TRIBHUWAN UNIVERSITY

INSTITIUE OF ENGINEERING

PULCHOWK CAMPUS

A PROJECT REPORT ON

LUDO

USING OBJECT ORIENTED PROGRAMMING WITH C++

SUBMITTED BY:

Bishal Lamichhane(078BCT035)

Bipin Bashyal(078BCT033)

Ayush K.C.(078BCT025)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

PULCHOWK CAMPUS

ACKNOWLEDGEMENT

First and foremost, we extend our heartfelt thanks to our lecturer Daya Sagar Baral, for his unwavering encouragement, insightful feedback, expert guidance, and constant support.

We are indebted to the authors of the various textbooks, online tutorials, and programming forums that served as invaluable resources during the research and implementation stages. Their dedication to sharing knowledge is truly commendable and has been instrumental in expanding my programming skills.

We would like to thank the Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering for providing us opportunity of collaborative undertaking which has helped us to implement the knowledge and provide us an experience in teamwork along with OOP.

Finally, we would like to express our deep appreciation to our fellow classmates and friends who provided us with their perspectives and insights during the brainstorming and development phases of this project. Their constructive criticism and collaborative spirit helped shape the project into its current form.

Authors:

Bishal Lamichhane

Bipin Bashyal

Ayush K.C.

ABSTRACT

The main aim of this project was to develop a game program using Object Oriented Programming in C++. For this project, we made a classic two to four player Ludo game using Simple and Fast Multimedia Library (SFML) for graphical interface. The goal of creating this game is also to learn about game development and to be familiar with OOP. Ludo is a strategic, four-player board game that originated in India. There are 4 pawns of each player which they move with the number resulting in dice roll until they all reach home. There are various rules in meantime of path movement.

The Ludo game developed in this project serves as a practical example of how OOP principles can be effectively applied to create a complex software system. By utilizing inheritance, encapsulation, and polymorphism, the codebase achieves a high level of organization and extensibility, making it straightforward to introduce new features or modify existing ones.

Keywords: Ludo, SFML, OOP, C++

TABLE OF Contents

1.OBJECTIVES

This project does not aim to advance our Game development skills within a matter of days or weeks. Instead, the major objective of this project is to encourage us to implement an object-oriented approach to think, model and develop basic algorithms into program code. The main objectives of this project can be summed as:

- To understand Object Oriented Programming paradigm and build a project using it. C++ is preferred over other languages like Python because C++ offers better efficiency and speed and is suitable for almost every platform including embedded systems whereas Python can be used only on certain platforms that support high-level languages.
- To explore the basic attributes of C++ programming languages.
- To be familiar with software development library designed to provide a simple Application Programming Interface to various multimedia components in computers using SFML library.
- To develop effective and efficient programs by optimizing time and space constraints.
- To learn the fundamental concepts about game development.
- To acquire teamwork and communication skills as a result of working as a team.

2. INTRODUCTION

The Ludo game, a classic and widely recognized board game, serves as an excellent platform to explore and apply various programming concepts in a practical context. The game is renowned for its blend of strategy and chance, where players navigate their tokens across a colorful game board with the goal of reaching the central "home" area. The game's mechanics involve rolling a dice to determine token movement, capturing opponent tokens, and progressing towards victory.

In this lab project, our aim was to implement a digital version of the Ludo game using the C++ programming language and Object-Oriented Programming (OOP) principles. By creating this digital adaptation, we sought to gain insights into the practical application of programming paradigms and concepts that are crucial in software development. The project allowed us to explore various aspects of OOP, such as class design, encapsulation, inheritance, and polymorphism, in the context of a real-world problem.

Our implementation covered essential game mechanics, including token movement, dice rolling, player interactions, and rule enforcement. The digital Ludo game not only replicates the traditional gameplay experience but also leverages the advantages of digital technology, such as automated rule checking and a graphical user interface (GUI) for user interaction. This project served as an opportunity to enhance our programming skills, algorithmic thinking, and software design strategies.

In the subsequent sections of this lab report, we will delve into the project's design choices, implementation details, challenges encountered, and the role of OOP principles in shaping the outcome. Through this exploration, we aim to showcase our ability to apply theoretical knowledge to practical scenarios, as well as to reflect on the significance of OOP in developing efficient, organized, and maintainable software systems.

3.APPLICATION

Ludo is a classic board game that combines luck and strategy. It is played by two to four players, each with a set of colored tokens. The game's objective is to move all of one's tokens from their starting area around the board to the central "home" area.

The Ludo game finds practical applications in various contexts, including:

1. **Educational Learning:** Ludo can be employed as an engaging tool to teach children concepts like counting, turn-taking, and basic math skills in a playful manner.
2. **Cognitive Development:** Playing Ludo enhances critical thinking, decision-making, and planning abilities, fostering cognitive development in individuals of all ages.
3. **Social Bonding:** The multiplayer nature of Ludo promotes social interaction, making it an excellent activity for building relationships and connections.
4. **Probability Understanding:** The dice-rolling aspect introduces probability concepts, aiding in understanding randomness and basic statistics.
5. **Team Building:** Ludo can be utilized in team-building exercises to encourage collaboration, communication, and teamwork among participants.
6. **Algorithmic Thinking:** Developing a digital Ludo game involves implementing algorithms for game mechanics, honing algorithmic thinking skills.
7. **Stress Relief:** Engaging in a game of Ludo can serve as a stress-relieving activity, providing a temporary escape from daily pressures.
8. **Entertainment Industry:** Ludo remains a popular choice for mobile apps, online platforms, and digital gaming, offering recreational enjoyment to a wide audience.

In summary, the Ludo game's versatility allows it to be more than just a leisure activity. It contributes to education, skill development, social interaction, and even therapeutic settings, making it a valuable tool with applications across diverse domains.

4. LITERATURE REVIEW

4.1. C++

C++ is a general-purpose programming language that was developed by Danish computer scientist Bjarne Stroustrup as an extension of the C programming language. It is an intermediate level language, as it comprises confirmation of both high level and low-level language features.

It was originally created as a successor to C with more features, hence the name" C++". It is an Object-Oriented Programming language but is not purely Object Oriented. It is also sometimes referred to as" C with classes" due to its Object-Oriented features and machine level control.

4.1.1. Structure of program

A general C++ program includes the following parts:

1. Standard Libraries Section
2. Class Definition Section
3. Functions Definition Section
4. Main Function Section

Standard Libraries Section: This section is written at the top of the program and consists of the first lines of code read by the compiler. This is also known as the preprocessor section. This section is where all the header files and namespaces are declared in the program such as iostream and std. This includes standard libraries as well as user-defined header files and programs.

Class Definition section: The classes are used to map real world entities into programming. The classes are key building blocks of any object-oriented program. A C++ program may include several class definitions.

Functions Definition Section: Functions are the features of Procedure Oriented Programming but are very useful in OOP as well. In our program there are many functions used to perform various activities as required.

Main Function Section: The main function is the main body of the program which the compiler executes first after all preprocessing. The main function is where all other functions of the program are called from. In C++, we write the main function such as an int return type such that it returns 0 if the program is successfully executed and 1 if there was an error in the execution.

4.2. SFML

Simple and Fast Multimedia Library (SFML) is a cross-platform software development library designed to provide a simple application programming interface (API) to various multimedia components in computers. It is written in C++ with bindings available for C++, Crystal, Java, Julia, .NET, Python, Ruby, and Rust.

SFML handles creating and input to window and provides a graphics module for simple hardware acceleration of 2D computer graphics which includes text rendering and audio rendering. SFML consists of various modules such as:

1. System – vector and string classes, portable threading, and timer facilities

2. Window – window and input device management

3. Graphics ‑ hardware acceleration of 2D graphics including sprites, polygons, and text rendering.

4. Audio ‑ hardware-accelerated spatialized audio playback and recording

5. Network ‑ TCP and UDP network sockets, data encapsulation facilities, HTTP and FTP classes.

4.3. Theoretical Background on Object Oriented Programming, C++

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a compiled language.

C++ is a middle-level language rendering the advantage of programming low-level (drivers, kernels) and even higher-level application (games, GUI, desktop apps etc.). The basic syntax and code structure of both C and C++ are the same.

Applications of C++:

Operating Systems Programming. e.g., Linux-based OS (Ubuntu etc.)
Browsers (Chrome Firefox)
Graphics Game engines (Photoshop, Blender, Unreal-Engine)
Database Engines (MySQL, MongoDB, Redis etc.)
Cloud/Distributed Systems

Simple and Fast Multimedia Library (SFML) is a cross-platform software development library designed to provide a simple application programming interface (API) to various multimedia components in computers. SFML provides a simple interface to the various components of the PC, to ease the development of games and multimedia applications. It is composed of five modules: system, window, graphics, audio, and network. With SFML, the application can be compiled and run out of the box on the most common operating systems: Windows, Linux, MacOS and soon Android iOS. SFML has official bindings for the C++ language. And thanks to its active community, it is also available in many other languages such as Java, Ruby, Python, Go, and more.

4.4. OBJECT ORIENTED FEATURES

The primary objective of the project was to learn object-oriented programming concepts by practically implementing them. So, we have tried our best to implement the features of object-oriented programming in our project.

4.5. Objects and Classes

As a basic building block of OOP, it is common to include these concepts in our program. Since game elements can be treated as objects, the classes Players and Pieces are made along with many other small classes like coordinate which in turn are useful in small tasks. Pieces are instantiated into various objects array that act as pieces in the board. Each piece is itself an object of their respective class such that each piece all 4 of them have a single player with total up to maximum number of players.

4.6. Abstraction

Abstraction is a feature of object-oriented programming that hides the internal details of how object does its work and only provides the interface to use the service. We can manage complexity through abstraction. In OOP, classes are used for creating user-defined data for abstraction. When data and its operation are presented together, it is called ADT (Abstract Data Type). Hence, a class is an implementation of abstract data type. So, in OOP, classes are used in creating Abstract Data Type. For instance, we have created a class coordinate and made it available in the program. Now we can implement the class in creating objects and its manipulation without knowing its implementation. We have also used the class implementation of SFML available to us to create desired graphics images. While we know how to create and manipulate these objects, the exact details are hidden from us.

4.7. Encapsulation And Data Hiding

Combining data and function together into a single unit is called encapsulation. We can say encapsulation is a protective box that prevents the data from being accessed by other code that is defined outside the box. We can easily achieve abstraction by making use of encapsulation. We can achieve encapsulation through classes. In classes, each data or function is kept under an access specifier (private, protected, or public).

1. **Public:** It contains data and functions that the external users of the class may know about.

2. **Private:** This section can only be accessed by code that is a member of a class.

3. **Protected:** This section is visible to class members, derived class members, friend functions of class, friend classes and friend classes of derived classes.

The insulation of data from direct access by the program is called data hiding. Data hiding by making them private or protected makes it safe from accidental alteration. Understanding this, we have tried to make our data private as much as possible.

4.8. Inheritance

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that facilitates the creation of a hierarchical relationship between classes. In C++, it allows a new class (the derived class) to inherit properties and behaviors from an existing class (the base class). This promotes code reuse and a structured codebase, as the derived class can access and extend the features of the base class. Inheritance establishes an "is-a" relationship, where the derived class is a specialized version of the base class. This concept enables developers to efficiently model complex systems by building upon existing components, ultimately contributing to modular design, enhanced readability, and reduced redundancy in C++ programs.

4.9. Polymorphism

Polymorphism is another important feature of OOP. It allows different objects to respond to the same operation in different ways. The different ways of using the same function or operator depending on what they are operating on is called polymorphism. In C++, polymorphism is mainly divided into two types:

1. Compile time polymorphism
2. Run time polymorphism.

Pure Virtual Functions: Pure virtual functions are functions that only has a declaration but doesn't have a definition. Since they have no definition, these functions cannot be called and objects consisting of pure virtual function cannot be created. Its usefulness comes from the fact that any class that derives from a base class consisting of a pure virtual function must implement the function for the derived class.

Abstract Base Class: A class that has a pure virtual function is an Abstract Base Class. These classes cannot be used to instantiate an object but serve the following function.

1. Deter from creation of the base class.
2. Act as a base class from any derived classes and allow for easy polymorphism

5.EXISTING SYSTEM

A Ludo game system has been developed as a software application. This system embodies the principles of OOP by representing game elements as objects with associated properties (attributes) and behaviors (methods). The Ludo game, a classic board game, is structured using classes such as "Player," "Board," "Dice," and "Game," each encapsulating relevant functionality. The "Player" class manages individual player attributes, the "Board" class oversees the game board layout and position of tokens, the "Dice" class simulates dice rolling, and the "Game" class orchestrates the overall gameplay, facilitating interactions between players and the board. By adopting OOP concepts like encapsulation, inheritance, and polymorphism, this Ludo game system promotes modularity, reusability, and maintainability in its codebase.

Ludo games like this are already developed for mobile and pc applications with their own change in few rules of game. We created a classic original ludo game to play on the computer.

6.METHODOLOGY

To complete our destined project, we aimed to follow the given methods:

6.1. Initiating and planning

We initialized by planning and dividing the work among 3 of our members. We familiarized ourselves with the required library (SFML). Besides, we refreshed our knowledge and game logic of the game itself for the rules needed for Ludo.

6.2. Algorithm Design

After getting the required rules and information, we designed the algorithm and flowchart of the game project. A basic working model algorithm was designed for further tests, validation, and continuity.

6.3. Software Design

With the basic set of algorithms in our hand, our focus then shifted to writing the code in Object Oriented paradigm. The project was carried out on C++ as its main skeletal and SFML as its graphics implementation since it was easy to learn and use. As for IDE and compiler, we opted for Visual Studio and Visual Studio Code as our IDE and Ming-W g++ as compiler in Windows operating system and GCC (GNU Compiler Collection) C++ Compiler(g++) in Linux.

6.4. Testing and debugging

We first developed Minimum Viable Product (MVP) sample of project for testing and debugging. Further testing and debugging were done to add features and edit the project code as per our need and capability.

7.IMPLEMENTATION

Implementing a complete Ludo game project in Object-Oriented Programming (OOP) using C++ involves several components and classes. Here's an outline of how we structured the implementation:

1. **Class Definitions:**
    - **Coordinates:** It represents the pixel x and y coordinate of the position of the pieces to be set in window. There are various functions in this class to operate with coordinates.
    - **Dice:** It represents the class containing dice and all its faces along with its different functions and variables to roll and draw dice.
    - **Players:** It represents a class with has most of the functions and variables in our program. Each player has 4 pieces and many attributes like color, name, etc.
    - **Pieces:** It represents the pawn of each player which is to be moved along with its different functions.

   Along with these major classes there are many other minor classes for various work in our program.

2. **Class Implementation:**

   These classes are used at various part of program to perform specific different functions as required in program each classes are declared and defined above main so that we can use them in main.
3. **Main Function:**

   In the main function the loop to display window every time the piece is moved or display the board and pieces is run along with all the many functions to run game properly in required order. All interactions and logic are called here.
4. **Error handling:**

   Ensure that moves are valid, according to game rules and board layout. Check for winning conditions after each turn.

7.1. System Block Diagram

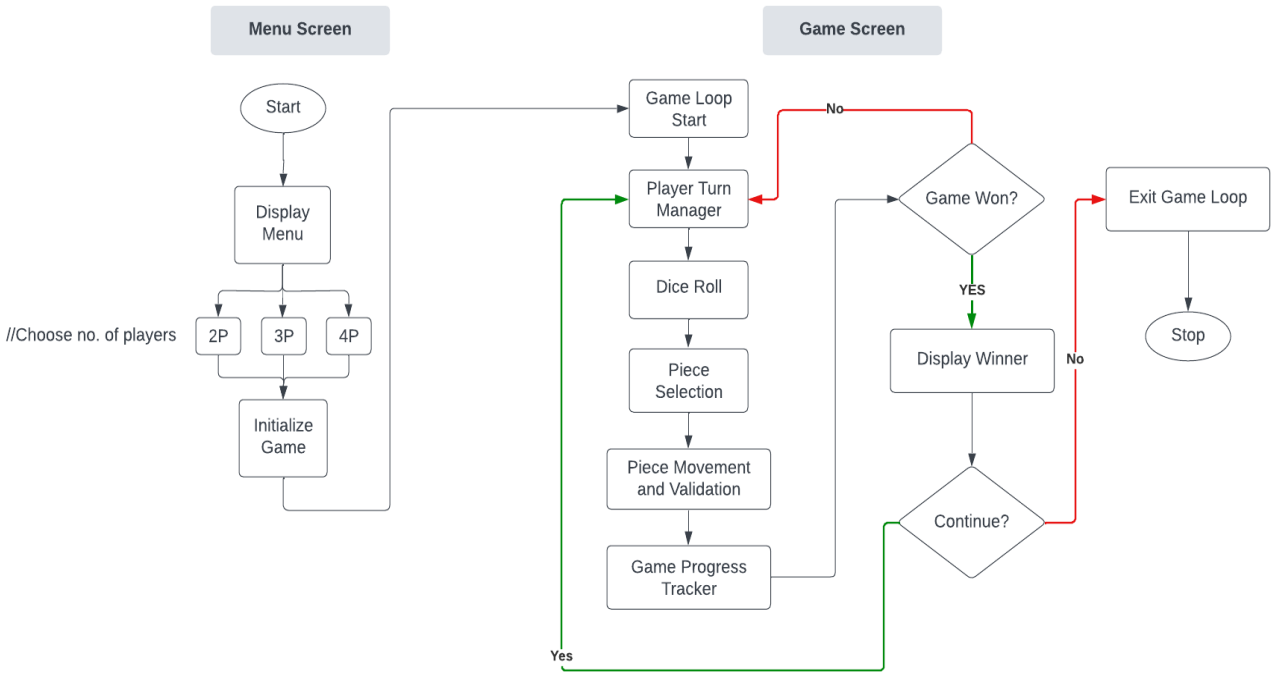


Figure: Block Diagram

In the above block diagram when a game is opened first a menu screen is displayed to choose human player. Then another screen appears to choose between the number of players from 2 to 4. Now the game board is loaded with the initial position of pieces of each player and dice ready to roll. The game starts with rolling of dice and then game runs according to rule i.e., if 1 comes in dice then player can pick piece out of base and game continues as usual turn by turn until all players make all their pieces home. When one player reaches home then a screen appears to choose between continuing the game or quitting then game continues as your choice. The game loop is arranged in such a pattern that that every condition of game winning killing other pieces are checked at right time to make program smooth and efficient. Finally, game ends with displaying winners piece.

8.RESULTS

The Ludo game project developed using Object-Oriented Programming (OOP) principles in C++ has been successfully completed and executed. The objective of this project was to create a functional and interactive digital version of the classic Ludo board game while adhering to the core tenets of OOP.

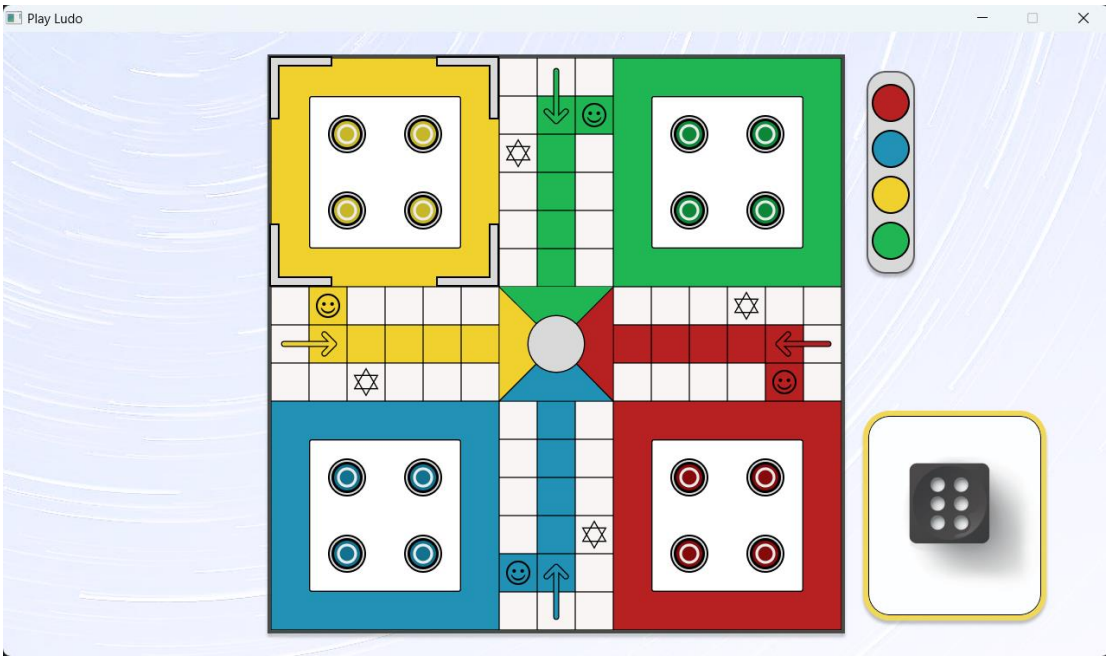
During the execution of the Ludo game, players took turns rolling the dice and moving their tokens according to the dice's outcome. Tokens were placed on the designated starting positions and navigated through the board's pathways. The implementation ensured that players could only make valid moves based on the dice roll and game rules. Token collisions, safe zones, and winning conditions were accurately handled by the program, creating a realistic and engaging gameplay experience.

Through multiple tests runs and interactions with the Ludo game system, several observations were made the following are few screenshots of different screens of game illustrating final product of project:

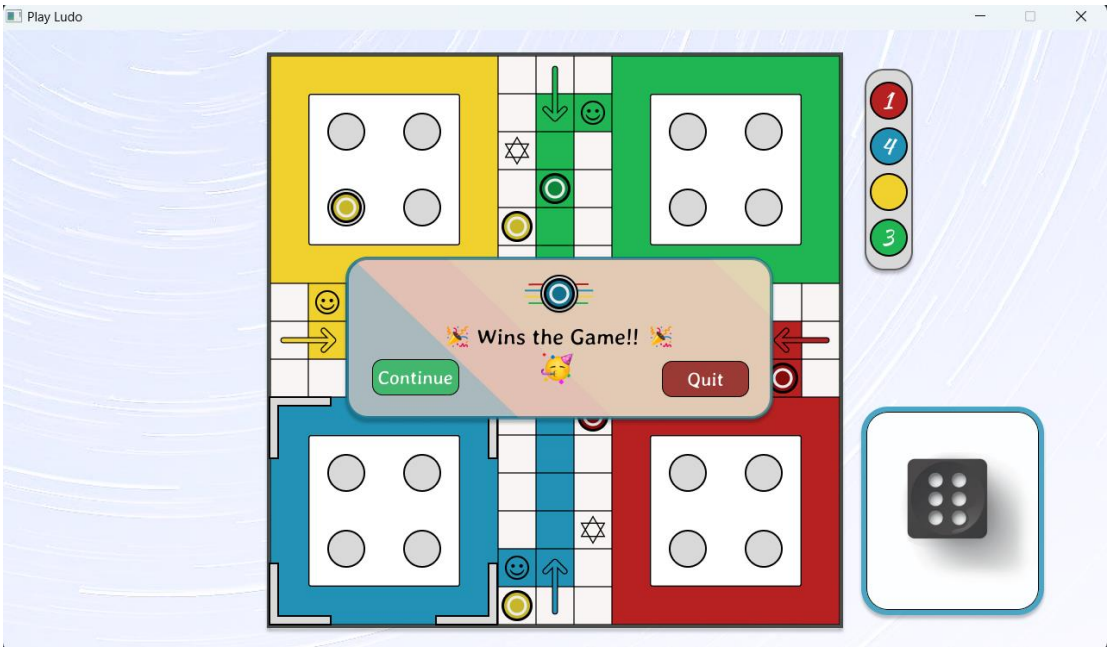
8.1 Main Menu

8.2 Second Menu

8.3 Game Board



8.4 Winners Screen



9.PROBLEMS FACED AND SOLUTIONS

We faced several problems over the course of creating a Ludo game. Here are some c problems that we faced during the process, along with possible solutions:

1. **Designing the Class Hierarchy:** Defining the appropriate classes and their relationships was complex. Deciding what attributes and methods each class should have, and how they interact, is crucial. To tackle this, we started with a clear understanding of the game's mechanics and break down its components into distinct classes with well-defined responsibilities.

2. **Token Movement and Collision Detection:** Implementing token movement on the board while ensuring that tokens don't overlap incorrectly or collide with one another requires careful logic. Keep track of the positions of all tokens on the board and validate each move to prevent illegal or overlapping moves.

3. **Mouse Clicking the token:** While we already created game implementing rules to play in console entering which piece to move it was difficult to create a logic which moves the piece which was clicked by mouse it was very difficult because we used SFML which was limited, and the proper use of available functions was very tricky finally with hours of time we did it.

4. **Turn-Based Gameplay:** Managing the sequence of turns for players, ensuring that each player gets a chance, and switching turns correctly was intricate. Utilize loops and conditional statements to manage the flow of turns and consider implementing a turn counter or a flag to track the current player along with turn skipping for player whose all tokens are already home.

5. **Winning Conditions:** Detecting when a player has won the game involves verifying that their tokens have reached the designated final positions. Implementing the logic to check for winning conditions after each move or turn and handled the game's conclusion accordingly.
6. **Error Handling and Validation:** Invalid moves, incorrect input, or unexpected behavior was to be handled gracefully. Implement error checks and validation mechanisms to ensure that the game doesn't crash or produce incorrect results due to unforeseen inputs or scenarios.

7. **User Interface (UI):** While the focus might be on the backend logic, providing a clear and user-friendly interface can enhance the overall experience. However, creating a graphical user interface (GUI) was an additional challenge.

8. **Testing and Debugging:** As with any software project, testing and debugging was crucial. Bugs related to logic, edge cases, or class interactions was challenging to identify and fix. Use debugging tools, print statements, and testing different scenarios to catch and address issues.

9. **Code Maintainability:** As the project grows, managing code structure and avoiding code duplication became more critical. We applied good software design principles to keep your codebase maintainable and modular.

10. **Learning Curve:** We were new to C++ or OOP concepts, there was a learning curve associated with understanding language features, syntax, and OOP

principles. Taking advantage of online resources, tutorials, and documentation enhanced our understanding.

10.Limitations and Future Enhancements

Due to time limitation, and various examination schedules, we were unable to add more additional features which might have enhanced our program. There are many more features that can still be added to make the program more attractive, result oriented and useful. We also tried to add AI of ludo but due to unsolved error appearance we couldn't complete that. Some of the limitation include:

1. No online multiplayer via networking.

2. Only local human players mode available.

4. Lack of proper total game settings.

5. Dynamic screen size to make the program work in multiple screen sizes.

The possible future enhancements are as follows.

1. A multiplayer mode, in which two people from any place can play with each other via networking.

2. An overhaul in the way we store the state of the game with the help of stack to track moves. This would allow for move take backs and will make it so that we don't need to copy the state of the board again and again to make changes.

3. A better AI for single player, implementing Alpha-Beta pruning alongside Minimax algorithm.

4. Board variation or newer board design along with some rule change according to players choice.

11.CONCLUSION AND RECOMMENDATIONS

The creation of the Ludo game using the Simple and Fast Multimedia Library (SFML) in C++, employing Object-Oriented Programming (OOP) principles, has resulted in a fully functional and visually engaging digital rendition of the classic board game. This project aimed to combine the power of OOP and the graphical capabilities of SFML to create an interactive and enjoyable gaming experience.

The successful completion of the project has brought to light several achievements and observations:

1. **SFML Integration:** The integration of SFML provided a platform for rendering graphics, animations, and user interactions. The graphical representation of the game board, tokens, dice, and user interface elements significantly enhanced the gaming experience.

2. **OOP Application:** The project successfully demonstrated the utilization of OOP principles in software design. The modular organization of classes—such as **Player**, **Pieces**, **Dice**, and **Coordinate**—facilitated code readability, reusability, and maintenance. Inheritance, encapsulation, and polymorphism were effectively employed to streamline class interactions.

3. **Visual Appeal:** The incorporation of SFML allowed for visually appealing graphics, including vibrant token colors, smooth animations, and a user-friendly interface. This visual aspect is crucial in capturing the essence of the original Ludo game.

4. **User Experience:** The inclusion of graphical elements and user prompts improved the overall user experience, making the game more engaging and interactive. This aligns with modern gaming expectations and enhances player enjoyment.

Recommendations

While the project has achieved its primary objectives, there are areas that could be further improved for an even more polished final product:

1. **User Interface Refinement:** Enhance the user interface with more intuitive buttons, indicators, and clearer feedback during gameplay. This will help guide players through their turns and interactions with the game.

2. **Animations and Effects:** Implement additional animations and effects, such as token movement animations, dice rolling animations, and winning celebrations. These visual cues can add depth to the gaming experience.

3. **AI Opponents:** Incorporating computer-controlled opponents would make the game enjoyable for single players. Design AI strategies that mimic human decisions and offer different levels of difficulty.

4. **Multiplayer Support:** Extend the game to support local or networked multiplayer modes. This would involve managing player turns, token movements, and communication between players' devices.

5. **Sound Effects and Music:** Adding sound effects for dice rolls, token movements, and game events can contribute to the game's immersion. Background music could further enhance the ambiance.

6. **Error Handling:** Implement robust error handling mechanisms to gracefully manage unexpected scenarios, input errors, or glitches without causing crashes.

In conclusion, the Ludo game project utilizing SFML in C++ with a focus on OOP principles has been a valuable learning experience. It highlighted the synergy between OOP's modular design and SFML's graphical capabilities, resulting in a functional and enjoyable game. By embracing the project's achievements and acting on the recommendations, we can refine the game to provide a polished and captivating gaming experience for players of all ages.

12.REFRENCES

12.1. Book References:

“The Secrets of Object-oriented Programming”, Daya Sagar Baral, Diwakar Baral

“The C++ Programming Language”, Bjarne Stroustrup

“C++ How To Program”, Paul Deitel, Harvey Deitel

“Object Oriented Programming with C++”, Robert Lafore

“SFML Game Development by Example”, Raimondas Pupius

12.2. Web References:

https://www.sfml-dev.org/

https://devdocs.io/cpp/

https://docs.microsoft.com/en-us/cpp/