KATHMANDU UNIVERSITY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT REPORT ON IMPOSTER GAME

[Code No: COMP 342]

For partial fulfillment of Year III / Semester I in Computer Science

Submitted By:

Lujaw Dhunju (Roll No. 19)
Jatin Madhikamri (Roll No. 32)

Submitted To:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

February 9, 2026

1 Abstract

The "Imposter Game" is a localized social deduction application developed to play among your friends in realtime rather an online game which is meant to play on the go i.e an "Pass-and-Play" game which also explore the intersection of 2D computer graphics and real-time state management. Utilizing the Python-based Pygame framework, the project implements a robust game loop capable of handling variable player counts and dynamic role assignment. A key feature of the implementation is the use of transformations—specifically scaling—to simulate 3D card-flip animations within a strictly 2D coordinate system. By incorporating culturally relevant word databases (e.g., local Nepali landmarks and cuisine) and other different variety of words, the project demonstrates how localized content can be integrated into a scalable game engine while maintaining rigorous graphical performance standards.

2 Introduction

The development of interactive digital systems requires a seamless integration of user input, logical state transitions, and real-time graphical rendering. This project, titled the "Imposter Game," is a social deduction application designed to explore these core components within a 2D environment. The game serves as a technical case study in building a "Pass-and-Play" interface that necessitates secure information hiding and fluid visual feedback.

2.1 Graphical Objectives and Framework

The project is built upon the **Pygame** framework, which acts as a wrapper for the Simple DirectMedia Layer (SDL). This provides the application with low-level access to the graphics hardware via OpenGL and Direct3D. A primary objective of this implementation was to move beyond static, binary User Interface (UI) elements by implementing **Animations**.

Instead of a simple image swap, the system calculates "in-between" frames to simulate a 3D card flip. This process involves:

- **Frame-Rate Independence:** Ensuring that the animation speed remains consistent regardless of the CPU's processing power.
- **Double Buffering:** Utilizing a primary and secondary buffer to render scenes off-screen. This technique is essential in computer graphics to prevent "flickering" or "tearing," as the user only sees the completed frame once the *display.flip()* command is executed.
- **Rasterization:** Converting vector-based font data into pixel-grid surfaces in real-time to display player names and secret words.

3 Technical Architecture

The software architecture is divided into two primary systems: the Game Loop and the State Management system.

3.1 The Game Loop

The heartbeat of the application is a high-frequency loop that ensures a consistent 60 Frames Per Second (FPS). The loop follows a strictly sequential execution:

1. **Event Polling:** Capturing hardware signals (Keyboard, Mouse).

```
for event in pygame.event.get():
```

Figure 1: Event Queue

Explaination: This is the essentially the starting point of our program and is the event handler i.e everytime a user does something like presssing the W key,a KEY-DOWN action event is added i.e everytime the user does something an "event" joins the queue. When you call get() it essentailly grabs every event currently waiting in that queue and puts them into a list. And when you use the for loop you are you look at the each one of the event individually and decide what to do with it.

```
if event.type == pygame.QUIT:
    pygame.quit(); sys.exit()
```

Figure 2: Command to quit the program

In the above figure if the event.type == pygame.QUIT i.e the user crosses/quits the program then we close the program.

2. **State Update:** Updating the state/screen according to the flow of the program

```
if self.state == "MENU":
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_RIGHT: self.current_cat_idx = (self.current_cat_idx + 1) % len(self.categories)
        if event.key == pygame.K_UP and self.player_count < 10: self.player_count += 1
        if event.key == pygame.K_DOWN and self.player_count > 3: self.player_count -= 1
        if event.key == pygame.K_SPACE: self.state = "INPUT_NAMES"
```

Figure 3: State input actions for the current and future states

Explaination: Here the state variable contains the value of the current state i.e the current screen that the players are in.The initial state is the MENU which is the starting screen/landing screen and this block of code is inside the code of Figure 1 and we know that we can use the different events like if the user entered different keys like Right,Up which can be interpreted as pygame.K_RIGHT and pygame.K_UP and according to these values we perform different actions in the background which the players can view in realtime and the last line in the above code is "if event.key == pygame.K_SPACE: self.state = "INPUT_NAMES" which translates to if the players pressed the Space bar then move to the next state which is the "INPUT_NAMES" where the players enter thier respective names.

3. **Rendering:** Drawing pixels to the back buffer. In Computer Graphics we use "Painter's Algorithm." that works on the basic that we first make the initial canvas all black(in this case) then we draw an object which is rectangle in this case and lastly we draw the text.Each subsequent calls overwrites the pixel underneath.

```
screen.fill(BLACK)
```

Figure 4: Code to make the screen black

```
pygame.draw.rect(screen,q_color,quit_rect,2,border_radius=10)
draw_text("QUIT",font_ui,q_color,SCREEN_WIDTH//2,455)
```

Figure 5: Code to first render the rectangle and then text

Explanation: The above program executes in the order of Figure 4 i.e the screen is first intially painted black and then the code from Figure 5 which is inside a drawing function is ran which translates to first draw the rectangle and then the text.

4. **Buffering:** Swapping the buffer to the screen to prevent flickering [1]. Swanpping the buffer is an important thing to do as if we don't do it then the new frame will be drawn on top the previous bufffer which creates a very bas visual effect.so, to prevent t**tearing** (where you see half of a new frame and half of an old one), the GPU maintains two buffers.

```
pygame.display.flip()
```

Figure 6: Code to display smooth transition

Explaination: The code in Figure 6 tells the graphics card to switch the pointer from the "Front Buffer" (displayed) to the "Back Buffer" (just rendered), which ensures the transition is perfectly smooth and flicker-free.

3.2 Finite State Machine (FSM)

To handle transitions between the Menu, Player Input, Role Reveal, and Voting screens, the project utilizes a Finite State Machine. Each screen is treated as a "Node," and transitions are "Edges" triggered by specific events. This ensures that the global game state is synchronized across different views.

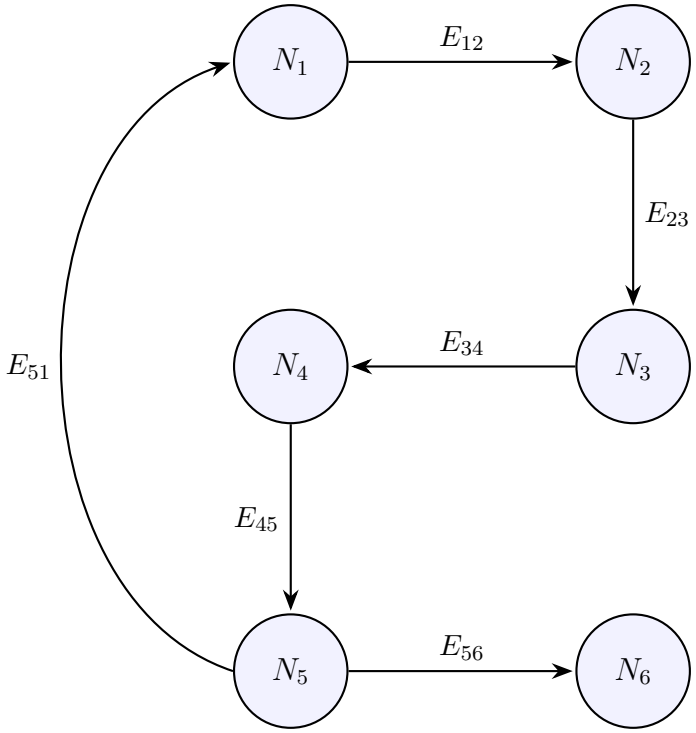

Figure 7: Formal Finite State Machine (FSM) Graph Representation.

Node	**Screen Name**	**Technical Function**
N_1	MENU	The main landing page which is the very starting page which prompts the player to choose the category and no of players
N_2	INPUT_NAMES	Takes the name of the players `player_names` list.
N_3	ROLE_ASSIGNMENT	Assigns the roles "Civilian" and "Imposter" on a random basics to the players
N_4	VOTING_SCREEN	Voting screen where the player votes off the players which they think is the imposter
N_5	GAME_OVER	Performs the final result calculation and renders the Win/Loss UI depending upon the results
N_6	QUIT	Triggers the termination of the program and releases memory

Table 1: Definition and Functional Purpose of FSM Nodes.

[HTML]EFEFEF **Edge**	**Trigger / Event**	**Logic Execution**
E_{12}	Mouse Click (Start)	Updates `self.state` to initiate player entry.
E_{23}	List Completion	Occurs when `len(player_names)` meets the player count threshold.
E_{34}	Iterator Overflow	Triggered when the final player completes the role viewing sequence.
E_{45}	Counter Threshold	Triggered when the `voted_count` matches the number of active players.
E_{51}	Mouse Click (Reset)	Calls `self.__init__()` to perform a complete state reset.
E_{56}	Mouse Click (Exit)	Executes `sys.exit()` to terminate the main execution thread.

Table 2: Definition of FSM State Transitions (Edges).

4 Computer Graphics Theory

The project emphasizes mathematical transformations to enhance visual immersion.

4.1 2D Transformations

The most significant graphical feature is the "Card Flip" animation. Since Pygame is a 2D engine, a 3D rotation is simulated using a scaling transformation matrix S:

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
def update_animation(self):
    """The math for the 2D scaling (pseudo-rotation)"""
    if self.is_flipping:
        self.anim_width -= self.flip_speed
        if self.anim_width <= 0:
            self.anim_width = 0
            self.showing_back = not self.showing_back
            self.flip_speed *= -1
        if self.anim_width > 200:
            self.anim_width = 200
            self.is_flipping = False
            self.flip_speed = abs(self.flip_speed)
```

Figure 8: 2D Transformation Code

4.1.1 The Scaling Mechanism

- State Transtition: The variable self.anim_width acts as the s_x factor. When a user clicks to reveal a card, the code decreases self.anim_width by self.flip_speed until it reaches 0.
- The Flip: When self.anim_width <= 0, the code performs a logical swap. It changes the boolean self.show_word from False to True. This is the exact moment the card is "edge-on" to the viewer and invisible.
- Inverse Scaling: The code then increases self.anim_width back to the original card width. This creates the visual effect of the card's "back" expanding toward the viewer and the player can view the the secret word or the word "IMPOSTER".

Explaination: The revealing of the word is timed in this way to maintain Graphical Continuity. If the word changed while the card was at full width , it would look like a glitch. By swapping the text when the width is 0, you use the "edge" of the card to hide the transition, making the flip look like a physical object rotating.

The animation cycle is divided into two distinct phases relative to the card's width i.e anim_width= (s_x):

1. **Contraction Phase:** The width decreases from *Full Width* ($s_x = 1$) to *Zero Width* ($s_x = 0$). During this phase, the obverse side ("Tap to Reveal") is rendered.
2. **The Midpoint Swap:** At $s_x \leq 0$, the state variable `show_word` is toggled. This is the "Critical Point" where the secret identity (e.g., "IMPOSTER") is loaded into the text surface.
3. **Expansion Phase:** The width increases from 0 back to *Full Width*. Because the swap occurred at the zero-point, the player perceives a natural rotation revealing the hidden identity.

4.2 Collision Detection

Interaction with buttons and player fields utilizes Axis-Aligned Bounding Box (AABB) logic. A click at point $P(x, y)$ is registered if:

$$x_{min} \leq P_x \leq x_{max} \quad \text{and} \quad y_{min} \leq P_y \leq y_{max} \tag{1}$$

```
if play_rect.collidepoint(event.pos):
```

Figure 9: Code for collision detection

Explaination: Here in the above Figure 9 the we check for collison between the rectangle and the mouse position which can be considered as a point, which is achieved my pygame default function collidepoint.

5 Conclusion and Future Works

5.1 Conclusion

This project successfully integrates fundamental Computer Graphics principles into a playable application. It demonstrates that complex visual effects, like 3D rotation, can be effectively simulated using 2D linear algebra.We have successfully created a game which can be played without any errors while having a funtime with your friends.

5.2 Future Works

- Future iterations could include Network Sockets for remote multiplayer.
- OpenGL shaders for advanced lighting effects.
- Addition of more variety of topics and world.
- Deployment of the game.

6 Google Drive Link

`https://drive.google.com/drive/folders/1no2Z5KcAmSvhck_ZByoEMmD9fEU0ZE2z?usp=sharing`

References

[1] S. McGugan, *Beginning Game Development with Python and Pygame*, Apress, 2007.

[2] J. Almog, *State Machines and Game Loops*, 2018.

[3] D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 4th ed., 2010.

A Appendix: Game Screenshots

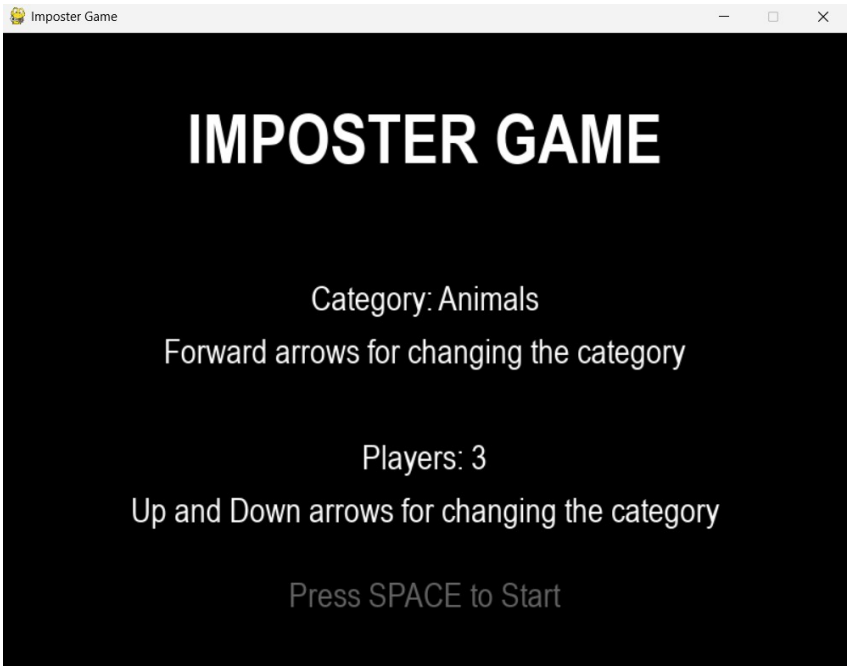

Figure 10: Main Menu

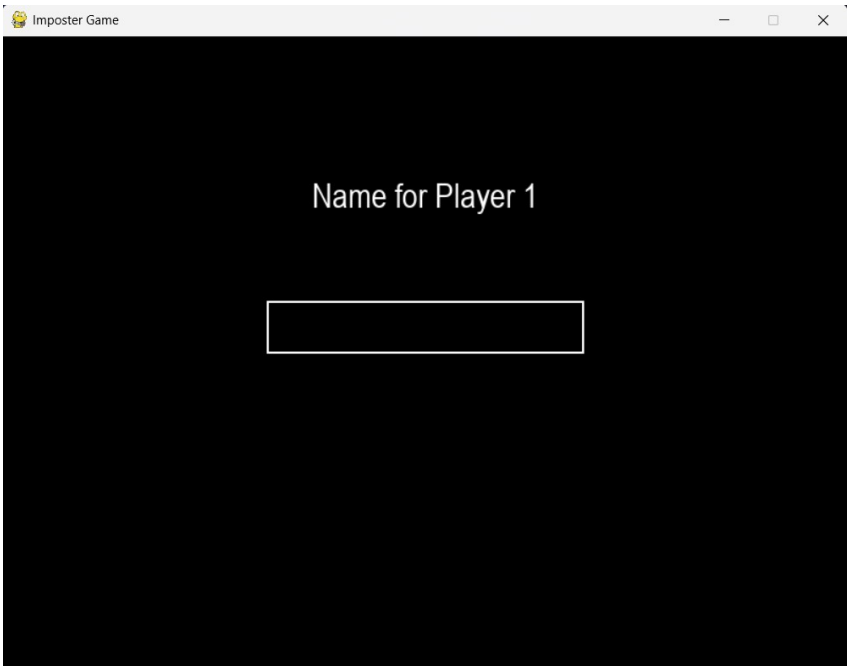

Figure 11: Input Menu

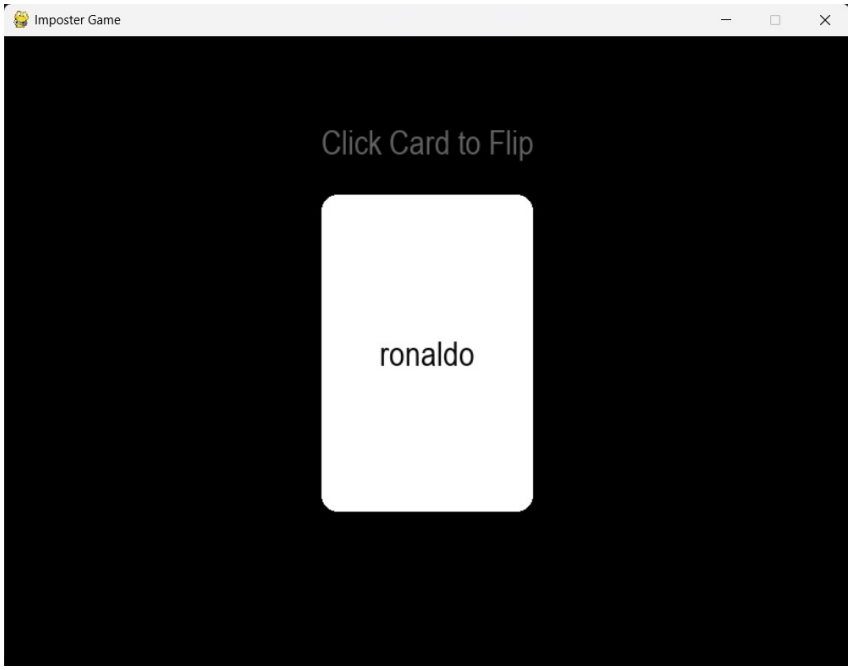

Figure 12: Player's Name

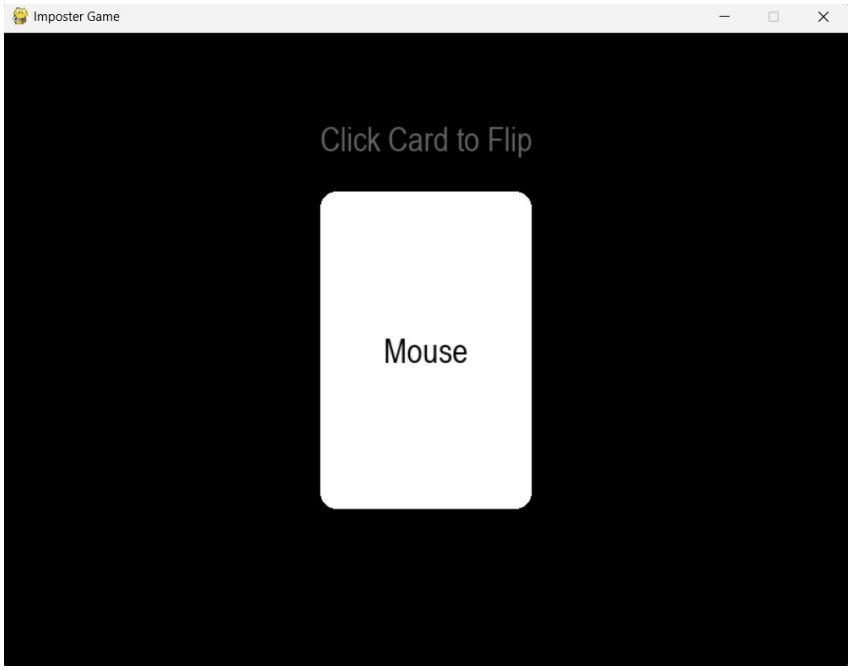

Figure 13: Player's Role

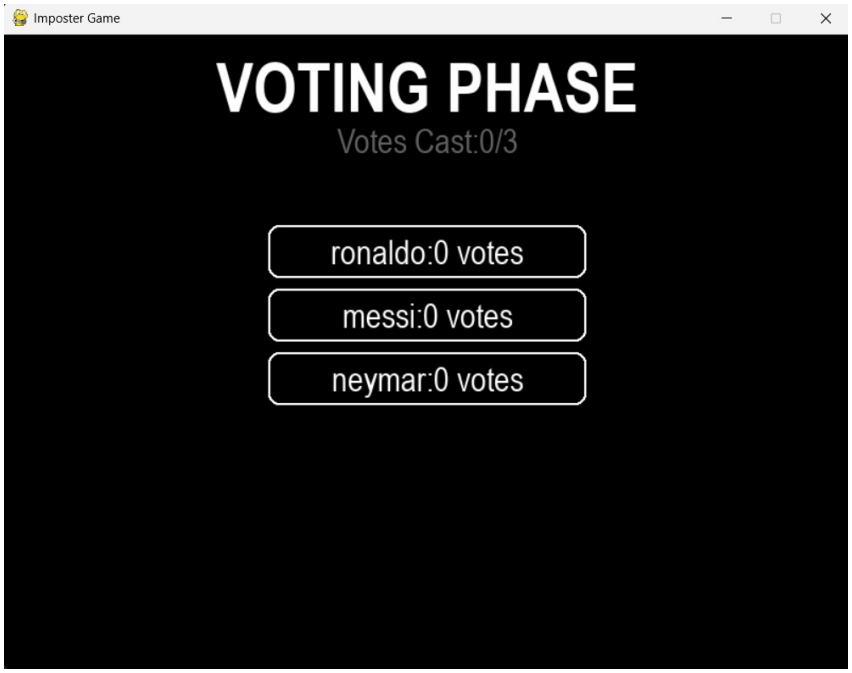

Figure 14: Voting Screen

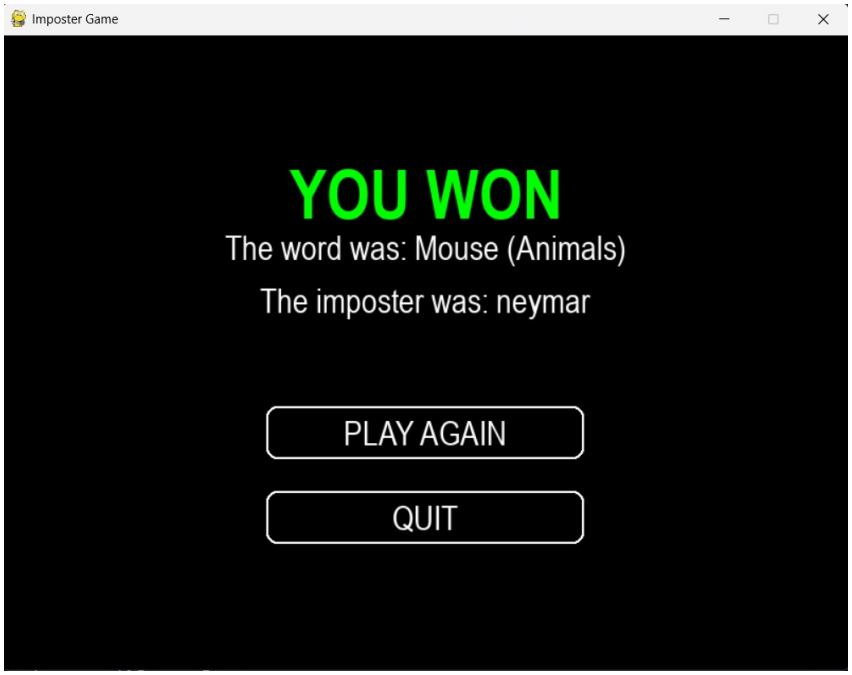

Figure 15: Game Over Screen