Title Page

- **Project Title**: SOFTWARE ENGINEERING PROJECT DOCUMENTATION

- **Student Name**:Andrew Lou

- **Date**:4/7/25

- **Course**: Software Engineering Stage 6

- **GitHub URL**:https://github.com/AndrewLou1/software-engineering/tree/main

Table of Contents

1. Identifying and Defining

1.1 Problem Statement

What problem are you solving? Who is affected?

The problem I am aiming to solve is the growing education crisis happening around the world. This is caused by issues such as not having enough teachers, a lack of good learning resources, and students not performing as well as they used to. This mainly affects young children and school students, who are missing out on important learning experiences. If this continues, it could lead to bigger problems in the future, such as fewer opportunities for students and wider gaps in education.

1.2 Project Purpose and Boundaries

What is the project trying to achieve? What is in and out of scope?

The purpose of my project is to help students who are interested in technology learn the basics of software development. This will be done through a simple and user-friendly game where players answer questions to test and build their knowledge. The goal is to make learning fun and engaging while teaching important skills in a way that's easy to understand.

Some factors within the scope of my project include creating a user-friendly game, providing accurate information, and helping students gain basic knowledge about software development. Things outside the scope include in depth coding tutorials, and covering every area of the concept of software.

1.3 Stakeholder Requirements

Who are your stakeholders? What do they need from this software?

The stakeholders of my game include young children who are interested in software development. They need the software to provide accurate, engaging, and friendly information that will help them understand key points of software development.

1.4 Functional Requirements

- Interactable blocks
- Log in/sign up
- Score system / life system

1.5 Non-Functional Requirements

- UI
- Mode difficulty

1.6 Constraints

There were many constraints in my project, including time, budget, and technical limits. In time, there were constraints on my time as we were only given 10 weeks to complete the project, this led to an average game with limited features. There were also constraints for budget, as it was a school project and we were not given money, so we could only use free resources.

2. Research and Planning

2.1 Development Methodology

Agile methodology is the process of breaking projects down into smaller and manageable cycles called sprints. This allows for continuous improvement on the project based on problems/feedback you find during development.

The reason I chose it is because it suits game development well, as it lets me test features early, fix bugs quickly, and improve the game step by step. It also helps me stay organised, work more efficiently, and make sure each part of the game is working before moving on to the next.

2.2 Tools and Technology

Languages, IDEs, libraries, frameworks used.

My game was developed using Python for its simplicity, pygame was the main library used to create the game interface and handle gameplay mechanics, and crypt was used for secure password hashing during login and signup. My project was built and tested in Visual Studio Code

2.3 Gantt Chart / Timeline

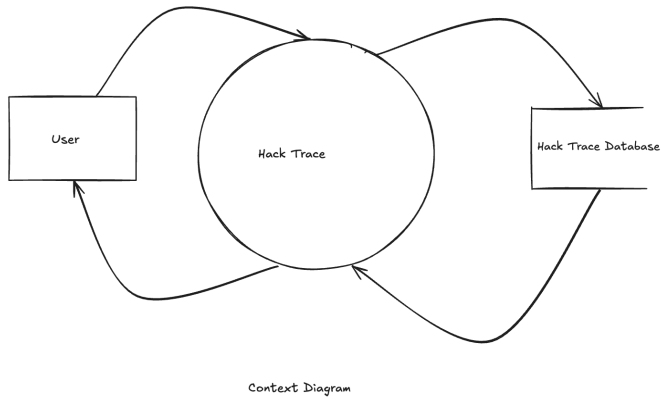|	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
Planning										
Design										
Development sprint 1										
Development sprint 2										
testing/debugging										
Evaluation / deployment										

2.4 Communication Plan

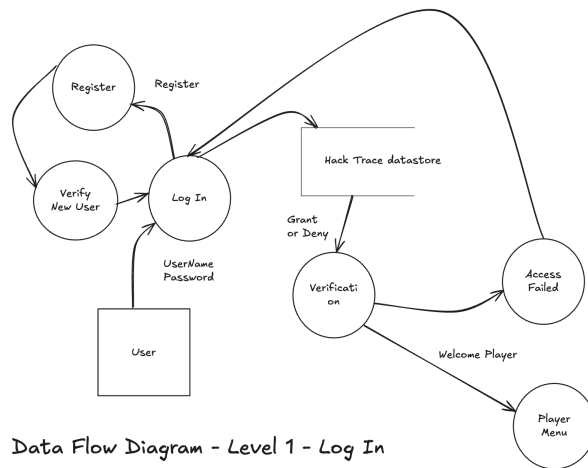How will you engage with your client or simulate feedback?

I will engage with my clients by implementing new features that users/clients may want or need, and fix any bugs reported from clients, for a more user friendly game
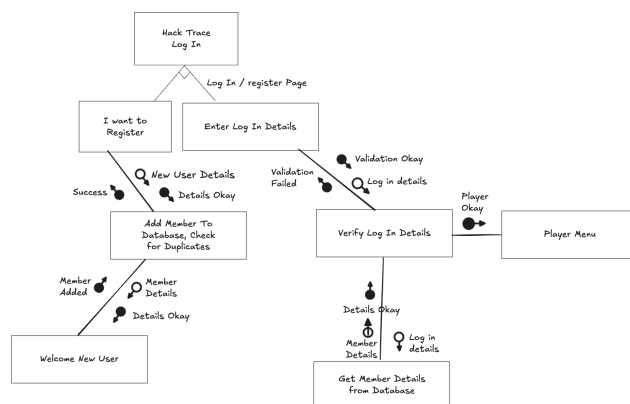
3. System Design

3.1 Context Diagram

User

Hack Trace

Hack Trace Database

Context Diagram

3.2 Data Flow Diagrams (Level 0 and 1)

Register

Register

Verify
New User

Log In

Hack Trace datastore

Grant
or Deny

UserName
Password

Verificati
on

Access
Failed

User

Welcome Player

Player
Menu

Data Flow Diagram - Level 1 - Log In

3.3 Structure Chart

Hack Trace
Log In

Log In / register Page

I want to
Register

Enter Log In Details

Validation Okay

New User Details

Log in details

Details Okay

Validation
Failed

Success

Add Member To
Database, Check
for Duplicates

Verify Log In Details

Player
Okay

Player Menu

Member
Added

Member
Details

Details Okay

Member
Details

Details Okay

Log in
details

Welcome New User

Get Member Details
from Database

3.4 IPO chart

Input	Process	Output
Username, Password	- Validate password- Hash password using bcrypt- Store in users.json if username is new	Message indicating success or reason for failure
Username, Password	- Load users from users.json- Verify username and check password hash	Message if login successful or failed
Mouse clicks	- Detect selected button (LOGIN, SIGN UP, INFORMATION, QUIT)	Opens corresponding screen
Mouse click on difficulty button	- Load corresponding question set (easy, medium, hard)- Display loading screen	Starts selected game room
WASD keys	- Update player sprite position	Updated position of player on screen
Timer-based	- Randomly choose a question from the list- Place a QuestionBlock sprite at random position	New question block appears on screen
E key near block	- Show question and answer options- Wait for 1–4 key input- Check answer correctness	Score/lives updated, feedback message shown
Player actions, timer	- Track score- Track lives- Remove expired blocks- Trigger win/lose screens	Game state changes or ends
Password string	- Check length, symbols, uppercase, not same as username	Error message or allow sign-up
Keyboard input	- Render text with optional masking for passwords	Captured string for use in login/signup
Long string, font, max width	- Split text into lines that fit inside a box (used for questions/options)	List of strings that are displayed properly

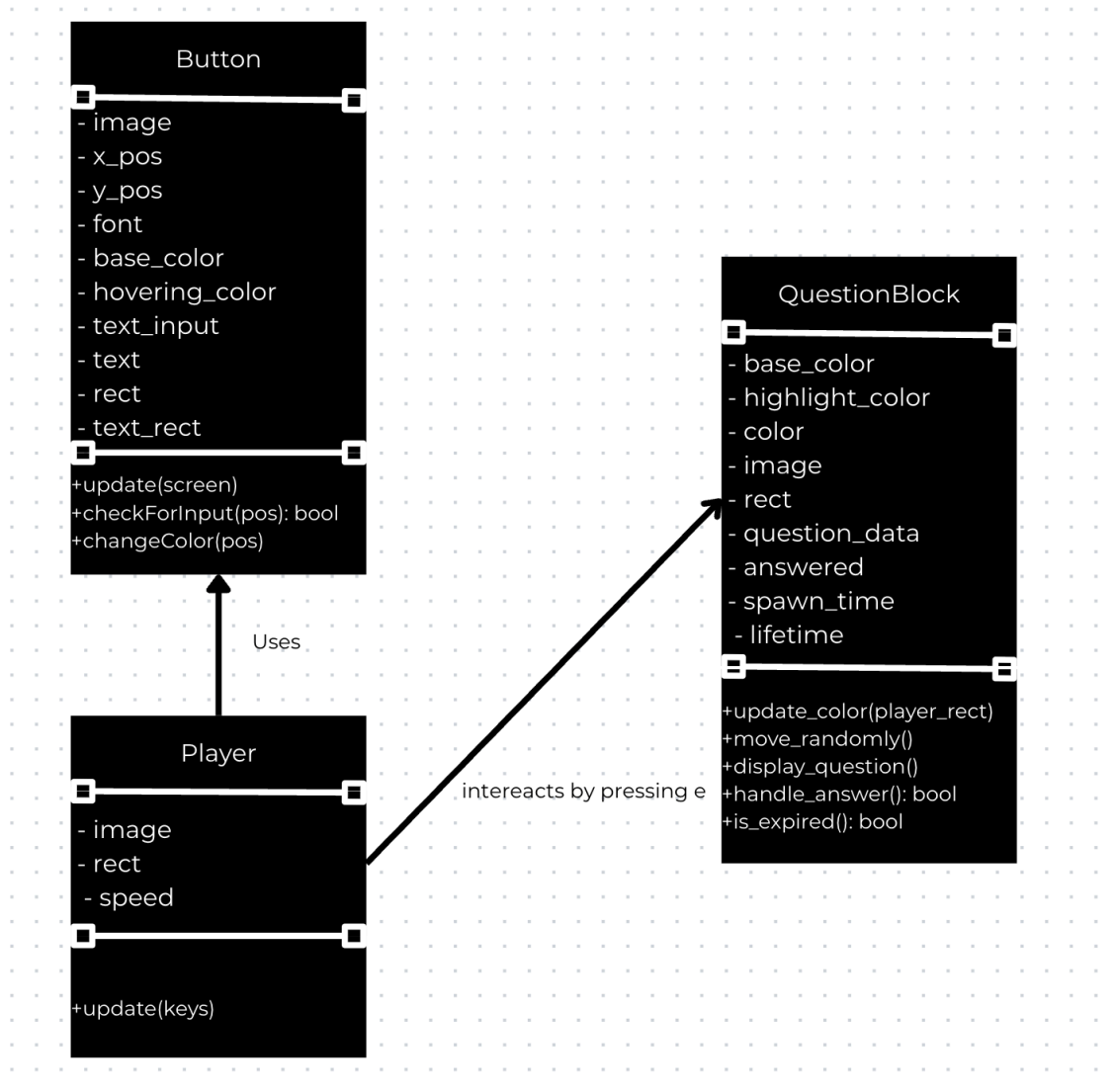| Mouse click	- Detect resume, return to menu, or mode select	Game continues or switches screen
Answer correctness or timeout	- Display "Correct!" or "Wrong! -1 Life" or "Too Slow!" message for 4 seconds	On-screen feedback
Read/write access to users.json	- Read/write JSON-formatted dictionary	Persistent user data storage

3.5 Data Dictionary

Name	Type	Description
username	String	Stores the username of the logged-in player
password	String (hashed)	Hashed password stored for account authentication
USERS_FILE	String (filename)	File path for storing user account data in JSON format
users	Dictionary	Key-value pairs of usernames and their hashed passwords
questions_easy	List of Dict	List of question objects for Easy level, with options and correct answer index
questions_medium	List of Dict	List of question objects for Medium level
questions_hard	List of Dict	List of question objects for Hard level
Player	Class	Represents the player character sprite
QuestionBlock	Class	Represents an interactable question cube on the screen
Button	Class	UI element that handles user interactions

text_input()	Function	Custom input field to collect usernames or passwords from the user
SCREEN	pygame.Surface	Main display surface where all rendering occurs
BG	pygame.Surface	Background image displayed in the main menu
score	Integer	Player's current score in a room
lives	Integer	Player's remaining lives in a level
question_data	Dictionary	A single question object with text, options, and answer
answered	Boolean	Tracks if a question block has been interacted with already
active_blocks	List	List of currently visible and unexpired question blocks
rect	pygame.Rect	Position and size of an object
font_path	String	Path to the TTF font file used in the game
wrap_text()	Function	Handles word wrapping for long text to fit the screen
get_font()	Function	Returns a pygame font object for a given size
sign_up()	Function	Registers a new user and saves their hashed password
login()	Function	Authenticates a user by verifying their password
save_users()	Function	Writes updated user data to the JSON file
load_users()	Function	Loads user data from the JSON file
hash_password()	Function	Hashes a password using bcrypt
verify_password()	Function	Verifies if a plain password matches the stored hash

3.6 UML Class Diagram (if OOP)

Button

- image
- x_pos
- y_pos
- font
- base_color
- hovering_color
- text_input
- text
- rect
- text_rect

+update(screen)
+checkForInput(pos): bool
+changeColor(pos)

QuestionBlock

- base_color
- highlight_color
- color
- image
- rect
- question_data
- answered
- spawn_time
- lifetime

+update_color(player_rect)
+move_randomly()
+display_question()
+handle_answer(): bool
+is_expired(): bool

Uses

Player

- image
- rect
- speed

+update(keys)

intereacts by pressing e

4. Producing and Implementing

4.1 Development Process
How did you approach the building of your solution?

To develop my project, I began by creating a proposal and planning diagrams to map out how the game was gonna look and work. I then used the agile methodology, breaking my work down into smaller sprints, which allowed me to focus on certain parts of the game step by step, such as user login, question handling, and game mechanics. This helped me stay organised, test regularly, and make improvements as I developed the code.

4.2 Key Features Developed
Describe and justify the core features

1. Interactive question system

   The game includes an active question and answer mechanic, where the users walk up to the block and press a key to answer educational questions

   Each question is about cyber security and software development, with increasing difficulty depending on levels.

   Justification: The system reinforces knowledge through the use of repetition and recallment.

2. Score and life system

   Players earn points through correct answers and lose lifes for wrong answers or if they take too long.

   The game ends when they reach enough points or they lose all lives

   Justification: Making the project more game with consequences leads to increased motivation and encourages more active thinking

3. Login and sign up with validation

   My project includes a secure signup/login system with password rules like minimum length, symbols and upper case letters.

   Passwords are hashed and validated

   Justification: Allows for a more secure app, and encourages good cybersecurity habits and demonstrates real-world authentication systems.
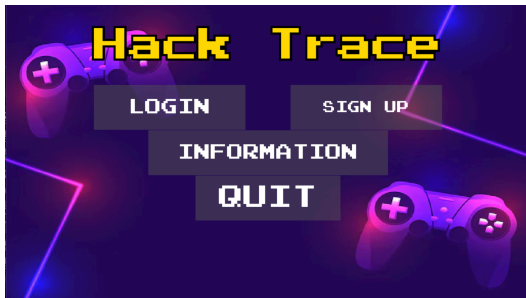
4. Difficulty levels

PLayers can choose from 3 different difficulties, each with its own set of questions and increasing score requirements.

Allows for players of different skill levels

Justification: a variety of levels support differentiated learning and provide a sense of accomplishment as players advance.

4.3 Screenshots of Interface

Main menu lets users navigate between login, signup and information	Sign in screen, lets users make an account

Mode select Allows users to select between 3 different modes	Select a Difficulty Easy Medium Hard BACK
Log in screen Allows users to login using a made account	Enter username: | BACK Enter password: | SHOW BACK

5. Testing and Evaluation

5.1 Testing Methods Used

Unit testing, system testing, integration testing, and acceptance testing.

5.2 Test Cases and Results

UT- unit testing
ST- system testing
IT- integration testing
At-acceptance testing

Test ID	Description	Expected Result	Actual Result	Pass/Fail

UT01	Login with valid user	Success message	Success message	pass
UT02	Invalid user login	Error message	Error message	pass
UT03	Sign up with existing username	Error message	Error message	pass
UT04	Sign up with valid credentials	success message	Success message	pass
UT05	Password same as username	Error message	Error message	pass
UT06	Password symbol missing	Error message	Error message	pass
IT01	Correct answer in game	+1 score, "Correct!" shown	+1 score, "Correct!" shown	pass
IT02	Wrong answer in game	-1 score, "wrong" message	-1 score, "wrong" message	pass
IT03	Time out (no answer)	-1 life, "too slow" message	-1 life, "too slow" message	pass
ST01	Pause menu opens from level	Menu screen displays	Menu screen displays	pass
AT01	Game launches correctly	Mainu menu appears with options	Mainu menu appears with options	pass
AT02	User can sign up with valid credentials	Sign up successful message appears	Sign up successful message appears	pass
AT03	Users login with valid username and password	Moved to difficulty selection screen	Moved to difficulty selection screen	pass

5.3 Evaluation Against Requirements
How well does the solution meet goals?

My solution met my goal very well as my goal was originally gonna be a game to fix the educational crisis currently happening around the world, and my game helps young students gain a deeper understanding of software development while keeping it fun and engaging.

5.4 Improvements and Future Work
What would you improve or add if you had more time?

The things I would improve/add would include the visuals like sound effects, animations, and better character designs. I would also like to make the gameplay more smooth and responsive. Additionally I would have loved to add features such as leaderboards for fastest completions, user progress tracking and more unique questions to make the game more fun and engaging.

6. (Client) Feedback and Reflection

6.1 Summary of Client Feedback
List of client comments or peer feedback.

Person	Feedback
eden	Decent game, could be improved by implementing simpler UI and a more intuitive experience, found some areas hard to understand
Joshua	I, Joshua eum, sister of jennifer eum, i believe this project is good. However, it could be improved by implementing a wider variety of levels and gameplay and enhancing graphics.
Vincent	Overall decent game, can be improved by implementing cleaner UI and a more intuitive experience, found some components hard to understand

6.2 Personal Reflection
What did you learn? What skills did you gain?

From making this project I improved my python and pygame skills to create a game, and gained a stronger grip on managing my time through a strong system of software development.

7. Appendices

- Full Gantt Chart

- Complete Data Dictionary

- Full Test Logs

- Raw Interview Notes / Feedback Forms

- Exemplar Code Snippets

Question block

```python
class QuestionBlock(pygame.sprite.Sprite):

    def __init__(self, pos, question_data):

        super().__init__()

        self.base_color = pygame.Color('yellow')

        self.highlight_color = pygame.Color('white')

        self.color = self.base_color

        self.image = pygame.Surface((50, 50))

        self.image.fill(self.color)

        self.rect = self.image.get_rect(center=pos)

        self.question_data = question_data

        self.answered = False

        self.spawn_time = pygame.time.get_ticks()

        self.lifetime = 7000  # Block disappears after 7 seconds if not
answered


    def update_color(self, player_rect):

        # Highlight if player is close and it hasn't been answered

        if self.rect.colliderect(player_rect.inflate(30, 30)) and not
self.answered:
```

```python
                self.color = self.highlight_color

        else:

                self.color = self.base_color

        self.image.fill(self.color)


    def move_randomly(self):

        # Move block to a random position on screen

        self.rect.center = (random.randint(100, 1180), random.randint(100,
620))


    def display_question(self):

        SCREEN.fill("black")

        font = get_font(28)


        # Draw the wrapped question text

        wrapped_lines = wrap_text(self.question_data['question'], font,
1000)

        for i, line in enumerate(wrapped_lines):

            rendered = font.render(line, True, "white")

            SCREEN.blit(rendered, (140, 50 + i * 35))


        # Draw each answer option

        for i, opt in enumerate(self.question_data['options']):

            opt_lines = wrap_text(f"{i+1}. {opt}", get_font(24), 1000)

            for j, line in enumerate(opt_lines):

                rendered = get_font(24).render(line, True, "white")

                SCREEN.blit(rendered, (160, 150 + i * 70 + j * 28))


        pygame.display.update()
```

Rooms

```python
def room_loop(title, questions):

    clock = pygame.time.Clock()  # To control framerate

    player = Player((640, 360))  # Create player at center of screen

    player.speed = 6  # Set movement speed

    all_sprites = pygame.sprite.Group(player)  # Add player to sprite
group


    question_blocks = pygame.sprite.Group()  # Group to hold question
blocks

    score = 0  # Player's current score


    # Determine how many questions need to be answered to win, based on
difficulty

    if title == "Easy Room":

        required_score = 3

    elif title == "Medium Room":

        required_score = 5

    elif title == "Hard Room":

        required_score = 7

    else:

        required_score = len(questions)


    lives = 5  # Number of lives player starts with

    feedback_text = ""  # Message to show after answering

    feedback_timer = 0  # Time to show feedback message

    question_active = False  # True when a question is open

    active_block = None  # Currently interacting question block
```

```python
next_block_time = 3000  # Time between spawns (ms)

block_spawn_timer = 0  # Tracks time since last spawn

active_blocks = []  # List of current on-screen blocks


# Create a button to pause game

menu_button = Button(None, (1180, 20), "MENU", get_font(25), "white",
"gray")


# Start of game loop

while True:

    dt = clock.tick(60)  # Limit frame rate and get delta time

    keys = pygame.key.get_pressed()  # Get current key states

    block_spawn_timer += dt  # Increase timer by frame time


    # Handle all events (keyboard/mouse/quit)

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            sys.exit()


        # Pause menu interaction

        if event.type == pygame.MOUSEBUTTONDOWN:

            mouse_pos = pygame.mouse.get_pos()

            if menu_button.checkForInput(mouse_pos):

                pause_menu()


        # Player presses E to answer a question

        if event.type == pygame.KEYDOWN and event.key == pygame.K_e
and not question_active:
```

```python
                for block in active_blocks:

                    if player.rect.colliderect(block.rect) and not
block.answered:

                        active_block = block

                        question_active = True

                        correct = block.handle_answer()  # Show question

                        question_active = False

                        active_block = None

                        block.answered = True

                        block.kill()  # Remove block from screen

                        active_blocks.remove(block)

                        if correct:

                            score += 1

                            feedback_text = "Correct!"

                        else:

                            lives -= 1

                            feedback_text = "Wrong! -1 Life"

                        feedback_timer = 4000  # Show feedback for 4
seconds

                        break


        # Spawning new question blocks

        if block_spawn_timer >= next_block_time and score < required_score
and lives > 0:

            question_data = random.choice(questions)

            new_block = QuestionBlock((random.randint(100, 1180),
random.randint(100, 620)), question_data)

            question_blocks.add(new_block)

            active_blocks.append(new_block)
```

```python
        block_spawn_timer = 0


        # Handle expired blocks (too slow to answer)

        for block in active_blocks[:]:

            if block.is_expired() and not block.answered:

                block.kill()

                active_blocks.remove(block)

                lives -= 1

                feedback_text = "Too Slow! -1 Life"

                feedback_timer = 4000


        # Update color of blocks based on player proximity

        for block in active_blocks:

            block.update_color(player.rect)


        # Game rendering

        SCREEN.fill("black")

        all_sprites.update(keys)  # Update player movement

        all_sprites.draw(SCREEN)  # Draw player


        # Draw question blocks

        for block in active_blocks:

            SCREEN.blit(block.image, block.rect)


        # Draw HUD text

        title_text = get_font(30).render(title, True, "white")

        SCREEN.blit(title_text, title_text.get_rect(center=(640, 30)))
```

```python
        score_text = get_font(20).render(f"Score:
{score}/{required_score}", True, "white")

        SCREEN.blit(score_text, (10, 10))

        lives_text = get_font(20).render(f"Lives: {lives}", True, "white")

        SCREEN.blit(lives_text, (10, 40))


        # Feedback text (Correct / Wrong)

        if feedback_timer > 0:

            feedback_surface = get_font(24).render(feedback_text, True,
"green" if "Correct" in feedback_text else "red")

            SCREEN.blit(feedback_surface,
feedback_surface.get_rect(center=(640, 80)))

            feedback_timer -= dt


        # Show controls at bottom-left

        control_text = get_font(18).render("WASD to move | E to interact",
True, (200, 200, 200))

        SCREEN.blit(control_text, (10, 680))


        # Update pause/menu button

        mouse_pos = pygame.mouse.get_pos()

        menu_button.changeColor(mouse_pos)

        menu_button.update(SCREEN)


        pygame.display.update()


        # Check for win/lose condition

        if score == required_score:

            win_screen()
```

```
        elif lives <= 0:

            game_over_screen()
```

Text input

```python
def text_input(prompt, hide_input=False):
    user_text = ""

    font = get_font(30)

    active = True

    show_password = not hide_input  # initial visibility state (if input
is password, starts hidden)


    BACK_BUTTON = Button(None, (100, 650), "BACK", get_font(30), "white",
"red")

    TOGGLE_BUTTON = Button(None, (1100, 300), "SHOW", get_font(20),
"white", "blue") if hide_input else None


    while active:

        SCREEN.fill((0, 0, 0))


        # Render user input safely (truncate if too long)

        display_text = user_text if show_password or not hide_input else
"*" * len(user_text)

        full_text = prompt + display_text


        max_width = SCREEN.get_width() - 380  # margin of 100 on left, 20
on right

        truncated_text = display_text


# Truncate the start if the full text is too wide
```

```python
        while font.size(prompt + truncated_text)[0] > max_width and
len(truncated_text) > 0:

            truncated_text = truncated_text[1:]


# Optional: add blinking cursor

        cursor = "|" if pygame.time.get_ticks() % 1000 < 500 else ""

        rendered_text = font.render(prompt + truncated_text + cursor,
True, (255, 255, 255))

        SCREEN.blit(rendered_text, (100, 300))




        mouse = pygame.mouse.get_pos()



        # Draw BACK button

        BACK_BUTTON.changeColor(mouse)

        BACK_BUTTON.update(SCREEN)



        # Draw toggle button if it's password input

        if hide_input and TOGGLE_BUTTON:

            TOGGLE_BUTTON = Button(None, (1100, 300), "HIDE" if
show_password else "SHOW", get_font(20), "white", "blue")

            TOGGLE_BUTTON.changeColor(mouse)

            TOGGLE_BUTTON.update(SCREEN)




        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit(); sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN:
```

```python
                if BACK_BUTTON.checkForInput(mouse):

                    return None

                if hide_input and TOGGLE_BUTTON and
TOGGLE_BUTTON.checkForInput(mouse):

                    show_password = not show_password  # Toggle show/hide


            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K_RETURN:

                    active = False

                    break

                elif event.key == pygame.K_BACKSPACE:

                    user_text = user_text[:-1]

                else:

                    user_text += event.unicode


        pygame.display.flip()


    return user_text
```