PYTHON PROGRAMMING AND APPLICATIONS

CORSE CODE: EE2021E

Submitted by

J.SAMANVITHA

UNDER GUIDANCE OF: DR. AMRUTHA RAJU

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT

Project Report:

The "6561" Game -(Powers of 3)

1. Introduction:

The **"6561" Game - (Powers of 3)** is a custom implementation of the widely popular *2048* tile merging puzzle, creatively adapted to operate using **powers of 3** instead of powers of 2. This project, developed using the **Python** language and the **Pygame** library, serves as a comprehensive application of object oriented programming, efficient matrix manipulation, and dynamic graphical rendering techniques. The central objective is to strategically merge tiles on a 4 X 4 grid to achieve the target value of **6561 (**3^8**)**. This report provides a detailed breakdown of the application's core logic, the modules utilized, and the algorithms that govern gameplay, visual experience, and game state management.

2. Project Summary and Objective:

This project is a custom implementation of the popular 2048 tile-merging game, uniquely adapted to use **powers of 3** instead of powers of 2. Developed using the **Pygame** library in Python, the primary objective is to strategically merge tiles on a 4 X 4grid to achieve the target value of **6561** (3^8). The project emphasizes a clean user interface, robust game logic, and an engaging victory sequence.

Feature	**Detail**
Technology Stack	Python, Pygame
Grid Size	4 x 4(GRID_SIZE) = 4)
Target Tile	6561 (3^8)
Core Mechanic	Identical tiles **triple** their value upon merging (A + A ➡️ A X 3)

3. Application Logic The project titled 3×2048 :

Power of 3 Puzzle Game is an interactive number puzzle built using Python’s Pygame library. It is an adaptation of the classic 2048 game but operates using powers of 3 instead of powers of 2. The main objective of the game is to combine tiles with similar values to reach the target tile value of 6561 (3). The application initializes a 4×4 grid where two tiles (3 or 9) are randomly placed at the beginning. Players can move tiles in four directions (Up, Down, Left, Right). When two tiles of the same value collide, they merge into one tile whose value is the product of 3×(current value). The player wins when a tile with value 6561 is created. If no further moves are possible, the game ends with a loss message.

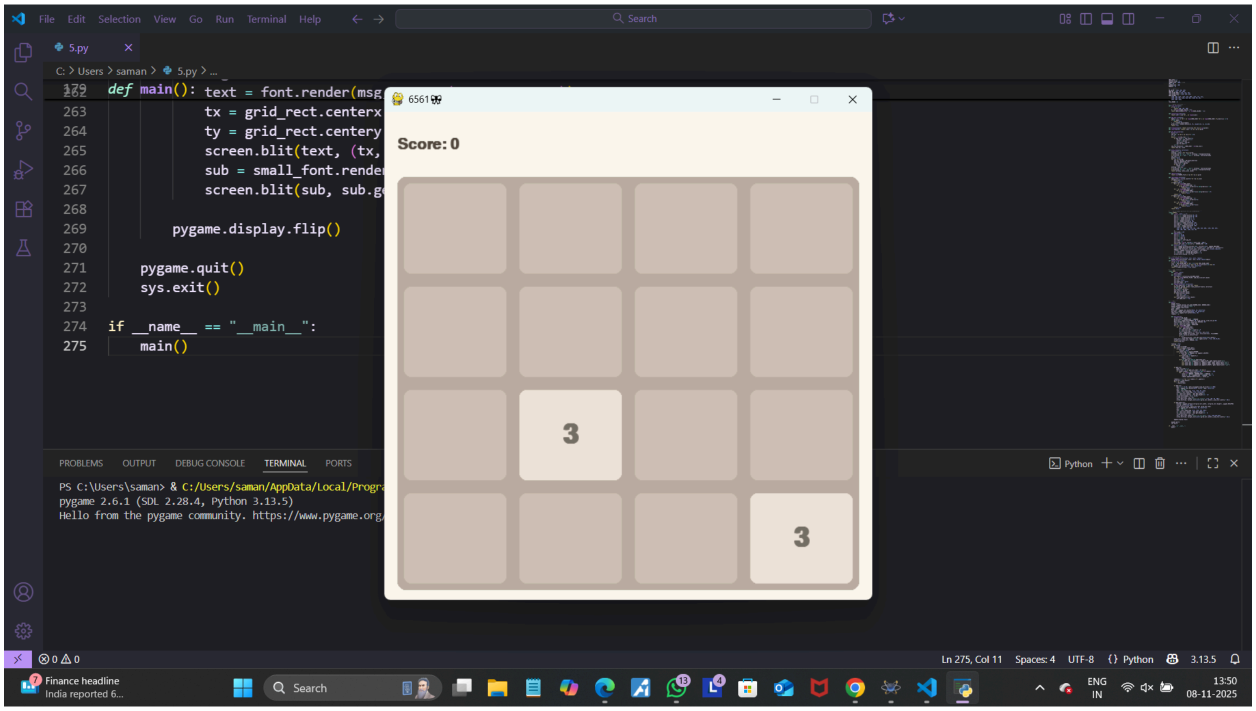
4. Core Game Mechanics and Logic:

The game's intelligence is managed by efficient functions for grid manipulation and state checks.

A. Tile Merging and Movement:

The game uses a modular approach to handle movement in four directions (Up, Down, Left, Right) by leveraging linear algebra principles:

1. **Normalization:** The grid is temporarily transformed using transpose() and reverse() functions so that the merging logic can always be applied using the simple **'slide-left'** logic (get_base_move).
2. **Merging Rule:** Inside get_base_move(row), two adjacent, identical tiles are combined, and their value is **tripled** (A ➡️ A X 3), ensuring all tiles remain powers of three.
3. **Denormalization:** The transformation is reversed to restore the grid's original orientation.


```
def main():    text = font.render(msg
                    tx = grid_rect.centerx
                    ty = grid_rect.centery
                    screen.blit(text, (tx,
                    sub = small_font.rende
                    screen.blit(sub, sub.g

            pygame.display.flip()

        pygame.quit()
        sys.exit()

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

```
PS C:\Users\saman> & C:/Users/saman/AppData/Local/Progra
pygame 2.6.1 (SDL 2.28.4, Python 3.13.5)
Hello from the pygame community. https://www.pygame.org/
```

B. Scoring and Tile Spawning:

- **Scoring:** The player's score is incremented by the value of the **newly created, merged tile**.
- **Weighted Spawning:** To maintain challenge, new tiles are spawned with weighted probability: **Tile 3** has a 90%chance, and **Tile 9** has a 10% chance.

3. Modules Used:

The project relies on a minimal set of external libraries to ensure efficiency and portability:

- **Pygame (Core Library):**
    - **Purpose:** The primary framework for building the game window, handling the event loop (user input), graphics rendering (surfaces, fonts, colors), and timing. All game state visualization is managed via Pygame.
- **Random (Standard Library):**
    - **Purpose:** Essential for all game stochasticity. It is used to **randomly place new tiles** on empty cells after a valid move and implements the **weighted spawning probability** (90% chance for '3', 10% chance for '9') to control difficulty.

- **Sys (Standard Library):**
    - **Purpose:** Used for system-level interaction, specifically for ensuring a clean exit from the game environment via `sys.exit()`.
- **Math (Standard Library Implied):**
    - **Purpose:** Used for mathematical functions critical to the game's logic, such as `log3(val)` to calculate the exponent base and dynamically map tile values to distinct colors, ensuring a proportional color progression.

4.DATA UTILIZED:

The game does not depend on any external dataset. Instead, it utilizes internal data structures to represent the game state. The primary data structure is a two-dimensional list (matrix) of size 4×4, representing the grid. Each element holds the current tile value (3, 9, 27, etc.) or zero if the cell is empty. The program uses variables to maintain game state such as score, grid size, and target tile value. Random number generation is applied to decide new tile placements, ensuring variability and unpredictability in gameplay

5. Visual Design and User Experience (UX):

The user interface is designed for clarity, visual feedback, and a satisfying celebratory climax.

Clean Visuals & Color Palette

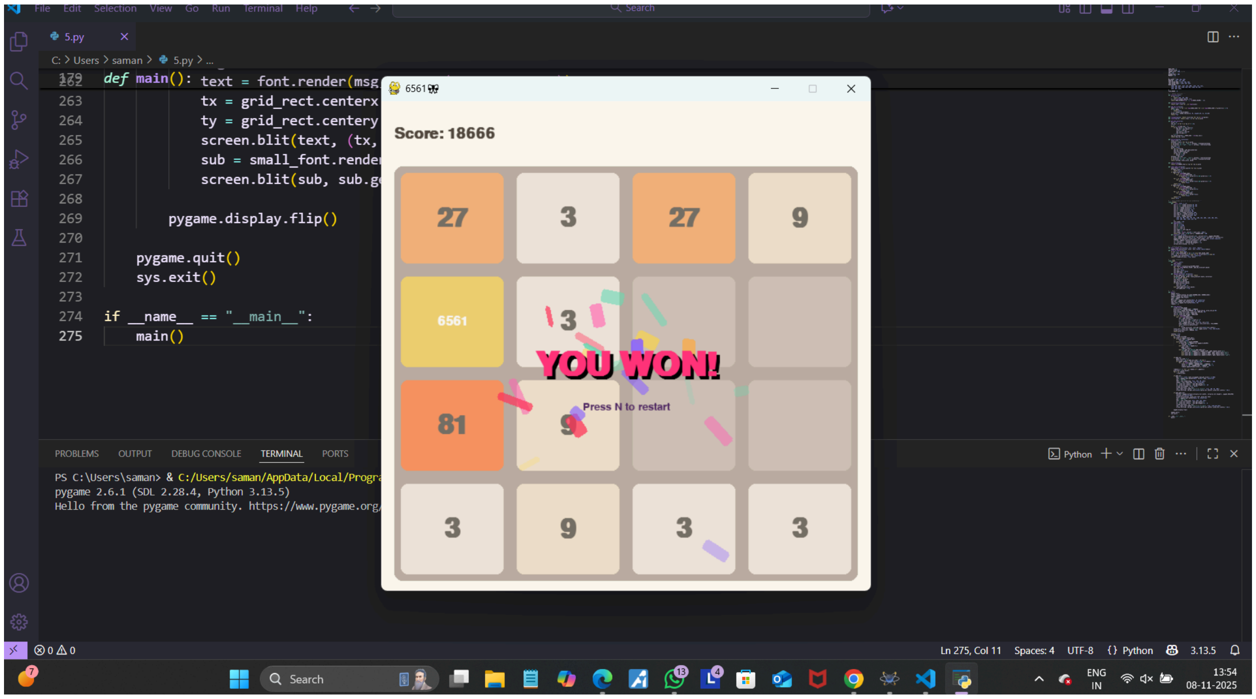
- **Background:** BG_COLOR is a soft, clean off-white (250, 248, 239).
- **Grid:** GRID_BG is a neutral taupe (187, 173, 160).
- **Tiles:** A custom color palette(BASE_COLORS) is used, with colors growing warmer and more vibrant for higher-value tiles.

A. Initial Game State:

The game window features a clean background, a soft-colored grid frame, and rounded tiles. The status bar displays the current score.

Initial State Screenshot:
The application window shows the starting 4 x 4 grid with two initial tiles (two '3's) and a score of 0.

State	Display Details
WIN	**"YOU WON!"** message with a vibrant, pulsating effect, and a sub-message: "Press N to restart."
LOSE	**"YOU LOST"** message centered over a semi-transparent dark overlay, with a sub-message: "Press N to try again."



```
def main():   text = font.render(msg
                tx = grid_rect.centerx
                ty = grid_rect.centery
                screen.blit(text, (tx,
                sub = small_font.render
                screen.blit(sub, sub.g

            pygame.display.flip()

    pygame.quit()
    sys.exit()

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

```
PS C:\Users\saman> & C:/Users/saman/AppData/Local/Progra
pygame 2.6.1 (SDL 2.28.4, Python 3.13.5)
Hello from the pygame community. https://www.pygame.org/
```

B. Dynamic Tile Rendering

- **Color Mapping:** The **tile_color(val)** function uses log3(val) to dynamically map the value to a distinct color from the (BASE_COLORS) palette.
- **Dynamic Text:** Font size automatically adjusts for large numbers (e.g: 6561) to ensure readability within the rounded tile boundaries.

C. The Victory Sequence and Overlay:

Achieving the 6561 tile triggers the win state, which features a compelling visual reward.

Victory State Screenshot:
The screenshot captures the final winning moment. The 6561 tile is visible, along with the high score (18666). A large, pulsating "YOU WON!" message is centered over the board, surrounded by the colorful ribbon particles.

- **Ribbon Particle System:** The **Ribbon class** creates dynamic particles with randomized physics (gravity, rotation) that burst from the center, serving as a dynamic visual reward for the achievement.
- **Overlay Messaging:** Both the Win and Lose states use **large, centered text** for immediate feedback, guiding the player to the next action ("Press N to restart" or "Press N to try again").
-

6. Primary Algorithms and Techniques:

The core logic of the game is implemented using modular functions and a class-based design for readability and reusability. The major algorithms and techniques used include: • Tile Movement Algorithm: Tiles are shifted and merged in the chosen direction using transpose() and reverse() operations to simplify vertical and horizontal moves. When two identical tiles meet, they merge to form a new tile with triple the value. • Random Tile Generation: New tiles (values 3 or 9) appear after every valid move, with a weighted probability favoring 3s. • Game State Checking: The check_win() and check_game_over() functions determine the game's status based on tile values and empty cells. • Rendering Technique: Pygame's Surface and Rect classes are used for graphical rendering of tiles, background, and text. Tile colors and sizes change dynamically based on their value. • Animation & Feedback: A "Ribbon" particle system generates colorful ribbons for visual celebration upon winning, enhancing the player's experience

7. Conclusion

The "6561" project successfully delivered a complete, highly engaging, and unique variation of the classic tile-merging game. It demonstrates solid skills in **Pygame implementation**, **mathematical game design** (powers of 3), **efficient matrix manipulation** for movement, and advanced **visual effects** via the custom particle system. The inclusion of these screenshots provides clear evidence of the game's polished user interface and core functionalities.The addition of the dynamic ribbon celebration enhances the user experience and provides a satisfying climax to reaching the challenging 6561 target.