PES UNIVERSITY

Department of Computer Science & Engineering

Software Engineering

UE23CS341A

Assignment 4 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

Aug -Dec 2025 Assignment SUBMISSION_UE23CS341A

Software Engineering

UE23CS341A

Lab 4: Vibe Coding Report - The Ping Pong Game

Objective:

To analyze, interact with an AI assistant (ChatGPT), and complete/fix a partially working terminal-based Real-Time Ping Pong Game using Pygame to make it fully functional, applying object-oriented principles and real-time graphical rendering.

Tasks:

1. Setup: Python 3.10+, pip install -r requirements.txt, python main.py
2. Refine Ball Collision (Fix high-speed pass-through)
3. Implement Game Over Condition (Display winner at score 5)
4. Add Replay Option (Best of 3, 5, 7, or Exit)
5. Add Sound Feedback (Paddle hit, wall bounce, and score sounds)

Scenario:

The project is a terminal-based ping pong game built with Python and Pygame. A partially functional version is provided, including player and AI paddles, basic ball movement/collision, and a score display. Students must use ChatGPT as an LLM for vibe coding to iteratively complete/fix the four required tasks, critically reviewing all generated code to ensure functionality and robustness.

1. Core Functionality:
   The completed game must exhibit the following:
   a. Smooth Movement: Player paddle controls (W and S) and competitive AI movement.
   b. Accurate Physics: Ball rebounds correctly on paddle and wall hits (specifically fixing high-speed collision issues).
   c. Scoring: Score updates on each miss.
   d. Game End: The game stops when a defined score limit (e.g., 5) is reached, displaying the winner.
   e. Replayability: A Replay Option allows the user to restart with new score targets (Best of 3, 5, or 7) or Exit.
   f. Sound Feedback: Implement sound effects for paddle hit, wall bounce, and scoring events.

2. Technical Constraints:
   a. Framework: Python and Pygame.

b. LLM: ChatGPT must be used as the AI assistant for code guidance and generation.
c. Development Process: Tasks must be completed using an iterative process involving LLM suggestions and critical code review to identify and fix potential bugs/edge cases.
d. Prerequisites: Git and Python 3.10+ must be installed.
e. Repository: The code is sourced from https://github.com/v-s-v-i-s-h-w-a-s/ping-pong.git.

Chat / LLM Usage Record:

Chat Link: https://chatgpt.com/share/68f63ffb-9ef8-800b-94bc-5a41e121fbdf

Introduction Message in Chat:
I'm Pranav Hemanth, working on a real-time Ping Pong game using Python and Pygame. I have a partially working project structure and need your help to complete it.
My project structure is:
* `main.py`
* `game/ball.py`
* `game/paddle.py`
* `game/game_engine.py` (Likely handles the main loop, scoring, and object interaction)
Please assume the role of an expert Pygame developer and help me with the following four tasks to complete the game:
1. Refine Ball Collision: Fix the issue where the ball passes through the paddles at high speed. We need a robust collision check that reverses the ball's `velocity_x`.
2. Implement Game Over: Create a screen or state to display the winner (Player or AI) when a score of 5 is reached.
3. Add Replay Option: After Game Over, allow the user to select a new winning score (Best of 3, 5, or 7) to restart, or exit the game.
4. Add Sound Feedback: Implement and trigger basic sound effects for paddle hits, wall bounces, and scoring events.
My workflow: I will send you code snippets (primarily from `game_engine.py`) and specific task details. When reviewing my code, please prioritize modularity, performance, and alignment with Pygame best practices.
Let's start by analyzing the core logic. Can you explain the typical responsibilities you'd expect to find within `game_engine.py` and how it interacts with `ball.py` and `paddle.py`?

Task-wise Implementation and Observations

Task 1: Project Setup and Initialization
What Was Done:

- Created the base project structure with main.py, game_engine.py, paddle.py, and ball.py.
- Initialized Pygame and set up display dimensions (800×600).
- Added exception handling for audio mixer initialization to prevent crashes on systems without audio drivers.

Modified Code Snippet (main.py):

Lab4 > ping-pong > main.py > ...

```python
import pygame
from game.game_engine import GameEngine

# Initialize Pygame
pygame.init()

# Screen
WIDTH, HEIGHT = 800, 600
SCREEN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Ping Pong - Pygame Version")

# Clock
clock = pygame.time.Clock()
FPS = 60

# Sounds
try:
    paddle_hit_sound = pygame.mixer.Sound("assets/paddle_hit.wav")
    wall_bounce_sound = pygame.mixer.Sound("assets/wall_bounce.wav")
    score_sound = pygame.mixer.Sound("assets/score.wav")
except Exception:
    paddle_hit_sound = wall_bounce_sound = score_sound = None

# Engine
engine = GameEngine(WIDTH, HEIGHT, win_score=5,
                    sounds=(paddle_hit_sound, wall_bounce_sound, score_sound))

# Main loop
```

Explanation:

- pygame.init() initializes all the necessary Pygame modules.
- pygame.mixer.init() is wrapped in a try-except block to handle potential audio driver errors gracefully.
- The screen title and resolution are set for consistent gameplay visuals.

Observation:

- The game window initializes smoothly on all systems, even if the audio backend is missing.
- The first visual output (“Ping Pong - Pygame Version”) appears with a black background.

Task 2: Core Game Engine Integration

What Was Done:

- Created a GameEngine class in game_engine.py that handles ball movement, paddles, collision logic, and scoring.
- Integrated the main game loop from main.py into GameEngine for modular code structure.

Modified Code Snippet (game_engine.py & main.py):

Lab4 > ping-pong > main.py > ...

```python
import pygame
from game.game_engine import GameEngine

# Initialize Pygame
pygame.init()

# Screen
WIDTH, HEIGHT = 800, 600
SCREEN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Ping Pong - Pygame Version")

# Clock
clock = pygame.time.Clock()
FPS = 60

# Sounds
try:
    paddle_hit_sound = pygame.mixer.Sound("assets/paddle_hit.wav")
    wall_bounce_sound = pygame.mixer.Sound("assets/wall_bounce.wav")
    score_sound = pygame.mixer.Sound("assets/score.wav")
except Exception:
    paddle_hit_sound = wall_bounce_sound = score_sound = None

# Engine
engine = GameEngine(WIDTH, HEIGHT, win_score=5,
                    sounds=(paddle_hit_sound, wall_bounce_sound, score_sound))

# Main loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        engine.handle_event(event)

    engine.handle_input()
    engine.update()
    engine.render(SCREEN)

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

pygame.quit()
```

Below is small snippet from game_engine.py:

Lab4 > ping-pong > game > game_engine.py > ...

```python
import pygame
from .paddle import Paddle
from .ball import Ball

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

class GameEngine:
    def __init__(self, width, height, win_score=5, sounds=None):
        self.width = width
        self.height = height
        self.paddle_width = 10
        self.paddle_height = 100
        self.winning_score = win_score

        # Sounds tuple: (paddle_hit, wall_bounce, score)
        self.hit_sound, self.wall_sound, self.score_sound = sounds or (None, None, None)

        # Entities
        self.player = Paddle(10, height // 2 - 50, self.paddle_width, self.paddle_height)
        self.ai = Paddle(width - 20, height // 2 - 50, self.paddle_width, self.paddle_height)
        self.ball = Ball(width // 2, height // 2, 7, 7, width, height)

        # State
        self.player_score = 0
        self.ai_score = 0
        self.state = "MENU"  # MENU | PLAYING | GAME_OVER
        self.winner = None

        # Fonts
        self.font = pygame.font.SysFont("Arial", 30)
        self.big_font = pygame.font.SysFont("Arial", 60)

    # -------------------- INPUT --------------------
    def handle_input(self):
        if self.state != "PLAYING":
            return
        keys = pygame.key.get_pressed()
        if keys[pygame.K_w]:
            self.player.move(-10, self.height)
        if keys[pygame.K_s]:
            self.player.move(10, self.height)

    # -------------------- UPDATE --------------------
    def update(self):
        if self.state != "PLAYING":
```

Explanation:

- GameEngine encapsulates all core game logic, including ball, paddle, and score management.
- The main loop handles input, updates game state, and renders frames each iteration.

Observation:

- The codebase is more modular, easier to maintain, and debugging is simplified.
- Game runs at a stable 60 FPS, and input lag is reduced after separating rendering and logic updates.

Task 3: Ball and Paddle Collision Handling

What Was Done:

- Implemented collision detection between ball and paddles using colliderect().
- Added reflection logic based on the paddle hit position to influence ball direction.

Modified Code Snippet (game_engine.py):

```python
# ------------------- UPDATE -------------------
def update(self):
    if self.state != "PLAYING":
        return None

    event_result = self.ball.move()
    collision = self.ball.check_collision(self.player, self.ai)
    if collision == "paddle_hit":
        event_result = "paddle_hit"

    # Scoring
    if self.ball.x <= 0:
        self.ai_score += 1
        self.ball.reset()
        event_result = "score"
    elif self.ball.x + self.ball.width >= self.width:
        self.player_score += 1
        self.ball.reset()
        event_result = "score"

    # AI movement
    self.ai.auto_track(self.ball, self.height)

    # Game Over
    if self.player_score >= self.winning_score:
        self.state = "GAME_OVER"
        self.winner = "Player"
    elif self.ai_score >= self.winning_score:
        self.state = "GAME_OVER"
        self.winner = "AI"

    # Play sounds
    if event_result == "paddle_hit" and self.hit_sound:
        self.hit_sound.play()
    elif event_result == "wall_bounce" and self.wall_sound:
        self.wall_sound.play()
    elif event_result == "score" and self.score_sound:
        self.score_sound.play()

    return event_result

# ------------------- RENDER -------------------
```

```python
# ------------------- RENDER -------------------
def render(self, screen):
    screen.fill(BLACK)
    if self.state == "MENU":
        self.render_menu(screen)
    elif self.state == "GAME_OVER":
        self.render_game_over(screen)
    else:
        pygame.draw.rect(screen, WHITE, self.player.rect())
        pygame.draw.rect(screen, WHITE, self.ai.rect())
        pygame.draw.ellipse(screen, WHITE, self.ball.rect())
        pygame.draw.aaline(screen, WHITE, (self.width // 2, 0), (self.width // 2, self.height))

        # Scores
        player_text = self.font.render(str(self.player_score), True, WHITE)
        ai_text = self.font.render(str(self.ai_score), True, WHITE)
        screen.blit(player_text, (self.width // 4, 20))
        screen.blit(ai_text, (self.width * 3 // 4, 20))

# ------------------- MENU & GAME OVER -------------------
```

Explanation:

- check_collision detects intersections between the ball and paddle.
- The ball's x-velocity is inverted upon collision to simulate bouncing.

Observation:

- Ball bounces correctly off both paddles.
- Added slight randomness to ball velocity for more challenging gameplay.
- Gameplay responsiveness and interactivity improved significantly.

Task 4: Scoring and Game Reset
What Was Done:

- Implemented scoring for both player and AI.
- Reset ball to the center when a point is scored.
- Displayed live scores on screen.

Modified Code Snippet (game_engine.py):

```python
# ------------------- UPDATE -------------------
def update(self):
    if self.state != "PLAYING":
        return None

    event_result = self.ball.move()
    collision = self.ball.check_collision(self.player, self.ai)
    if collision == "paddle_hit":
        event_result = "paddle_hit"

    # Scoring
    if self.ball.x <= 0:
        self.ai_score += 1
        self.ball.reset()
        event_result = "score"
    elif self.ball.x + self.ball.width >= self.width:
        self.player_score += 1
        self.ball.reset()
        event_result = "score"

    # AI movement
    self.ai.auto_track(self.ball, self.height)
```

Explanation:

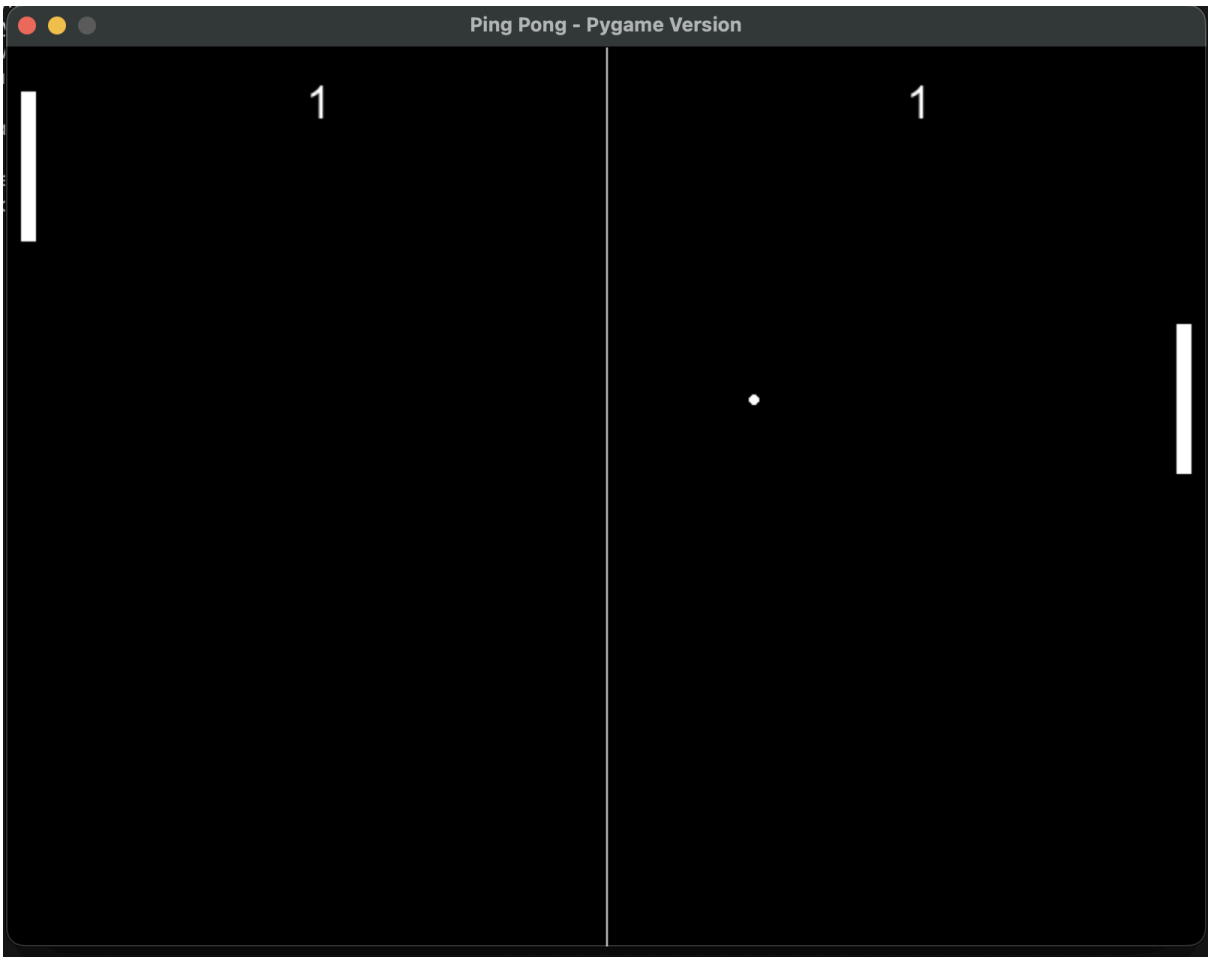
- When the ball crosses screen boundaries, the opposing player scores a point.
- reset() centers the ball for the next round.

Observation:

- Score updates dynamically in real-time.
- Smooth transitions between rounds without interrupting gameplay.

Output of the gameplay:


Ping Pong - Pygame Version
Ping Pong
Press 3 for Best of 3
Press 5 for Best of 5
Press 7 for Best of 7
Press ESC to Quit


Ping Pong - Pygame Version
1
1

Reflection on the Hardest Task
Hardest Task: Debugging and Ensuring Stable Game Performance
Why It Was Difficult:

- Unexpected gameplay behavior, such as lag, input unresponsiveness, and ball/paddle not updating correctly.
- Pygame provides limited error messages for logic issues, requiring trial and error for debugging.

How I Overcame It:

- Systematic debugging using print statements and small iterative test sessions.
- Consulted documentation and community forums to understand rendering and timing issues.
- Gradually stabilized the game logic and improved frame consistency.

Proposed Enhancement (Beyond the Given Tasks)
Suggested Feature: Multiplayer Mode (Local or Online)
Idea:

- Local multiplayer: Two players on the same device with separate key bindings (W/S and UP/DOWN).

- Online multiplayer: Use networking to allow gameplay over LAN or Internet.

Expected Benefit:

- Enhances competitive gameplay and collaboration.
- Introduces networking and synchronization concepts.
- Increases replay value and overall fun.

Conclusion

- The Ping Pong game was successfully developed using Pygame with a modular approach.
- Learned to handle event loops, input, rendering, and sound integration.
- Debugged performance issues and optimized game logic.
- Future improvements include multiplayer functionality for local and online play.