**Turn-Based** **Game** **with** **Integrated** **DBMS**

Submitted by

THIRUMALAI.J

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 "THIRUMALAI.J" who carried out the project work under my supervision.

Submitted for the Practical Examination held on .

Signature of Internal Examiner

Signature of External Examiner

# INDEX

1. Project Synopsis
2. System Requirements
3. Functions and Modules Used
4. Use of Technology
5. Code
6. Output
7. Bibliography

## 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/](http://www.pygame.org/docs/)
3. SQLite Documentation – https://sqlite.org/docs.html