Hangmam Game

Project Title: Hangman Game Implementation and Analysis

Authors: Gautham Srinivasan, Connor Nealis, Liam Kelly

Northeastern University

EECE 2140. Computing Fundamentals for Engineers

Fall 2024

Date: 11/22/2024

Abstract

This report outlines the design and development of a Hangman game, based on the classic word-guessing game. The project implements a Hangman game using Python's Tkinter library for the graphical user interface and custom game logic for word selection and gameplay. The game uses 2 main classes, one for managing the logic, and the other for controlling the UI and general gameplay loop. The UI consisted of a category selection screen and a main game screen. The main game screen has buttons to input guesses, a hangman drawing, and also displays what letters have been guessed so far. The game logic class contains function to check guesses, and to update the current state of the mystery word. It also can check the status of the game to see if the player has won or lost. We had several key takeaways from this project, mainly about the limitations of Python and Tkinter as game development tools. All features were fully implemented, providing users with a streamlined experience. We had other ideas to implement in future iterations of the game, such as a multiplayer mode and dynamic animations. Overall we met our objectives of creating a Python based video game, and there are several ways we can expand and improve the project in the future.

Contents

Chapter 1

Introduction

1.1 Background

Hangman is a popular word-guessing game where players attempt to find a hidden word by guessing letters within a limited number of tries. Our project recreated this using Python.

1.2 Problem Statement

To design a Hangman game that combines game play with word selection from a set of predetermined categories in simplistic, easy to follow user interface.

1.3 Objectives

Our project objectives were to design a word randomizer with several different categories to choose from, and then prompt the player to start guessing letters. If the letter inputted are contained within the word, all instances of the letter will be revealed. If the letter is not contained within the word, one of the attempts will be used up and the hangman picture will advance to the next stage. Once all 6 attempts have been exhausted the game will end, prompting the player to pic another category for the next round.

1.4 Scope

The project focuses on a single-player Hangman game with predefined word banks. While the player has the option to combine the word banks into one general bank, the individual words are limited to what we specifically code into the game.

Chapter 2

Technical Approaches and Code UML

2.1 Development Environment

This project was developed using Python 3.12.4 and the Spyder IDE. The Tkinter library was used to create the UI and the Random module was used for word selection. Image handling was done with the PIL library.

2.2 Implementation Details

The word selection algorithm uses a class of objects called WordBank that contain an array with all the words in a category. When a the function getRandomWord() is called, a random word from that array is returned.

The game logic is managed by the HangmanGame class. It contains several functions used to check the game status and verify guesses.

The setCategory() function is responsible for selecting the word to guess based on the category that is selected. It does this with a series of if statements for each category.

The displayWord() function is responsible for return what letters of of the word should be revealed. It checks to see if the each letter is on the list of correct guesses and then replaces them with an underscore if they are not

The makeGuess() function inputs a letter, and then checks to see if that letter is contained within the word. If it is, the function returns true and adds the letter to the list of correct guesses. If it isn't, it returns false and subtracts 1 from the amount of guesses left.

The gameStatus() function checks to see if the player has won or lost. If the amount of attempts ever reaches zero it returns that they lost. If the player fully guesses the word it returns that they won. If neither of these conditions have been met it returns that the game should continue.

The HangmanUI class manages the UI and gameplay loop of the program.

The initialization function creates all of the UI elements and also the first category selection screen. It first sets the background to the chalkboard image. After the background is set it displays all categories, and then waits for the player to select one before moving on to the main game. Each category button calls the setCategory() function for whatever category it corresponds to. Once a category and word have been selected, The rest of the main game elements, like win counter, letter buttons, and hangman are loaded in.

The createLetterBUttons() method is called at the end of the initialization stage to create an array of buttons. It uses a for loop to cycle through each letter of the alphabets, creating a button for each one. Each button has the calls a command onLetterButtonClick("A").

The onLetterButtonClick() functions activates whenever a button is clicked. It first calls the makeGuess function to see if the letter is correct. If it is it calls the displayWord function to update the players progress. It also updates the amount of attempts left and what letters have been guessed. It calls the updateHangman function before creating a variable to store the output of the gameStatus function. If the player has won, it creates a message box with a congratulatory message, before incrementing the amount of wins by one and calling the resetGame function. If the player lost it displays a messages stating they lost, and then calls the resetGame function. It also disables the button so it cannot be clicked on again.

The updateHangman function clears the current hangman image, and then updates it to the current stage.

The resetGame function, as the name suggests, resets the game. It resets all of the guesses, attempts, and buttons to their default state and brings back the category selection screen. Once a category button is clicked, a new round begins.

Chapter 3

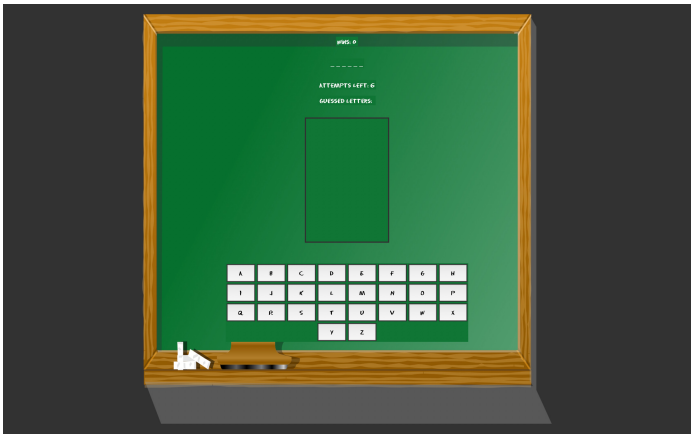
Project Demonstration

3.1 Screenshots and Code Snippets

When the player first opens up the program, they see a category selection screen.


```
# Category selection screen (move everything down with padding)
self.pick_category_label = tk.Label(root, text="Pick a category:", font=self.custom_font, bg=self.bg_color)
self.pick_category_label.pack(side='top', pady=(100, 10))  # Increased padding to move it down

self.var = tk.IntVar()
self.btn1 = tk.Button(root, text='fruits', command=lambda: [self.game.setcategory("fruits"), self.var.set(1)],
                      bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
self.btn1.pack(side='top', pady=5)
self.btn2 = tk.Button(root, text='places', command=lambda: [self.game.setcategory("places"), self.var.set(1)],
                      bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
self.btn2.pack(side='top', pady=5)
self.btn3 = tk.Button(root, text='coding', command=lambda: [self.game.setcategory("coding"), self.var.set(1)],
                      bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
self.btn3.pack(side='top', pady=5)
self.btn4 = tk.Button(root, text='all', command=lambda: [self.game.setcategory("all"), self.var.set(1)],
                      bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
self.btn4.pack(side='top', pady=5)

# Wait until a button is pressed and then clears selection screen elements
self.btn1.wait_variable(self.var)
self.btn1.destroy()
self.btn2.destroy()
self.btn3.destroy()
self.btn4.destroy()
self.pick_category_label.destroy()
```

Once a button is selected it advances to the main game stage.

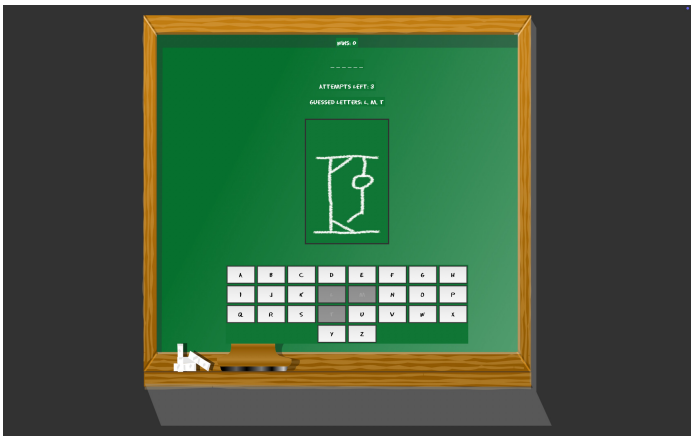


From here the play will continue to start selecting letters to try to guess the mystery word. When a letter is button is clicked, the program calls a function to check whether that letter is contained in the word. If it is it updates the UI to display all position of the guessed letter in the word.

```
def on_letter_button_click(self, letter):
    if not self.game.make_guess(letter):
        return

    self.word_label.config(text=self.game.display_word())
    self.guessed_label.config(text=f"Guessed Letters: {', '.join(sorted(self.game.guessed_letters))}")
    self.attempts_label.config(text=f"Attempts Left: {self.game.attempts_left}")

    self.letter_buttons[letter].config(state=tk.DISABLED)

    # Update hangman shape based on attempts left
    self.update_hangman(self.game.attempts_left)

    status = self.game.game_status()
    if status == "win":
        messagebox.showinfo("Congratulations!", f"You guessed the word: {self.game.word_to_guess}")
        self.wins += 1
        self.reset_game()
    elif status == "lose":
        messagebox.showinfo("Game Over", f"You lost! The word was: {self.game.word_to_guess}")
        self.reset_game()
```


```
def on_letter_button_click(self, letter):
    if not self.game.make_guess(letter):
        return

    self.word_label.config(text=self.game.display_word())
    self.guessed_label.config(text=f"Guessed Letters: {', '.join(sorted(self.game.guessed_letters))}")
    self.attempts_label.config(text=f"Attempts Left: {self.game.attempts_left}")

    self.letter_buttons[letter].config(state=tk.DISABLED)

    # Update hangman shape based on attempts left
    self.update_hangman(self.game.attempts_left)

    status = self.game.game_status()
    if status == "win":
        messagebox.showinfo("Congratulations!", f"You guessed the word: {self.game.word_to_guess}")
        self.wins += 1
        self.reset_game()
    elif status == "lose":
        messagebox.showinfo("Game Over", f"You lost! The word was: {self.game.word_to_guess}")
        self.reset_game()
```


If the letter guessed is not contained in the word, the number of attempts left is reduced by one and the Hangman picture updates to the next stage. The user can then input another guess.



This cycle repeats until all letters of the word has been guessed, or until the user is out of attempts. At which point the game will display a message stating whether they won or lost, and then reset back to the category selection screen. If the player won that round, the win counter will also increment by 1.

```python
def game_status(self):
    """Check if the game has been won or lost."""
    if self.attempts_left == 0:
        return "lose"
    if set(self.word_to_guess) == self.correct_guesses:
        return "win"
    return "continue"
```

```python
def reset_game(self):
    """Reset the game and show only the category selection buttons."""
    self.game = HangmanGame()

    # Remove all UI elements related to the game
    try:
        self.word_label.destroy()
        self.attempts_label.destroy()
        self.guessed_label.destroy()
        self.wins_label.destroy()
        self.canvas.destroy()
        self.letter_buttons_frame.destroy()
    except AttributeError:
        pass  # Handle if elements are already destroyed in case of multiple game resets

    # Show the category selection screen again
    self.pick_category_label = tk.Label(self.root, text="Pick a category:", font=self.custom_font, bg=self.bg_color)
    self.pick_category_label.pack(side='top', pady=(100, 10))  # Increased padding to move it down

    self.var = tk.IntVar()
    self.btn1 = tk.Button(self.root, text='fruits', command=lambda: [self.game.setcategory("fruits"), self.var.set(1)],
                          bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
    self.btn1.pack(side='top', pady=5)
    self.btn2 = tk.Button(self.root, text='places', command=lambda: [self.game.setcategory("places"), self.var.set(1)],
                          bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
    self.btn2.pack(side='top', pady=5)
    self.btn3 = tk.Button(self.root, text='coding', command=lambda: [self.game.setcategory("coding"), self.var.set(1)],
                          bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
    self.btn3.pack(side='top', pady=5)
    self.btn4 = tk.Button(self.root, text='all', command=lambda: [self.game.setcategory("all"), self.var.set(1)],
                          bg=self.bg_color, relief="flat", highlightthickness=0, font=self.custom_font)
    self.btn4.pack(side='top', pady=5)

    # Wait until a button is pressed and then clear the selection screen elements
    self.btn1.wait_variable(self.var)
    self.btn1.destroy()
    self.btn2.destroy()
    self.btn3.destroy()
    self.btn4.destroy()
    self.pick_category_label.destroy()

    # Recreate the game UI elements (word display, attempts, guessed letters, and canvas)
    self.wins_label = tk.Label(self.root, text=f"Wins: {self.wins}", font=self.custom_font, bg=self.bg_color)
    self.wins_label.pack(side='top', pady=(80, 10))  # Increased padding to move it further down

    self.word_label = tk.Label(self.root, text=self.game.display_word(), font=self.custom_font, bg=self.bg_color)
    self.word_label.pack(pady=20)

    self.attempts_label = tk.Label(self.root, text=f"Attempts left: {self.game.attempts_left}", font=self.custom_font, bg=self.bg_color)
    self.attempts_label.pack(pady=(10, 5))

    self.guessed_label = tk.Label(self.root, text=f"Guessed Letters: ", font=self.custom_font, bg=self.bg_color)
    self.guessed_label.pack(pady=10)

    # Hangman Canvas (add more padding to ensure it stays below the text)
    self.canvas = tk.Canvas(self.root, width=200, height=300, bg=self.bg_color)  # Remove grey background
    self.canvas.pack(pady=(20, 30))  # Increased padding to give more space between the canvas and text

    # Letter buttons (move all buttons down a bit)
    self.letter_buttons_frame = tk.Frame(self.root, bg=self.bg_color)  # Remove background color from frame
    self.letter_buttons_frame.pack(pady=(20, 40))  # Increased padding to move buttons further down

    self.create_letter_buttons()

    # Reset the hangman state and image
    self.canvas.delete("all")  # Clear the canvas
    self.canvas.create_image(100, 250, image=self.stand_photo_image, anchor=tk.S)  # Show initial hangman stand image
    self.update_hangman(self.game.attempts_left)
```

Chapter 4

Discussion

4.1 Future Improvements

This program can be improved in the future by utilizing Pygame to develop the UI. While Tkinter is very useful for developing UI for programs, in the context of game design there are areas where it falls short. Pygame is a library specifically designed for video games and its graphics engine would be much better suited for this project. Animated sprites and sound effects would be good things to add in the future. A better word bank system is also something that could come with future versions of this project. Currently, we have to manually add each word. A better approach would be to use some sort of dictionary or word list library, such as PyDictionary.

4.2 Project Limitations

We were largely limited by our experience in developing games in Python. We had many other planned features that we did not have the knowledge to implement, such as multiplayer modes. We were also heavily limited by the capabilities of Tkinter.

Chapter 5

Conclusion

5.1 Summary

The project successfully achieved its objectives by implementing a category-based Hangman game with a functional graphical interface. The program utilized a class-based design to seperately handle the UI and game logic. The UI was further enhance by using custom fonts and images to animate the hangman. It serves as a foundation for further enhancements in gameplay features.

5.2 Recommendations for Future Work

- Dynamic word banks with external sources (e.g., APIs).
- Additional gameplay modes (e.g., timed challenges).
- Enhanced visual design for better engagement.

Bibliography

[1] Python, "Graphical User Interfaces with Tk – Python 3.7.4 documentation," Python.org, 2019. [Online]. Available: `https://docs.python.org/3/library/tk.html`. [Accessed: Nov. 22, 2024].

[2] "Pillow: Python Imaging Library (Fork)," PyPI, Oct. 15, 2024. [Online]. Available: `https://pypi.org/project/Pillow/`. [Accessed: Nov. 22, 2024].

[3] W3Schools, "Python Classes," W3schools.com, 2019. [Online]. Available: `https://www.w3schools.com/python/python_classes.asp`. [Accessed: Nov. 22, 2024].

[4] R. Bansal, "Python GUI - tkinter - GeeksforGeeks," GeeksforGeeks, Jun. 17, 2017. [Online]. Available: `https://www.geeksforgeeks.org/python-gui-tkinter/`. [Accessed: Nov. 22, 2024].