Title: Word Castle

Author: Ahmet Furkan Atal

Project Description

'World Castle' is an engaging Python-based game developed primarily using the Pygame module. Designed to sharpen keyboard skills, this game challenges players with a series of levels, each presenting a specific set of words to be typed within a designated time frame. As players progress through the levels, they encounter floating blocks at the top of the screen, with the number of blocks corresponding to the current level (for instance, three blocks appear at level 3). To successfully pop these floating blocks, players must first pop the current floating block. (meaning that there are no two floating blocks in the screen at the same time)

One of the unique features of 'World Castle' is its integration of the Tkinter module's filedialog, which allows players to upload their own word lists in a text format. This customization enables players to practice typing specific words of their choice, enhancing the game's educational value. By engaging with 'World Castle', players not only improve their typing speed and accuracy but also have the opportunity to memorize vocabulary, making it an excellent tool for those looking to develop their keyboard proficiency and language skills in a fun, interactive environment.

The game is case sensitive(so be aware when you want to add new txt file, the user would want to change them or keep them as to make the game harder), pressing ‘enter’ when next level button is visible proceeds to the next level too(same as clicking to the next level button but as game begins user is used to keyboard and may not want to click with mouse), user can stop the music with clicking on the sound icon(I didn’t make an extra icon to turn off the sound effect(you need to turn off your computers sound for that) ),at the end of the game user can see the reached level and the score he/she achieved. I Left 2 more txt files in the files they can be loaded too when tkinter opens a file dialog.

Code Blocks

1.Global variables and imports and some initializations

A screenshot of a computer program

Description automatically generated

1. **Library Imports**: Imports Pygame for game development, along with other libraries for system functions, randomness, and graphical user interfaces.
2. **Pygame Initialization**: Initializes Pygame and its sound mixer module.
3. **Sound Setup**: Loads and loops background music, and loads a sound effect for block removal.
4. **Graphics for Sound Control**: Loads and resizes images for play and mute sound buttons.
5. **Game State and Configuration**:
   * Tracks the music playing state.
   * Sets the position for the sound button.
   * Defines the game window size (800x600 pixels).
   * Establishes color schemes and a default font for text.

This code sets up the basic environment, sound, and visual elements for the game

2. Block class

A screen shot of a computer program

Description automatically generated

This code defines a **Block** class for the game, which represents the blocks that appear on the screen:

1. **Class Definition**: **Block** class is defined with a default white color, and a black color option for text.
2. **Constructor (\_\_init\_\_)**: Initializes a block with:
   * **rect**: A Pygame rectangle object defining the block's position and size.
   * **word**: The word associated with the block.
   * **color**: The block's color, defaulting to white if not specified.
   * **floating**: A boolean indicating if the block is floating.
   * **velocity**: Sets the block's movement direction, initially upwards and to the left.
3. **Drawing the Block (draw method)**:
   * Draws the block on the screen using its color and rectangle dimensions.
   * Renders the associated word in black, centered within the block.

This class is essential for creating and displaying the blocks that players interact with during the game, each carrying a word for the typing challenge.

3.Button Class

A computer screen shot of code

Description automatically generated

This code defines a **Button** class in the game:

* **Initialization**: Sets up a button with position, size, text, background color, text color, and visibility.
* **Drawing**: If visible, it draws the button and centers the text on it.
* **Click Detection**: Includes a method to detect if the button is clicked based on mouse position.

3.Game Class

A screen shot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

* **Initialization**: In my game, the **Game** class is initialized with various buttons (start, next level, exit, load) and essential game attributes like score, level, and blocks. The class also sets up the game screen and loads a background image.
* **Block Initialization (initialize\_blocks)**: This method resets level-specific attributes and creates a grid of blocks using a word list. Each block is color-coded and positioned on the screen.
* **Removing a Block (remove\_block)**: This function manages the removal of blocks from the game. It plays a removal sound, updates the score, and handles the addition of new floating blocks based on the level progression.
* **Adding Floating Blocks (add\_floating\_blocks)**: My game adds floating blocks at random positions on the upper half of the screen. The number of these blocks corresponds to the current level number.
* **Updating Floating Blocks (update\_floating\_blocks)**: This part of the game moves the floating blocks around the screen. Their direction and color change when they hit the screen's edges or the midpoint vertically.

This class forms the core gameplay mechanics, handling block interactions, level progression, and the dynamic movement of floating blocks in my game.

4.Draw functions

A screen shot of a computer program

Description automatically generated

1. **Drawing the Menu (draw\_menu)**: Displays different buttons depending on the game state. If there are no blocks, the "next level" button is shown; otherwise, the "start" button is displayed. The "load" button appears only at level 0, and the "exit" button is always visible.
2. **Drawing the Game (draw\_game)**:
   * Draws each block on the screen.
   * Displays the user's current input and the game's score.
   * Manages and shows a countdown timer, updating every second. If it reaches zero, the game ends, and the game-over state is activated.
   * If all blocks are cleared, it returns to the menu.
3. **Drawing the Game Over Screen (draw\_game\_over)**:
   * Shows a "Game Over" message, the level reached, and the total score.
   * Displays the "exit" button and dynamically creates a "Play Again" button for restarting the game.

These methods control the visual presentation of the game, ensuring players are provided with the necessary information and options based on their progress and the game's state.

5.Handling Events

A computer screen shot of a program code

Description automatically generated

1. **Menu Events (handle\_menu\_events)**: Responds to button clicks to start the game, advance levels, exit, or load a custom word list. Also handles starting the game via the enter key.
2. **Game Events (handle\_game\_events)**: Processes keyboard inputs for typing and deleting text, and checks for matching words to remove blocks. Mouse clicks on blocks also trigger their removal.
3. **Game Over Events (handle\_game\_over\_events)**: Manages button clicks for exiting the game or replaying from the game over screen, including resetting the game state for a new round.

References

* Used the help of GitHub copilot specifically for the part where blocks are initialized.
* In the initialize\_blocks function,

