Turn-Based Game with Integrated DBMS

Submitted by

SURYA M V

In partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science

Your Institution Name

Year: 2024 - 2025

Bonafide Certificate

Certified that this project report "Turn-Based Game with Integrated DBMS" is the bonafide work of "SURYA M V" who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____.

Signature of External Examiner

Signature of Internal Examiner

INDEX

Project Synopsis

The Turn-Based Game with Integrated DBMS is designed to provide an engaging gaming experience while maintaining a database of player statistics using SQLite. The project demonstrates the integration of game mechanics with a lightweight DBMS for storing player wins, losses, and other data.

The game uses Python's Pygame library for graphics and animations, with SQLite handling the backend database operations. Players can perform attacks, heal, and view their rankings on a leaderboard.

System Requirements

System:

OS: Windows 10 or later

Processor: Intel Core i3 or equivalent

Memory: 4 GB RAM

Software:

Python 3.10 or later

SQLite 3

Pygame Library

Functions and Modules Used

Functions:

- attack_animation(): Animates character attacks.
- heal_animation(): Animates character healing.

- update_player_record(): Updates player statistics in SQLite.

Modules:

- pygame: For graphics and animations.

- sqlite3: For database operations.

Use of Technology

SQLite:

SQLite is a lightweight relational database management system that stores player data in a local database. It is used to track wins, losses, and leaderboard rankings.

Python with Pygame:

Python is used for the game's core logic, with Pygame handling animations, user interface, and game mechanics.

Code

The game's implementation includes character classes, turn-based mechanics, and database functions. The full code is available in the project directory.

```
import pygame
import random
import math
import sqlite3

# Initialize Pygame
```

```
pygame.init()

# Set up the display
WIDTH, HEIGHT = 800,600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Epic Turn-Based Game")

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)

# Fonts
font = pygame.font.Font(None, 36)
big_font = pygame.font.Font(None, 72)

def setup_database():
    conn = sqlite3.connect('game_results.db')
    c = conn.cursor()

    # Drop the existing table if it exists
    c.execute('DROP TABLE IF EXISTS players')

    # Create the table with all required columns
    c.execute('''CREATE TABLE players
```

```
                (name TEXT PRIMARY KEY,
                 wins INTEGER DEFAULT 0,
                 losses INTEGER DEFAULT 0,
                 level INTEGER DEFAULT 1,
                 exp INTEGER DEFAULT 0)''')

    conn.commit()
    return conn, c

# Initialize database connection with proper schema
conn, c = setup_database()

# Load background image
background_image = pygame.image.load('background.png')
background_image = pygame.transform.scale(background_image, (WIDTH, HEIGHT))

class Character:
    def __init__(self, name, x, y, color, image_path, is_player=False):
        self.name = name
        self.hp = 100
        self.max_hp = 100
        self.level = 1
        self.exp = 0
        self.x = x
        self.y = y
        self.original_x = x
        self.original_y = y
        self.color = color
```

```
        self.width = 50
        self.height = 100
        self.is_hit = False
        self.image = pygame.image.load(image_path)
        self.image = pygame.transform.scale(self.image, (self.width, self.height))
        self.is_player = is_player
        self.abilities = [
            {"name": "Basic Attack", "damage": 20, "heal": 0, "cooldown": 0},
            {"name": "ka me ha me haa", "damage": 35, "heal": 0, "cooldown": 2},
            {"name": "senzu bean", "damage": 0, "heal": 25, "cooldown": 3},
            {"name": "sprit bomb", "damage": 50, "heal": 0, "cooldown": 5}
        ]
        self.cooldowns = [0, 0, 0, 0]
        self.particles = []
        self.energy_particles = []
        self.aura_particles = []

    def draw(self, screen):
        screen.blit(self.image, (self.x, self.y))
        # Health bar
        bar_width = 100
        bar_height = 10
        outline_rect = pygame.Rect(self.x - 25, self.y - 60, bar_width, bar_height)
        fill_rect = pygame.Rect(self.x - 25, self.y - 60, int(self.hp / self.max_hp * bar_width), bar_height)
        pygame.draw.rect(screen, RED, outline_rect)
        pygame.draw.rect(screen, GREEN, fill_rect)

        health_text = font.render(f"{self.name} HP: {self.hp}", True, WHITE)
```

```
    level_text = font.render(f"Level {self.level}", True, YELLOW)
    screen.blit(health_text, (self.x - 20, self.y - 100))
    screen.blit(level_text, (self.x, self.y - 130))

    if self.is_hit:
        pygame.draw.rect(screen, RED, (self.x - 5, self.y - 5, self.width + 10, self.height + 10), 3)

    # Draw particles
    for particle in self.particles:
        particle.draw(screen)

    self.particles = [particle for particle in self.particles if particle.lifetime > 0]

    for particle in self.energy_particles:
        particle.draw(screen)

    # Draw aura particles
    for particle in self.aura_particles:
        particle.draw(screen)

    self.energy_particles = [p for p in self.energy_particles if p.lifetime > 0]
    self.aura_particles = [p for p in self.aura_particles if p.lifetime > 0]

def take_damage(self, damage):
    self.hp = max(0, self.hp - damage)
    return self.hp <= 0

def heal(self, amount):
```

```
    self.hp = min(self.max_hp, self.hp + amount)


def gain_exp(self, amount):
    self.exp += amount
    if self.exp >= 100:
        self.level_up()


def level_up(self):
    self.level += 1
    self.exp -= 100
    self.max_hp += 20
    self.hp = self.max_hp
    for ability in self.abilities:
        ability["damage"] = int(ability["damage"] * 1.1)
        ability["heal"] = int(ability["heal"] * 1.1)


def attack_animation(self, target):
    frames = 60
    for i in range(frames):
        progress = i / frames
        if progress < 0.3:
            # Charge up
            self.charge_up_animation()
        elif progress < 0.6:
            # Release energy blast
            self.energy_blast_animation(target)
        else:
            # Impact and aftermath
```

```
            self.impact_animation(target)


        self.draw_frame(target)
        pygame.time.delay(30)


    target.is_hit = False


def heal_animation(self, other_character):
    frames = 60
    for i in range(frames):
        progress = i / frames
        self.healing_aura_animation()
        self.draw_frame(other_character)
        pygame.time.delay(30)


def draw_frame(self, other_character, scale=1):
    screen.blit(background_image, (0, 0))
    scaled_image = pygame.transform.scale(self.image, (int(self.width * scale), int(self.height * scale)))
    screen.blit(scaled_image, (self.x - (scaled_image.get_width() - self.width) // 2,
                               self.y - (scaled_image.get_height() - self.height) // 2))
    if other_character:
        other_character.draw(screen)
    self.draw(screen)
    draw_buttons(self)
    pygame.display.flip()


def charge_up_animation(self):
    for _ in range(5):
```

```
        angle = random.uniform(0, 2 * math.pi)
        distance = random.uniform(30, 50)
        x = self.x + self.width // 2 + math.cos(angle) * distance
        y = self.y + self.height // 2 + math.sin(angle) * distance
        self.energy_particles.append(EnergyParticle(x, y, self.color))

def energy_blast_animation(self, target):
    start_x = self.x + self.width
    start_y = self.y + self.height // 2
    end_x = target.x
    end_y = target.y + target.height // 2
    for _ in range(10):
        progress = random.uniform(0, 1)
        x = start_x + (end_x - start_x) * progress
        y = start_y + (end_y - start_y) * progress
        self.energy_particles.append(EnergyBlast(x, y, self.color))

def impact_animation(self, target):
    target.is_hit = True
    for _ in range(20):
        angle = random.uniform(0, 2 * math.pi)
        speed = random.uniform(2, 5)
        x = target.x + target.width // 2
        y = target.y + target.height // 2
        self.energy_particles.append(ImpactParticle(x, y, self.color, angle, speed))

def healing_aura_animation(self):
    for _ in range(5):
```

```
            angle = random.uniform(0, 2 * math.pi)
            distance = random.uniform(0, self.width // 2)
            x = self.x + self.width // 2 + math.cos(angle) * distance
            y = self.y + self.height + math.sin(angle) * distance
            self.aura_particles.append(AuraParticle(x, y, GREEN))



class Particle:
    def __init__(self, x, y, color, move_up=False):
        self.x = x
        self.y = y
        self.color = color
        self.radius = random.randint(2, 5)
        self.lifetime = random.randint(20, 40)
        self.move_up = move_up
        if move_up:
            self.speed = random.uniform(1, 3)
            self.angle = random.uniform(-0.5, 0.5)
        else:
            self.speed = random.uniform(2, 5)
            self.angle = random.uniform(0, 2 * math.pi)

    def draw(self, screen):
        self.lifetime -= 1
        if self.move_up:
            self.y -= self.speed
            self.x += math.sin(self.angle) * 0.5
        else:
```

```
            self.x += math.cos(self.angle) * self.speed
            self.y += math.sin(self.angle) * self.speed
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.radius)

    def take_damage(self, damage):
        self.hp = max(0, self.hp - damage)
        return self.hp <= 0

    def heal(self, amount):
        self.hp = min(self.max_hp, self.hp + amount)

    def gain_exp(self, amount):
        self.exp += amount
        if self.exp >= 100:
            self.level_up()

    def level_up(self):
        self.level += 1
        self.exp -= 100
        self.max_hp += 20
        self.hp = self.max_hp
        for ability in self.abilities:
            ability["damage"] = int(ability["damage"] * 1.1)
            ability["heal"] = int(ability["heal"] * 1.1)

class EnergyParticle:
    def __init__(self, x, y, color):
        self.x = x
```

```
        self.y = y
        self.color = color
        self.size = random.randint(2, 5)
        self.lifetime = random.randint(10, 20)

    def draw(self, screen):
        self.lifetime -= 1
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.size)

class EnergyBlast:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
        self.size = random.randint(5, 10)
        self.lifetime = random.randint(20, 30)

    def draw(self, screen):
        self.lifetime -= 1
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.size)

class ImpactParticle:
    def __init__(self, x, y, color, angle, speed):
        self.x = x
        self.y = y
        self.color = color
        self.angle = angle
        self.speed = speed
```

```
        self.size = random.randint(2, 5)
        self.lifetime = random.randint(20, 30)

    def draw(self, screen):
        self.lifetime -= 1
        self.x += math.cos(self.angle) * self.speed
        self.y += math.sin(self.angle) * self.speed
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.size)

class AuraParticle:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
        self.size = random.randint(2, 5)
        self.lifetime = random.randint(20, 30)
        self.speed = random.uniform(1, 2)

    def draw(self, screen):
        self.lifetime -= 1
        self.y -= self.speed
        self.x += random.uniform(-0.5, 0.5)
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), self.size)

def create_enemy(player_level):
    enemy = Character("Enemy", 650, 400, RED, 'enemy.png')
    enemy.level = max(1, player_level - 1)  # Enemy level is player level - 1, but at least 1
```

```
    # Scale enemy stats based on level
    enemy.max_hp = 100 + (enemy.level - 1) * 20
    enemy.hp = enemy.max_hp

    for ability in enemy.abilities:
        ability["damage"] = int(ability["damage"] * (1 + 0.1 * (enemy.level - 1)))
        ability["heal"] = int(ability["heal"] * (1 + 0.1 * (enemy.level - 1)))

    return enemy

def draw_button(screen, text, x, y, width, height, color, text_color=BLACK):
    pygame.draw.rect(screen, color, (x, y, width, height))
    pygame.draw.rect(screen, WHITE, (x, y, width, height), 2)
    text_surface = font.render(text, True, text_color)
    text_rect = text_surface.get_rect(center=(x + width // 2, y + height // 2))
    screen.blit(text_surface, text_rect)
    return pygame.Rect(x, y, width, height)

def draw_buttons(player):
    buttons = []
    for i, ability in enumerate(player.abilities):
        color = GREEN if player.cooldowns[i] == 0 else RED
        button = draw_button(screen, ability["name"], 50 + i*180, 500, 170, 50, color, WHITE)
        buttons.append(button)
    return buttons

def show_message(message, color=BLACK):
    text = big_font.render(message, True, color)
```

```
    text_rect = text.get_rect(center=(WIDTH // 2, HEIGHT // 2))
    screen.blit(text, text_rect)
    pygame.display.flip()
    pygame.time.delay(1000)

def get_user_input():
    input_box = pygame.Rect(WIDTH // 2 - 100, HEIGHT // 2 - 16, 200, 32)
    color_inactive = pygame.Color('lightskyblue3')
    color_active = pygame.Color('dodgerblue2')
    color = color_inactive
    active = False
    text = ''
    done = False

    while not done:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                return None
            if event.type == pygame.MOUSEBUTTONDOWN:
                if input_box.collidepoint(event.pos):
                    active = not active
                else:
                    active = False
                color = color_active if active else color_inactive
            if event.type == pygame.KEYDOWN:
                if active:
                    if event.key == pygame.K_RETURN:
```

```
                done = True
            elif event.key == pygame.K_BACKSPACE:
                text = text[:-1]
            else:
                text += event.unicode

        screen.fill(BLACK)
        txt_surface = font.render(text, True, color)
        width = max(200, txt_surface.get_width() + 10)
        input_box.w = width
        screen.blit(txt_surface, (input_box.x + 5, input_box.y + 5))
        pygame.draw.rect(screen, color, input_box, 2)

        prompt_text = font.render("Enter your name:", True, WHITE)
        screen.blit(prompt_text, (WIDTH // 2 - 100, HEIGHT // 2 - 50))

        pygame.display.flip()

    return text

def update_player_record(name, won):
    c.execute("SELECT * FROM players WHERE name=?", (name,))
    player = c.fetchone()
    if player:
        if won:
            c.execute("UPDATE players SET wins = wins + 1, level = ? WHERE name=?", (player[3] + 1, name))
        else:
            c.execute("UPDATE players SET losses = losses + 1 WHERE name=?", (name,))
```

```
    else:
        if won:
            c.execute("INSERT INTO players VALUES (?, 1, 0, 1)", (name,))
        else:
            c.execute("INSERT INTO players VALUES (?, 0, 1, 1)", (name,))
    conn.commit()

def show_player_stats(name):
    c.execute("SELECT * FROM players WHERE name=?", (name,))
    player = c.fetchone()
    if player:
        stats_text = f"Player: {player[0]} | Wins: {player[1]} | Losses: {player[2]} | Level: {player[3]}"
    else:
        stats_text = f"New player: {name}"

    text = font.render(stats_text, True, WHITE)
    text_rect = text.get_rect(center=(WIDTH // 2, 30))
    screen.blit(text, text_rect)
    pygame.display.flip()
    pygame.time.delay(3000)

def show_main_menu(player_name):
    screen.fill(BLACK)
    title = big_font.render(f"Welcome, {player_name}!", True, WHITE)
    screen.blit(title, (WIDTH // 2 - title.get_width() // 2, 100))

    play_button = draw_button(screen, "Play", WIDTH // 2 - 100, 250, 200, 50, GREEN, WHITE)
    leaderboard_button = draw_button(screen, "Leaderboard", WIDTH // 2 - 100, 320, 200, 50, BLUE, WHITE)
```

```
    pygame.display.flip()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return "quit"
            if event.type == pygame.MOUSEBUTTONDOWN:
                if play_button.collidepoint(event.pos):
                    return "play"
                elif leaderboard_button.collidepoint(event.pos):
                    return "leaderboard"

def show_leaderboard():
    screen.fill(BLACK)
    title = big_font.render("Leaderboard", True, WHITE)
    screen.blit(title, (WIDTH // 2 - title.get_width() // 2, 50))

    # Updated SQL query to calculate win percentage
    c.execute("""SELECT name, wins, losses, level, CASE WHEN (wins + losses) > 0 THEN
ROUND(CAST(wins AS FLOAT) / (wins + losses) * 100, 2)ELSE 0
        END as win_percentage
        FROM players
        WHERE (wins + losses) > 0
        ORDER BY win_percentage DESC, level DESC, wins DESC
        LIMIT 10
    """)
    players = c.fetchall()
```

```
    y_offset = 120
    for i, player in enumerate(players, 1):
        name, wins, losses, level, win_percentage = player
        player_text = f"{i}. {name}: {win_percentage}% (W: {wins}, L: {losses}, Lvl: {level})"
        text_surface = font.render(player_text, True, WHITE)
        screen.blit(text_surface, (WIDTH // 2 - text_surface.get_width() // 2, y_offset))
        y_offset += 40

    back_button = draw_button(screen, "Back", WIDTH // 2 - 100, 500, 200, 50, RED, WHITE)

    pygame.display.flip()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return "quit"
            if event.type == pygame.MOUSEBUTTONDOWN:
                if back_button.collidepoint(event.pos):
                    return "back"

def play_game(player_name):
    player = Character(player_name, WIDTH * 0.2, HEIGHT * 0.6, BLUE, 'player.png', is_player=True)

    # Fetch player level from database
    c.execute("SELECT level FROM players WHERE name=?", (player_name,))
    result = c.fetchone()
    if result:
```

```
    player.level = result[0]


enemy = create_enemy(player.level)
enemy.x = WIDTH * 0.8 - enemy.width
enemy.y = HEIGHT * 0.6


clock = pygame.time.Clock()
player_turn = True


def end_game(winner):
    show_message(f"{winner} wins!", GREEN if winner == player_name else RED)
    print(f"{winner} wins!")  # Print the winner
    update_player_record(player_name, winner == player_name)
    pygame.time.delay(2000)
    return False  # Ends the game loop


running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            return
        if event.type == pygame.MOUSEBUTTONDOWN and player_turn:
            mouse_pos = pygame.mouse.get_pos()
            buttons = draw_buttons(player)
            for i, button in enumerate(buttons):
                if button.collidepoint(mouse_pos) and player.cooldowns[i] == 0:
                    ability = player.abilities[i]
                    if ability["damage"] > 0:
```

```
                if enemy.take_damage(ability["damage"]):
                    player.attack_animation(enemy)
                    running = end_game(player_name)
                else:
                    player.attack_animation(enemy)
                    show_message(f"Player used {ability['name']} for {ability['damage']} damage!")
            if ability["heal"] > 0:
                player.heal(ability["heal"])
                player.heal_animation(enemy)
                show_message(f"Player healed for {ability['heal']} HP!")
            player.cooldowns[i] = ability["cooldown"]
            player_turn = False
            break

if not player_turn and running:
    # Enemy turn logic
    available_abilities = [i for i, cd in enumerate(enemy.cooldowns) if cd == 0]
    if available_abilities:
        chosen_ability = random.choice(available_abilities)
        ability = enemy.abilities[chosen_ability]
        if ability["damage"] > 0:
            if player.take_damage(ability["damage"]):
                enemy.attack_animation(player)
                running = end_game("Enemy")
            else:
                enemy.attack_animation(player)
                show_message(f"Enemy used {ability['name']} for {ability['damage']} damage!")
        if ability["heal"] > 0:
```

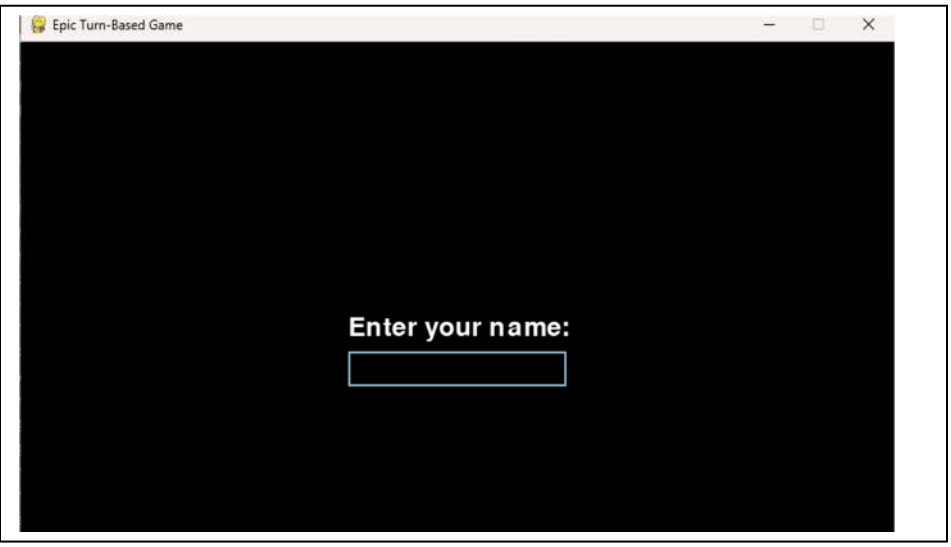
```
                enemy.heal(ability["heal"])
                enemy.heal_animation(player)
                show_message(f"Enemy healed for {ability['heal']} HP!")
            enemy.cooldowns[chosen_ability] = ability["cooldown"]
        else:
            show_message("Enemy is stunned!")
        player_turn = True

    # Reduce cooldowns
    player.cooldowns = [max(0, cd - 1) for cd in player.cooldowns]
    enemy.cooldowns = [max(0, cd - 1) for cd in enemy.cooldowns]

    if running:
        screen.blit(background_image, (0, 0))  # Draw background
        player.draw(screen)
        enemy.draw(screen)
        buttons = draw_buttons(player)

        # Draw cooldown timers
        for i, cooldown in enumerate(player.cooldowns):
            if cooldown > 0:
                cooldown_text = font.render(str(cooldown), True, WHITE)
                screen.blit(cooldown_text, (buttons[i].centerx - cooldown_text.get_width() // 2, buttons[i].bottom +
5))

        # Draw experience bar
        exp_bar_width = 200
        exp_bar_height = 20
```

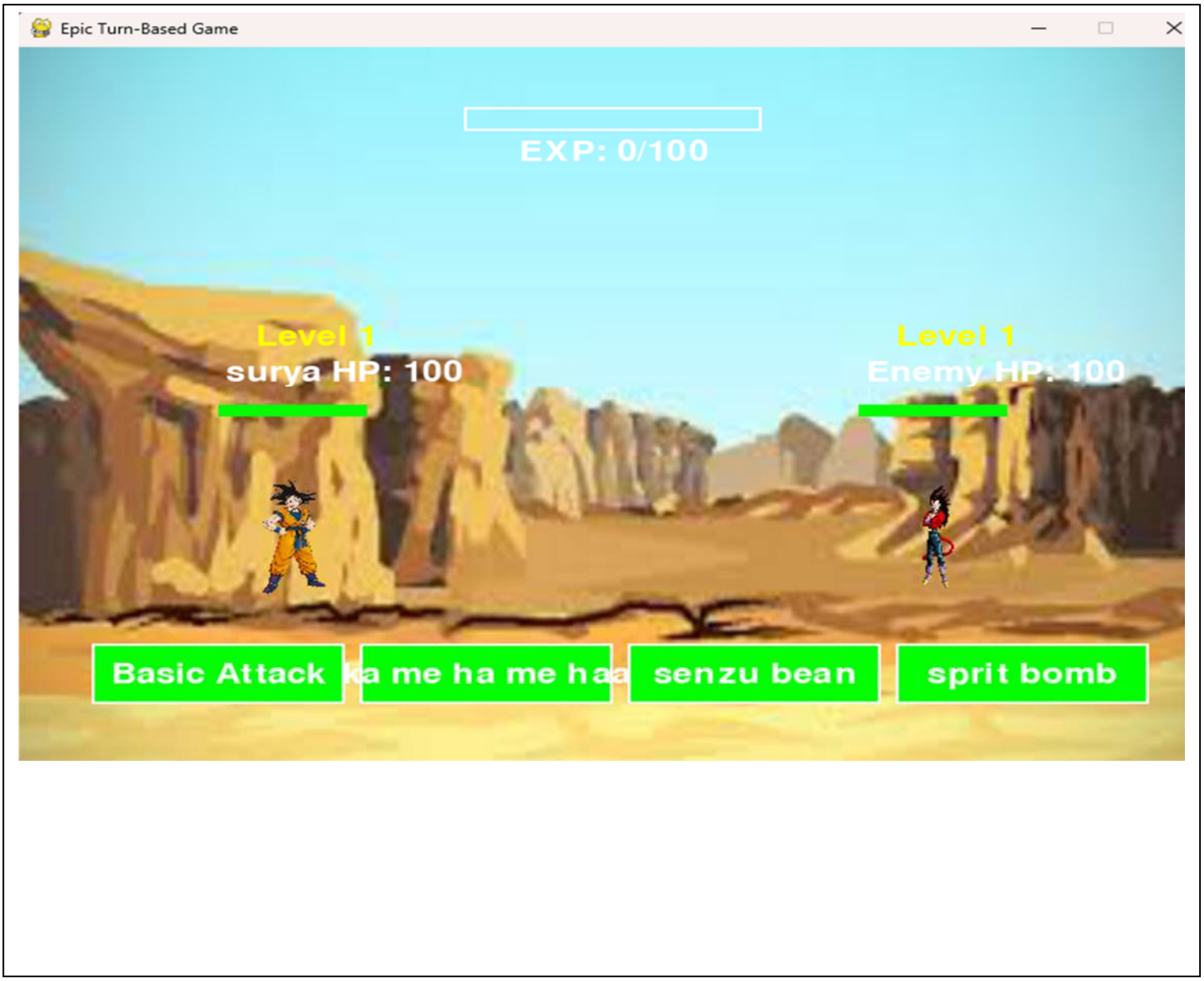
```
            exp_bar_x = WIDTH // 2 - exp_bar_width // 2
            exp_bar_y = 50
            pygame.draw.rect(screen, WHITE, (exp_bar_x, exp_bar_y, exp_bar_width, exp_bar_height), 2)
            pygame.draw.rect(screen, BLUE, (exp_bar_x, exp_bar_y, int(player.exp / 100 * exp_bar_width),
exp_bar_height))
            exp_text = font.render(f"EXP: {player.exp}/100", True, WHITE)
            screen.blit(exp_text, (exp_bar_x + exp_bar_width // 2 - exp_text.get_width() // 2, exp_bar_y +
exp_bar_height + 5))

            pygame.display.flip()
            clock.tick(60)

    # End of battle
    if player.hp > 0:
        exp_gain = random.randint(20, 50) + (enemy.level * 5)  # More exp for higher level enemies
        player.gain_exp(exp_gain)
        show_message(f"You gained {exp_gain} EXP!")
        if player.exp >= 100:
            show_message(f"Level Up! You are now level {player.level}!")

def main():
    player_name = get_user_input()
    if not player_name:
        return

    while True:
        action = show_main_menu(player_name)
```

```
        if action == "quit":
            break
        elif action == "play":
            play_game(player_name)
        elif action == "leaderboard":
            if show_leaderboard() == "quit":
                break

    pygame.quit()
    conn.close()

if __name__ == "__main__":
    main()
```

OUTPUT

The game features an interactive graphical interface where players can battle opponents, heal, and view their stats. The leaderboard dynamically updates based on player performance





Bibliography

1. Python Documentation - https://docs.python.org/3/

2. Pygame Library - https://www.pygame.org/docs/

3. SQLite Documentation – https://sqlite.org/docs.html