```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <cstdlib>
#include <SFML/Audio.hpp>
#include <cmath>
#include <SFML/System.hpp>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
using namespace sf;
int yellowPath[58][2] = {
    {650, 140}, {650, 200}, {650, 260}, {650, 320}, {650, 380}, {710, 440}, {770,
440}, {830, 440}, {890, 440}, {950, 440}, {1010, 440}, {1010, 500}, {1010, 560},
{950, 560}, {890, 560}, {830, 560}, {770, 560}, {710, 560}, {650, 620}, {650, 680},
{650, 740}, {650, 800}, {650, 860}, {650, 920}, {590, 920}, {530, 920}, {530, 860},
{530, 800}, {530, 740}, {530, 680}, {530, 620}, {470, 560}, {410, 560}, {350, 560},
{290, 560}, {230, 560}, {170, 560}, {170, 500}, {170, 440}, {230, 440}, {290, 440},
{350, 440}, {410, 440}, {470, 440}, {530, 380}, {530, 320}, {530, 260}, {530, 200},
{530, 140}, {530, 80}, {590, 80}, {590, 140}, {590, 200}, {590, 260}, {590, 320},
{590, 380}, {590, 440}},
    yellowHome[4][2] = {{770, 140}, {950, 140}, {770, 320}, {950, 320}},
bluePath[58][2] = {{530, 860}, {530, 800}, {530, 740}, {530, 680}, {530, 620},
{470, 560}, {410, 560}, {350, 560}, {290, 560}, {230, 560}, {170, 560}, {170, 500},
{170, 440}, {230, 440}, {290, 440}, {350, 440}, {410, 440}, {470, 440}, {530, 380},
{530, 320}, {530, 260}, {530, 200}, {530, 140}, {530, 80}, {590, 80}, {650, 80},
{650, 140}, {650, 200}, {650, 260}, {650, 320}, {650, 380}, {710, 440}, {770, 440},
{830, 440}, {890, 440}, {950, 440}, {1010, 440}, {1010, 500}, {1010, 560}, {950,
560}, {890, 560}, {830, 560}, {770, 560}, {710, 560}, {650, 620}, {650, 680}, {650,
740}, {650, 800}, {650, 860}, {650, 920}, {590, 920}, {590, 860}, {590, 800}, {590,
740}, {590, 680}, {590, 620}, {590, 560}}, blueHome[4][2] = {{230, 680}, {410,
680}, {230, 860}, {410, 860}}, redPath[58][2] = {{230, 440}, {290, 440}, {350,
440}, {410, 440}, {470, 440}, {530, 380}, {530, 320}, {530, 260}, {530, 200}, {530,
140}, {530, 80}, {590, 80}, {650, 80}, {650, 140}, {650, 200}, {650, 260}, {650,
320}, {650, 380}, {710, 440}, {770, 440}, {830, 440}, {890, 440}, {950, 440},
{1010, 440}, {1010, 500}, {1010, 560}, {950, 560}, {890, 560}, {830, 560}, {770,
560}, {710, 560}, {650, 620}, {650, 680}, {650, 740}, {650, 800}, {650, 860}, {650,
920}, {590, 920}, {530, 920}, {530, 860}, {530, 800}, {530, 740}, {530, 680}, {530,
620}, {470, 560}, {410, 560}, {350, 560}, {290, 560}, {230, 560}, {170, 560}, {170,
500}, {230, 500}, {290, 500}, {350, 500}, {410, 500}, {470, 500}, {530, 500}},
redHome[4][2] = {{230, 140}, {410, 140}, {230, 320}, {410, 320}}, greenPath[58][2]
= {{950, 560}, {890, 560}, {830, 560}, {770, 560}, {710, 560}, {650, 620}, {650,
680}, {650, 740}, {650, 800}, {650, 860}, {650, 920}, {590, 920}, {530, 920}, {530,
860}, {530, 800}, {530, 740}, {530, 680}, {530, 620}, {470, 560}, {410, 560}, {350,
560}, {290, 560}, {230, 560}, {170, 560}, {170, 500}, {170, 440}, {230, 440}, {290,
440}, {350, 440}, {410, 440}, {470, 440}, {530, 380}, {530, 320}, {530, 260}, {530,
200}, {530, 140}, {530, 80}, {590, 80}, {650, 80}, {650, 140}, {650, 200}, {650,
260}, {650, 320}, {650, 380}, {710, 440}, {770, 440}, {830, 440}, {890, 440}, {950,
440}, {1010, 440}, {1010, 500}, {950, 500}, {890, 500}, {830, 500}, {770, 500},
{710, 500}, {650, 500}}, greenHome[4][2] = {{770, 680}, {950, 680}, {770, 860},
{950, 860}};
class LudoPiece
{
private:
    int currentX, currentY;

public:
    void drawLudoPiece(sf::RenderWindow &window, sf::Color color)
    {
        sf::CircleShape piece(24);
        piece.setOutlineColor(Color::Black);
        piece.setOutlineThickness(2);
            piece.setPosition(currentX-25 , currentY-25);
        piece.setFillColor(color);
        window.draw(piece);
    }
    void setPiece(int x, int y)
    {
        currentX = x;
        currentY = y;
    }
    int getX()
    {
        return currentX;
    }
    int getY()
    {
        return currentY;
    }
};
class Player
{
private:
    sf::Color color;
    LudoPiece piece[4];
    bool status;

public:
    Player() {}
    Player(sf::Color color) : color(color),status(false) {}
    int homePosition[4][2], path[58][2];

    void setPath(int playerHome[4][2], int playerPath[58][2])
    {
        for (int i = 0; i < 4; i++)
        {
            homePosition[i][0] = playerHome[i][0];
            homePosition[i][1] = playerHome[i][1];
        }
        for (int i = 0; i < 58; i++)
        {
            path[i][0] = playerPath[i][0];
            path[i][1] = playerPath[i][1];
        }
    }
    void setStatus(bool status){
      this->status = status;
      }
      bool getStatus(){
            return status;
      }
    void setColor(sf::Color color){
      this->color = color;
      }
    sf::Color getColor() const
    {
        return color;
    }
    LudoPiece *getPiece()
    {
        return piece;
    }
};
class Coordinates
{
      private:
      int midX;
    int midY;
    int row;
    int col;
public:

    Coordinates getcord(sf::RenderWindow &window)
    {
        int clickedRow = -1, clickedColumn = -1;
        int startX = 500, startY = 50, TILESIZE = 60;

        while (true)
        {
            sf::Event event;
            while (window.pollEvent(event))
            {
                if (event.type == sf::Event::Closed)
                {
                    window.close();
                    exit(0);
                }
                else if (event.type == sf::Event::MouseButtonPressed &&
event.mouseButton.button == sf::Mouse::Left)
                {
                    sf::Vector2i mouseClickPos = sf::Mouse::getPosition(window);
                    // current position of the mouse click
                    int mouseClickX = mouseClickPos.x;
                    int mouseClickY = mouseClickPos.y;

                    // identifying potential row
                    clickedRow = (mouseClickY - startY) / TILESIZE;

                    // for vertical paths
                    if ((clickedRow <= 5 || (clickedRow >= 9 && clickedRow <= 14))
&&
                        (mouseClickX >= startX && mouseClickX <= startX + (3 *
TILESIZE)) &&
                        (mouseClickY >= startY && mouseClickY <= startY + (15 *
TILESIZE)))
                    {
                        clickedColumn = (mouseClickX - startX) / TILESIZE;

                        int topx = startX + clickedColumn * TILESIZE;
                        int topy = startY + clickedRow * TILESIZE;
                        int botx = topx + TILESIZE;
                        int boty = topy + TILESIZE;

                        midX = (topx + botx) / 2;
                        midY = (topy + boty) / 2;
                        row = clickedRow;
                        col = clickedColumn;

                        return *this;

                        // for horizontal paths
                    }
                    else if ((clickedRow > 5 && clickedRow <= 8) &&
                             (mouseClickX >= startX + (3 * TILESIZE)) &&
                             (mouseClickY >= startY + (6 * TILESIZE)) &&
                             (mouseClickX <= startX + (9 * TILESIZE)) &&
                             (mouseClickY <= startY + (9 * TILESIZE)))
                    {
                        clickedColumn = (mouseClickX - (startX + (3 * TILESIZE))) /
TILESIZE;
                        int topx = startX + (3 * TILESIZE) + (clickedColumn *
TILESIZE);
                        int topy = startY + (6 * TILESIZE) + ((clickedRow - 6) *
TILESIZE);
                        int botx = topx + TILESIZE;
                        int boty = topy + TILESIZE;

                        midX = (topx + botx) / 2;
                        midY = (topy + boty) / 2;
                        row = clickedRow;
                        col = clickedColumn;

                        return *this;
                    }
                    else if ((clickedRow > 5 && clickedRow <= 8) &&
                             (mouseClickX >= startX - (6 * TILESIZE)) &&
                             (mouseClickY >= startY + (6 * TILESIZE)) &&
                             (mouseClickX <= startX) &&
                             (mouseClickY <= startY + (9 * TILESIZE)))
                    {
                        clickedColumn = (mouseClickX - (startX - (6 * TILESIZE))) /
TILESIZE;
                        int topx = startX - (6 * TILESIZE) + clickedColumn *
TILESIZE;
                        int topy = startY + (6 * TILESIZE) + (clickedRow - 6) *
TILESIZE;
                        int botx = topx + TILESIZE;
                        int boty = topy + TILESIZE;

                        midX = (topx + botx) / 2;
                        midY = (topy + boty) / 2;
                        row = clickedRow;
                        col = clickedColumn;

                        return *this;
                    }

                    // for Yellow and Green Home
                    else if ((clickedRow >= 1 && clickedRow <= 4 || (clickedRow >=
10 && clickedRow <= 13)) &&
                             mouseClickX >= startX + (4 * TILESIZE) && mouseClickX
<= startX + (8 * TILESIZE) &&
                             ((mouseClickY >= startY + TILESIZE && mouseClickY <=
startY + (5 * TILESIZE)) ||
                              (mouseClickY >= startY + (10 * TILESIZE) &&
mouseClickY <= startY + (14 * TILESIZE))))

                    {
                        clickedColumn = (mouseClickX - (startX + (4 * TILESIZE))) /
TILESIZE;

                        if (clickedRow >= 1 && clickedRow <= 4)
                        {
                            int topx = startX + (4 * TILESIZE) + clickedColumn *
TILESIZE;
                            int topy = (startY + TILESIZE) + (clickedRow - 1) *
TILESIZE;
                            int botx = topx + TILESIZE;
                            int boty = topy + TILESIZE;

                            midX = (topx + botx) / 2;
                            midY = (topy + boty) / 2;
                            row = clickedRow;
                            col = clickedColumn;

                            return *this;
                        }

                        else if (clickedRow >= 10 && clickedRow <= 14)
                        {

                            int topx = startX + (4 * TILESIZE) + clickedColumn *
TILESIZE;
                            int topy = (startY + (10 * TILESIZE)) + (clickedRow -
10) * TILESIZE;
                            int botx = topx + TILESIZE;
                            int boty = topy + TILESIZE;

                            midX = (topx + botx) / 2;
                            midY = (topy + boty) / 2;
                            row = clickedRow;
                            col = clickedColumn;

                            return *this;
                        }
                    }
                    else if ((clickedRow >= 1 && clickedRow <= 4 || (clickedRow >=
10 && clickedRow <= 13) &&
                                                                        mouseClickX
>= startX - (5 * TILESIZE) && mouseClickX <= startX - (TILESIZE)) &&
                             ((mouseClickY >= startY + TILESIZE && mouseClickY <=
startY + (5 * TILESIZE)) ||
                              mouseClickY >= startY + (10 * TILESIZE) &&
mouseClickY <= startY + (14 * TILESIZE)))
                    {
                        clickedColumn = (mouseClickX - (startX - (5 * TILESIZE))) /
TILESIZE;
                    }

                    if (clickedRow >= 1 && clickedRow <= 4)
                    {
                        int topx = (startX - (5 * TILESIZE)) + clickedColumn *
TILESIZE;
                        int topy = (startY + TILESIZE) + (clickedRow - 1) *
```

```
TILESIZE;
                        int botx = topx + TILESIZE;
                        int boty = topy + TILESIZE;

                        midX = (topx + botx) / 2;
                        midY = (topy + boty) / 2;
                        row = clickedRow;
                        col = clickedColumn;

                        return *this;
                    }

                    else if (clickedRow >= 10 && clickedRow <= 13)
                    {

                        int topx = (startX - (5 * TILESIZE)) + clickedColumn *
TILESIZE;
                        int topy = (startY + (10 * TILESIZE)) + (clickedRow - 10) *
TILESIZE;
                        int botx = topx + TILESIZE;
                        int boty = topy + TILESIZE;

                        midX = (topx + botx) / 2;
                        midY = (topy + boty) / 2;
                        row = clickedRow;
                        col = clickedColumn;

                        return *this;
                    }
                }
            }
        }
    }
    int getmidX(){
      return midX;
      }
      int getmidY(){
            return midY;
      }
};
class LudoBoard
{
public:
    void drawLudoBoard(sf::RenderWindow &window)
    {
        int TILESIZE = 60;
        int rows = 15, columns = 0;
        int startX = 500, startY = 50;

        sf::RectangleShape rectangle;
        rectangle.setOutlineThickness(1.5);

        for (int i = 0; i < rows; i++)
        {
            if (i <= 5)
            {
                columns = 3;
                rectangle.setOutlineColor(sf::Color::Black);
                for (int j = 0; j < columns; j++)
                {
                    int topx = startX + j * TILESIZE;
                    int topy = startY + i * TILESIZE;

                    rectangle.setSize(sf::Vector2f(TILESIZE, TILESIZE));
                    rectangle.setPosition(topx, topy);
                    rectangle.setFillColor(sf::Color::White);

                    window.draw(rectangle);

                    if (i == 1 && j == 2)
                    {
                        rectangle.setFillColor(sf::Color::Yellow);
                        window.draw(rectangle);
                    }
                    if (i >= 1 && i <= 5 && j == 1)
                    {
                        rectangle.setFillColor(sf::Color::Yellow);
                        window.draw(rectangle);
                    }

                    if (i == 2 && j == 0)
                    {
                        sf::ConvexShape star;
                        star.setOutlineThickness(2);
                        star.setPointCount(12);
                        star.setPosition(startX + (0.5 * TILESIZE), startY + (2.5 *
TILESIZE));
                        float outerRadius = (TILESIZE / 2) - 6;
                        float innerRadius = (TILESIZE / 2) - 16;

                        star.setFillColor(sf::Color::Yellow);
                        star.setOutlineColor(sf::Color::Black);

                        // Calculate the angle between each point of the star
                        float angle = 2 * 3.14159265358979323846 / 12; // 2 * PI /
pointCount

                        // Set the position of each point of the star
                        for (int i = 0; i < 12; i++)
                        {
                            float radius = (i % 2 == 0) ? outerRadius :
innerRadius;
                            float x = radius * std::sin(i * angle);
                            float y = -radius * std::cos(i * angle);
                            star.setPoint(i, sf::Vector2f(x, y));
                        }

                        window.draw(star);
                    }
                }
            }

            if (i >= 9 && i <= 14)
            {
                columns = 3;
                rectangle.setOutlineColor(sf::Color::Black);
                for (int j = 0; j < columns; j++)
                {
                    int topx = startX + j * TILESIZE;
                    int topy = startY + i * TILESIZE;

                    rectangle.setSize(sf::Vector2f(TILESIZE, TILESIZE));
                    rectangle.setPosition(topx, topy);
                    rectangle.setFillColor(sf::Color::White);

                    window.draw(rectangle);

                    if (i == 13 && j == 0)
                    {
                        rectangle.setFillColor(sf::Color::Blue);
                        window.draw(rectangle);
                    }
                    if (i >= 9 && i <= 13 && j == 1)
                    {
                        rectangle.setFillColor(sf::Color::Blue);
                        window.draw(rectangle);
                    }

                    if (i == 13 && j == 2)
                    {
                        sf::ConvexShape star;
                        star.setOutlineThickness(2);
                        star.setPointCount(12);
                        star.setPosition(startX + (2.5 * TILESIZE), startY + (12.5
* TILESIZE));
                        float outerRadius = (TILESIZE / 2) - 6;
                        float innerRadius = (TILESIZE / 2) - 16;

                        star.setFillColor(sf::Color::Blue);
                        star.setOutlineColor(sf::Color::Black);

                        // Calculate the angle between each point of the star
                        float angle = 2 * 3.14159265358979323846 / 12; // 2 * PI /
pointCount

                        // Set the position of each point of the star
                        for (int i = 0; i < 12; i++)
                        {
                            float radius = (i % 2 == 0) ? outerRadius :
innerRadius;

                            float x = radius * std::sin(i * angle);
                            float y = -radius * std::cos(i * angle);
                            star.setPoint(i, sf::Vector2f(x, y));
                        }

                        window.draw(star);
                    }
                }
            }

            if (i > 5 && i <= 8)
            {
                columns = 6;
                rectangle.setOutlineColor(sf::Color::Black);
                for (int j = 0; j < columns; j++)
                {
                    int topx = (startX + TILESIZE * 3) + j * TILESIZE;
                    int topy = (startY + TILESIZE * columns) + (i - columns) * 
TILESIZE;

                    rectangle.setSize(sf::Vector2f(TILESIZE, TILESIZE));
                    rectangle.setPosition(topx, topy);
                    rectangle.setFillColor(sf::Color::White);

                    window.draw(rectangle);

                    if (i == 8 && j == 4)
                    {
                        rectangle.setFillColor((Color(4,155,75)));
                        window.draw(rectangle);
                    }
                    if (i == 7 && j >= 0 && j <= 4)
                    {
                        rectangle.setFillColor((Color(4,155,75)));
                        window.draw(rectangle);
                    }

                    if (i == 6 && j == 3)
                    {
                        sf::ConvexShape star;
                        star.setOutlineThickness(2);
                        star.setPointCount(12);
                        star.setPosition(startX + (6.5 * TILESIZE), startY + (6.5 * 
TILESIZE));
                        float outerRadius = (TILESIZE / 2) - 6;
                        float innerRadius = (TILESIZE / 2) - 16;

                        star.setFillColor((Color(4,155,75)));
                        star.setOutlineColor(sf::Color::Black);

                        // Calculate the angle between each point of the star
                        float angle = 2 * 3.14159265358979323846 / 12; // 2 * PI / 
pointCount

                        // Set the position of each point of the star
                        for (int i = 0; i < 12; i++)
                        {
                            float radius = (i % 2 == 0) ? outerRadius : 
innerRadius;

                            float x = radius * std::sin(i * angle);
                            float y = -radius * std::cos(i * angle);
                            star.setPoint(i, sf::Vector2f(x, y));
                        }

                        window.draw(star);
                    }
                }
                for (int j = 0; j < columns; j++)
                {
                    int topx = (startX - TILESIZE * columns) + j * TILESIZE;
                    int topy = (startY + TILESIZE * columns) + (i - columns) * 
TILESIZE;
```

```
                    rectangle.setSize(sf::Vector2f(TILESIZE, TILESIZE));
                    rectangle.setPosition(topx, topy);
                    rectangle.setFillColor(sf::Color::White);

                    window.draw(rectangle);

                    if (i == 6 && j == 1)
                    {
                        rectangle.setFillColor(sf::Color::Red);
                        window.draw(rectangle);
                    }
                    if (i == 7 && j >= 1 && j <= 5)
                    {
                        rectangle.setFillColor(sf::Color::Red);
                        window.draw(rectangle);
                    }

                    if (i == 8 && j == 2)
                    {
                        sf::ConvexShape star;
                        star.setOutlineThickness(2);
                        star.setPointCount(12);
                        star.setPosition(startX - (3.5 * TILESIZE), startY + (8.5 *
TILESIZE));
                        float outerRadius = (TILESIZE / 2) - 6;
                        float innerRadius = (TILESIZE / 2) - 16;

                        star.setFillColor(sf::Color::Red);
                        star.setOutlineColor(sf::Color::Black);

                        // Calculate the angle between each point of the star
                        float angle = 2 * 3.14159265358979323846 / 12; // 2 * PI /
pointCount

                        // Set the position of each point of the star
                        for (int i = 0; i < 12; i++)
                        {
                            float radius = (i % 2 == 0) ? outerRadius :
innerRadius;
                            float x = radius * std::sin(i * angle);
                            float y = -radius * std::cos(i * angle);
                            star.setPoint(i, sf::Vector2f(x, y));
                        }

                        window.draw(star);
                    }
                }
            }
        }
        // RED QUADRANT
        sf::RectangleShape rectanglered(sf::Vector2f(6 * TILESIZE, 6 * TILESIZE));
        rectanglered.setOutlineColor(sf::Color::Black);
        rectanglered.setOutlineThickness(2);
        rectanglered.setPosition(startX - (6 * TILESIZE), startY);
        rectanglered.setFillColor(sf::Color::Red);
        window.draw(rectanglered);

        // Draw the red circles
        sf::CircleShape circle((TILESIZE / 2) - 1);
        circle.setFillColor(sf::Color::White);
        circle.setOutlineColor(sf::Color::Black);
        circle.setOutlineThickness(2);

        circle.setPosition(startX - (5 * TILESIZE), startY + (1 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX - (5 * TILESIZE), startY + (4 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX - (2 * TILESIZE), startY + (1 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX - (2 * TILESIZE), startY + (4 * TILESIZE));
        window.draw(circle);

        // YELLOW QUADRANT
        sf::RectangleShape rectangleyellow(sf::Vector2f(6 * TILESIZE, 6 *
TILESIZE));
        rectangleyellow.setOutlineColor(sf::Color::Black);
        rectangleyellow.setOutlineThickness(2);
        rectangleyellow.setPosition(startX + (3 * TILESIZE), startY);
        rectangleyellow.setFillColor(sf::Color::Yellow);
        window.draw(rectangleyellow);

        // Draw the yellow circles
        circle.setPosition(startX + (4 * TILESIZE), startY + (1 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX + (7 * TILESIZE), startY + (1 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX + (4 * TILESIZE), startY + (4 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX + (7 * TILESIZE), startY + (4 * TILESIZE));
        window.draw(circle);

        // BLUE QUADRENT
        sf::RectangleShape rectangleblue(sf::Vector2f(6 * TILESIZE, 6 * TILESIZE));
        rectangleblue.setOutlineColor(sf::Color::Black);
        rectangleblue.setOutlineThickness(2);
        rectangleblue.setPosition(startX - (6 * TILESIZE), startY + (9 *
TILESIZE));
        rectangleblue.setFillColor(sf::Color::Blue);
        window.draw(rectangleblue);

        // Draw the blue circles
        circle.setPosition(startX - (5 * TILESIZE), startY + (10 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX - (2 * TILESIZE), startY + (10 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX - (5 * TILESIZE), startY + (13 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX - (2 * TILESIZE), startY + (13 * TILESIZE));
        window.draw(circle);
        // GREEN QUADRENT
        sf::RectangleShape rectanglegreen(sf::Vector2f(6 * TILESIZE, 6 *
TILESIZE));
        rectanglegreen.setOutlineColor(sf::Color::Black);
        rectanglegreen.setOutlineThickness(2);
        rectanglegreen.setPosition(startX + (3 * TILESIZE), startY + (9 *
TILESIZE));
        rectanglegreen.setFillColor((Color(4,155,75)));
        window.draw(rectanglegreen);

        // Draw the green circles
        circle.setPosition(startX + (4 * TILESIZE), startY + (10 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX + (7 * TILESIZE), startY + (10 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX + (4 * TILESIZE), startY + (13 * TILESIZE));
        window.draw(circle);

        circle.setPosition(startX + (7 * TILESIZE), startY + (13 * TILESIZE));

        window.draw(circle);

        // RED HOME
        sf::Vector2f point1(startX + (0.05 * TILESIZE), startY + (6.05 *
TILESIZE));
        sf::Vector2f point2(startX + (0.05 * TILESIZE), startY + (8.95 *
TILESIZE));
        sf::Vector2f point3(startX + (1.5 * TILESIZE), startY + (7.5 * TILESIZE));
        sf::ConvexShape trianglered;
        trianglered.setOutlineColor(sf::Color::Black);
        trianglered.setOutlineThickness(1);
        trianglered.setPointCount(3);
        trianglered.setPoint(0, point1);
        trianglered.setPoint(1, point2);
        trianglered.setPoint(2, point3);
        trianglered.setFillColor(sf::Color::Red);
        window.draw(trianglered);

        // YELLOW HOME
        sf::Vector2f point1yellow(startX + (0.05 * TILESIZE), startY + (6.05 *
TILESIZE));
        sf::Vector2f point2yellow(startX + (2.95 * TILESIZE), startY + (6.05 *
TILESIZE));
        sf::Vector2f point3yellow(startX + (1.5 * TILESIZE), startY + (7.5 *
TILESIZE));
        sf::ConvexShape triangleyellow;
        triangleyellow.setOutlineColor(sf::Color::Black);
        triangleyellow.setOutlineThickness(1);
        triangleyellow.setPointCount(3);
        triangleyellow.setPoint(0, point1yellow);
        triangleyellow.setPoint(1, point2yellow);
        triangleyellow.setPoint(2, point3yellow);
        triangleyellow.setFillColor(sf::Color::Yellow);
        window.draw(triangleyellow);

        // BLUE HOME
        sf::Vector2f point1blue(startX + (0.05 * TILESIZE), startY + (8.95 *
TILESIZE));
        sf::Vector2f point2blue(startX + (2.95 * TILESIZE), startY + (8.95 *
TILESIZE));
        sf::Vector2f point3blue(startX + (1.5 * TILESIZE), startY + (7.5 *
TILESIZE));
        sf::ConvexShape triangleblue;
        triangleblue.setOutlineColor(sf::Color::Black);
        triangleblue.setOutlineThickness(1);
        triangleblue.setPointCount(3);
        triangleblue.setPoint(0, point1blue);
        triangleblue.setPoint(1, point2blue);
        triangleblue.setPoint(2, point3blue);
        triangleblue.setFillColor(sf::Color::Blue);
        window.draw(triangleblue);

        // GREEN HOME
        sf::Vector2f point1green(startX + (2.95 * TILESIZE), startY + (6.05 *
TILESIZE));
        sf::Vector2f point2green(startX + (2.95 * TILESIZE), startY + (8.95 *
TILESIZE));
        sf::Vector2f point3green(startX + (1.5 * TILESIZE), startY + (7.5 *
TILESIZE));
        sf::ConvexShape trianglegreen;
        trianglegreen.setOutlineColor(sf::Color::Black);
        trianglegreen.setOutlineThickness(1);
        trianglegreen.setPointCount(3);
        trianglegreen.setPoint(0, point1green);
        trianglegreen.setPoint(1, point2green);
        trianglegreen.setPoint(2, point3green);
        trianglegreen.setFillColor((Color(4,155,75)));
        window.draw(trianglegreen);
    }
};
class Dice
{
private:
    int dotSize;
    CircleShape dotShape;
    int diceX;
    int diceY;
    int diceSize;
    int lastDiceNumber;
public:
      Dice(){
            this->dotSize = 7;
            this->diceX = 1420;
            this->diceY = 450;
            this->diceSize = 85;
            this->lastDiceNumber = 1;
            dotShape = CircleShape(dotSize);
      }

    void drawDotShape(int diceNumber, sf::RenderWindow &window)
    {
      dotShape.setFillColor(sf::Color::Black);
        switch (diceNumber)
        {
        case 1:
            dotShape.setPosition(diceX - dotSize, diceY - dotSize);
            window.draw(dotShape);
            break;
        case 2:

            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            break;
        case 3:

            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition(diceX - dotSize, diceY - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            break;
        case 4:

            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            break;
        case 5:

            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition(diceX - dotSize, diceY - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            break;
        case 6:

            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY -
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX - diceSize / 4) - dotSize, diceY -
dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, diceY -
dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX - diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            dotShape.setPosition((diceX + diceSize / 4) - dotSize, (diceY +
diceSize / 4) - dotSize);
            window.draw(dotShape);
            break;
        }
    }
    void drawTempDice(int lastDiceNumber,sf::RenderWindow& window){
      sf::Texture diceTexture;
if (!diceTexture.loadFromFile("DICE SHAPE.png")) {
    cout<<"not loaded";
}
    sf::Sprite diceSprite(diceTexture);
diceSprite.setOrigin(diceTexture.getSize().x / 2, diceTexture.getSize().y / 2);
diceSprite.setPosition(diceX, diceY);
diceSprite.setScale(static_cast<float>(diceSize) / diceTexture.getSize().x,
                    static_cast<float>(diceSize) / diceTexture.getSize().y);
                     window.draw(diceSprite);
                     drawDotShape(lastDiceNumber,window);
      }
   void drawDice(sf::RenderWindow& window, Player* player, LudoBoard board, int
playerCount, Text playerTurn,Text rollDice,int flag)
{
    srand(static_cast<unsigned int>(std::time(NULL)));

    window.clear();
    int rollDuration = 5000;
    bool diceRolling = false;
    Clock clock;
    sf::Texture diceTexture;
if (!diceTexture.loadFromFile("DICE SHAPE.png")) {
    cout<<"not loaded";
}
    sf::Sprite diceSprite(diceTexture);
diceSprite.setOrigin(diceTexture.getSize().x / 2, diceTexture.getSize().y / 2);
diceSprite.setPosition(diceX, diceY);
diceSprite.setScale(static_cast<float>(diceSize) / diceTexture.getSize().x,
                    static_cast<float>(diceSize) / diceTexture.getSize().y);
                     window.draw(diceSprite);
                     drawDotShape(lastDiceNumber,window);
                     board.drawLudoBoard(window);
      for(int i= 0;i<playerCount;i++){
            LudoPiece *piece = player[i].getPiece();
                  for(int j = 0;j<4;j++){
                        piece[j].drawLudoPiece(window,player[i].getColor());
                  }
            }
            if(flag==0){
            window.draw(playerTurn);
      }
      else{
            window.draw(rollDice);
      }
                     window.display();
        Time elapsedTime;
        bool diceRolled = false;

        while(!diceRolled){

            Event event;
            label:
                while (window.pollEvent(event))
        {
            if (event.type == Event::Closed)
            {
                window.close();
            }
            else if (event.type == Event::MouseButtonPressed &&
event.mouseButton.button == Mouse::Left && !diceRolling)
            {
                sf::Vector2i mouseClickPos = sf::Mouse::getPosition(window);
                int mouseClickX = mouseClickPos.x;
                int mouseClickY = mouseClickPos.y;
                if (mouseClickX >= diceX - (diceSize / 2) && mouseClickX <= diceX +
(diceSize / 2) && mouseClickY >= diceY - (diceSize / 2) && mouseClickY <= diceY +
(diceSize / 2))
                {
                    diceRolling = true;
                    elapsedTime = clock.restart();
                    break;
                }
                else{
                  cout<<"\nClick ON the dice!\n";
                  goto label;
                        }
            }
        }
        while (diceRolling)
                {

                    window.clear();
                    board.drawLudoBoard(window);
      for(int i= 0;i<playerCount;i++){
            LudoPiece *piece = player[i].getPiece();
                  for(int j = 0;j<4;j++){
                        piece[j].drawLudoPiece(window,player[i].getColor());
                  }
            }

                    window.draw(diceSprite);

                    int diceNumber = (rand() % 6) + 1;
                    drawDotShape(diceNumber, window);

                    window.display();
                    rollDuration -= elapsedTime.asSeconds();

                    if (rollDuration <= 0)
                    {
                    lastDiceNumber = diceNumber;
                        diceRolling = false;
                        diceRolled = true;

                    }
            }
}
}
    int getLastDiceNumber(){
      return lastDiceNumber;
      }
};

class LudoGame
{
private:
    Player *player;
    LudoBoard board;
    Dice dice;
    Coordinates coord;
    int playerCount;
    vector <int> storeDice;
public:
    void setPlayerCount(int playerCount)
    {
      storeDice.resize(3);
        this->playerCount = playerCount;
        player = new Player[playerCount];
        cout << sizeof(player) * playerCount << endl;
    }

    Player *getPlayer()
    {
        return player;
    }

    int getPlayerCount()
    {
        return playerCount;
    }
    int validChoice(LudoPiece *piece, int clickedX, int clickedY)
    {
        for (int i = 0; i < 4; i++)
        {
            if (piece[i].getX() == clickedX && piece[i].getY() == clickedY)
            {
                return i;
            }
        }
        return -1;
    }
    bool isWinner(int currentPlayer){
      LudoPiece *piece = player[currentPlayer].getPiece();
            for(int i = 0;i<4;i++){
```

```
            if(piece[i].getX()==player[currentPlayer].path[57][0] &&
piece[i].getY()==player[currentPlayer].path[57][1]){
                  continue;
                  }
                  else{
                        return false;
                  }
            }
            return true;
      }
      bool isHome(int currentPlayer){
            LudoPiece *piece = player[currentPlayer].getPiece();
            int count = 0;
            for(int i = 0;i<4;i++){
                  for(int j = 0;j<4;j++){
                        if(piece[i].getX()==player[currentPlayer].homePosition[j]
[0] && piece[i].getY()==player[currentPlayer].homePosition[j][1]){
                              count++;
                              break;
                        }
                  }
            }
            if(count==4){
                  return true;
            }
            return false;
      }
      int getHome(int knockedPlayer){
            LudoPiece *piece = player[knockedPlayer].getPiece();
            for(int i = 0;i<4;i++){
                  for(int j = 0;j<4;j++){
                  if(piece[i].getX()!=player[knockedPlayer].homePosition[j][0] &&
piece[i].getX()!=player[knockedPlayer].homePosition[j][1]){
                        return j;
                  }
            }
            }
      }
      bool isOccupied(LudoPiece *actualPiece,int index,int currentPlayer){

            int stopPosition[8][2]={{650,140},{530,200},{890,440},{950,560},
{650,740},{530,860},{290.560},{230,440}};
            int count = 0,storedJ;
            for(int i = 0;i<getPlayerCount();i++){
                  if(i==currentPlayer){
                        continue;
                  }
                  LudoPiece *piece = player[i].getPiece();
                  for(int j = 0;j<4;j++){
                        if(piece[j].getX()==actualPiece[index].getX() &&
piece[j].getY()==actualPiece[index].getY()){
                              storedJ = j;
                              count++;
                        }
                  }
                  if(count==1){

                        for(int k=0;k<8;k++){
                              if(piece[storedJ].getX()==stopPosition[k][0] &&
piece[storedJ].getY()==stopPosition[k][1]){
                                    return false;
                              }
                        }


                        int homePos = getHome(i);

      piece[storedJ].setPiece(player[i].homePosition[homePos]
[0],player[i].homePosition[homePos][1]);
                              return true;
                        }
                        count = 0;
            }
            return false;
      }
      int isPieceHome(int currentPlayer){
            LudoPiece *piece = player[currentPlayer].getPiece();
            for(int j = 0;j<4;j++){
            for(int i=0;i<53;i++){
                  if(piece[j].getX()==player[currentPlayer].path[i][0] &&
piece[j].getY()==player[currentPlayer].path[i][1]){
                        return j;
                  }
            }
            return -1;
      }
      }
      bool movePiece(int index,int currentPlayer,int diceNumber){
            LudoPiece *piece = player[currentPlayer].getPiece();
            int flag = 0;
            for(int i = 0;i<4;i++){
                  if(piece[index].getX()==player[currentPlayer].homePosition[i][0]
&& piece[index].getY()==player[currentPlayer].homePosition[i][1] && diceNumber==6){
                        piece[index].setPiece(player[currentPlayer].path[0]
[0],player[currentPlayer].path[0][1]);
                        flag = 1;
                        return false;
                  }
            }
            for(int k = 0;k<58;k++){
                  if(isPieceHome(currentPlayer)!=-1){
                        index = isPieceHome(currentPlayer);
                        if(piece[index].getX()==player[currentPlayer].path[k][0] &&
piece[index].getY()==player[currentPlayer].path[k][1]){

      piece[index].setPiece(player[currentPlayer].path[k+diceNumber]
[0],player[currentPlayer].path[k+diceNumber][1]);
                        flag = 1;
                        return false;
                  }
                  }
            }
            if(flag==0){
            for(int i = 0;i<58;i++){
                  if(piece[index].getX()==player[currentPlayer].path[i][0] &&
piece[index].getY()==player[currentPlayer].path[i][1]){
                  if(i+diceNumber<=57){
```

```
      piece[index].setPiece(player[currentPlayer].path[i+diceNumber]
[0],player[currentPlayer].path[i+diceNumber][1]);
                        if(isOccupied(piece,index,currentPlayer)){
                              player->setStatus(true);
                        return true;
                  }
                  else{
                        return false;
                  }
                  }
                  else{
                        return false;
                  }

                  }
            }
      }
      }
      void drawBoardandPieces(sf::RenderWindow &window){
            board.drawLudoBoard(window);
            for (int i = 0; i < getPlayerCount(); i++)
            {
                LudoPiece *currentPiece = player[i].getPiece();
                for (int j = 0; j < 4; j++)
                {
                    currentPiece[j].drawLudoPiece(window, player[i].getColor());
                }
            }
      }
      void youRolledADice(sf::RenderWindow &window,Text diceNumber){
            stringstream dn;
            dn << dice.getLastDiceNumber();
std::string diceNumberString = "You Rolled A " + dn.str();
diceNumber.setString(diceNumberString);
        window.draw(diceNumber);
            cout<<"DICE NUMBER : "<<dice.getLastDiceNumber();
            dice.drawTempDice(dice.getLastDiceNumber(),window);
      }

      int drawText(sf::RenderWindow &window,Font pt){
    Text t1("Enter Move : ",pt,60);
    t1.setFillColor(Color::White);
    t1.setPosition(1100,720);
    sf::Text text;
    text.setFont(pt);
    text.setCharacterSize(60);
    text.setPosition(1500,720);
    text.setFillColor(sf::Color::White);
    int inputInt = 0,flag = 0;
    string inputText;
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed){
                window.close();
            }
            if (event.type == sf::Event::KeyPressed && event.key.code ==
sf::Keyboard::Return){
            stringstream e(inputText);
            e>>inputInt;
            window.clear();

        drawBoardandPieces(window);
        window.display();
            return inputInt;
            }
            if (event.type == sf::Event::TextEntered)
            {
                if (event.text.unicode < 128)
                {
                    if (event.text.unicode == '\b' && !inputText.empty())
                    {
                        // Handle backspace, remove the last character
                        inputText.erase(inputText.size() - 1);
                    }
                    else if (event.text.unicode != '\b')
                    {
                        // Append the entered character to the input text
                        inputText += static_cast<char>(event.text.unicode);
                    }
                }
            }
        }

        window.clear();

        drawBoardandPieces(window);

            // If conversion is successful, convert it back to a string and set the
string to the text
            text.setString(inputText);

        // Draw the text on the window
        window.draw(t1);
        window.draw(text);

        window.display();

    }
}

    void startGame(sf::RenderWindow &window)
    {
      vector<int> result;
        int count = 0,num=0, flag = 0;
        board.drawLudoBoard(window);

        for (int i = 0; i < getPlayerCount(); i++)
        {
            cout << getPlayerCount() << endl;
            LudoPiece *piece = player[i].getPiece();
            for (int j = 0; j < 4; j++)
            {
                piece[j].setPiece(player[i].homePosition[j][0],
player[i].homePosition[j][1]);
                piece[j].drawLudoPiece(window, player[i].getColor());
            }
        }
        Font pt;
        pt.loadFromFile("COD.ttf");
        Text playerTurn("",pt,110),diceNumber("",pt,110),rollDice("Roll The Dice 
Again!",pt,80);
        diceNumber.setPosition(1100,130);
        rollDice.setPosition(1080,140);
        playerTurn.setFillColor(player[count].getColor());
        playerTurn.setPosition(1100,130);
        std::stringstream ss;
ss << count + 1;
std::string playerTurnString = "Player " + ss.str() + " Turn";
playerTurn.setString(playerTurnString);
        window.draw(playerTurn);
        int index = 0;
        window.display();
        int numberX = 1300,rectX = 1270,numClicked = 0;
        while (count < playerCount)
        {
            if(isWinner(count)){
                  cout<<"Game Over!"<<"Player "<<count+1<<" Won";
                  window.clear();
                  window.close();
                  break;
                  }
                  LudoPiece *piece = player[count].getPiece();
            playerTurn.setFillColor(player[count].getColor());
            std::stringstream ss;
ss << count + 1;
std::string playerTurnString = "Player " + ss.str() + " Turn";
playerTurn.setString(playerTurnString);


            dice:
            window.clear();
            window.draw(playerTurn);
            sleep(milliseconds(500));

dice.drawDice(window,player,board,playerCount,playerTurn,rollDice,flag);
            result.push_back(dice.getLastDiceNumber());
            if(dice.getLastDiceNumber()==6){
                  diceNumber.setFillColor(player[count].getColor());
                  youRolledADice(window,diceNumber);
                  window.display();
                  sleep(milliseconds(100));
                  flag = 1;
                  goto dice;
                  }
                  flag = 0;
                  if(result.size() >= 3) {
        int last = result.size()-1;
        if(result[last] == 6 && result[last-1] == 6 && result[last-2] == 6) {
            count++;
                  if(count==getPlayerCount()){
                        count=0;
                  }
                  result.clear();
                  sleep(seconds(1));
                  continue;
        }
    }
            board.drawLudoBoard(window);
            for(int i= 0;i<getPlayerCount();i++){
            LudoPiece *piece = player[i].getPiece();
                  for(int j = 0;j<4;j++){
                        piece[j].drawLudoPiece(window,player[i].getColor());
                  }
            }
            diceNumber.setFillColor(player[count].getColor());
            youRolledADice(window,diceNumber);
            if(result.size()>1){
            Text text[result.size()];
            RectangleShape rectDice[result.size()];
            for(int i = 0;i<result.size();i++){
                  stringstream tt;
                  tt << result[i];
                  string pst = tt.str();
                  text[i] = Text(pst,pt,70);
                  rectDice[i] = RectangleShape((Vector2f(100.0f,100.0f)));
                  text[i].setFillColor(Color::White);
                  rectDice[i].setFillColor(Color::Black);
                  rectDice[i].setOutlineColor(Color::White);
                  rectDice[i].setOutlineThickness(2);
                  text[i].setPosition(numberX,530);
                  rectDice[i].setPosition(rectX,520);
                  window.draw(rectDice[i]);
                  window.draw(text[i]);
                  numberX+=110;
                  rectX+=110;
            }
      }
            window.display();
            numberX = 1300;
            rectX = 1270;

            if(isHome(count)&&result.size()==1){
                  count++;
                  if(count==getPlayerCount()){
                        count=0;
                  }
                  result.clear();
                  sleep(seconds(1));
                  continue;
            }

            for(int i = 0;i<result.size();i++){
label:
            coord.getcord(window);
            index = validChoice(piece, coord.getmidX(), coord.getmidY());
            if (index == -1)
            {
                cout << "\n\nInvalid Click\nClick On your own piece\n";
                goto label;
            }
            if(result.size()>1){
                  num = drawText(window,pt);
                  for(int k = 0;k<result.size();k++){
```

```
              if(result[k]==num){
                    if(movePiece(index,count,num)){
                          window.clear();
        board.drawLudoBoard(window);
        for (int i = 0; i < getPlayerCount(); i++)
        {
            LudoPiece *currentPiece = player[i].getPiece();
            for (int j = 0; j < 4; j++)
            {
                currentPiece[j].drawLudoPiece(window, player[i].getColor());
            }
        }
        window.display();
  result.erase(result.begin()+k);
        goto dice;
                          }
                    }
                    else{
                          cout<<endl<<"invalid number";
                          goto label;
                    }
                    result.erase(result.begin()+k);
                    break;
              }
}
    else{
        num = dice.getLastDiceNumber();
        for(int i = 0;i<result.size();i++){
              if(num==result[i]){
                    result.erase(result.begin()+i);
                    break;
                    }
              }
              if(movePiece(index,count,num)){
   window.clear();
        board.drawLudoBoard(window);
        for (int i = 0; i < getPlayerCount(); i++)
        {
            LudoPiece *currentPiece = player[i].getPiece();
            for (int j = 0; j < 4; j++)
            {
                currentPiece[j].drawLudoPiece(window, player[i].getColor());
            }
        }
        window.display();
  goto dice;
  }
        }

window.clear();
        board.drawLudoBoard(window);
        for (int i = 0; i < getPlayerCount(); i++)
        {
            LudoPiece *currentPiece = player[i].getPiece();
            for (int j = 0; j < 4; j++)
            {
                currentPiece[j].drawLudoPiece(window, player[i].getColor());
            }
        }
```

```
            window.display();
                              if(!result.empty()){
                                    goto label;
                              }
}
            count++;
            result.clear();
                if (count == getPlayerCount())
            {
                count = 0;
            }
            sleep(milliseconds(10));
        }
}
    int gameLoadingScreen(sf::RenderWindow &window){
    window.clear();
    window.display();

    sf::Font font;
    if (!font.loadFromFile("JUMPMAN.ttf"))
    {
        // Handle font loading error
        return EXIT_FAILURE;
    }

    sf::Text text("Game Starting", font, 50);
    sf::FloatRect textBounds = text.getLocalBounds();
    text.setFillColor(sf::Color::Yellow);
    text.setOrigin(textBounds.left + textBounds.width / 2, textBounds.top +
textBounds.height / 2);
    text.setPosition(window.getSize().x / 2, window.getSize().y / 2);

    sf::CircleShape circle(200);
    circle.setOrigin(circle.getRadius(), circle.getRadius());
    circle.setPosition(window.getSize().x / 2, window.getSize().y / 2);
    circle.setFillColor(sf::Color(0, 0, 128));
    circle.setOutlineColor(sf::Color::Black);
    circle.setOutlineThickness(2);

    bool dotsVisible = false;
    bool countdownStarted = false;
    std::string dotsString = "";
    int animationCycles = 0;
    int countdownValue = 3;

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        // Clear the window
        window.clear();

        if (animationCycles < 3) // Show initial animation
        {
            // Draw the circle and text
            window.draw(circle);
            window.draw(text);

            if (!dotsVisible)
            {
                // Show the dots one by one
                dotsVisible = true;
                for (int i = 0; i < 3; i++)
                {
                    dotsString += ".";
                    text.setString("Game Starting" + dotsString);

                    // Draw the updated text
                    window.draw(circle);
                    window.draw(text);
                    window.display();

                    // Delay between each dot
                    sf::sleep(sf::seconds(0.4f));
                }
            }

            // Check if animation cycle is complete
            if (dotsVisible && dotsString == "...")
            {
                dotsVisible = false;
                dotsString = "";
                animationCycles++;
            }
        }
        else if (!countdownStarted) // Show countdown animation
        {
            text.setString(intToString(countdownValue));
            text.setCharacterSize(100);
            sf::FloatRect countdownBounds = text.getLocalBounds();
            text.setOrigin(countdownBounds.left + countdownBounds.width / 2,
countdownBounds.top + countdownBounds.height / 2);
            countdownStarted = true;
        }
        else if (countdownValue > 0)
        {
            // Draw the circle and text
            window.draw(circle);
            window.draw(text);

            countdownValue--;
            text.setString(intToString(countdownValue));
            window.display();
            sf::sleep(sf::seconds(1.0f));
        }
        else
        {
            // Countdown animation complete, do something else or exit the loop
            break;
        }

        // Display the window
        window.display();
```

```
    }

    return 0;
}

std::string intToString(int value)
{
    std::stringstream ss;
    ss << value;
    return ss.str();
}
   int GameMenu(sf::RenderWindow &window)
    {
        const int MENU_ITEMS = 3;
        window.clear(sf::Color::Blue);

         sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("menu_select.wav"))
    {
        // Error loading the sound file
        return 1;
    }

    sf::Sound sound;
    sound.setBuffer(buffer);
        sf::Texture texture;
        if (!texture.loadFromFile("LUDO LOADING.png"))
            return -1;

        sf::Sprite sprite(texture);
        int windowWidth = 1840;
        int windowHeight = 1080;
        int imageWidth = 362;
        int imageHeight = 312;

        // Calculate the position
        int spritePosX = (windowWidth - imageWidth) / 2;
        int spritePosY = (windowHeight - imageHeight) / 2;

        // Set the position
        sprite.setPosition(spritePosX, spritePosY - 70);

        // Calculate the scale
        float scaleX = static_cast<float>(imageWidth) / texture.getSize().x;
        float scaleY = static_cast<float>(imageHeight) / texture.getSize().y;

        // Set the scale
        sprite.setScale(scaleX, scaleY);

        sf::Texture backgroundTexture;
        if (!backgroundTexture.loadFromFile("MAIN2.jpg"))
            return -1;

        sf::Sprite backgroundSprite(backgroundTexture);
        backgroundSprite.setScale(window.getSize().x /
static_cast<float>(backgroundTexture.getSize().x),
                                  window.getSize().y /
static_cast<float>(backgroundTexture.getSize().y));
```

```
        sf::RectangleShape loadingBar;
        loadingBar.setPosition(735, 655);
        loadingBar.setFillColor(sf::Color(255, 215, 0));
        sf::Font font;
        if (!font.loadFromFile("JUMPMAN.ttf"))
            return -1;

        sf::Font arrowFont;
        if (!arrowFont.loadFromFile("ariblk.ttf"))
            return -1;

        std::string menuItems[MENU_ITEMS] = {
            "Multiple Players",
            "About",
            "Exit Game"};

        sf::Text menuText[MENU_ITEMS];
        sf::Font heading;
        heading.loadFromFile("Samurai.ttf");

        sf::Text titleText("LUDO GAME", heading, 130);
        titleText.setFillColor(sf::Color::White);
        titleText.setPosition((window.getSize().x -
titleText.getLocalBounds().width) / 2.0f, 155);

        sf::Text arrowLeft("<", arrowFont, 70);
        arrowLeft.setFillColor(sf::Color(255, 223, 0));

        sf::Text arrowRight(">", arrowFont, 70);
        arrowRight.setFillColor(sf::Color(255, 223, 0));

        int selectedMenuIndex = 0;
        bool loadingComplete = false;
        int playerCount = 0;
        int selectedOption = 0;
        while (window.isOpen())
        {
            sf::Event event;
            while (window.pollEvent(event))
            {
                if (event.type == sf::Event::Closed)
                    window.close();

                if (event.type == sf::Event::KeyPressed)
                {
                    if (event.key.code == sf::Keyboard::Up && loadingComplete)
                    {
                        selectedMenuIndex = (selectedMenuIndex - 1 + MENU_ITEMS) %
MENU_ITEMS;
                        sound.play();
                    }
                    else if (event.key.code == sf::Keyboard::Down &&
loadingComplete)
                    {
                       sound.play();
                        selectedMenuIndex = (selectedMenuIndex + 1) % MENU_ITEMS;
                    }
                    else if (event.key.code == sf::Keyboard::Return &&
loadingComplete)
```

```
                    {
                        // Handle menu item selection
                        switch (selectedMenuIndex)
                        {
                        case 0:
                        {

                            std::cout << "Selected menu item: Multiplayer\n";
                            // multiplayerSelected = true;

                            // Clear the window
                            window.clear();
                            sf::Text multiText("CHOOSE NUMBER OF PLAYERS", heading,
110);
                            multiText.setFillColor(sf::Color::White);
                            multiText.setStyle(sf::Text::Bold);

                            // Calculate the position to center the text
horizontally and place it at the top
                            float xPos = (window.getSize().x -
multiText.getGlobalBounds().width) / 2;
                            float yPos = 40;
                            cout << endl
                                 << multiText.getGlobalBounds().width;

                            multiText.setPosition(xPos, yPos);
                            int j = 380, k = 30;
                            RectangleShape rectangle[3];

                            Text rectText[3];
                            rectText[0] = Text("2 Player", heading, 52);
                            rectText[1] = Text("3 Player", heading, 52);
                            rectText[2] = Text("4 Player", heading, 52);
                            for (int i = 0; i < 3; i++)
                            {
                                rectText[i].setPosition(j + k, 340);
                                rectangle[i] = RectangleShape(sf::Vector2f(300.0f,
200.0f));

                                rectangle[i].setPosition(j, 280);
                                rectangle[i].setFillColor(Color(107, 107, 107));
                                rectangle[i].setOutlineColor(Color::White);
                                rectangle[i].setOutlineThickness(2);
                                j += 400;
                                window.draw(rectangle[i]);
                                window.draw(rectText[i]);
                            }
                            window.draw(multiText);
                            window.display();

                            // ...

                            while (window.isOpen())
                            {
                                sf::Event event;
                                while (window.pollEvent(event))
                                {
                                    if (event.type == sf::Event::Closed)
                                        window.close();
                                    if (event.type == sf::Event::KeyPressed)
                                    {
                                        if (event.key.code == sf::Keyboard::Left &&
loadingComplete)
                                        {
                                            // Move selection to the left
                                            selectedOption = (selectedOption - 1 +
3) % 3;
                                        }
                                        else if (event.key.code ==
sf::Keyboard::Right && loadingComplete)
                                        {
                                            // Move selection to the right
                                            selectedOption = (selectedOption + 1) %
3;
                                        }
                                        else if (event.key.code ==
sf::Keyboard::Return && loadingComplete)
                                        {
                                            // Enter key is pressed, handle the
selected option
                                            switch (selectedOption)
                                            {
                                            case 0:

                                                playerCount = 2;
                                                std::cout << "Selected option: 2
Players\n";

                                                setPlayerCount(2);
                                                for (int i = 0; i < playerCount; i+
+)
                                                {
                                                    if (i == 0)
                                                    {

player[i].setPath(yellowHome, yellowPath);

player[i].setColor(Color(240, 225, 48));
                                                    }
                                                    if (i == 1)
                                                    {
                                                        player[i].setPath(blueHome,
bluePath);

player[i].setColor((Color::Blue));
                                                    }
                                                }
                                                //window.clear();
                                                //window.draw(rectangle[0]);
                                                //window.draw(rectangle[1]);
                                                //window.display();
                                                //sleep(seconds(100));
                                                gameLoadingScreen(window);
                                                window.clear();
                                                startGame(window);
                                                sleep(seconds(100));
                                                break;
                                            case 1:
                                                playerCount = 3;
                                                std::cout << "Selected option: 3
Players\n";

                                                setPlayerCount(3);
                                                for (int i = 0; i < playerCount; i+
+)
                                                {
                                                    if (i == 0)
                                                    {

player[i].setPath(yellowHome, yellowPath);

player[i].setColor(Color(240, 225, 48));
                                                    }
                                                    if (i == 1)
                                                    {
                                                        player[i].setPath(blueHome,
bluePath);

player[i].setColor((Color::Blue));
                                                    }
                                                    if (i == 2)
                                                    {
                                                        player[i].setPath(redHome,
redPath);

player[i].setColor((Color::Red));
                                                    }
                                                }
                                                gameLoadingScreen(window);
                                                window.clear();
                                                startGame(window);
                                                sleep(seconds(100));
                                                break;
                                            case 2:

                                                playerCount = 4;
                                                std::cout << "Selected option: 4
Players\n";

                                                setPlayerCount(4);
                                                for (int i = 0; i < playerCount; i+
+)
                                                {
                                                    if (i == 0)
                                                    {

player[i].setPath(yellowHome, yellowPath);

player[i].setColor(Color(240, 225, 48));
                                                    }
                                                    if (i == 1)
                                                    {
                                                        player[i].setPath(blueHome,
bluePath);

player[i].setColor((Color::Blue));
```

```
                                                    }
                                                    if (i == 2)
                                                    {
                                                        player[i].setPath(redHome, 
redPath);

player[i].setColor((Color::Red));
                                                    }
                                                    if (i == 3)
                                                    {

player[i].setPath(greenHome, greenPath);

player[i].setColor((Color(26, 163, 109)));
                                                    }
                                                }
                                                gameLoadingScreen(window);
                                                window.clear();
                                                startGame(window);
                                                sleep(seconds(100));
                                                break;
                                            }
                                        }
                                    }
                                }

                                // ...
                                window.clear();
                                for (int i = 0; i < 3; i++)
                                {
                                    // ...

                                    if (i == selectedOption)
                                    {
                                        // Change appearance of the selected option

rectangle[i].setOutlineColor(sf::Color::Red);
                                        rectangle[i].setFillColor(sf::Color::Red);
                                        rectText[i].setFillColor(sf::Color::White);
                                    }
                                    else
                                    {
                                        // Reset appearance of other options

rectangle[i].setOutlineColor(sf::Color::White);

rectangle[i].setFillColor(sf::Color::Black);
                                        rectText[i].setFillColor(sf::Color::White);
                                    }

                                    // Draw the option
                                    window.draw(rectangle[i]);
                                    window.draw(rectText[i]);

                                    // ...
                                }
                                window.draw(multiText);
                                // Display the changes
                                window.display();
```

```
                            }
                            break;
                        }

                        case 1:
                            std::cout << "Selected menu item: About\n";
                            std::cout << "Hello World\n";

                            window.clear();
                            if (true)
                            {
                                Text about("", font, 24);
                                about.setFillColor(Color::White);
                                about.setString("Experience the classic board game
of Ludo in a new digital form");
                                about.setPosition(window.getSize().x / 4,
window.getSize().y / 2);
                                window.draw(about);
                                window.display();
                                sleep(seconds(1000));
                            }
                            break;
                        case 2:
                            std::cout << "Selected menu item: Exit Game\n";
                            std::cout << "Hello World\n";
                            window.clear();
                            window.display();
                            exit(1);
                            break;
                        }
                    }
                }
            }

            window.clear();
            // window.display();

            if (loadingComplete)
        {
            window.draw(backgroundSprite);
            window.draw(titleText);

            for (int i = 0; i < MENU_ITEMS; i++)
            {
                menuText[i] = sf::Text(menuItems[i], font, 60);
                menuText[i].setFillColor(sf::Color(54, 54, 54));
                menuText[i].setPosition((window.getSize().x -
menuText[i].getLocalBounds().width) / 2.0f,
                                        (window.getSize().y -
menuText[i].getLocalBounds().height) / (MENU_ITEMS + 3) * (i + 3) + 20);

                if (i == selectedMenuIndex)
                    menuText[i].setFillColor(sf::Color::White);

                window.draw(menuText[i]);
            }

            arrowLeft.setPosition(menuText[selectedMenuIndex].getPosition().x - 60,
menuText[selectedMenuIndex].getPosition().y - 10);
```

```
            arrowRight.setPosition(menuText[selectedMenuIndex].getPosition().x +
menuText[selectedMenuIndex].getLocalBounds().width + 20,
menuText[selectedMenuIndex].getPosition().y - 10);

            window.draw(arrowLeft);
            window.draw(arrowRight);
        }
            else
            {
                int i = 0;
                while (i < 330)
                {
                    loadingBar.setSize(sf::Vector2f(50 + i, 30));

                    window.draw(sprite);
                    window.draw(loadingBar);
                    window.display();

                    sf::sleep(sf::milliseconds(70));
                    i += 10;
                }

                loadingComplete = true;
            }

            window.display();
        }

        return 0;
    }
};

int main()
{
    LudoGame game;
    sf::RenderWindow window(sf::VideoMode(1840, 1080), "Ludo Game OOP Project");
    game.GameMenu(window);
    sleep(milliseconds(100000));
}
```