MINI PROJECT

Title:-Snake and Ladders Game in Python

Objective:- To design and implement a Python-based Snake and Ladders game that simulates dice rolls, player turns, and win conditions while reinforcing core programming concepts like OOP, loops, and randomization.

Theory:-

The Snake and Ladders game is a classic board game that originated in ancient India as Moksha Patam.It is a multiplayer game where players roll a die to move their tokens across a numbered grid, typically from 1 to 100. The game features ladders, which act as shortcuts to advance players forward, and snakes, which penalize players by moving them backward. The gameplay is driven by randomized dice rolls and turn-based mechanics, with the first player to reach the final position (usually 100) declared the winner. This simple yet engaging game serves as an excellent project for learning fundamental programming concepts in Python.

The game mechanics are governed by a set of basic rules. Players start at position 0 or 1, depending on the design, and take turns rolling a six-sided die to determine their movement. If a player lands at the base of a ladder, they climb to the top, while landing on a snake's head sends them sliding down to its tail. A player must roll the exact number required to reach position 100 to win, adding an element of strategy and anticipation. The game board is typically represented using a one-dimensional list or dictionary in Python, with snakes and ladders predefined as key-value pairs for efficient lookup and movement.

In the Python implementation, Object-Oriented Programming (OOP) principles are employed to encapsulate the game logic. A `SnakeAndLadder` class is used to manage player positions, snakes, ladders, and turn sequences. Key methods include `roll_dice()` for generating random die rolls, `move_player()` for updating positions and checking for snakes or ladders, and `play_turn()` to handle player moves. Control flow is managed using a `while` loop that runs until a player wins, with conditional checks to detect snakes, ladders, and win conditions. Data structures like dictionaries and lists are utilized to store game elements and track player progress efficiently.

Randomization plays a crucial role in ensuring fair and dynamic gameplay. The `random` module in Python generates unbiased dice rolls, preventing predictability and maintaining the game’s excitement. This project not only reinforces core Python concepts such as loops, conditionals, and OOP but also provides a foundation for more advanced developments, such as GUI implementations using Pygame or Tkinter. By understanding the underlying logic of this game, learners gain valuable insights into game development and Python’s versatility in creating interactive applications. Future enhancements could include multiplayer networking, AI opponents, or customizable game boards, further expanding the project’s scope and educational value.

The Snake and Ladder game, while simple in appearance, presents rich theoretical challenges in probability, game theory, and algorithmic design. Its implementation touches on fundamental computer science concepts while providing a tangible framework for exploring stochastic processes. Future research directions could formalize:

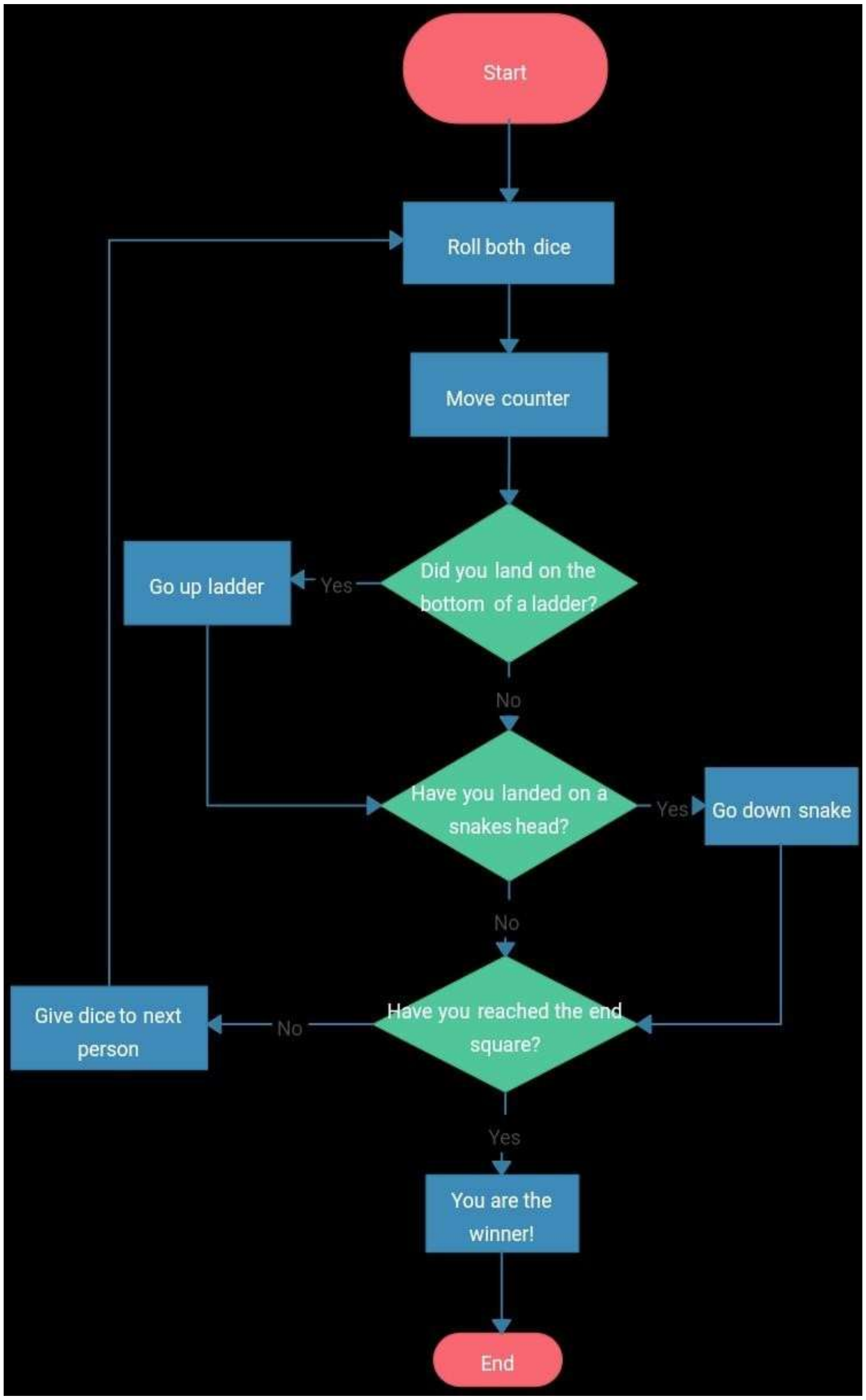
- Exact solutions for expected game length

- Nash equilibria in modified competitive versions

- Computational complexity of optimal play determination

This analysis demonstrates how traditional games can serve as vehicles for sophisticated theoretical exploration while maintaining accessibility for practical implementation.

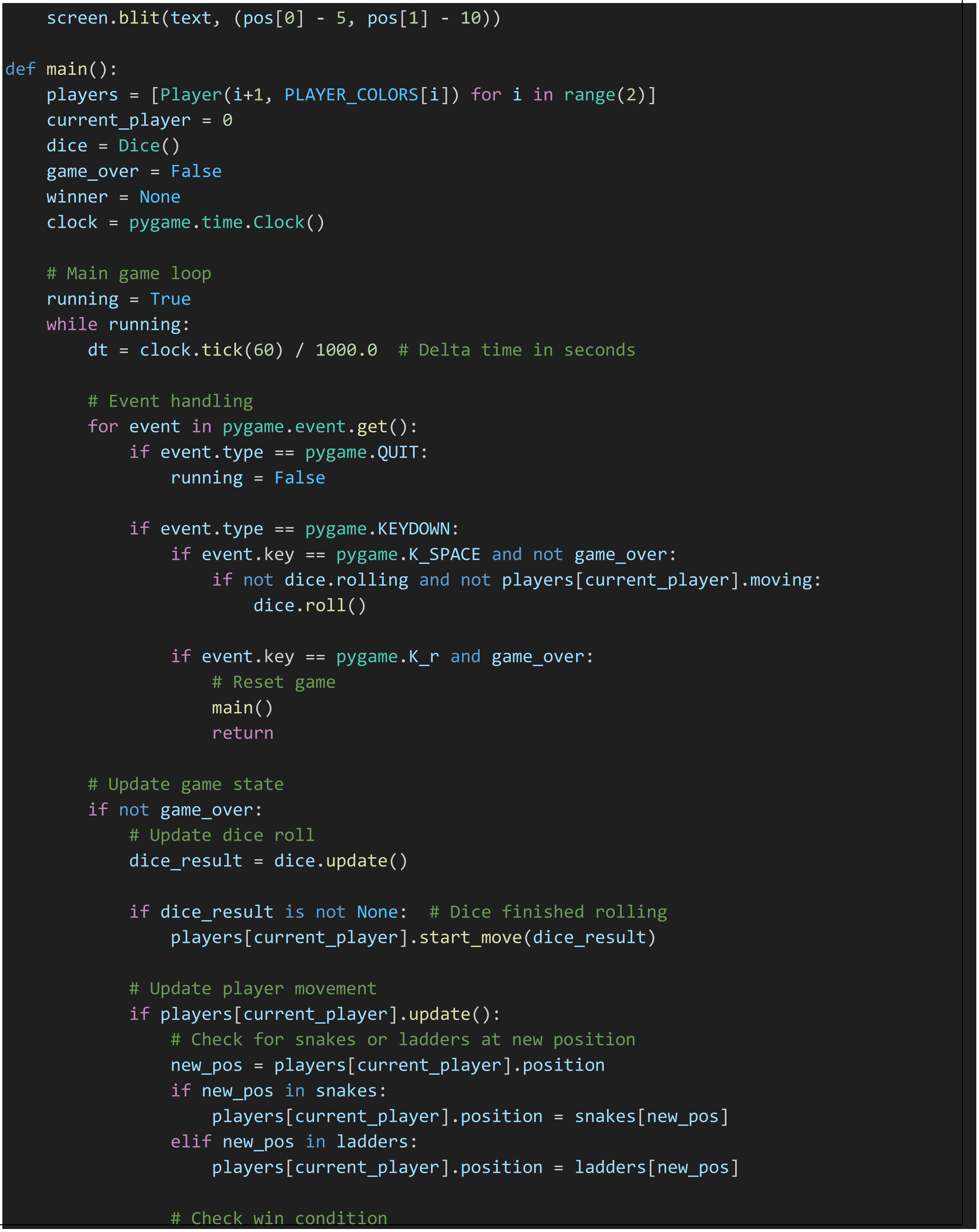
Flowchart :


Start
Roll both dice
Move counter
Did you land on the bottom of a ladder?
Yes
Go up ladder
No
Have you landed on a snakes head?
Yes
Go down snake
No
Have you reached the end square?
No
Give dice to next person
Yes
You are the winner!
End

Program code :

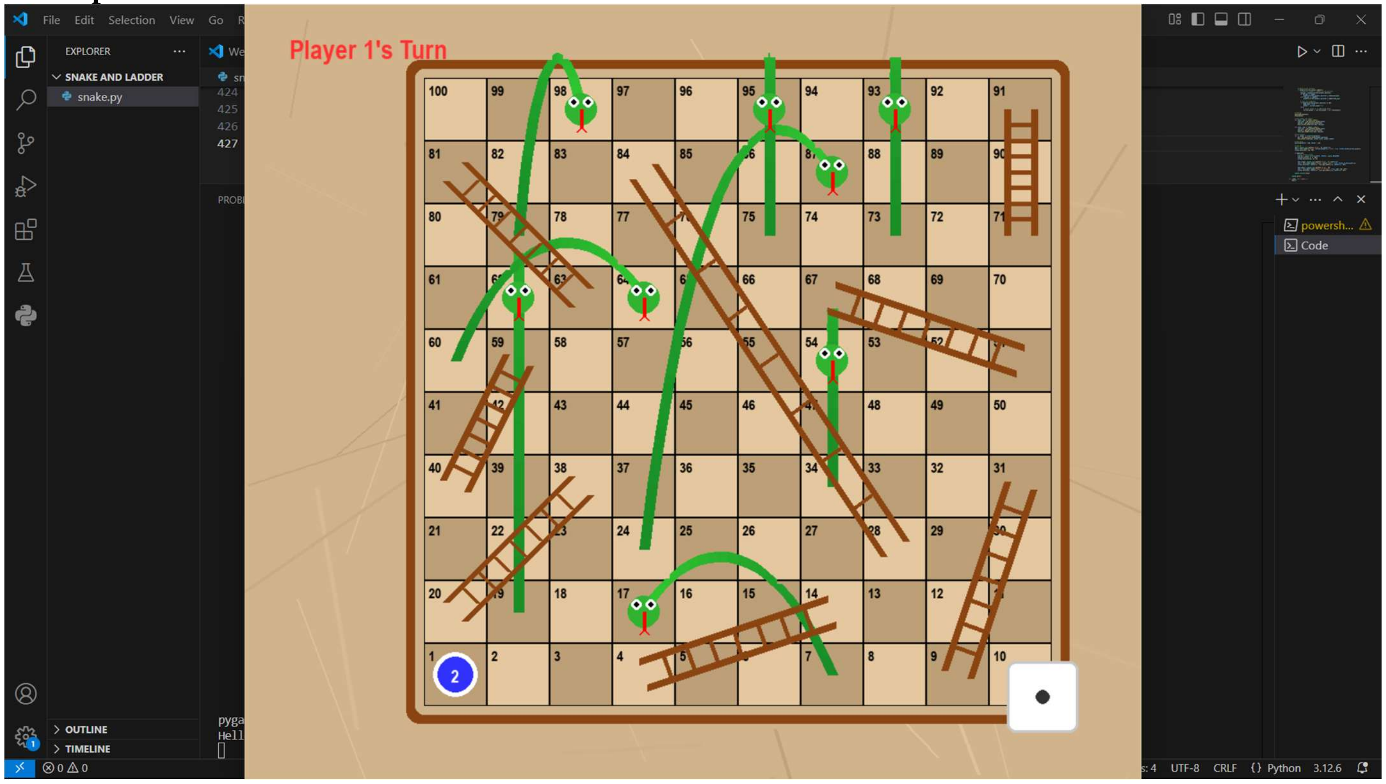
```python
import pygame
import random
import math
import time
from pygame import gfxdraw

# Initialize pygame
pygame.init()
pygame.mixer.init()

# Screen setup
WIDTH, HEIGHT = 1000, 900
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Ultimate Snake & Ladder")

# Colors
WOOD_COLOR = (210, 180, 140)  # Realistic wood color
DARK_WOOD = (139, 69, 19)     # Darker wood for accents
SNAKE_COLOR = (50, 200, 50)   # Vibrant green for snakes
PLAYER_COLORS = [
    (255, 50, 50),   # Red
    (50, 50, 255),   # Blue
    (255, 255, 50),  # Yellow
    (50, 255, 50)    # Green
]

# Game settings
GRID_SIZE = 10
SQUARE_SIZE = 70
BOARD_OFFSET_X = 200
BOARD_OFFSET_Y = 100
DICE_SIZE = 80

# Snake and ladder positions
snakes = {
    17: 7, 54: 34, 62: 19, 64: 60, 87: 24, 93: 73, 95: 75, 98: 79
}
ladders = {
    4: 14, 9: 31, 20: 38, 28: 84, 40: 59, 51: 67, 63: 81, 71: 91
}

class Dice:
    def __init__(self):
        self.value = 1
        self.rolling = False
        self.roll_start_time = 0
        self.roll_duration = 1.0  # seconds
        self.faces = [
            self.create_dice_face(1),
            self.create_dice_face(2),
            self.create_dice_face(3),
            self.create_dice_face(4),
            self.create_dice_face(5),
            self.create_dice_face(6)
        ]

    def create_dice_face(self, value):
        """Create a 3D-looking dice face"""
        face = pygame.Surface((DICE_SIZE, DICE_SIZE), pygame.SRCALPHA)
        pygame.draw.rect(face, (255, 255, 255), (0, 0, DICE_SIZE, DICE_SIZE),
border_radius=10)
        pygame.draw.rect(face, (200, 200, 200), (0, 0, DICE_SIZE, DICE_SIZE), width=3,
border_radius=10)

        # Draw dots based on value
        dot_color = (50, 50, 50)
        dot_radius = 8
        if value % 2 == 1:  # Center dot for odd numbers
            pygame.draw.circle(face, dot_color, (DICE_SIZE//2, DICE_SIZE//2), dot_radius)
        if value > 1:
            pygame.draw.circle(face, dot_color, (DICE_SIZE//4, DICE_SIZE//4), dot_radius)  #
Top-left
            pygame.draw.circle(face, dot_color, (3*DICE_SIZE//4, 3*DICE_SIZE//4),
dot_radius)  # Bottom-right
        if value > 3:
            pygame.draw.circle(face, dot_color, (3*DICE_SIZE//4, DICE_SIZE//4),
dot_radius)  # Top-right
            pygame.draw.circle(face, dot_color, (DICE_SIZE//4, 3*DICE_SIZE//4),
dot_radius)  # Bottom-left
        if value == 6:
            pygame.draw.circle(face, dot_color, (DICE_SIZE//4, DICE_SIZE//2), dot_radius)  #
Mid-left
            pygame.draw.circle(face, dot_color, (3*DICE_SIZE//4, DICE_SIZE//2),
dot_radius)  # Mid-right

        return face

    def roll(self):
        if not self.rolling:
            self.rolling = True
            self.roll_start_time = time.time()
            return True
        return False

    def update(self):
        if self.rolling:
            elapsed = time.time() - self.roll_start_time
            if elapsed >= self.roll_duration:
                self.rolling = False
                self.value = random.randint(1, 6)
                return self.value
            else:
                # Animate by rapidly changing values during roll
                self.value = random.randint(1, 6)
        return None

    def draw(self, x, y):
        current_face = self.faces[self.value - 1]

        if self.rolling:
            # Add spinning effect
            angle = (time.time() - self.roll_start_time) * 720  # 2 rotations per second
            current_face = pygame.transform.rotate(current_face, angle)
            # Get new rect centered at original position
            rect = current_face.get_rect(center=(x + DICE_SIZE//2, y + DICE_SIZE//2))
            screen.blit(current_face, rect)
        else:
            screen.blit(current_face, (x, y))

class Player:
    def __init__(self, number, color):
        self.number = number
        self.color = color
        self.position = 1
        self.target_position = 1
        self.moving = False
        self.move_start_time = 0
        self.move_duration = 0.5  # seconds
        self.path = []
        self.current_path_index = 0

    def start_move(self, steps):
        self.target_position = min(self.position + steps, 100)
        self.calculate_path()
        self.moving = True
        self.move_start_time = time.time()

    def calculate_path(self):
        self.path = []
        current = self.position
        while current < self.target_position:
            current += 1
            self.path.append(current)
        self.current_path_index = 0

    def update(self):
        if not self.moving:
            return False

        if self.current_path_index < len(self.path):
            self.position = self.path[self.current_path_index]
            self.current_path_index += 1
            return False
        else:
            self.moving = False
            return True

def draw_wood_texture():
    """Draws a realistic wood texture background"""
    # Base wood color
    screen.fill(WOOD_COLOR)

    # Add wood grain effect
    for i in range(50):
        x = random.randint(0, WIDTH)
        y = random.randint(0, HEIGHT)
        length = random.randint(100, 300)
        width = random.randint(2, 5)
        angle = random.uniform(0, math.pi)

        end_x = x + length * math.cos(angle)
        end_y = y + length * math.sin(angle)

        color_variation = random.randint(-20, 20)
        grain_color = (
            max(0, min(255, WOOD_COLOR[0] + color_variation)),
            max(0, min(255, WOOD_COLOR[1] + color_variation)),
            max(0, min(255, WOOD_COLOR[2] + color_variation))
        )

        pygame.draw.line(screen, grain_color, (x, y), (end_x, end_y), width)

def draw_board():
    """Draws the game board with realistic wood texture"""
    # Board background
    board_rect = pygame.Rect(
        BOARD_OFFSET_X - 20, BOARD_OFFSET_Y - 20,
        GRID_SIZE * SQUARE_SIZE + 40, GRID_SIZE * SQUARE_SIZE + 40
    )
    pygame.draw.rect(screen, DARK_WOOD, board_rect, border_radius=15)
    pygame.draw.rect(screen, WOOD_COLOR, (
        BOARD_OFFSET_X - 10, BOARD_OFFSET_Y - 10,
        GRID_SIZE * SQUARE_SIZE + 20, GRID_SIZE * SQUARE_SIZE + 20
    ), border_radius=10)

    # Grid squares
    for row in range(GRID_SIZE):
        for col in range(GRID_SIZE):
            rect = pygame.Rect(
                BOARD_OFFSET_X + col * SQUARE_SIZE,
                BOARD_OFFSET_Y + row * SQUARE_SIZE,
                SQUARE_SIZE, SQUARE_SIZE
            )

            # Alternate square shading
            if (row + col) % 2 == 0:
                shade = 20
            else:
                shade = -20

            square_color = (
                max(0, min(255, WOOD_COLOR[0] + shade)),
                max(0, min(255, WOOD_COLOR[1] + shade)),
                max(0, min(255, WOOD_COLOR[2] + shade))
            )
            pygame.draw.rect(screen, square_color, rect)
            pygame.draw.rect(screen, (0, 0, 0), rect, 1)  # Border

            # Draw number
            number = (GRID_SIZE - row - 1) * GRID_SIZE + (col + 1 if (GRID_SIZE - row - 1) %
2 == 0 else GRID_SIZE - col)
            font = pygame.font.SysFont('Arial', 16, bold=True)
            text = font.render(str(number), True, (0, 0, 0))
            screen.blit(text, (rect.x + 5, rect.y + 5))

def draw_real_ladder(start_pos, end_pos):
    """Draws a realistic ladder with side rails and rungs"""
    # Calculate direction vector
    dx = end_pos[0] - start_pos[0]
    dy = end_pos[1] - start_pos[1]
    length = math.sqrt(dx*dx + dy*dy)

    # Normalize and get perpendicular vector
    nx, ny = dx/length, dy/length
    perp_x, perp_y = -ny * 15, nx * 15  # Ladder width

    # Draw side rails (with wood texture)
    pygame.draw.line(screen, DARK_WOOD,
                   (start_pos[0] + perp_x, start_pos[1] + perp_y),
                    (end_pos[0] + perp_x, end_pos[1] + perp_y),
                    8)
    pygame.draw.line(screen, DARK_WOOD,
                    (start_pos[0] - perp_x, start_pos[1] - perp_y),
                    (end_pos[0] - perp_x, end_pos[1] - perp_y),
                    8)

    # Draw rungs (7 rungs per ladder)
    for i in range(1, 8):
        t = i / 8
        rung_start = (
            start_pos[0] + dx*t - perp_x*0.8,
            start_pos[1] + dy*t - perp_y*0.8
        )
        rung_end = (
            start_pos[0] + dx*t + perp_x*0.8,
            start_pos[1] + dy*t + perp_y*0.8
        )
        pygame.draw.line(screen, DARK_WOOD, rung_start, rung_end, 5)

def draw_curved_snake(start_pos, end_pos):
    """Draws a mobile-style snake with curved body"""
    # Control points for bezier curve
    control1 = (
        start_pos[0] + (end_pos[0] - start_pos[0]) * 0.3,
        start_pos[1] - 100
    )
    control2 = (
        start_pos[0] + (end_pos[0] - start_pos[0]) * 0.7,
        start_pos[1] - 80
    )

    # Calculate curve points
    points = []
    for t in [i/20 for i in range(21)]:
        x = (1-t)**3 * start_pos[0] + 3*(1-t)**2*t * control1[0] + 3*(1-t)*t**2 * control2[0]
+ t**3 * end_pos[0]
        y = (1-t)**3 * start_pos[1] + 3*(1-t)**2*t * control1[1] + 3*(1-t)*t**2 * control2[1]
+ t**3 * end_pos[1]
        points.append((x, y))

    # Draw snake body with gradient
    if len(points) > 1:
        for i in range(len(points)-1):
            # Gradient from green to darker green
            color_factor = i / (len(points)-1)
            color = (
                int(SNAKE_COLOR[0] * (1 - color_factor * 0.5)),
```

```python
                int(SNAKE_COLOR[1] * (1 - color_factor * 0.3)),
                int(SNAKE_COLOR[2] * (1 - color_factor * 0.2))
            )
            pygame.draw.line(screen, color, points[i], points[i+1], 12)

    # Draw snake head
    draw_snake_head(start_pos)

def draw_snake_head(pos):
    """Draws a detailed snake head"""
    # Head
    pygame.draw.circle(screen, (50, 180, 50), pos, 18)

    # Eyes
    pygame.draw.circle(screen, (255, 255, 255), (pos[0]-8, pos[1]-8), 6)
    pygame.draw.circle(screen, (255, 255, 255), (pos[0]+8, pos[1]-8), 6)
    pygame.draw.circle(screen, (0, 0, 0), (pos[0]-8, pos[1]-8), 3)
    pygame.draw.circle(screen, (0, 0, 0), (pos[0]+8, pos[1]-8), 3)

    # Tongue
    pygame.draw.line(screen, (255, 0, 0), pos, (pos[0], pos[1]+20), 4)
    pygame.draw.line(screen, (255, 0, 0), (pos[0], pos[1]+20), (pos[0]-5, pos[1]+25), 2)
    pygame.draw.line(screen, (255, 0, 0), (pos[0], pos[1]+20), (pos[0]+5, pos[1]+25), 2)

def get_position_coords(position):
    """Converts board position to screen coordinates"""
    if position < 1 or position > 100:
        return (0, 0)

    row = GRID_SIZE - ((position - 1) // GRID_SIZE) - 1
    col = (position - 1) % GRID_SIZE

    if (GRID_SIZE - row - 1) % 2 != 0:
        col = GRID_SIZE - col - 1

    x = BOARD_OFFSET_X + col * SQUARE_SIZE + SQUARE_SIZE // 2
    y = BOARD_OFFSET_Y + row * SQUARE_SIZE + SQUARE_SIZE // 2
    return (x, y)

def draw_player_token(pos, color, number):
    """Draws a player token with highlight effect"""
    # Outer glow
    pygame.draw.circle(screen, (255, 255, 255, 100), pos, 25)
    # Main token
    pygame.draw.circle(screen, color, pos, 20)
    # Number
    font = pygame.font.SysFont('Arial', 20, bold=True)
    text = font.render(str(number), True, (255, 255, 255))
```



```python
    screen.blit(text, (pos[0] - 5, pos[1] - 10))

def main():
    players = [Player(i+1, PLAYER_COLORS[i]) for i in range(2)]
    current_player = 0
    dice = Dice()
    game_over = False
    winner = None
    clock = pygame.time.Clock()

    # Main game loop
    running = True
    while running:
        dt = clock.tick(60) / 1000.0  # Delta time in seconds

        # Event handling
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE and not game_over:
                    if not dice.rolling and not players[current_player].moving:
                        dice.roll()

                if event.key == pygame.K_r and game_over:
                    # Reset game
                    main()
                    return

        # Update game state
        if not game_over:
            # Update dice roll
            dice_result = dice.update()

            if dice_result is not None:  # Dice finished rolling
                players[current_player].start_move(dice_result)

            # Update player movement
            if players[current_player].update():
                # Check for snakes or ladders at new position
                new_pos = players[current_player].position
                if new_pos in snakes:
                    players[current_player].position = snakes[new_pos]
                elif new_pos in ladders:
                    players[current_player].position = ladders[new_pos]

                # Check win condition
                if players[current_player].position == 100:
                    game_over = True
                    winner = current_player + 1
                else:
                    # Switch player if no additional moves
                    current_player = (current_player + 1) % len(players)

        # Drawing
        draw_wood_texture()
        draw_board()

        # Draw snakes and ladders
        for start, end in snakes.items():
            start_pos = get_position_coords(start)
            end_pos = get_position_coords(end)
            draw_curved_snake(start_pos, end_pos)

        for start, end in ladders.items():
            start_pos = get_position_coords(start)
            end_pos = get_position_coords(end)
            draw_real_ladder(start_pos, end_pos)

        # Draw players
        for i, player in enumerate(players):
            pos = get_position_coords(player.position)
            draw_player_token(pos, player.color, player.number)

        # Draw dice
        dice.draw(WIDTH - 150, HEIGHT - 150)

        # Draw UI
        font = pygame.font.SysFont('Arial', 30, bold=True)
        text = font.render(f"Player {current_player+1}'s Turn", True,
PLAYER_COLORS[current_player])
        screen.blit(text, (50, 50))

        if game_over:
            # Draw victory screen
            overlay = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)
            overlay.fill((0, 0, 0, 180))
            screen.blit(overlay, (0, 0))

            font_large = pygame.font.SysFont('Arial', 72, bold=True)
            text = font_large.render(f"Player {winner} Wins!", True, PLAYER_COLORS[winner-1])
            screen.blit(text, (WIDTH//2 - text.get_width()//2, HEIGHT//2 - 50))

            font_small = pygame.font.SysFont('Arial', 36)
            text = font_small.render("Press R to Restart", True, (255, 255, 255))
            screen.blit(text, (WIDTH//2 - text.get_width()//2, HEIGHT//2 + 30))

        pygame.display.flip()

    pygame.quit()

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

Output:-


File Edit Selection View Go
EXPLORER
SNAKE AND LADDER
snake.py
OUTLINE
TIMELINE
Player 1's Turn
2
UTF-8 CRLF {} Python 3.12.6
powersh...
Code