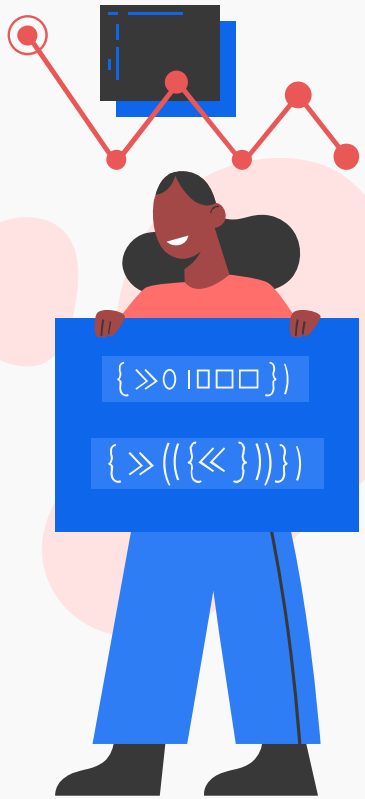
Word Search

by Hello World

Table of contents



Get to know our project...

Game features

Allows players to search for words across different levels.

User can choose difficulty levels (easy, medium, hard).

Originally developed for CLI, now enhanced with a user-friendly GUI.

Accessibility

User-friendly and inbuilt interface.

Players can navigate through a grid of words easily.

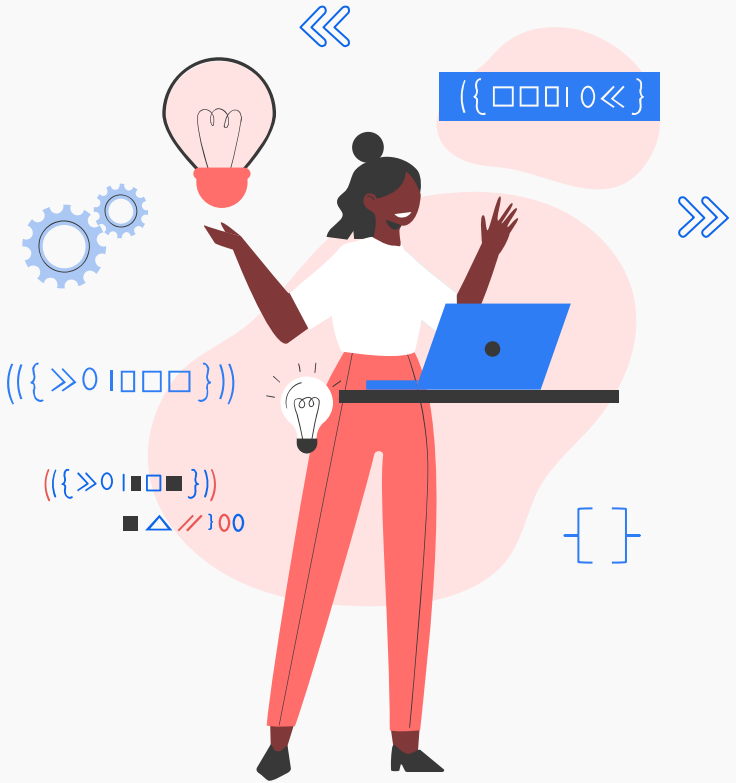
Words can be highlighted in various directions (vertical, horizontal, diagonal, and backward).

Challenges & Entertainment

Offers three difficulty levels to cater to players of all ages.

Provides a mix of challenge and entertainment.

Suitable for both relaxation and vocabulary improvement.



Theory Behind the Design

Category and Difficulty Level Selection

- Offers users the choice of selecting a category from a predefined set
- Difficulty levels determine the number of words, grid size, and maximum allowed attempts

Word Placement and Grid Generation

- Dynamically reads word lists from text files corresponding to the selected category
- Words are randomly placed on the grid, considering orientations such as horizontal, vertical, and diagonal
- The remaining grid cells are filled with random letters

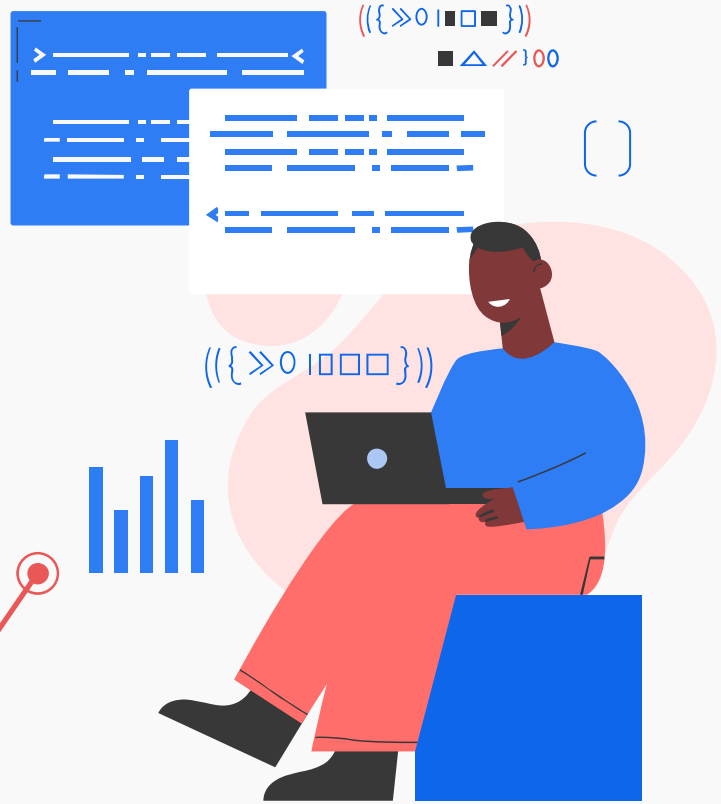
Scoring Mechanism

- Earn points based on the length of each word found and the user's speed
- Time bonus is applied for finding words quickly
- Total score accumulates as players successfully identify words within the grid

User Interaction and Visual Elements

- The graphical user interface features interactive elements, including input entries for name and category selection.
- Game grid is displayed as a set of labeled Tkinter labels
- Users can select letters by clicking on the corresponding labels.

Functions



word_choice()

```python
# Function that starts the game by managing the player's category choice and name input
def word_choice():
    # Create the main application window
    global root
    root = tk.Tk()
    root.title("Setup")

    # Get the dimensions of the current display
    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()

    # Set variables for window size
    window_width = 575
    window_height = 325

    # Calculate the x and y coordinates for the center of the screen
    x = (screen_width - window_width) // 2
    y = (screen_height - window_height) // 2

    # Set the window position
    root.geometry(f"{window_width}x{window_height}+{x}+{y}")

    # Create two input entries for player name and category choice
    create_input_entry(root, "Enter your name:")
    create_input_entry(root, "Choose category (1 for Animals, 2 for Countries and Capitals, 3 for Fruits and Vegetables, 4 for Sports):")

    # Create a menu button for difficulty selection
    difficulty_menu = tk.Menubutton(root, text="New Game", relief=tk.RAISED)
    difficulty_menu.pack(side="top", pady=7)
    difficulty_menu.menu = tk.Menu(difficulty_menu, tearoff=0)
    difficulty_menu["menu"] = difficulty_menu.menu

    # Add menu options for each difficulty
    # The argument for the difficulty parameter is the driver for the difference in modes
    difficulty_menu.menu.add_command(label="Easy", command=lambda: mode(root, "easy"))
    difficulty_menu.menu.add_command(label="Medium", command=lambda: mode(root, "medium"))
    difficulty_menu.menu.add_command(label="Hard", command=lambda: mode(root, "hard"))

    # Define and pack a button that shows all existing save files
    load_menu_button = tk.Menubutton(root, text="Load Game", relief=tk.RAISED)
    load_menu_button.pack(side="top", pady=7)
    load_menu_button.menu = tk.Menu(load_menu_button, tearoff=0)
    load_menu_button["menu"] = load_menu_button.menu

    # Use the function for checking save files to fill the menu
    existing_saves = get_existing_saves()
    if existing_saves:
        # Add the save file name as a button under the menu
        for save_file in existing_saves:
            load_menu_button.menu.add_command(label=save_file, command=lambda sf=save_file: load_game(sf))
    # If no save files are detected, display a disabled button
    else:
        load_menu_button.menu.add_command(label="No saved games found", state=tk.DISABLED)

    # Start the Tkinter event loop
    root.mainloop()
```

create_input_entry (window, label_text)

```
# Function to create an input entry and a label above it in a chosen window
def create_input_entry(window, label_text):
    label = tk.Label(window, text=label_text)
    label.pack(side="top", pady=10)

    entry = tk.Entry(window)
    entry.pack(side="top", pady=10)
```

```python
# Function for starting the game according to the difficulty button clicked by the user (different word count and grid size)
def mode(window, difficulty):
    # Declare a global variable to store the words to find
    global words_to_find
    # Declare the number of maximum tries as a global
    global max_tries
    # Initialize an empty list for the words to find in the grid
    words_to_find = []
    # Getting the player's input from their respective widget and assigning them to new variables
    entry_name = window.winfo_children()[1].get()
    category_choice = window.winfo_children()[3].get()

    # Try and except block to handle a prevent upcoming error coming from the player's input
    # Try to convert the player's category of choice to an integer
    try:
        category_choice = int(category_choice)
    # If the player's input cannot be converted to an integer
    except ValueError:
        # Display an error message for invalid input
        invalid_category = tk.Label(root, text="Enter a valid category number")
        invalid_category.pack(side="top", pady=8)
        root.after(1500, lambda: invalid_category.pack_forget())
        # Exit the whole function and repeat when the button is pressed until a valid input is given
        return

    if 1 <= category_choice <= 4:
        # Construct the file path for the selected category
        category_file_name = f'Word Lists\\{categories[category_choice]}.txt'
        # Read words from the selected category file
        words = read_words_from_file(category_file_name)

        # Determine the number of words, maximum tries and grid size based on the argument given for the difficulty parameter
        if difficulty == "easy":
            num_words = 5
            grid_size = 8
            max_tries = 2

        elif difficulty == "medium":
            num_words = 7
            grid_size = 10
            max_tries = 4

        elif difficulty == "hard":
            num_words = 10
            grid_size = 12
            max_tries = 6

        while True:
            # Choose a random word from the list and strip whitespace
            new_word = random.choice(words).strip()
            # Exit the loop if the number of words needed is attainted
            if len(words_to_find) == num_words:
                break
            # Add the word to the list defined above if its not already in the list
            elif new_word not in words_to_find:
                words_to_find.append(new_word)
            else:
                continue

        # Construct a hint message based on the word randomly chosen from the text file
        # Words will be displayed in a normal sequence, separated by a comma
        capitalized_words = [word.capitalize() for word in words_to_find]
        hint_text = f"Hi {entry_name}, how many words can you find?\nHint: the words are {', '.join(capitalized_words)}"

        # Start the word search game with the arguments based on the difficulty chosen
        create_word_search(grid_size, grid_size, words_to_find, max_tries, hint=hint_text)

    # Display an error message for invalid category number
    else:
        invalid_category = tk.Label(root, text="Invalid category number\n Choose from (1, 2, 3 or 4)")
        invalid_category.pack(side="top", pady=8)
        root.after(1500, lambda: invalid_category.pack_forget())
```

mode(window, difficulty)

read_words_from_file (file_path)

```
# Function to read words from a file and store each line as an element of a list
def read_words_from_file(file_path):
    with open(file_path, 'r') as list_file:
        return list_file.readlines()
```

create_word_search (rows, cols, words_to_find, tries, hint)

```python
# Function to create the word search game window
def create_word_search(rows, cols, words_to_find, tries, hint=None):
    # Global variables to avoid variable errors
    global start_time
    global total_score
    # Create a new top-level fullscreen window for the game
    global game_window
    game_window = tk.Toplevel()
    game_window.title("Word Search by Hello World")
    game_window.attributes('-fullscreen', True)
    game_window.configure(bg="#C6DEF1")

    # Create and pack a frame within the window
    grid_frame = tk.Frame(game_window)
    grid_frame.pack()
    # Create and pack a submit button below the game grid
    submit_button = tk.Button(game_window, text="SUBMIT", font=("Impact", 15), width=10, height=3, bg="lightgreen", relief=tk.RAISED, command=lambda: checkWords(grid))
    submit_button.pack(side="bottom", pady=10)

    # Initialize an empty list to represent the grid of labels
    grid = []
    for _ in range(rows):
        # Initialize an empty list to represent a row in the grid
        row = []
        for _ in range(cols):
            # Create a label for each cell in the grid and determine its configuration
            cell_label = tk.Label(grid_frame, text='_', font=("Arial", 9, "bold"), width=6, height=4, borderwidth=2, bg="white", relief=tk.RIDGE)
            row.append(cell_label)
        # Add the row of labels to the grid list
        grid.append(row)

    # Add an ESC key bind to the game window
    game_window.bind("<Escape>", lambda event: on_escape_key(root))

    # Get the time at the beginning of the game for score calculation
    start_time = time.time()
    # Initialize the score counter
    total_score = 0

    # Define and pack a hint message to help the player
    hint_label = tk.Label(game_window, text=hint, bg="#C6DEF1", fg="darkorange", bd=4, font=("Verdana", 14, "bold italic"))
    hint_label.pack(side="top", pady=15)

    # Call the functions that create the random letter grid and display it
    add_words_to_grid(grid, words_to_find)
    fill_empty_spaces(grid)
    display_grid(grid)
```

add_words_to_grid (game_grid, words)

```
# Function used for word placement and/or arrangement
def add_words_to_grid(game_grid, words):
    # Loop over every word in the list of word that we want to add
    for word in words:
        # Flag that acts as an on/off switch to the loop as words get placed
        placed = False
        # Loop until name is placed and flag (placed) is True
        while not placed:
            # Picking a random starting coordinate (row, column) on the grid
            row = random.randint(0, len(game_grid) - 1)
            col = random.randint(0, len(game_grid[0]) - 1)
            # Picking a random direction for the word using x and y coordinate translation
            orientations = [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (1, -1), (-1, -1), (-1, 1)]
            direction = random.choice(orientations)

            # Another flag variable used in the inner loop
            valid_placement = True
            # Iterating over every letter of the word
            for i in range(len(word)):
                # Calulating the exact coordinates of each letter using the chosen random direction
                new_row = row + i * direction[0]
                new_col = col + i * direction[1]

                # Checking if the letter placement is in the grid and does not overlap with others letters
                if not (0 <= new_row < len(game_grid) and 0 <= new_col < len(game_grid[0])) or game_grid[new_row][new_col]['text'] != '_':
                    valid_placement = False
                    break

            # If all letters are eligible to the above conditions, we can start placing the individual letters
            if valid_placement:
                for i in range(len(word)):
                    # Uses the calculated row and column values to reach the specific cell for the letter being placed, then replaces the original underscore (_), with the the letter
                    game_grid[row + i * direction[0]][col + i * direction[1]]['text'] = word[i]  # Place the word in the grid
                placed = True
```

fill_empty_spaces (game_grid)

```
# Function for filling the empty spaces in the grid (containing an underscore)
def fill_empty_spaces(game_grid):# Iterate over each row in the grid
    # Iterate over each row in the grid
    for row in range(len(game_grid)):
        # Iterate over each column in the current row
        for col in range(len(game_grid[0])):
            # Check if the current cell is empty/contains an underscore (_)
            if game_grid[row][col]["text"] == '_':
                # If it's empty, fill it with a random capital letter
                game_grid[row][col]["text"] = random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ')  # Fill empty spaces with random letters
```

display_grid (game_grid)

```
# Function for displaying the grid as separate labels in the window
def display_grid(game_grid):
    # Nested loop to reach a single cell
    for r in range(len(game_grid)):
        for c in range(len(game_grid[0])):
            # Grid the label in the specified row and column
            game_grid[r][c].grid(row=r, column=c)
            # Bind a click event to each label to change the background color when it is clicked using Mouse Button 1
            game_grid[r][c].bind("<Button-1>", lambda event, label=game_grid[r][c]: on_cell_click(label))
```

on_cell_click(label)

```
# Function for changing the color of each label when a certain button input is given
def on_cell_click(label):
    # Get the current background color of the label
    current_bg = label.cget("bg")
    # Change the background color to yellow if it's currently white
    if current_bg == "white":
        label.config(bg="yellow")
    elif current_bg == "yellow":
            # Change the background color to white if it's currently not white
        label.config(bg="white")
```

checkWords (game_grid)

```python
# Function for checking right and wrong attempts by the user
def checkWords(game_grid):
    # Set some variables as globals for easier accessibility
    global found_counter
    global color_counter
    global tries_counter
    global max_tries
    global total_score
    global start_time

    # Empty set to store the player's selected letters
    selected_letters = set()
    # Convert words_to_find list into a list of sets
    words_to_find_sets = [set(word) for word in words_to_find]
    # List of colors to be used for highlighting correct answers
    answer_colors = ["red", "blue", "green", "grey", "purple", "orange", "pink", "turquoise", "brown"]

    # Loop over every row in the grid
    for r in range(len(game_grid)):
        # Loop over every column of each row
        for c in range(len(game_grid[0])):
            # Get the text for all labels with a yellow background
            if game_grid[r][c].cget("bg") == "yellow":
                selected_letters.add(game_grid[r][c].cget("text"))

    # Check if the selected letters match any of the word sets
    for word_set in words_to_find_sets:
        if selected_letters == word_set:
            # Scoring mechanism
            # Get the time at the moment of the submission
            end_time = time.time()
            # Calculate the time taken by subtracting end from start times
            time_taken = int(end_time) - int(start_time)
            # Give bonus points based on the length of the word found
            word_length_bonus = len(word_set)
            # Give bonus points if the word is found in less than 15 seconds
            time_bonus = max(0, 15 - time_taken)
            word_score = int(word_length_bonus + time_bonus)

            # Add individual submission points to the total
            total_score += word_score
            # Increment counters by 1
            found_counter += 1
            color_counter += 1
            # Define and pack a correct answer label widget that disappears after 1.5 seconds
            correctLabel = tk.Label(game_window, text=f"Correct Answer\nScore: {total_score}", font=("Courier New", 13, "bold"), fg="darkgreen", bg="#C6DEF1")
            correctLabel.pack(side="bottom", pady=5)
            game_window.after(1500, lambda: correctLabel.pack_forget())
```

checkWords (game_grid) continued

```
        # Restart time for the next submission
        start_time = time.time()

        # Check for yellow background cells again and turn their background to a different color, so they can no longer be selected
        for r in range(len(game_grid)):
            for c in range(len(game_grid[0])):
                # Assign the background color based on the color counter value
                if game_grid[r][c].cget("bg") == "yellow" and color_counter <= 9:
                    game_grid[r][c].config(bg=answer_colors[color_counter - 1])

        # Display a congratulations message when the player gets all words
        if found_counter == len(words_to_find):
            messagebox.showinfo("Congratulations", f"You Win!\nTotal Score: {total_score}")
            game_window.after(1000, game_window.destroy())
        return

    # Display a disappearing wrong answer label widget when the selected letters do not resemble a word
    incorrectLabel = tk.Label(game_window, text=f"Wrong Answer\n Remaining tries:{max_tries-1}", font=("Courier New", 13, "bold"), fg="red", bg="#C6DEF1")
    incorrectLabel.pack(side="bottom", pady=5)
    game_window.after(1500, lambda: incorrectLabel.pack_forget())
    # Decrement tries left by one
    max_tries -= 1
    # Exit the game if all tries are used up
    if max_tries == 0:
        messagebox.showinfo("Game Over", "You've run out of tries.")
        game_window.destroy()
        root.destroy()
```

get_existing_saves()

```python
# Function for retrieving all json save files
def get_existing_saves():
    # Checking for files in a specific folder with a specific ending and extension and adding them to a list
    try:
        save_files = [file for file in os.listdir("Saves") if file.endswith("_game_progress.json")]
        return save_files
    # Return an empty list if no files are detected
    except FileNotFoundError:
        return []
```

on_escape_key (window)

```python
# Function for binding the ESC key to a function
def on_escape_key(window):
    # Open a top level pause menu when ESC is pressed
    pause = tk.Toplevel(window)
    pause.geometry("250x125")
    pause.title("Game Paused")

    # Label widget for the pause menu window
    message_label = tk.Label(pause, text="Do you want to:\n(i) Exit and save?\n(ii) Exit without saving?")
    message_label.pack(pady=10)
    # Button for saving game progress and exiting
    save_and_exit_button = tk.Button(pause, text="i", command=lambda: handle_exit(root, pause, True))
    save_and_exit_button.pack(side="left", padx=20)
    # Button for exiting without saving
    exit_button = tk.Button(pause, text="ii", command=lambda: handle_exit(root, pause, False))
    exit_button.pack(side="right", padx=20)
```

handle_exit (primary, secondary, save)

```python
# Function for handling the process of closing the game
def handle_exit(primary, secondary, save):
    # If the passed parameter is true, the player's name will be used as a file name for the save file
    if save == True:
        player_name = primary.winfo_children()[1].get()
        save_game_to_json(player_name)
    # Close all windows after the player chooses to exit
    secondary.destroy()
    primary.destroy()
```

save_game_to_json (username)

```python
# Function for saving the neccesary game data to a json file
def save_game_to_json(username):
    data = {
        "player_name": username,
        "found_counter": found_counter,
    }
    # Transfer the game data to a json file named after the player's name
    file_name = f'Saves\\{username.replace(" ", "_")}_game_progress.json'
    with open(file_name, 'w') as save_file:
        json.dump(data, save_file, indent=4)
```

load_game(file_name)

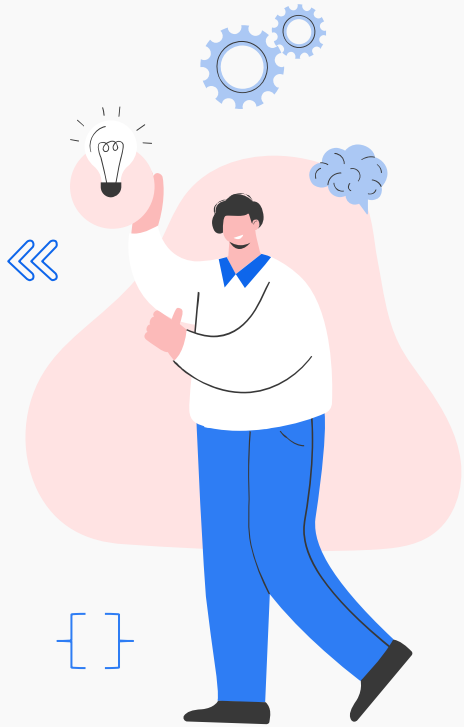
```
# Function for loading player's progress back from a json file (work in progress)
def load_game(file_name):
    global found_counter
    try:
        with open(file_name, 'r') as save_file:
            data = json.load(save_file)
            found_counter = data.get("found_counter", 0)
            messagebox.showinfo("Game Loaded", f"Welcome back! Your progress has been loaded from {file_name}.")
            root.destroy()
            create_word_search(8, 8, words_to_find)
    except FileNotFoundError:
        messagebox.showerror("Error", f"Error loading {file_name}. File not found.")
    except json.JSONDecodeError:
        messagebox.showerror("Error", "Error loading game data. Starting a new game.")
```

- Learning tKinter
- Grid generation and display
- Event handling
- File Handling
- Data Management
- Save and Load Mechanism

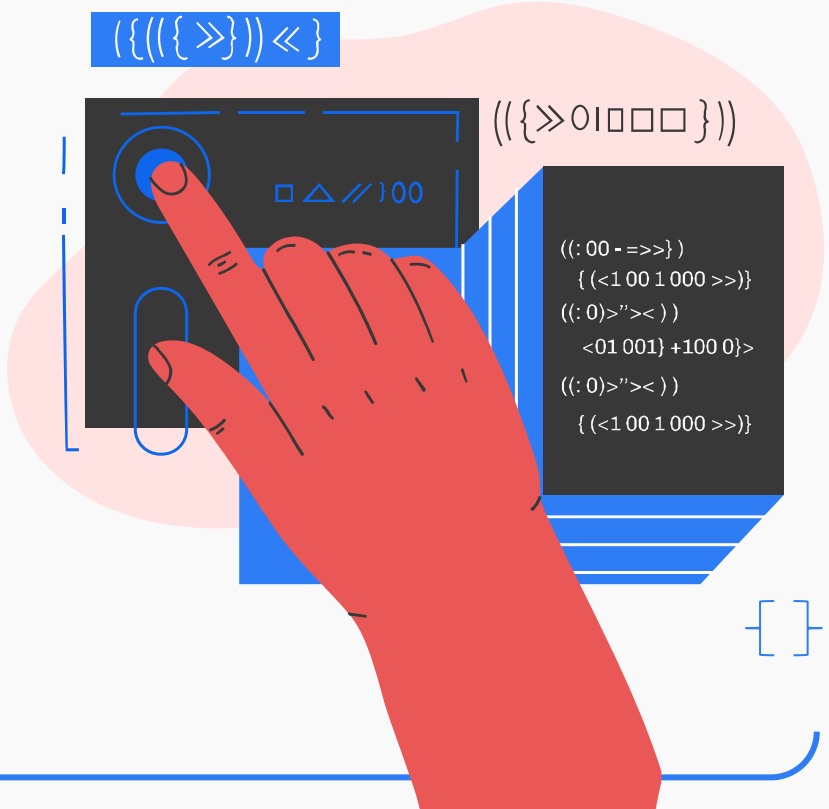
Challenges We Faced

Future Improvements/Work

- Complete loading functionality
- Reduce dependency on global variables
- Better visual appeal



Thanks!



CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**