Ludo Game

COURSE:
OOP (CT-260)

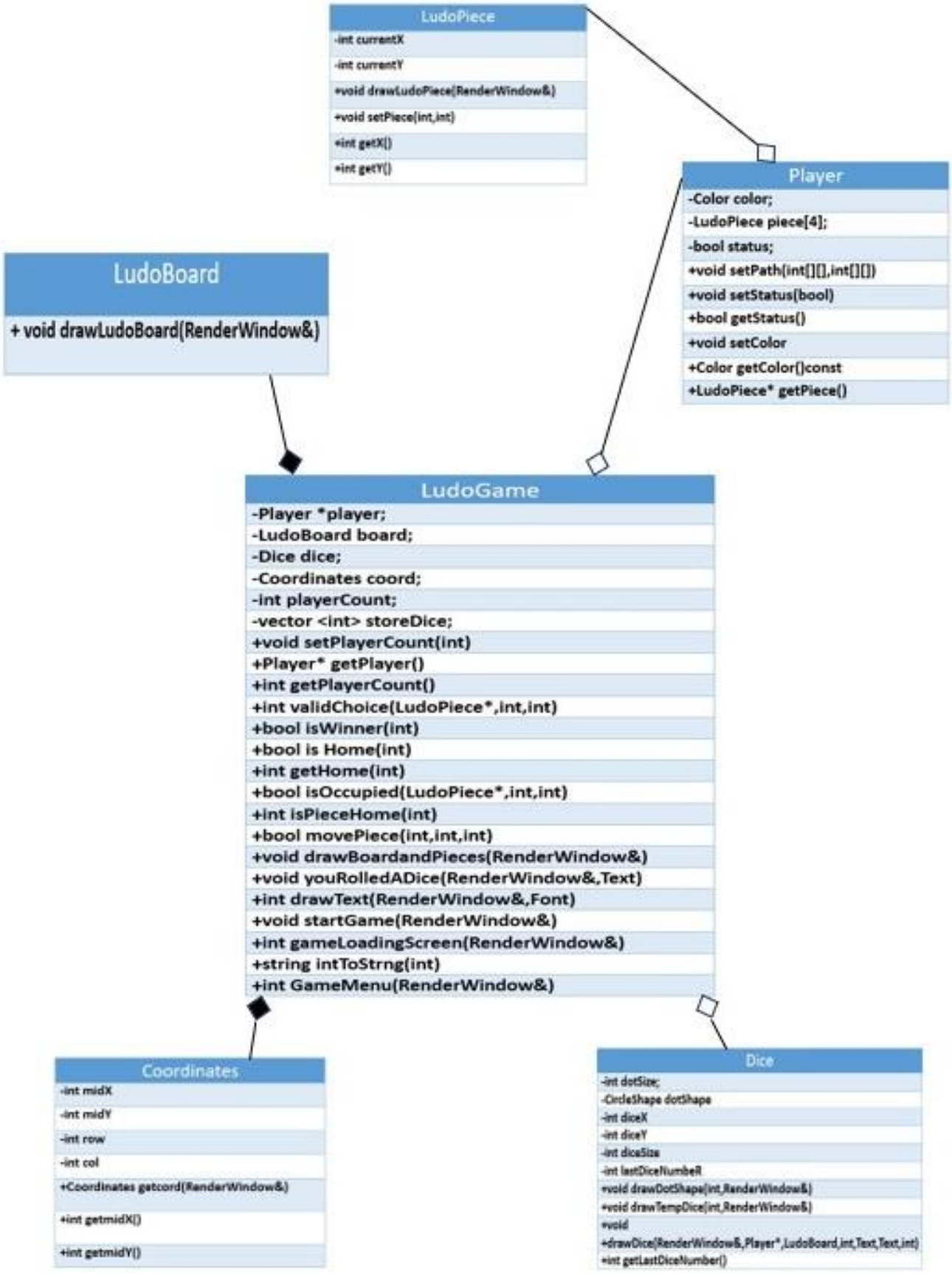
INTRODUCTION

The following report provides an overview of the Ludo game developed using the C++ programming language with an emphasis on object-oriented programming (OOP) principles. The purpose of this report is to explain the design and implementation of the game, highlighting key features and discussing the benefits of using OOP concepts.

GAME OVERVIEW

Ludo is a popular board game that can be played by two to four players. The objective of the game is to move all four tokens from the starting point to the player's home area by rolling a dice and strategically selecting which token to move. The game involves both luck, as the movement of tokens depends on the dice roll, and strategy, as players can send the opponent’s tokens back to the starting point.

UML DIAGRAM


LudoPiece
-int currentX
-int currentY
+void drawLudoPiece(RenderWindow&)
+void setPiece(int,int)
+int getX()
+int getY()
Player
-Color color;
-LudoPiece piece[4];
-bool status;
+void setPath(int[][],int[][])
+void setStatus(bool)
+bool getStatus()
+void setColor
+Color getColor()const
+LudoPiece* getPiece()
LudoBoard
+ void drawLudoBoard(RenderWindow&)
LudoGame
-Player *player;
-LudoBoard board;
-Dice dice;
-Coordinates coord;
-int playerCount;
-vector <int> storeDice;
+void setPlayerCount(int)
+Player* getPlayer()
+int getPlayerCount()
+int validChoice(LudoPiece*,int,int)
+bool isWinner(int)
+bool is Home(int)
+int getHome(int)
+bool isOccupied(LudoPiece*,int,int)
+int isPieceHome(int)
+bool movePiece(int,int,int)
+void drawBoardandPieces(RenderWindow&)
+void youRolledADice(RenderWindow&,Text)
+int drawText(RenderWindow&,Font)
+void startGame(RenderWindow&)
+int gameLoadingScreen(RenderWindow&)
+string intToStrng(int)
+int GameMenu(RenderWindow&)
Coordinates
-int midX
-int midY
-int row
-int col
+Coordinates getcord(RenderWindow&)
+int getmidX()
+int getmidY()
Dice
-int dotSize;
-CircleShape dotShape
-int diceX
-int diceY
-int diceSize
-int lastDiceNumbeR
+void drawDotShape(int,RenderWindow&)
+void drawTempDice(int,RenderWindow&)
+void
+drawDice(RenderWindow&,Player*,LudoBoard,int,Text,Text,int)
+int getLastDiceNumber()

OBJECT-ORIENTED DESIGN

The Ludo game was developed using object-oriented programming principles, which provide a modular and structured approach to software development. The primary classes used in the implementation of the game include:

1. PLAYER:

This class represents a player in the game and maintains information such as the player's name, home area, and current position of tokens.

2. TOKENS:

The Token class represents an individual token on the game board. It contains attributes such as the current position, owner, and methods to move the token.

3. DICE:

The Dice class simulates the rolling of a dice. It generates a random number between 1 and 6, which determines the movement of tokens.

4. BOARD:

The Board class represents the game board and manages the positions of tokens. It provides methods to move tokens, check for valid moves, and handle collisions between tokens.

5. GAME:

The Game class serves as the main controller for the Ludo game. It creates instances of the Player, Token, Dice, and Board classes and manages the flow of the game, including turns, dice rolls, and win conditions.

KEY FEATURES:

The implemented Ludo game offers several key features:

1. MULTIPLE PLAYERS:

The game supports two to four players, allowing friends or family members to play together.

2. DICE ROLLING:

The game simulates the rolling of a dice using the Dice class. Each player takes turns rolling the dice to determine the number of spaces their token can move.

3. TOKEN MOVEMENT:

Players can select a token to move based on the dice roll. The movement is handled by the Token class, which updates the token's position on the game board.

4. COLLISION DETECTION:

The Board class checks for collisions between tokens. If a token lands on a space occupied by another token, the latter is sent back to the starting point.

5. WIN CONDITION:

The game checks for a win condition where a player successfully moves all four tokens to their home area. The first player to achieve this wins the game.

BENEFITS OF OBJECT-ORIENTED PROGRAMMING

The use of object-oriented programming in developing the Ludo game provides several advantages, including:

1. MODULARITY:

The game's components are encapsulated within separate classes, promoting code reusability and maintainability.

2. ABSTRACTION:

The classes abstract away the complexity of individual components, allowing developers to focus on the high-level interactions between objects.

3. CODE ORGANIZATION:

OOP promotes a structured approach to software development, making the codebase easier to navigate and understand.

4. SCALABILITY:

The modular nature of OOP allows for easy expansion and the addition of new features to the game without affecting existing code.

5. TEAM COLLABORATION:

OOP facilitates collaborative development, as multiple developers can work on different classes simultaneously without interfering with each other's code.

OOP Principles Used In The Project:

1. **Abstraction**: Abstraction is the method of hiding the complex implementation details and only showing the essential features to the user. In your Ludo game, the player doesn't need to know the inner working details of the 'Dice' class. The player just knows when he rolls the dice, he will get a number between 1 to 6. The underlying process of how the dice generates these numbers is abstracted away from the player.
2. **Encapsulation**: This involves bundling data (attributes) and methods (functions) that work on the data into a single unit, i.e., class. Each 'Player'

is an instance of the 'Player' class which may have attributes like 'name', 'color', and 'ludoPieces'. The 'ludoPieces' attribute might be an array of 'LudoPiece' objects. Each 'Player' object would also have methods like 'rollDice' and 'movePiece' that manipulate these attributes in defined ways. This is encapsulation.

3. **Association**: This is a relationship between two classes that allows one object instance to cause another to perform an action on its behalf. In your game, association is present in the relationship between 'Player' and 'Dice', 'Player' and 'LudoPiece' classes. A player can roll a dice and move a piece, but the actual rolling and moving is performed by the 'Dice' and 'LudoPiece' objects.

Contribution of Each Group Member:

In the development of the Ludo game project, each member of the team had significant contributions, playing unique roles that catered to their strengths and areas of expertise. Each member’s distinct input was pivotal in shaping the final product, from conceptualization to the final implementation.

Firstly, **Group Member (DT-042)** significantly contributed by overseeing the implementation of the game mechanics. This involved a deep understanding of Ludo's rules and game flow, translating this into a logical structure that a computer program could follow. This member designed and developed the algorithms for gameplay, dictating how turns progressed, the movements of pieces based on dice rolls, and the win-loss conditions. Their work served as the backbone of the project, ensuring that the core of the Ludo game was faithfully represented in our digital version.

Group Member(DT-015) brought the game to life visually by overseeing the implementation of the SFML library, an important task that allowed the team to create a graphical interface for the game. The responsibility entailed creating windows, accepting user input, and rendering game states onto the screen. This member's work was critical in making the game interactive and user-friendly, elevating the gaming experience from a simple console application to a fully featured graphical game.

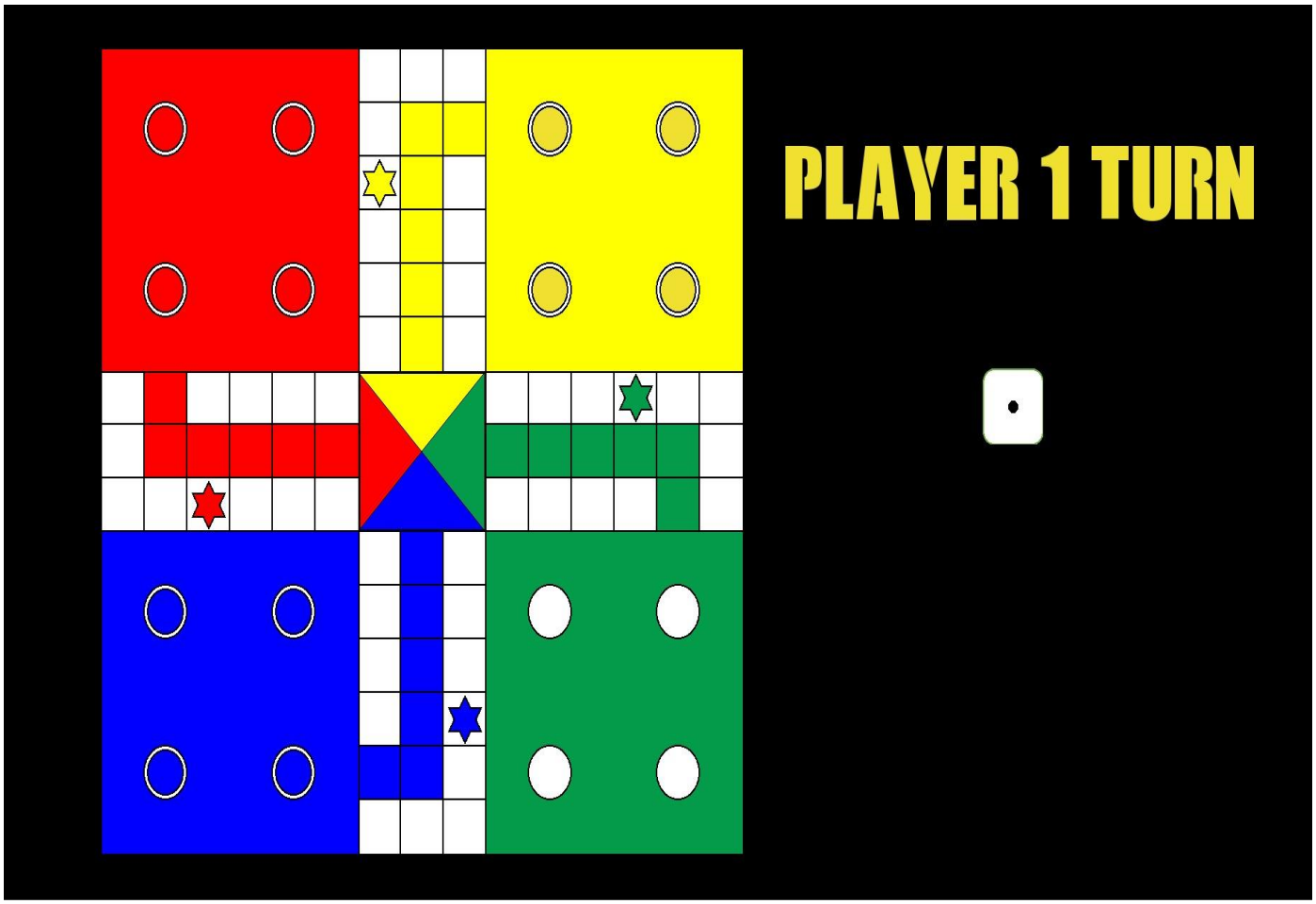
Finally, **Group Member(DT-049)** played a supportive and yet essential role in the project, aiding in the

implementation of the game mechanics. They worked closely with the first member, assisting in fine-tuning the game's rules and behaviors. Their collaborative efforts resulted in a more robust and reliable game, helping ensure that the digital representation of the game was as accurate and engaging as possible. This member was vital in providing an additional perspective and expertise, ensuring the project's success by fortifying the foundation upon which the game was built.

Each team member's contribution was valuable and indispensable, leading to the successful completion of the project. Their collective effort, cooperation, and dedication shaped the Ludo game project into an engaging and enjoyable digital experience.

ScreenShots:


CHOOSE NUMBER OF PLAYERS
2 PLAYER
3 PLAYER
4 PLAYER


PLAYER 1 TURN

CONCLUSION

The development of the Ludo game using object-oriented programming principles in C++ has resulted in a modular and scalable implementation. The game offers key features such as multiple players, dice rolling, token movement, collision detection, and win conditions. OOP has provided benefits such as

modularity, code organization, and scalability, making the codebase maintainable and facilitating team collaboration. Overall, the Ludo game serves as an excellent example of applying OOP concepts to create an enjoyable and interactive gaming experience.

GROUP MEMBERS:

Muhammad Baziq Khan (DT-015)

Muhammad Insha Khan (DT-042)

Syed Ahab Ali (DT-049)