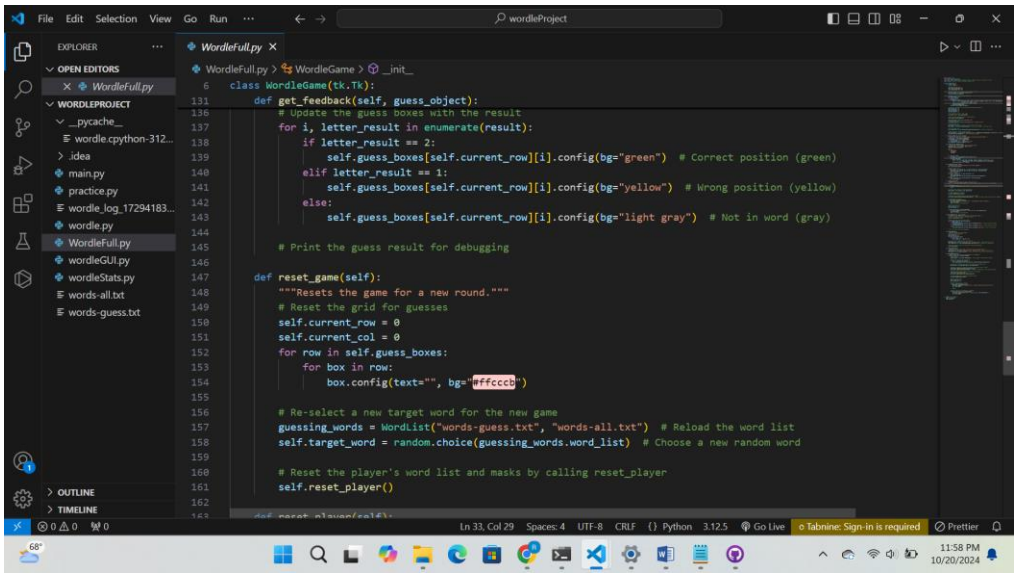
```python
class WordleGame(tk.Tk):
    def center_window(self, width, height):
        """Centers the window on the screen based on the given width and height."""
        screen_width = self.winfo_screenwidth()
        screen_height = self.winfo_screenheight()
        x = (screen_width // 2) - (width // 2)
        y = (screen_height // 2) - (height // 2) - 40
        self.geometry(f"{width}x{height}+{x}+{y}")

    def create_virtual_keyboard(self):
        """Creates the virtual keyboard below the guess grid."""
        keyboard_frame = tk.Frame(self, bg="#ffcccb")
        keyboard_frame.pack(pady=10)

        keys = [
            "QWERTYUIOP",
            "ASDFGHJKL",
            "ZXCVBNM"
        ]

        for key_row in keys:
            row_frame = tk.Frame(keyboard_frame, bg="#ffcccb")
            row_frame.pack(pady=5)
            for key in key_row:
                button = tk.Button(row_frame, text=key, width=4, height=1, font=("Arial", 14),
                                   command=lambda k=key: self.on_key_press(tk.Event(keysym=k)))
                button.pack(side="left", padx=2)

    def on_key_press(self, event):
```



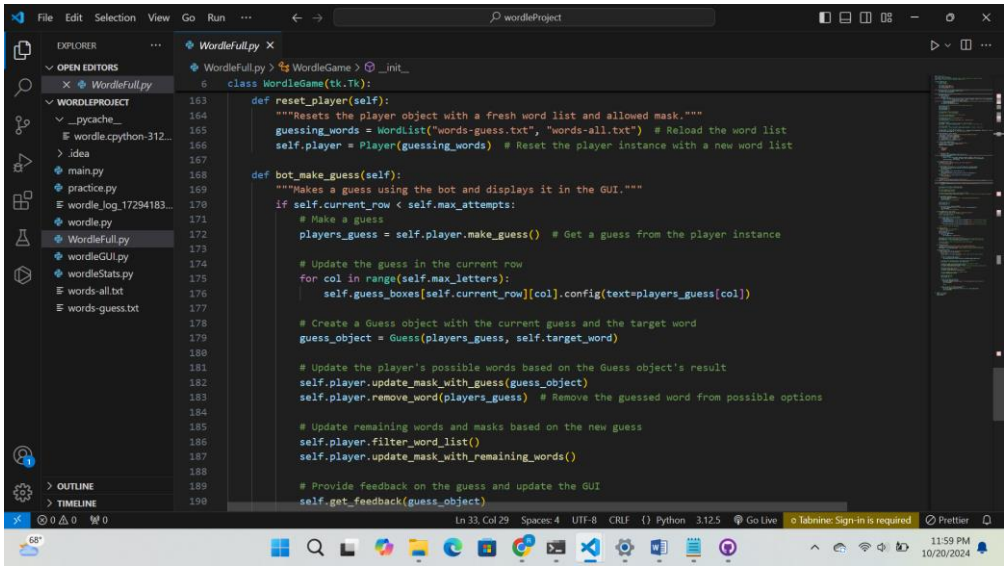
```python
class WordleGame(tk.Tk):
    def on_key_press(self, event):
        """Handle key press from both physical and virtual keyboard."""
        key = event.keysym.upper()

        # Handle letter input
        if key.isalpha() and len(key) == 1 and self.current_col < self.max_letters:
            self.guess_boxes[self.current_row][self.current_col].config(text=key)
            self.current_col += 1
        # Handle Backspace
        elif key == "BACKSPACE" and self.current_col > 0:
            self.current_col -= 1
            self.guess_boxes[self.current_row][self.current_col].config(text="")
        # Handle Enter
        elif key == "RETURN" and self.current_col == self.max_letters:
            self.submit_guess()

    def submit_guess(self):
        """Submit the current guess and move to the next row."""
        guess = "".join([self.guess_boxes[self.current_row][col].cget("text") for col in range(5)])

        if len(guess) != 5:
            messagebox.showwarning("Invalid Input", "Please enter a 5-letter word.")
            return

        # Create a Guess object to evaluate the guess
        guess_object = Guess(guess, self.target_word)

        # Provide feedback on the guess
```



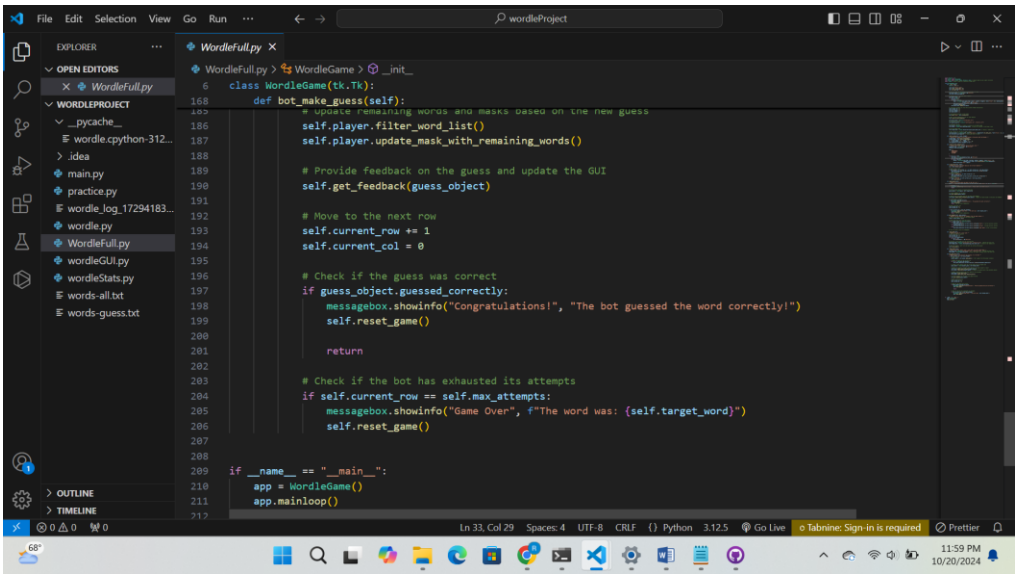
```python
class WordleGame(tk.Tk):
    def submit_guess(self):
        # Provide feedback on the guess
        self.get_feedback(guess_object)

        # Update player knowledge based on the guess and feedback
        self.player.update_mask_with_guess(guess_object)  # Update the masks based on the guess and feedback

        # If the guess is correct
        if guess_object.guessed_correctly:
            messagebox.showinfo("Congratulations!", "You guessed the word correctly!")
            self.reset_game()
            return  # Do not reset immediately

        # Move to the next row
        self.current_row += 1
        self.current_col = 0

        if self.current_row == self.max_attempts:
            messagebox.showinfo("Game Over", f"The word was: {self.target_word}")
            self.reset_game()
            return  # Do not reset immediately

    def get_feedback(self, guess_object):
        """Updates the GUI based on the feedback from the Guess object."""
        # Extract the guess result from the Guess object
        result = guess_object.result

        # Update the guess boxes with the result
```



```python
class WordleGame(tk.Tk):
    def get_feedback(self, guess_object):
        # Update the guess boxes with the result
        for i, letter_result in enumerate(result):
            if letter_result == 2:
                self.guess_boxes[self.current_row][i].config(bg="green")  # Correct position (green)
            elif letter_result == 1:
                self.guess_boxes[self.current_row][i].config(bg="yellow")  # Wrong position (yellow)
            else:
                self.guess_boxes[self.current_row][i].config(bg="light gray")  # Not in word (gray)

        # Print the guess result for debugging

    def reset_game(self):
        """Resets the game for a new round."""
        # Reset the grid for guesses
        self.current_row = 0
        self.current_col = 0
        for row in self.guess_boxes:
            for box in row:
                box.config(text="", bg="#ffcccb")

        # Re-select a new target word for the new game
        guessing_words = WordList("words-guess.txt", "words-all.txt")  # Reload the word list
        self.target_word = random.choice(guessing_words.word_list)  # Choose a new random word

        # Reset the player's word list and masks by calling reset_player
        self.reset_player()
```

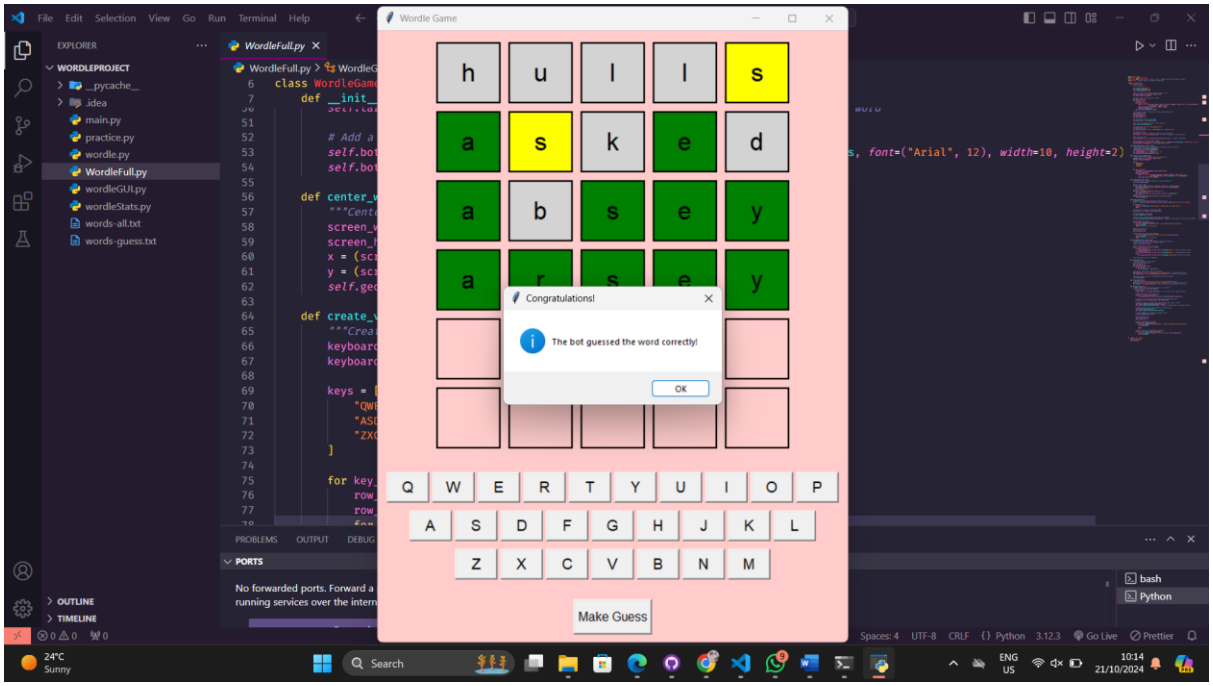


```
def reset_player(self):
    """Resets the player object with a fresh word list and allowed mask."""
    guessing_words = WordList("words-guess.txt", "words-all.txt")  # Reload the word list
    self.player = Player(guessing_words)  # Reset the player instance with a new word list

def bot_make_guess(self):
    """Makes a guess using the bot and displays it in the GUI."""
    if self.current_row < self.max_attempts:
        # Make a guess
        players_guess = self.player.make_guess()  # Get a guess from the player instance

        # Update the guess in the current row
        for col in range(self.max_letters):
            self.guess_boxes[self.current_row][col].config(text=players_guess[col])

        # Create a Guess object with the current guess and the target word
        guess_object = Guess(players_guess, self.target_word)

        # Update the player's possible words based on the Guess object's result
        self.player.update_mask_with_guess(guess_object)
        self.player.remove_word(players_guess)  # Remove the guessed word from possible options

        # Update remaining words and masks based on the new guess
        self.player.filter_word_list()
        self.player.update_mask_with_remaining_words()

        # Provide feedback on the guess and update the GUI
        self.get_feedback(guess_object)
```



```
        # Update remaining words and masks based on the new guess
        self.player.filter_word_list()
        self.player.update_mask_with_remaining_words()

        # Provide feedback on the guess and update the GUI
        self.get_feedback(guess_object)

        # Move to the next row
        self.current_row += 1
        self.current_col = 0

        # Check if the guess was correct
        if guess_object.guessed_correctly:
            messagebox.showinfo("Congratulations!", "The bot guessed the word correctly!")
            self.reset_game()

            return

        # Check if the bot has exhausted its attempts
        if self.current_row == self.max_attempts:
            messagebox.showinfo("Game Over", f"The word was: {self.target_word}")
            self.reset_game()


if __name__ == "__main__":
    app = WordleGame()
    app.mainloop()
```

Output



Link Video: https://youtu.be/c4CEVlsmKjQ