TRADERS

EARN RENT SPENT

MAJOR PROJECT

On

GAME DESIGN AND DEVELOPMENT

SUBBMITTED IN THE PARTIAL FULFILMENT OF THE REQUIREMENT OF DEGREE

OF

MASTERS OF COMPUTER APPLICATION

(4th Semester, 2023)

SUBBMITTED BY

Aman Rawat
MCA (Final year)

SUBBMITTED TO

Ms. Sugandhi Vij
Project Supervisor

AVVIARE EDUCATIONAL HUB, NOIDA
(GLOCAL UNIVERSITY, SAHARANPUR, UTTAR PRADESH)

CERTIFICATE

This is to certify that the project entitled "Traders" has been successfully completed by Aman Rawat, a student of Avviare Educational Hub, in fulfillment of the requirements for the degree of Master of Computer Application.

The project "Traders" is an authentic work undertaken by Aman Rawat under the supervision and guidance provided by the faculty of Avviare Educational Hub. This project demonstrates his knowledge, skills, and expertise in the field of computer application.

We certify that this project has not been submitted to any other institution or university for the award of a diploma or degree.

Congratulations to Aman Rawat on the successful completion of the project "Traders"!

Date: June 15, 2023

Place: Avviare Educational Hub, Noida, Uttar Pradesh

Signature of Director of Operation	Signature of HOD of IT Department	Signature of Project Supervisor

ABSTRACT

"Traders" is an engaging tabletop board game that immerses 2 to 4 players in the captivating world of real estate. With a roll of the dice, players embark on a journey where they navigate the game "Traders" is an engaging tabletop board game that immerses 2 to 4 players in the captivating board, strategically purchase properties, engage in trades, and outmaneuver their rivals in a thrilling race to financial dominance. The objective of "Traders" is to emerge as the last player standing, avoiding bankruptcy while accumulating wealth and assets. The game incorporates elements of luck, strategy, and negotiation, challenging players to make calculated decisions and employ shrewd tactics to outsmart their opponents. The scope of "Traders" extends beyond traditional board games by offering an expanded game board featuring over 45+ countries and 10+ special tiles. This larger playing area creates a sense of exploration and provides players with fresh and exciting gameplay possibilities. Additionally, the game introduces three unique routes: air, water, and land, allowing players to strategically navigate the board and unlock new strategic opportunities. The versatility of gameplay modes in "Traders" adds to its appeal. Players can engage in multiplayer online matches with random opponents, create and join rooms with friends, or opt for local play by sharing a single device. This flexibility ensures that players can enjoy the game according to their preferences and playstyle. With its visually stunning 2D art style and immersive 3D effect, "Traders" captivates players with its visually striking aesthetics. The skillful use of perspective creates a sense of depth, making players feel as though they are navigating a three-dimensional world. The educational value of "Traders" lies in its ability to teach players about real estate, money management, and decision-making within a business context. By engaging with economic concepts and strategic thinking, players can enhance their knowledge and skills in these areas while enjoying an entertaining gameplay experience. "Traders" also has global appeal, featuring various countries and neighboring regions on the game board. This inclusion promotes cultural awareness and provides relatability to players around the world. The use of Indian currency (rupees) adds a unique touch and further enriches the game's global theme. The game mechanics of "Traders" are carefully balanced to ensure fair gameplay for all players. Luck, strategy, and skill all play crucial roles in determining success, ensuring that every player has an equal opportunity to thrive and emerge victorious. With its accessible user interface, "Traders" caters to players of all ages and skill levels. The intuitive design and easy navigation make it suitable for both newcomers to board games and experienced players, fostering inclusivity and enjoyment for all.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to **Ms. Kanika Singh, Director of Operations**, Avviare Educational Hub, Noida , for providing me with the opportunity to carry out my major project work. Her support and belief in my abilities have been instrumental in the successful completion of this project.

I would also like to extend my sincere appreciation to **Mr. Ashutosh Rathore**, **HOD of the IT Department**, Avviare Educational Hub, Noida, for his guidance and encouragement throughout the project. His expertise and valuable insights have greatly contributed to the development and execution of this work.

Special thanks are also due to **Ms. Sughandhi Vij , Project Supervisor** for her exceptional guidance, provision of resources, and invaluable instructions. Without her continuous support and mentorship, I would not have been able to undertake and progress this project to its current stage.

I would like to acknowledge the contributions of the **other faculty members of the IT Department**, whose expertise and assistance have been instrumental in shaping this project. Their feedback and input have been invaluable in refining my ideas and ensuring the project's success.

I would also like to express my heartfelt thanks to my family members for their unwavering support and motivation throughout this journey. Their belief in my abilities has been a constant source of encouragement.

Lastly, I would like to extend my gratitude to my friend, **Gagandeep Singh Bartwal, former student of B.C.A** , Avviare Educational Hub, for his unwavering support and motivation. I am grateful for his generosity in providing his laptop, which has played a crucial role in the execution of this project.

I am indebted to all these individuals for their unwavering support, guidance, and motivation, without which this project would not have been possible.

Name: Aman Rawat, M.C.A

Thursday, June 15, 2023

List of Figures

Contents

Contents

SYNOPSIS

Introduction

"**Traders**" is an engaging table top **board game** set in a world where **2 to 4 players** delve into the realm of real estate.

With a roll of the dice, players navigate the game board, purchasing properties, engaging in trades, and strategizing their way to success by outsmart their rivals in a **race to financial dominance**.

Players are tasked with **buying properties** in various countries, encountering special tiles, and engaging in a variety of events and interactions. The game continues until players successfully outmaneuver their opponents, avoiding bankruptcy, and emerging as the last player standing—the **ultimate winner.**

The core mechanics of "Traders" revolve around the concept of **earning, renting, and spending**. Players have the opportunity to earn income from their properties, collect rent from opponents who land on their owned spaces, and make strategic expenditures to grow their portfolios.

The game also incorporates chance elements, bonus opportunities, community chest interactions, jail scenarios, club visits, resort visits, and other exciting events that can either boost their financial standing or hinder their progress

Multiple gameplay options are available, including **multiplayer online** with random players, creating and **joining rooms** with friends, and **local play** by sharing a single device.

It combines elements of **luck, strategy, and negotiation**. The game encourages players to develop skills such as critical thinking, financial planning, and resource allocation, making it an educational and entertaining experience.

With its appealing music and charming sound effects. "Traders" delivers an **interactive and enjoyable experience**. Each playthrough offers **endless possibilities**, where players can experience both **financial pitfalls and triumphant comebacks**.

Key Features

1. **Visually Stunning 2D Art Style with Immersive 3D Effect:**
"Traders" boasts a visually striking 2D art style that incorporates an immersive 3D effect. The skillful use of perspective creates a sense of depth and dimension, giving players the feeling of navigating a three-dimensional world.

2. **Expanded Game Board with More Tiles:**
Experience a larger game board than regular board games. "Traders" offers a vast playing area with over 45 countries and 10+ special tiles. These special tiles, along with new interactive elements, enhance the classic Monopoly game experience, providing fresh and exciting gameplay.

3. **Three Routes for Strategic Navigation:**
Stand out with the game's unique feature—three different routes to choose from: air, water, and land. This allows players to strategically traverse the game board, offering diverse paths and unlocking new strategic opportunities.

4. **Versatile Gameplay Modes**:
Enjoy a variety of gameplay modes tailored to your preferences. Challenge AI opponents in bot vs. player mode, form alliances and compete against other teams in team vs. team mode, or test your skills against friends in thrilling player vs. player matches. "Traders" offers engaging options for every type of player.

5. **User-Friendly Interface:**
The game's user interface is designed to be simple, intuitive, and easy to navigate. This ensures that players of all ages and experience levels can easily grasp the mechanics and enjoy a seamless gaming experience.

6. **Balanced and Fair Gameplay:**
"Traders" emphasizes fair gameplay for all players. The game mechanics are carefully balanced to create a level playing field, ensuring that luck, strategy, and skill are all factors in determining success. Every player has an equal opportunity to thrive and emerge victorious.

Software Development Life Cycle Used

Spiral Model

The Spiral model is an iterative software development methodology that combines elements of both waterfall and iterative development approaches. It is a risk-driven model that emphasizes flexibility and allows for continuous refinement and enhancement throughout the development process.

In the context of developing "Traders," the Spiral model would have been utilized to guide the game's creation and ensure its successful implementation. Here are some key details about the Spiral model:

1. **Iterative Approach:** The Spiral model breaks the development process into a series of iterations, or spirals, where each iteration represents a phase of the project. This allows for continuous feedback and improvement throughout the development lifecycle.

2. **Risk Analysis:** The Spiral model places a strong emphasis on risk analysis and management. It involves identifying potential risks early in the process and incorporating risk mitigation strategies into the development plan. This helps to minimize uncertainties and address potential challenges proactively.

3. **Phases:** The Spiral model typically consists of four main phases: Planning, Risk Analysis, Engineering, and Evaluation. Each phase involves specific activities and deliverables, and the process moves through these phases in an iterative manner.

4. **Incremental Development:** The Spiral model supports incremental development, which means that the project is divided into smaller increments or releases. This allows for the delivery of functional components or features at different stages of the development process, providing early value to the end-users.

5. **Flexibility and Adaptability:** The Spiral model allows for flexibility and adaptability throughout the development process. It enables adjustments and refinements based on feedback and changing requirements, ensuring that the final product meets the desired goals and objectives.

6. **Continuous Improvement:** The Spiral model promotes continuous improvement through regular evaluation and feedback loops. This allows for the incorporation of lessons learned from each iteration into subsequent cycles, leading to a more refined and high-quality end product.

By adopting the Spiral model, the development team behind "Traders" would have followed a systematic approach that allowed for risk management, iterative development, and continuous improvement. This would have helped in creating a robust and polished game that meets the requirements and expectations of the players.

Software and Hardware Requirement

Pixel Lab: For creating the game assets, I used utilized Pixel Lab, a free application available on the Play Store for Android. However, alternative software options like Adobe Photoshop or Lightroom can also be used for asset creation. Pixel Lab does not have specific RAM requirements mentioned. **However, to ensure smooth performance, it is recommended to have a device (Android) with a minimum of 2 GB RAM.**

Game Engine: Unity is the chosen game engine for development. To get started, download Unity Hub and install Unity from there. Unity provides all the necessary libraries and tools for developing Android games. You may need to add additional packages through the Unity Editor, such as JDK, IDE, and relevant Game Development kits, which are available in the Unity installer package. **RAM: Unity recommends a minimum of 8 GB RAM. However, depending on the complexity on projects.**

Visual Basics: Visual Basics was used for coding purposes. During installation, you can set it up to be compatible with Unity for game development. RAM: Visual Studio **requires a minimum of 2 GB RAM. However, Microsoft recommends having at least 8 GB RAM or more for better performance**, especially when working on large projects or multiple instances of the IDE.

In summary, the software requirements for developing "Traders" include utilizing Pixel Lab for graphic asset creation, installing Unity through Unity Hub, and configuring Visual Basics to work seamlessly with Unity for coding purposes. **The System Specifications that I used for Game Development:**

Operating System: Microsoft Windows 11 Home

Processor: 11th Gen Intel Core i3-1115G4 @ 3.00 GHz, 2995 MHz, 2 Cores, 4 Logical Processors

RAM: 8 GB

System Name: GAGANBARTWAL

Game Details and Requirements to Play

Genre: Board Game, Strategy

Built-in: Unity game engine

Target Audience: 8+

Platform: Android

Minimum Requirements: 2 GB RAM

Play Time: Up to 180 minutes

Version 1.0 (demo)

"Traders" is a captivating board game that falls under the genres of **board game and strategy**. Developed using the powerful **Unity game engine**, this game is optimized for Android devices and requires a **minimum of 2 GB RAM** to ensure **smooth gameplay**.

With its engaging mechanics and strategic gameplay, "Traders" offers an immersive experience for **players aged 8 and above**. Whether you're a seasoned strategist or new to board games, this title provides a challenging and rewarding gameplay experience.

The game's playtime can range **up to 180 minutes**, offering hours of fun and excitement. Players will find themselves immersed in the world of commerce, making strategic decisions, negotiating trades, and managing their resources to emerge victorious.

"Traders" is designed to provide a visually appealing and intuitive user interface, ensuring a seamless gaming experience even on devices with modest specifications. Its **optimized** performance and engaging gameplay make it an ideal choice for board game enthusiasts and strategy lovers alike.

Embark on a thrilling journey of wealth accumulation, strategic planning, and intense competition in "Traders" on your **Android device.**

Immerse yourself in the visually captivating world of "Traders" where expanded game boards, multiple routes, versatile gameplay modes, and balanced mechanics await. Whether you're a casual player or a seasoned strategist, this game promises an exciting and engaging experience for all.

Property Rate Logic (Research)

The sites I used to find the property rate logic for the project, mention below:

Wikipedia (www.wikipedia.org) is a widely used online encyclopedia that offers detailed information about various topics, including countries' GDP, population, and land size. It provides well-referenced articles written by contributors from around the world. When using information from Wikipedia, it's always a good practice to cross-reference the cited sources and verify the accuracy of the data.

WorldOmeter (www.worldometers.info) is a trusted source for real-time statistics on various global topics, including population, GDP, and other key indicators. It provides updated data from official sources and offers comprehensive information in an easily accessible format. WorldOmeter can be a valuable resource for obtaining the latest population and GDP figures for different countries.

Assets and Others

The best sites I used to download images, mention below:

Freepik (www.freepik.com) provides a vast collection of free and premium graphics, including illustrations, icons, templates, and more. It offers a user-friendly interface and allows you to filter assets based on your specific needs.

Pixabay (www.pixabay.com) is a well-known platform that offers a large library of high-resolution images and videos, all released under Creative Commons CC0. You can find a diverse range of visuals, from landscapes to objects, that can enhance the visual appeal of your game.

Unsplash (www.unsplash.com) is another excellent resource for high-quality, royalty-free images. It features a wide selection of photos contributed by talented photographers. The images on Unsplash are free to use and can bring an artistic touch to your game project.

I don't have the track of the original owner of the images, but in future I will mention them/ their attribute in "Traders" credits scene, but thanks to all of them providing me their Resources (JPG, PNG).

Scope of the Game

The scope of Traders as a game is extensive and offers unique opportunities for players. Here are some aspects of its scope:

1. Engaging Gameplay: Traders offers an engaging and immersive gameplay experience, combining elements of **strategy, negotiation, and resource management.** Players navigate the game board, buy properties, engage in trades, and make financial decisions to outwit their opponents.

2. Educational Value: Traders provides educational benefits by teaching players about real estate, money management, and decision-making in a business context. It offers insights into **economic concepts and strategic thinking, enhancing players' knowledge and skills.**

3. Global Appeal: Traders incorporates geographical elements by featuring various countries and neighboring regions, making it relatable to players around the world. **The use of Indian currency (rupees) adds a unique touch and promotes cultural awareness.**

4. Enhanced Game Board: Traders offers a larger game board compared to traditional board games, **featuring more tiles representing different countries and special interactive tiles.** This expands the gameplay possibilities and adds depth to the overall gaming experience.

5. Strategic Routes: The availability of three routes **(air, water, and land)** in Traders allows players to strategize their movements and make informed decisions about their journey across the game board. This adds a layer of strategic planning and provides players with different options for exploration.

6. Diverse Game Modes: Traders offers various game modes, including **bot vs. player, team vs. team, and player vs. player**, catering to different player preferences and allowing for diverse gameplay experiences.

7. Accessible User Interface: Traders features a user-friendly interface that ensures easy navigation and accessibility for players of all ages and skill levels. This makes it suitable for a **wide range of players**, including those new to board games or game development.

8. Balanced and Fair Gameplay: Traders aims to provide balanced and fair gameplay for all players, ensuring that everyone has an equal chance of success. This **promotes healthy competition and encourages strategic thinking.**

Through its captivating gameplay, educational value, and accessible design, Traders offers a unique and enjoyable gaming experience for players. And I hope its strategic elements, geographical immersion, and cultural references create a dynamic and engaging game that can be appreciated by a wide audience.

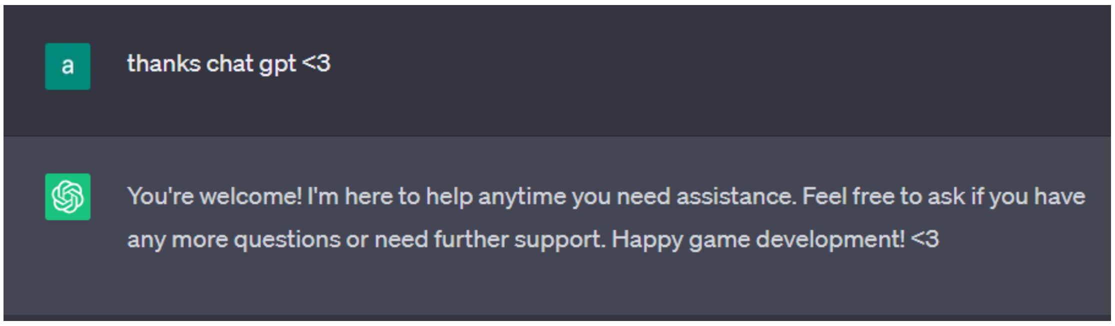
Chat GPT

As an **AI language model**, ChatGPT can be a **valuable assistant** in providing code snippets, guiding you with code structure, and offering instructions on how to use Unity. With its extensive knowledge base and programming expertise, ChatGPT can help you understand Unity's features, components, and best practices for game development.

When you need code snippets for specific functionalities or solutions, you can ask ChatGPT to provide examples in various programming languages commonly used with Unity, such as C# or JavaScript. It can **assist you with coding patterns, syntax, and logic to achieve desired outcomes** in your game projects.

Furthermore, ChatGPT can explain Unity's concepts and workflows, including scene creation, asset management, scripting, and game object manipulation. You can ask questions about Unity's user interface, editor features, and various features like Events, Prefabs or etc.

Whether you're a beginner seeking guidance on Unity's basics or an experienced developer looking for more advanced techniques**, ChatGPT can provide step-by-step instructions, explanations, and tips to help you achieve your goal**s. It can also assist in troubleshooting common errors or suggesting optimizations for your Unity projects.



a

thanks chat gpt <3

You're welcome! I'm here to help anytime you need assistance. Feel free to ask if you have any more questions or need further support. Happy game development! <3

Special thanks to ChatGPT for its invaluable assistance in providing code snippets, offering guidance on Unity, and helping game developers bring their creative visions to life.

Inspiration

MONOPOLY, a classic board game, is widely recognized and has been a source of inspiration for various adaptations and game designs. It involves strategic decision-making, property acquisition, and economic transactions, making it a popular choice for game developers.

INTERNATIONAL BUSINESS GAME by the brand EKTA, which originates from India. This further emphasizes the influence of real-world economic concepts and business strategies in the project.

MONOPOLY and **INTERNATIONAL BUSINESS GAME** inspired me a lot to build "Traders".

REPORT

Chapter: 1

Introduction of “TRADERS”

OVERVIEW

"Traders" is a tabletop board game where 2 to 4 players engage in real estate transactions and strategic gameplay. The objective is to outsmart opponents and accumulate wealth while avoiding bankruptcy. Players buy properties, engage in trades, and make calculated decisions to emerge as the last player standing. The game features over 45+ countries, special tiles, and three routes for strategic navigation. It combines luck, strategy, and negotiation, providing an educational and entertaining experience. With its visually stunning art style and versatile gameplay modes, "Traders" offers an immersive and balanced gaming experience.

OBJECTIVES & GAMEPLAY

In the game "Traders," the objective for each player is to become wealthier than their opponents and avoid bankruptcy. Players aim to accumulate assets and outmaneuver their rivals to secure financial dominance.

During gameplay, players use their tokens to navigate the game board and explore different tiles. They have the opportunity to purchase properties and build houses to generate income. The game incorporates various interactive tiles that can result in either profit or loss for the players, adding an element of chance and strategy.

Players must make calculated decisions on property purchases, trades with other players, and strategic investments to maximize their profits and minimize their losses. The game creates a competitive environment where players aim to strategically grow their wealth while hindering their opponents' progress.

Ultimately, the player who manages their resources effectively, makes astute decisions, and accumulates the most wealth while avoiding bankruptcy emerges as the winner of "Traders."

TARGET AUDIANCE

The target audience for "Traders" includes board game enthusiasts, strategy game lovers, and individuals looking for an immersive and competitive gaming experience. The game is designed to cater to players aged 8 and above, offering a balance between accessibility and strategic depth.

Board game enthusiasts will appreciate the engaging gameplay mechanics, which combine elements of luck, strategy, and negotiation. The game's emphasis on property acquisition, financial planning, and resource management will appeal to players who enjoy games that involve critical thinking and decision-making.

Strategy game lovers will find "Traders" intriguing, as it requires players to strategically navigate the game board, make calculated investments, and outsmart their opponents. The availability of multiple gameplay modes, such as multiplayer online matches, team vs. team battles, and player vs. player encounters, offers diverse strategic challenges for different playstyles.

"Traders" is designed to appeal to a global audience, featuring various countries and cultural elements on the game board. The inclusion of Indian currency (rupees) adds a unique touch and promotes cultural awareness.

Overall, the target audience for "Traders" encompasses both casual gamers and avid enthusiasts who enjoy immersive gameplay, strategic challenges, and educational elements related to finance and real estate.

SIGNIFICANCE OF THE GAME

The game "Traders" holds significant value due to several key aspects that set it apart from traditional board games. These aspects contribute to its unique gameplay experience and its potential impact on the gaming community:

1. New Board and Art Style: The introduction of a new game board featuring over 45 countries and 10+ special tiles offers players a fresh and expansive playing area. The visually stunning 2D art style with an immersive 3D effect enhances the aesthetic appeal and creates a sense of depth, making players feel like they are navigating a three-dimensional world. This innovative approach to board design adds a visually captivating element to the game.

2. Unique Perspective View: "Traders" incorporates a skillful use of perspective, further enhancing the immersive experience for players. The new perspective view adds a layer of realism and engagement, making players feel more connected to the game world. This fresh approach to gameplay presentation sets "Traders" apart and contributes to its significance in the gaming community.

3. Three Routes for Strategic Navigation: The introduction of three different routes (air, water, and land) provides players with strategic choices and opportunities. This adds depth and complexity to the gameplay, allowing players to devise unique strategies and explore different paths to achieve financial success. The inclusion of multiple routes expands the gameplay possibilities and offers a new level of strategic decision-making.

4. New Rules and Gameplay Mechanics: "Traders" introduces new rules and gameplay mechanics that differentiate it from traditional board games. The incorporation of chance elements, bonus opportunities, community chest interactions, and other exciting events adds unpredictability and excitement to the gameplay. These new rules and mechanics create a dynamic and engaging gameplay experience, making "Traders" a standout choice for board game enthusiasts.

5. Contribution to the Gaming Community: The unique features and innovative design of "Traders" contribute to the gaming community by offering a fresh and captivating gameplay experience. The game's expanded board, new art style, perspective view, and strategic routes provide inspiration and set a precedent for future game development. "Traders" showcases the creative ideas and concepts originating from India, emphasizing the country's potential for game innovation.

In summary, the significance of "Traders" lies in its introduction of a new board, innovative art style, perspective view, strategic routes, and gameplay mechanics. These elements contribute to a unique and immersive gaming experience, making the game stand out within the gaming community. Additionally, "Traders" serves as an example of creative ideas and concepts emerging from India, highlighting the country's potential in game development and fostering inspiration for future game creators.

DEVELOPMENT PROCESS

The development process of "Traders" involved a systematic approach to bring the game concept to life. The team's vision and goals were central to the process, aiming to create an engaging and immersive gameplay experience. Here's an overview of the development process, technologies used, challenges faced, and key milestones achieved:

1. Conceptualization and Planning: The development process began with brainstorming sessions to conceptualize the game idea and define its core features. The team outlined the gameplay mechanics, objectives, and unique selling points. Detailed planning was done to establish the scope, timeline, and resource allocation for the project.

2. Design and Artwork: The next step involved creating the game's visual elements and artwork. This included designing the game board, tokens, properties, and other in-game assets. The team focused on developing a visually appealing and immersive 2D art style with an immersive 3D effect, ensuring that the game's aesthetics aligned with the vision.

3. Technology and Tools: The development of "Traders" utilized modern technologies, tools, and frameworks to ensure a smooth and efficient development process. The team leveraged game development engines like Unity to build the game, which offered a wide range of features and functionality. Other tools such as graphic design software, code editors, and version control systems were used to streamline the development workflow.

4. Gameplay Development: The core gameplay mechanics were implemented during this phase. The team worked on coding the rules, interactions, and game logic to create an engaging and strategic gameplay experience. Challenges were encountered during balancing the gameplay mechanics, ensuring fair competition, and fine-tuning the player experience.

5. Testing and Iteration: A crucial aspect of the development process was testing the game extensively to identify and resolve any bugs, glitches, or gameplay imbalances. The team conducted internal playtesting sessions to gather feedback, analyze player experiences, and make necessary improvements. Iterations were made based on the feedback received to refine the gameplay and enhance the overall user experience.

7. Challenges Faced: Developing "Traders" came with its fair share of challenges. These included ensuring the game's stability and performance across various devices and screen sizes, optimizing resource management, and creating an intuitive user interface. The team also tackled challenges related to gameplay balancing, ensuring fair competition, and addressing any technical hurdles that arose during development.

8. Team's Vision and Goals: The development team's vision for "Traders" was to create an immersive and entertaining board game experience that would captivate players of all ages. Their goal was to deliver a visually stunning game with engaging gameplay mechanics, strategic depth, and a balanced competitive environment. They aimed to provide an enjoyable and memorable experience that showcased their creativity and passion for game development.

In conclusion, the development process of "Traders" involved meticulous planning, creative design, utilization of modern technologies, and thorough testing. The team overcame challenges and achieved significant milestones to bring their vision to life. The result is a visually stunning and engaging board game that offers a unique gameplay experience to players.

INDIVIDUAL CONTRIBUTION

Although "Traders" was envisioned as a team project, I was determined to carry it forward and make it a success. Despite the challenges of working individually, I was committed to pushing the boundaries and taking the project as far as possible. Here's how I approached the development of "Traders" as a solo endeavor:

1. Embracing Responsibility: As the sole contributor, I fully embraced the responsibility of taking charge of all aspects of the game's development. This included conceptualizing, designing, programming, and testing the game to ensure its quality and success.

2. Comprehensive Skill Set: To effectively handle the various development tasks, I leveraged my comprehensive skill set in game design, programming, graphics, and project management. This allowed me to take on all the necessary responsibilities and ensure a cohesive development process.

3. Effective Time Management: As an individual developer, time management became crucial. I established a clear development timeline, setting achievable goals and milestones. By efficiently allocating my time and resources, I ensured steady progress and timely completion of different development phases.

4. Seeking Feedback and Collaboration: While working individually, I recognized the importance of seeking feedback and collaborating with others. I actively sought input from peers, mentors, and the gaming community to gather diverse perspectives and refine the game's design and mechanics.

5. Iterative Development: I embraced an iterative development approach, continually improving and refining the game as I progressed. Regular playtesting and feedback sessions allowed me to identify areas of improvement and make necessary adjustments to enhance the gameplay experience.

6. Commitment to Quality: Despite working alone, I maintained a strong commitment to delivering a high-quality game. I paid attention to details, rigorously tested the game for bugs and glitches, and ensured smooth performance across different devices and platforms.

7. Showcasing Creativity: With "Traders," I aimed to showcase my creativity and innovative ideas. I incorporated unique elements such as the new board design, captivating artwork, and the introduction of the three strategic routes. These elements set "Traders" apart and contributed to its appeal in the gaming community.

8. Personal Growth and Learning: Developing "Traders" as an individual project provided an opportunity for personal growth and learning. I embraced the challenges, expanded my skill set, and gained valuable experience in project management, problem-solving, and game development.

In summary, although "Traders" was initially intended as a team project, I took on the challenge of designing & developing it individually. Through effective time management, comprehensive skills, seeking feedback, and commitment to quality, I carried the project forward, determined to make it a success. The experience allowed me to showcase my creativity, foster personal growth, and deliver an engaging and memorable gaming experience to the players.

Chapter: 2

Requirement Analysis and System Specification

Requirement Analysis is the process of identifying and documenting the needs and expectations of stakeholders for a particular system or project. It involves gathering and analyzing information to determine what features, functions, and capabilities are required.

System Specification refers to the detailed description of the hardware and software components, as well as the overall system configuration, needed to meet the identified requirements. It outlines the specific technical specifications and constraints for the development and implementation of the system.

REQUIREMENT ANALYSIS

To develop the game "Traders," a thorough requirement analysis was conducted to identify the necessary software and hardware components. The following software requirements were identified:

1. Pixel Lab: This application was utilized for creating game assets, such as graphics and visual elements. However, alternative software options like Adobe Photoshop or Lightroom can also be used for asset creation.

2. Unity Game Engine: Unity was chosen as the game engine for development. Unity Hub was downloaded and used to install Unity, which provides all the necessary libraries and tools for developing Android games.

3. Visual Basics: Visual Basics was used for coding purposes, specifically for integrating with Unity. It was set up during installation to be compatible with Unity for game development.

Additionally, the hardware requirements for the development environment were considered. The following specifications were used:

1. Operating System: Microsoft Windows 11 Home was the chosen operating system for development.

2. Processor: An 11th Gen Intel Core i3-1115G4 processor was used, operating at 3.00 GHz with 2 cores and 4 logical processors.

3. RAM: The development system was equipped with 8 GB of RAM, which is the minimum requirement for Unity and Visual Studio. However, having more RAM is recommended for better performance, especially when working on larger projects or multiple instances of the IDE.

SYSTEM SPECIFICATION

Based on the requirement analysis, the system specifications used for game development were as follows:

Operating System: Microsoft Windows 11 Home

Processor: 11th Gen Intel Core i3-1115G4 @ 3.00 GHz, 2 cores, 4 logical processors

RAM: 8 GB

System Name: GAGANBARTWAL

ADDITIONAL REQUIREMENT & SYSTEM SPECIFICATION

An audio editor: It is also necessary for editing sounds for the game. It is important to choose an audio editor that is suitable for game development and offers the required features.

However, it is specified that sound was not used in the game. If sound editing becomes necessary in the future, an audio editor can be added to the project.

For the asset creation using Pixel Lab, **I utilized my mother's phone, the Realme 1,** which provides sufficient resources for running Pixel Lab smoothly.

1. Processor: MediaTek Helio P60 octa-core processor
2. RAM: 6 GB
3. Internal Memory: 128 GB
4. Display: 6.0-inch IPS LCD
5. Resolution: 1080 x 2160 pixels
6. GPU: Mali-G72

These specifications contribute to the Realme 1's capabilities for graphic editing tasks, providing ample processing power, sufficient memory, and a vibrant display for an enhanced viewing experience.

Although the Realme 1 meets the requirements for asset creation in "Traders," it's important to note that users have the flexibility to choose alternative devices based on their familiarity and suitability. The specifications mentioned above are considered low-end by today's standards, but they are still adequate for creating assets using Pixel Lab. Users can opt for higher-end devices with more powerful processors, larger RAM capacity, and better displays if they prefer, as long as they are compatible with the required software and tools.

By conducting a comprehensive requirement analysis and specifying the system specifications, the development process for "Traders" can proceed efficiently with the appropriate software and hardware components in place.

Chapter: 3

System Design

In the context of "Traders," system design refers to the process of creating a detailed plan and blueprint for the various components and systems that make up the game. It involves defining the architecture, functionality, and interactions of the game's systems to ensure a cohesive and well-structured design.

GAME MECHANICS

The game mechanics of "Traders" revolve around strategic decision-making, resource management, and interactive gameplay elements. Here are the key game mechanics that drive the gameplay experience:

1. Dice Rolling: Players navigate the game board by rolling dice, determining the number of spaces they can move. The outcome of the dice roll introduces an element of chance and unpredictability into the game.

2. Property Acquisition: Players have the opportunity to purchase properties on the game board, which they can later rent out to opponents who land on their owned spaces. Property acquisition is a fundamental aspect of accumulating wealth and generating income in the game.

3. Trades and Negotiations: Players can engage in trades and negotiations with each other to acquire new properties, exchange resources, or gain advantages. Successful negotiations require strategic thinking, persuasion skills, and a keen understanding of market dynamics.

4. Income Generation: Players earn income from their owned properties, collecting rent from opponents who land on those spaces. The more properties a player owns, the higher their potential income generation, contributing to their overall financial success.

5. Strategic Expenditures: Players can strategically invest in their properties by building houses or other improvements, which can increase the rent value and potential income. These expenditures require careful planning and resource allocation.

6. Chance Events: The game incorporates chance events through special tiles or cards, introducing unpredictable outcomes that can either benefit or hinder players. These chance events add excitement, variety, and strategic considerations to the gameplay.

7. Bankruptcy and Elimination: Players aim to outmaneuver their opponents and drive them into bankruptcy by acquiring their properties or forcing them into unfavorable trades. The last player standing, who successfully avoids bankruptcy, emerges as the ultimate winner.

8. Multiple Gameplay Options: "Traders" offers various gameplay options, including multiplayer online with random players, creating and joining rooms with friends, and local play by sharing a single device. This allows players to choose their preferred mode of gameplay and enjoy the game with others.

By combining these game mechanics, "Traders" creates an immersive and strategic gameplay experience that challenges players to make shrewd decisions, manage their resources effectively, and outwit their opponents in the race to financial dominance.

USER INTERFACE UI

The user interface (UI) in "Traders" is designed to provide an immersive and intuitive gaming experience. With visually appealing graphics, clear menus, and responsive controls, the UI allows players to navigate the game world, manage their resources, and make strategic decisions effortlessly. It also provides informative notifications and feedback, ensuring players stay engaged and informed. The UI's responsive design and accessibility features make it adaptable to different devices and user preferences, enhancing the overall gameplay experience.

Some examples of UI given below :

Fig. 1 Background of Traders Concept Art

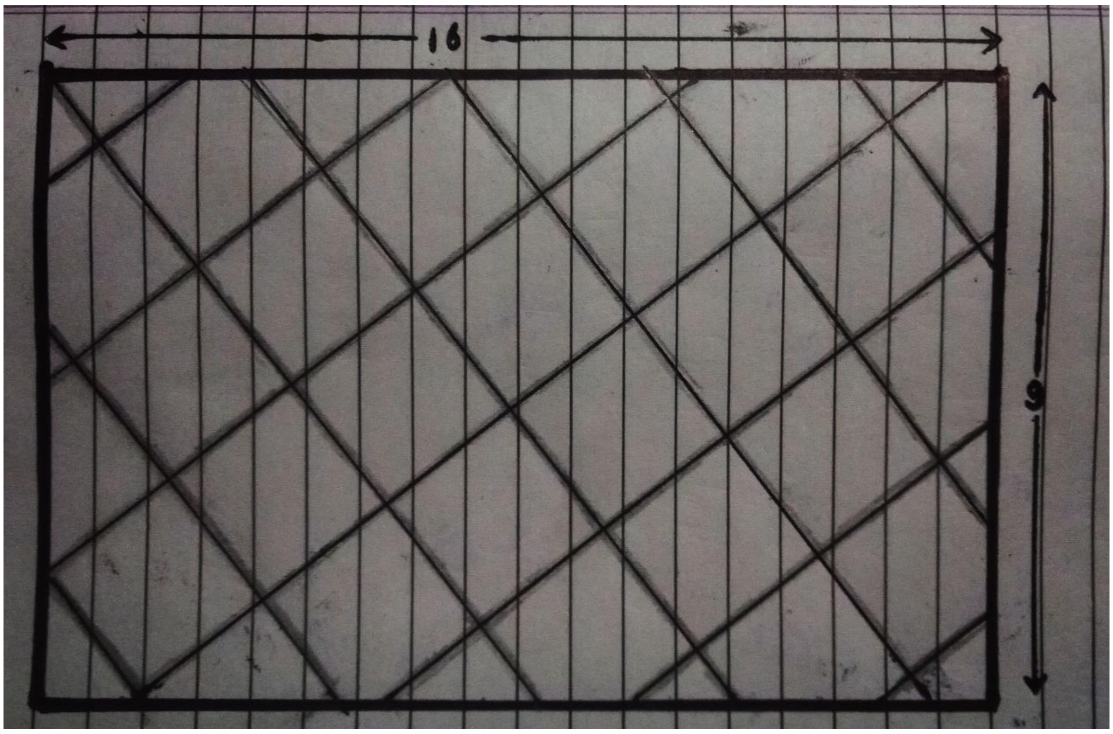
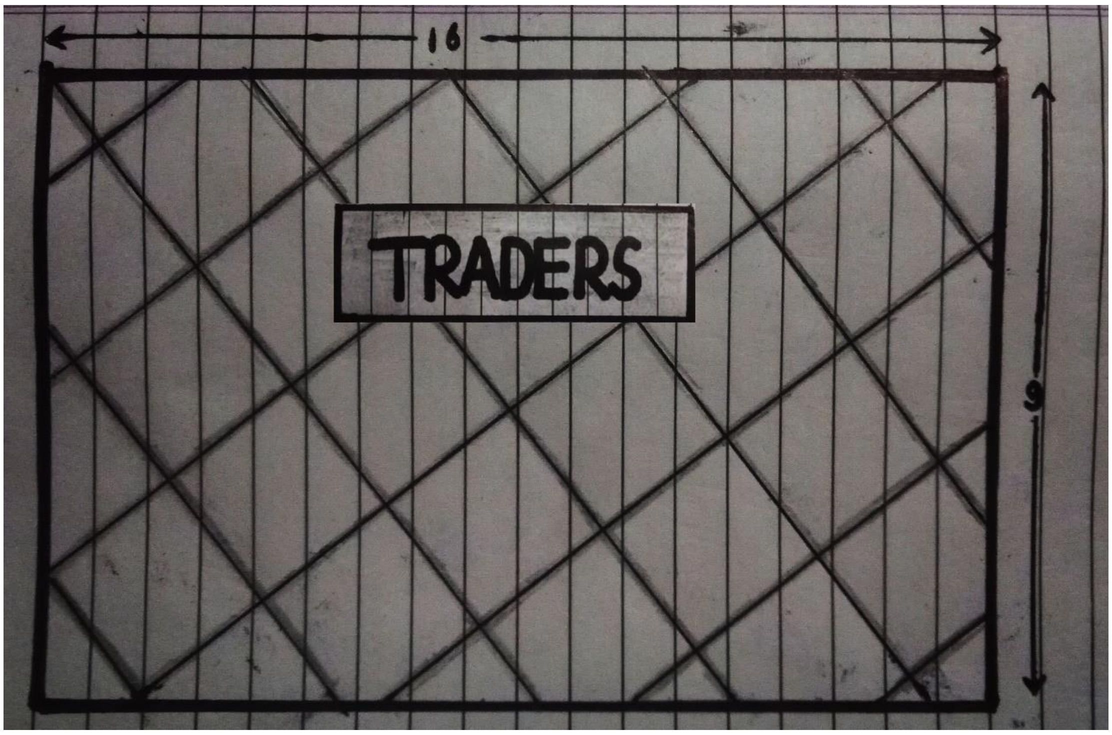


Fig. 2 Logo of Traders Concept Art

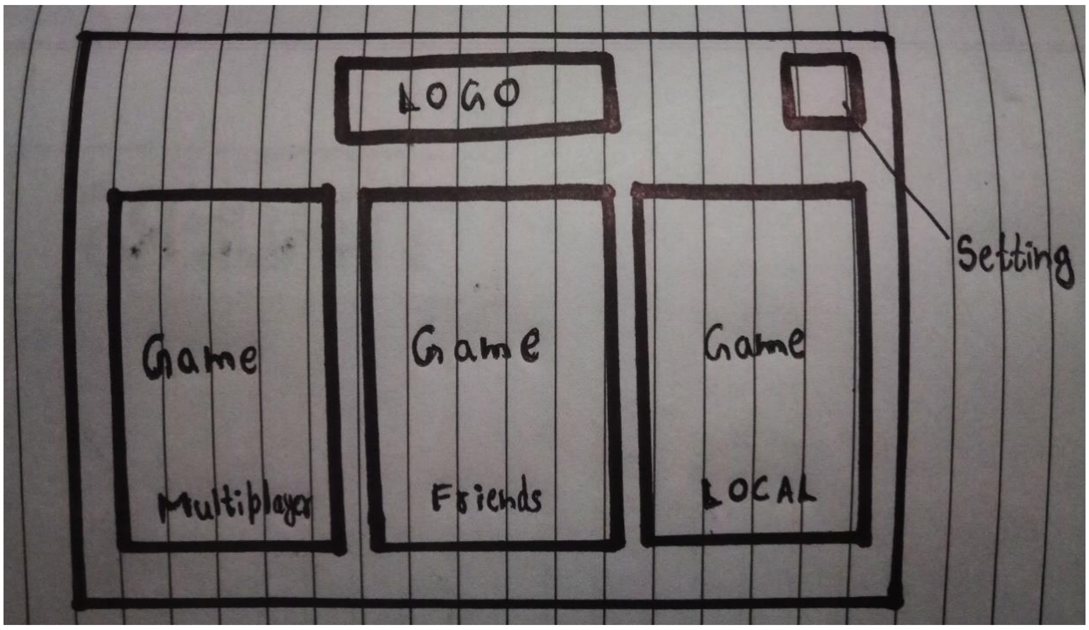
Fig. 3 Game Loading Scene Concept Art



The game loading scene in "Traders" is designed to provide a smooth and engaging transition for players as they enter the game. It serves as a visual and interactive element that keeps players entertained while the necessary game assets and data are loaded.

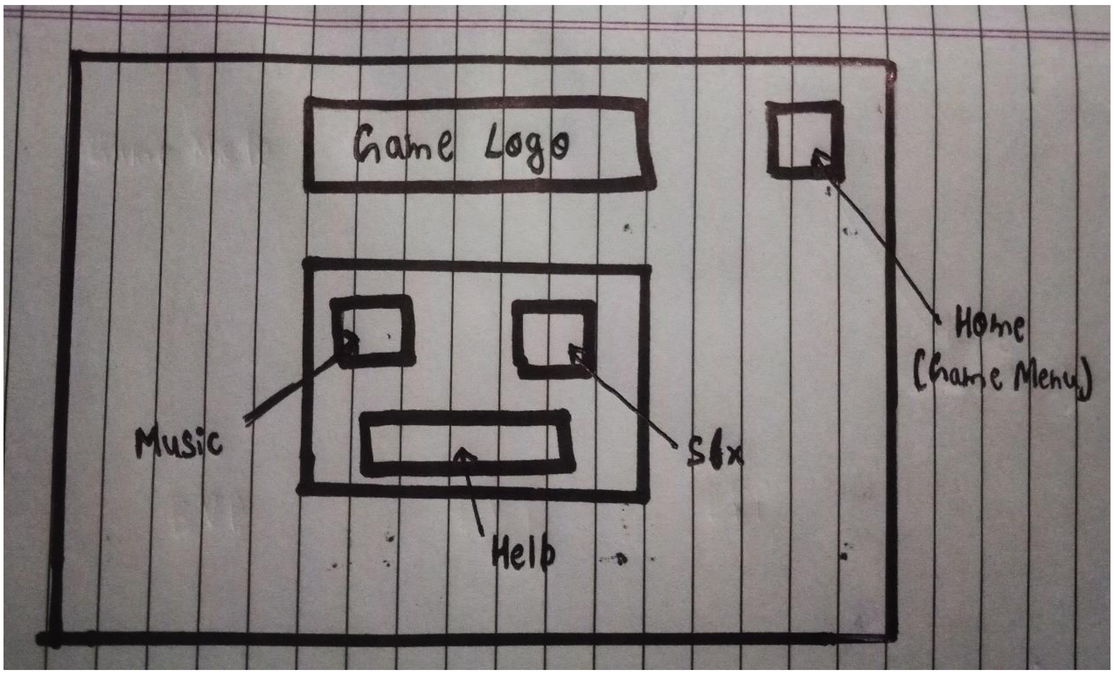
The loading scene begins with a visually appealing and dynamic loading screen that showcases the game's logo or artwork, creating anticipation and setting the tone for the gaming experience. The loading screen may also feature subtle animations or progress indicators to convey the progress of the loading process.

Fig. 4 Game Menu Scene Concept Art



The game menu in "Traders" offers versatile gameplay options, allowing players to choose between single-player, multiplayer, and local play modes. It also provides a settings menu for customization and personalization. With these features, players can enjoy a dynamic and tailored gaming experience, whether they prefer solo challenges, competitive multiplayer matches, or engaging gameplay sessions with friends.

Fig. 5 Setting Scene Concept Art



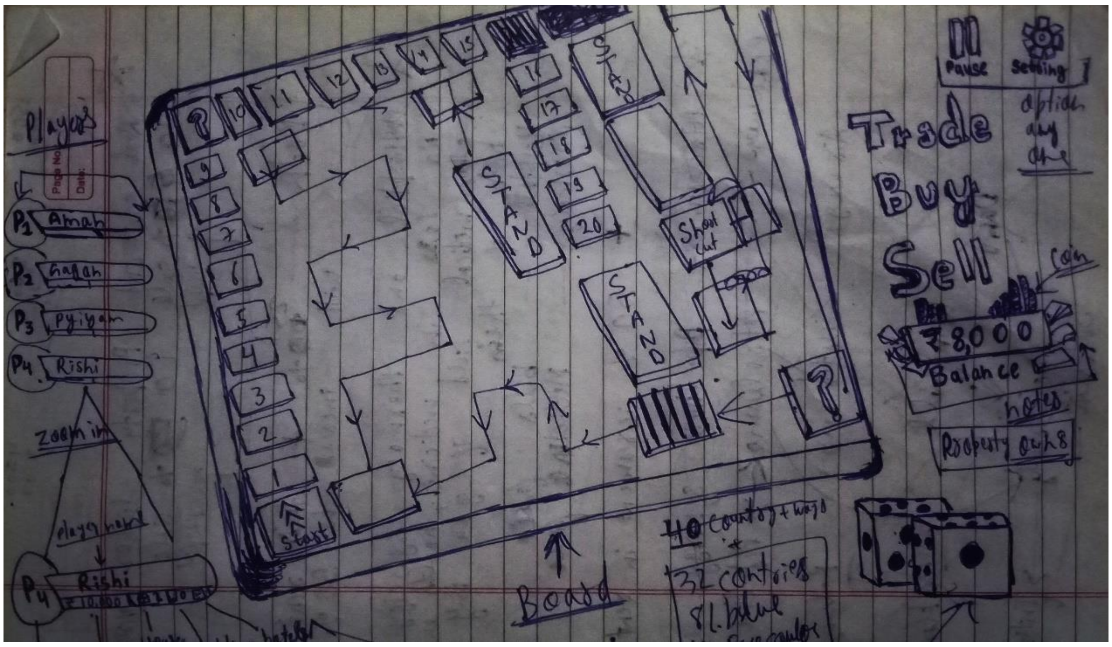
In the settings menu of "Traders," players have the ability to customize various aspects of the game to suit their preferences. It includes the following options:

1. Music On/Off: Players can toggle the music option to turn the in-game music on or off according to their preference. This allows them to create their desired audio atmosphere while playing the game.

2. SFX (Sound Effects) On/Off: The SFX option allows players to enable or disable the sound effects in the game. By toggling this option, players can control the volume and presence of in-game sound effects.

3. Help: The help option provides players with access to a comprehensive guide or tutorial that explains the game mechanics, rules, and strategies. This feature is designed to assist players in understanding the game better and making informed decisions during gameplay.

Fig. 6 First Concept Art of Game



Starting with my initial game design idea, I envisioned "Traders" to have a specific gameplay experience. However, as I progressed in the development process, I decided to make several changes and improvements to the game. One significant change was shifting from an isometric view to a top-down view due to resource limitations. Later on, I further modified the game's perspective, ultimately settling on a perspective view that enhanced the overall visual appeal.

These changes and improvisations allowed me to adapt the game design to overcome constraints and create a final design that I found visually appealing and suitable for the gameplay experience I envisioned. By being flexible and open to modifications, I was able to refine and shape the game to meet my goals and provide an enjoyable experience for the players.

GAME FLOW

The game flow in "Traders" follows a dynamic and engaging progression, keeping players immersed in the gameplay.

By providing these settings options, "Traders" ensures that players have control over the audio elements of the game, allowing them to tailor the audio experience to their liking. Additionally, the inclusion of a help option helps players access information and guidance whenever needed, promoting a smoother and more enjoyable gameplay experience.

1. Setup: Players begin by selecting their tokens and placing them on the starting point of the game board. They are also provided with an initial amount of virtual currency.

2. Turn-Based Gameplay: The game progresses in turns, with each player taking their actions one by one. Players roll the dice to determine their movement across the board.

3. Exploring Tiles: As players move their tokens, they explore different tiles on the board. These tiles represent properties, special events, or interactions. Players can choose to purchase properties, engage in trades, or interact with the tiles to gain advantages or face challenges.

4. Property Management: Players can buy properties they land on, which become part of their portfolio. They can construct houses on their properties to increase their value and earn more income from rent. Property management is a key aspect of the game, requiring strategic decisions on investments and resource allocation.

5. Interactions and Events: Throughout the game, players encounter various interactions and events on specific tiles. These can range from collecting bonuses or rewards, facing penalties, or engaging in special activities. These interactions add excitement and unpredictability to the gameplay.

6. Financial Decisions: Players must make wise financial decisions, such as selling properties, negotiating trades, or making strategic expenditures. Balancing income and expenses is crucial for long-term success and outmaneuvering opponents.

7. Bankruptcy and Victory: The game continues until all players except one have gone bankrupt. The remaining player emerges as the winner, having accumulated the most wealth and assets. The game flow ensures a competitive and strategic experience for all players.

8. Replayability: With different board configurations, random events, and strategic choices, each playthrough offers unique challenges and opportunities. Players can explore different strategies and approaches, enhancing the replayability of the game.

AUDIO DESIGN

The audio design in "Traders" encompasses two main components: music and sound effects (SFX). The game features a carefully selected soundtrack to enhance the overall experience. One of the tracks used in the game is "Blinding Lights" by the artist Weekend, which adds a dynamic and energetic atmosphere to the gameplay. This instrumental version of the song creates a captivating backdrop for players as they navigate through the game board.

In addition to the music, the game incorporates custom-designed UI sounds. These sounds were created by me to provide auditory feedback and enhance the user interface interactions. The UI sounds are carefully crafted to be intuitive, informative, and pleasing to the players' ears. These sounds add an extra layer of immersion and engagement to the game, making the user experience more enjoyable.

By combining well-curated music tracks and custom UI sounds, the audio design in "Traders" aims to create an immersive and memorable experience for players. The carefully chosen music track and the bespoke UI sounds contribute to the overall atmosphere of the game, adding depth and richness to the gameplay.

PERFORMANCE OPTIMIZATION

In "Traders," performance optimization plays a crucial role in ensuring a smooth and enjoyable gameplay experience on various screen sizes of Android phones. To achieve this, the visual assets of the game are optimized by reducing their original quality. By carefully balancing the visual fidelity and performance requirements, the game maintains a high level of visual appeal while minimizing the strain on the device's resources.

The optimization process involves optimizing textures, models, and other visual elements to reduce rendering demands. This ensures that the game runs efficiently on devices with different hardware capabilities, allowing a wider range of players to enjoy the game without experiencing lag or performance issues.

While the visual quality may be slightly decreased from the original assets, the development team ensures that the optimized visuals still meet players' expectations and deliver a beautiful visual experience. The goal is to strike a balance between visual appeal and optimized performance, providing players with an immersive and visually pleasing gameplay experience across various Android devices.

By implementing performance optimization techniques, "Traders" aims to provide a seamless and enjoyable gaming experience for players, regardless of their device's screen size or hardware specifications. The optimized performance ensures that players can fully immerse themselves in the game without any technical hindrances, while the visually appealing graphics maintain the game's overall aesthetic appeal.

Chapter: 4

Implementation, Testing and Maintenance

After finalizing the game design, I began the implementation phase by using Unity and C# programming language to bring the game to life. Following the spiral development life cycle, I integrated testing alongside the development process to ensure a high-quality and bug-free game.

Throughout the development and testing phases, I regularly conducted playtesting sessions to gather feedback and identify any issues or areas for improvement. This iterative approach allowed me to refine gameplay mechanics, address bugs, and enhance the overall user experience.

Maintenance plays a crucial role in keeping the game up-to-date and addressing any post-release issues. As the sole developer, I took responsibility for game maintenance, including bug fixes, performance optimizations, and updates to ensure the game remained enjoyable and compatible with the latest platforms and devices.

Regular updates and communication with the player community were also part of the maintenance process. I actively collected user feedback, listened to their suggestions, and implemented relevant improvements to provide an engaging and satisfying gaming experience.

Overall, the combination of implementation, testing, and maintenance ensured that "Traders" was developed with a focus on quality, user satisfaction, and continued support throughout its lifecycle.

There are some code of my scene that I am adding from next page:

1. GameIntro.cs:

- Handles the transition and fading effects for the game's intro scene.

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using System.Collections;

public class UnityIntro : MonoBehaviour
{
    public Image backgroundImage;
    public Image fadeInImage;
    public float colorTransitionDuration = 2.5f;
    public float fadeInDuration = 1.5f;
    public string nextSceneName;

    private void Start()
    {
        AudioManager.instance.PlayMusic("Game Opening"); // Play the game
opening music
        StartCoroutine(TransitionSequence()); // Start the transition sequence
coroutine
    }

    private IEnumerator TransitionSequence()
    {
        Color initialColor = Color.black; // Set the initial color to black
        Color targetColor = Color.white; // Set the target color to white
        float t = 0f;
        while (t < 1f)
        {
            t += Time.deltaTime / colorTransitionDuration; // Increment t based on
the elapsed time and transition duration
```

```
                backgroundImage.color = Color.Lerp(initialColor, targetColor, t); // Lerp the background image color from initial to target color
                yield return null; // Wait for the next frame
            }

            fadeInImage.color = Color.clear; // Set the fade-in image color to clear
            float fadeInT = 0f;
            while (fadeInT < 1f)
            {
                fadeInT += Time.deltaTime / fadeInDuration; // Increment fadeInT based on the elapsed time and fade-in duration
                fadeInImage.color = Color.Lerp(Color.clear, Color.white, fadeInT); // Lerp the fade-in image color from clear to white
                yield return null; // Wait for the next frame
            }

            yield return new WaitForSeconds(2.5f); // Wait for 2.5 seconds

            SceneManager.LoadScene(nextSceneName); // Load the next scene specified by the nextSceneName variable
        }
    }
```

This code is implementing an introductory scene in Unity, where the game transitions from a black background to a white background with a fading-in effect. Here's a breakdown of what the code does:

1. The code defines public variables for the background image, the fade-in image, the color transition duration, the fade-in duration, and the name of the next scene.

2. In the `Start()` method, the game opening music is played using the `AudioManager` class.

3. The `TransitionSequence()` coroutine is started. This coroutine handles the transition from the black background to the white background with a fading-in effect.

4. Inside the `TransitionSequence()` coroutine:

   - The initial color is set to black, and the target color is set to white.

- A `while` loop runs until `t` reaches 1 (based on the elapsed time and color transition duration).

- Inside the loop, the background image color is gradually changed from the initial color to the target color using `Color.Lerp()`.

- Another `while` loop runs until `fadeInT` reaches 1 (based on the elapsed time and fade-in duration).

- Inside this loop, the fade-in image color is gradually changed from clear to white using `Color.Lerp()`.

- After a delay of 2.5 seconds, the next scene specified by `nextSceneName` is loaded using `SceneManager.LoadScene()`.

Overall, this code sets up an introductory scene with a smooth transition from a black background to a white background and a fading-in effect.

2. GameOpening.cs:

- Simulates a loading screen sequence for the game's opening, allowing the player to continue after loading is complete.

```
using UnityEngine;

using UnityEngine.UI;

using TMPro;

using System.Collections;

using UnityEngine.SceneManagement;


public class GameOpening : MonoBehaviour

{

    public Slider loadingBar;                  // Reference to the loading bar UI slider

    public TextMeshProUGUI loadingText;         // Reference to the loading text UI

    public TextMeshProUGUI continueText;        // Reference to the continue text UI


    private bool isLoadingComplete = false;     // Flag to track if loading is complete


    private void Start()

    {

        loadingBar.value = 0f;                  // Set the initial value of the loading bar
        loadingText.text = "Loading...";         // Set the initial loading text
        continueText.gameObject.SetActive(false);   // Hide the continue text initially

        StartCoroutine(SimulateLoading());       // Start the loading simulation coroutine
    }

    private IEnumerator SimulateLoading()
    {
        float progress = 0f;                     // Progress variable for simulating loading
progress
        while (progress < 1f)
        {
            progress += Time.deltaTime;          // Increment progress based on elapsed time
            loadingBar.value = progress;         // Update the loading bar value
            yield return null;                   // Wait for the next frame
        }

        isLoadingComplete = true;                // Set the loading complete flag to true
        loadingText.gameObject.SetActive(false);   // Hide the loading text
        continueText.gameObject.SetActive(true);   // Show the continue text
        loadingBar.gameObject.SetActive(false);    // Hide the loading bar
    }

    private void Update()
    {
        if (isLoadingComplete)
        {
            // Check for user input to continue
            if (Input.GetMouseButtonDown(0) || Input.anyKeyDown || Input.touchCount
> 0)
            {
                SceneManager.LoadScene("Game Menu");           // Load the game menu
scene
```

```
            Resources.UnloadUnusedAssets();                // Unload any unused assets
from memory

          }

        }

    }

}
```

This code is responsible for simulating a loading screen in the game's opening scene. Here's a breakdown of what the code does:

The code defines public variables for the loading bar UI slider, loading text UI, and continue text UI.

In the Start() method, the initial values and visibility of the loading bar, loading text, and continue text are set.

The SimulateLoading() coroutine is started. This coroutine simulates the loading progress by incrementing the progress variable over time.

Inside the SimulateLoading() coroutine:

A while loop runs until the progress variable reaches 1 (signifying loading completion).

Inside the loop, the progress variable is incremented based on the elapsed time.

The loading bar value is updated accordingly.

The coroutine waits for the next frame before continuing.

After the loading is complete, the isLoadingComplete flag is set to true, and the loading text is hidden, while the continue text is shown. The loading bar is also hidden.

In the Update() method, when the loading is complete:

The code checks for user input (mouse click, any key press, or touch input).

If there is user input, the game menu scene is loaded, and any unused assets are unloaded from memory using Resources.UnloadUnusedAssets().

Overall, this code creates a loading screen effect, simulating loading progress until completion, and allowing the user to proceed to the game menu scene by providing input.

3. GameMenu.cs:

- Handles the game menu functionality, including loading the settings scene and the local game mode scene.

```
using UnityEngine;

using UnityEngine.SceneManagement;
```

```
public class GameMenu : MonoBehaviour
{
    private void Start()
    {
        // Play the main track music when the game menu scene starts.
        AudioManager.instance.PlayMusic("Main Track");
    }

    public void LoadSettingScene()
    {
        // Load the "Setting" scene.
        SceneManager.LoadScene("Setting");

        // Play a button click sound effect.
        AudioManager.instance.PlaySFX("Button Click");

        // Unload any unused assets from memory.
        Resources.UnloadUnusedAssets();
    }

    public void LoadLocalGameModeScene()
    {
        // Load the "Local Game Mode" scene.
        SceneManager.LoadScene("Local Game Mode");

        // Play a button click sound effect.
        AudioManager.instance.PlaySFX("Button Click");

        // Unload any unused assets from memory.
        Resources.UnloadUnusedAssets();
    }
}
```

This script represents the game menu functionality. It includes the following methods:

Start(): This method is executed when the game menu scene starts. It plays the main track music using AudioManager.instance.PlayMusic().

LoadSettingScene(): This method is called when the player wants to access the settings scene. It loads the "Setting" scene using SceneManager.LoadScene(), plays a button click sound effect using AudioManager.instance.PlaySFX(), and unloads any unused assets from memory using Resources.UnloadUnusedAssets().

LoadLocalGameModeScene(): This method is triggered when the player wants to play the local game mode. It loads the "Local Game Mode" scene using SceneManager.LoadScene(), plays a button click sound effect, and unloads any unused assets from memory.

These methods enable the player to navigate between the game menu and other scenes such as the settings scene or the local game mode scene. The scene loading, sound effects, and asset management are handled using Unity's SceneManager, AudioManager, and Resources APIs respectively.

4. Setting.cs:

- Manages the settings scene, allowing the player to toggle music and sound effects, open and close the help panel, and return to the main menu.

```
using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;


public class Setting : MonoBehaviour

{

   public Slider musicSlider;     // Reference to the music volume slider

   public Slider sfxSlider;       // Reference to the SFX volume slider

   public GameObject helpPanel;   // Reference to the help panel game object
```

```
    public Scrollbar scrollbar;    // Reference to the scrollbar within the help panel

    public void ToggleMusic()
    {
        // Toggle the music volume by setting it to 0 if it was greater than 0, and vice
versa
        float volume = musicSlider.value > 0 ? 0 : 1;
        AudioManager.instance.SetMusicVolume(volume);

        // Update the music slider value accordingly
        if (musicSlider.value > 0)
        {
            musicSlider.value = 0;
        }
        else
        {
            musicSlider.value = 1;
        }
    }

    public void ToggleSfx()
    {
        // Toggle the SFX volume by setting it to 0 if it was greater than 0, and vice versa
        float volume = sfxSlider.value > 0 ? 0 : 1;
        AudioManager.instance.SetSFXVolume(volume);

        // Update the SFX slider value accordingly
        if (sfxSlider.value > 0)
        {
            sfxSlider.value = 0;
        }
        else
        {
            sfxSlider.value = 1;
        }
    }

    public void OpenHelpPanel()
    {
        // Play a button click sound effect
        AudioManager.instance.PlaySFX("Button Click");

        // Set the help panel game object to active
        helpPanel.SetActive(true);

        // Adjust the scrollbar value to the top
        scrollbar.value = 1f;
    }

    public void CloseHelpPanel()
    {
        // Play a button click sound effect
        AudioManager.instance.PlaySFX("Button Click");

        // Set the help panel game object to inactive
        helpPanel.SetActive(false);
    }

    public void LoadMainMenu()
    {
        // Load the "Game Menu" scene
        SceneManager.LoadScene("Game Menu");

        // Play a button click sound effect
```

```
        AudioManager.instance.PlaySFX("Button Click");


        // Unload any unused assets from memory
        Resources.UnloadUnusedAssets();
    }
}
```

This script represents the functionality of the settings scene. It includes the following methods:

ToggleMusic(): This method is called when the player interacts with the music slider. It toggles the music volume by setting it to 0 if it was greater than 0, and vice versa. It also updates the music slider value accordingly.

ToggleSfx(): Similar to ToggleMusic(), this method is triggered when the player interacts with the sound effects (SFX) slider. It toggles the SFX volume and updates the slider value accordingly.

OpenHelpPanel(): This method is called when the player wants to open the help panel. It plays a button click sound effect, sets the help panel game object to active, and adjusts the scrollbar value to the top.

CloseHelpPanel(): This method is triggered when the player wants to close the help panel. It plays a button click sound effect and sets the help panel game object to inactive.

LoadMainMenu(): This method is used to return to the game menu scene. It loads the "Game Menu" scene, plays a button click sound effect, and unloads any unused assets from memory.

These methods provide functionality for adjusting the music and SFX volume, opening and closing the help panel, and navigating back to the game menu scene. The scene loading, sound effects, and asset management are handled using Unity's SceneManager, AudioManager, and Resources APIs respectively.

5. LocalGameMode.cs:

- Manages the local game mode, providing functions to load the local player versus player (PVP) scene and the main menu scene.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class LocalGameMode : MonoBehaviour
{
   // This method is called when the player wants to load the local player-versus-player (PVP) game mode.
   public void LoadLocalPVP()
   {
      // Load the "PVP Setting" scene.
      SceneManager.LoadScene("PVP Setting");

      // Play a button click sound effect.
      AudioManager.instance.PlaySFX("Button Click");

      // Unload any unused assets from memory.
      Resources.UnloadUnusedAssets();
   }

   // This method is called when the player wants to return to the main menu.
   public void LoadMainMenu()
   {
      // Load the "Game Menu" scene.
      SceneManager.LoadScene("Game Menu");

      // Play a button click sound effect.
      AudioManager.instance.PlaySFX("Button Click");

      // Unload any unused assets from memory.
      Resources.UnloadUnusedAssets();
   }
}
```

This code is a script attached to a game object that handles the local game mode functionality. It includes two methods:

1. `LoadLocalPVP()`: This method is called when the player wants to load the local player-versus-player (PVP) game mode. It uses `SceneManager.LoadScene()` to load the "PVP Setting" scene, plays a button click sound effect using `AudioManager.instance.PlaySFX()`, and unloads any unused assets from memory using `Resources.UnloadUnusedAssets()`.

2. `LoadMainMenu()`: This method is called when the player wants to return to the main menu. It uses `SceneManager.LoadScene()` to load the "Game Menu" scene, plays a button click sound effect using `AudioManager.instance.PlaySFX()`, and unloads any unused assets from memory using `Resources.UnloadUnusedAssets()`.

These methods allow the player to navigate between the local PVP game mode and the main menu. The scene transitions are handled by Unity's `SceneManager`, and the button click sound effect is played using an `AudioManager` instance. The `Resources.UnloadUnusedAssets()` call helps free up memory by unloading any assets that are no longer in use.

6. PVPSetting.cs:

- Handles the PVP game mode scene, allowing the player to set the number of players and initiate the loading of the game scene.

```
using UnityEngine;

using UnityEngine.UI;

using TMPro;

using UnityEngine.SceneManagement;


public class PVPSetting : MonoBehaviour

{

    public GameObject loadingScreen;

    public Slider loadingBar;

    public TextMeshProUGUI loadingText;


    private void Start()

    {

        loadingScreen.SetActive(false); // Disable the loading screen initially
    }

    // Load the PVP game scene with the specified number of players
    public void LoadPVPGameWithPlayers(int players)
    {
        PlayerPrefs.SetInt("NumberOfPlayers", players); // Set the number of players in
PlayerPrefs
        AudioManager.instance.PlaySFX("Button Click"); // Play button click sound

        loadingScreen.SetActive(true); // Enable the loading screen
        AudioManager.instance.StopMusic(); // Stop playing the music
        StartCoroutine(DelayedLoadGameAsync()); // Start the async loading process
with a small delay
    }

    // Add a small delay before starting the actual async loading process
    private System.Collections.IEnumerator DelayedLoadGameAsync()
    {
        yield return new WaitForSeconds(0.1f); // Wait for a small delay

        StartCoroutine(LoadGameAsync()); // Start loading the game async
    }

    // Load the game scene asynchronously
    private System.Collections.IEnumerator LoadGameAsync()
    {
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync("Game"); // Start
loading the game scene
        asyncLoad.allowSceneActivation = false; // Do not allow the scene to activate
immediately

        while (!asyncLoad.isDone)
        {
            float progress = Mathf.Clamp01(asyncLoad.progress / 0.9f); // Calculate the loading progress

            loadingBar.value = progress; // Update the loading bar value

            loadingText.text = "Loading... " + (progress * 100f).ToString("F0") + "%"; // Update the loading text

            if (progress >= 1f)

            {

                asyncLoad.allowSceneActivation = true; // Allow the scene to activate when the loading is complete

            }

            yield return null;

        }

    }

    // Load the local game mode scene

    public void LoadLocalGameModeScene()

    {

        SceneManager.LoadScene("Local Game Mode"); // Load the local game mode scene

        AudioManager.instance.PlaySFX("Button Click"); // Play button click sound

        Resources.UnloadUnusedAssets(); // Unload unused assets to free up memory

    }

}
```

This script manages the PVP setting scene. It allows the player to select the number of players for the PVP game and initiates the asynchronous loading of the game scene. It also displays a loading screen with a progress bar and updates the loading progress. Additionally, it provides functionality to load the local game mode scene.

And finally main codes of the TRADER;

```
using UnityEngine;
using UnityEngine.UI;
using System;

public class DiceRoller : MonoBehaviour
{
    public GameObject dice;
    public GameObject dice1;
    public GameObject dice2;
    public Button diceButton;
    public Sprite[] diceSideSprites;
    public Image[] diceImages;
    public MessageManager messageManager;

    private bool isRolling = false;
    private float rollDuration = 1f;
    private float rollTimer = 0f;
    private int dice1Result = 0;
    private int dice2Result = 0;
    private bool isDiceRolled = false;
    public DiceBGColorChanger diceBGColorChanger;


    public event Action OnDiceRolled;

    public void ActivateDice()
    {
       dice.gameObject.SetActive(true);
       diceButton.onClick.AddListener(OnDiceButtonClicked);
       diceButton.interactable = !isDiceRolled;

       diceBGColorChanger.EnableBGColorChanger();

    }

    private void Update()
    {
       if (isRolling)
       {
          messageManager.HideMessage();
          messageManager.HideMessage();
          messageManager.DisplayMessage("Dices Are Rolling");

          rollTimer += Time.deltaTime;

          if (rollTimer >= rollDuration)
          {
             isRolling = false;
             dice1Result = UnityEngine.Random.Range(1, 7);
             dice2Result = UnityEngine.Random.Range(1, 7);
             UpdateDiceVisuals(dice1Result, dice2Result);

             isDiceRolled = true;
             diceButton.interactable = !isDiceRolled;
             diceBGColorChanger.SetActive(false);
             OnDiceRolled?.Invoke();
          }
          else
          {
             RollDiceAnimation();
```

```
        }
      }
    }

    private void OnDiceButtonClicked()
    {
      if (!isRolling && !isDiceRolled)
      {
        isRolling = true;
        rollTimer = 0f;
      }
    }

    private void RollDiceAnimation()
    {
      int randomSide1 = UnityEngine.Random.Range(0, 6);
      int randomSide2 = UnityEngine.Random.Range(0, 6);

      Sprite sprite1 = diceSideSprites[randomSide1];
      Sprite sprite2 = diceSideSprites[randomSide2];

      diceImages[0].sprite = sprite1;
      diceImages[1].sprite = sprite2;
    }

    private void UpdateDiceVisuals(int roll1, int roll2)
    {
      diceImages[0].sprite = diceSideSprites[roll1 - 1];
      diceImages[1].sprite = diceSideSprites[roll2 - 1];
    }

    public int[] GetDiceResult()
    {
      int[] diceResults = new int[2];
      diceResults[0] = dice1Result;
      diceResults[1] = dice2Result;
      return diceResults;
    }

    public void DeactivateDice()
    {
      dice.gameObject.SetActive(false);
    }
}


using UnityEngine;
using UnityEngine.UI;
```

```
public class DiceBGColorChanger : MonoBehaviour
{
    public Image backgroundImage;
    public float colorChangeInterval = 1f;

    private string[] colorCodes = new string[]
    {  "#FF0000", "#FF0D00", "#FF1A00", "#FF2700", "#FF3400", "#FF4100",
"#FF4F00", "#FF5C00",
    "#FF6900", "#FF7600", "#FF8300", "#FF9000", "#FF9E00", "#FFAA00",
"#FFB800", "#FFC500",
    "#FFD300", "#FFDF00", "#FFEB00", "#FFF800", "#FAFF00", "#EDFF00",
"#E0FF00", "#D3FF00",
    "#C5FF00", "#B8FF00", "#AAFF00", "#9DFF00", "#90FF00", "#83FF00",
"#76FF00", "#69FF00",
    "#5CFF00", "#4FFF00", "#42FF00", "#35FF00", "#28FF00", "#1BFF00",
"#0EFF00", "#00FF00",
    "#00FF0D", "#00FF1A", "#00FF27", "#00FF34", "#00FF41", "#00FF4F",
"#00FF5C", "#00FF69",
    "#00FF76", "#00FF83", "#00FF90", "#00FF9E", "#00FFAA", "#00FFB8",
"#00FFC5", "#00FFD3",
    "#00FFDF", "#00FFEB", "#00FFF8", "#00FAFF", "#00EDFF", "#00E0FF",
"#00D3FF", "#00C5FF",
    "#00B8FF", "#00AAFF", "#009DFF", "#0090FF", "#0083FF", "#0076FF",
"#0069FF", "#005CFF",
    "#004FFF", "#0042FF", "#0035FF", "#0028FF", "#001BFF", "#000EFF",
"#0000FF", "#0D00FF",
    "#1A00FF", "#2700FF", "#3400FF", "#4100FF", "#4F00FF", "#5C00FF",
"#6900FF", "#7600FF",
    "#8300FF", "#9000FF", "#9E00FF", "#AA00FF", "#B800FF", "#C500FF",
"#D300FF", "#DF00FF",
    "#EB00FF", "#F800FF", "#FF00FA", "#FF00ED", "#FF00E0", "#FF00D3",
"#FF00C5", "#FF00B8",
    "#FF00AA", "#FF009D", "#FF0090", "#FF0083", "#FF0076", "#FF0069",
"#FF005C", "#FF004F",
    "#FF0042", "#FF0035", "#FF0028", "#FF001B", "#FF000E"
    };

    private int currentIndex = 0;
    private float timer = 0f;
    private bool isActive = false;

    public void EnableBGColorChanger()
    {
        isActive = true;
        SetBackgroundColor();
    }

    private void Update()
    {
        if (!isActive)
```

```
            return;

        timer += Time.deltaTime;

        if (timer >= colorChangeInterval)
        {
            timer = 0f;
            ChangeBackgroundColor();
        }
    }

    public void SetActive(bool isActive)
    {
        this.isActive = isActive;
    }

    private void SetBackgroundColor()
    {
        if (backgroundImage != null && colorCodes.Length > 0)
        {
            Color color;
            if (ColorUtility.TryParseHtmlString(colorCodes[currentIndex], out color))
            {
                backgroundImage.color = color;
            }
        }
    }

    private void ChangeBackgroundColor()
    {
        if (backgroundImage != null && colorCodes.Length > 0)
        {
            currentIndex = (currentIndex + 1) % colorCodes.Length;
            SetBackgroundColor();
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class PlayerDiceRoller : MonoBehaviour
{
    public GameObject dice1;
    public GameObject dice2;
    public TextMeshProUGUI diceResultText;
    public Sprite[] diceSideSprites;
    public Image[] diceImages;

    private float rollDuration = 1f;
    private bool isRolling = false;
    private float rollTimer = 0f;
    private int dice1Result = 0;
    private int dice2Result = 0;

    private PlayerTurnManager turnManager;

    public void Initialize(PlayerTurnManager turnManager)

    {
       this.turnManager = turnManager;
    }

    public void RollDice()
    {
       isRolling = true;
       rollTimer = 0f;

       dice1Result = Random.Range(1, 7);
       dice2Result = Random.Range(1, 7);

    }

    private void Update()
    {
       if (isRolling)
       {
          rollTimer += Time.deltaTime;

          if (rollTimer >= rollDuration)
          {
             isRolling = false;
             UpdateDiceVisuals(dice1Result, dice2Result);
             DisplayResult(dice1Result + dice2Result);

             int totalDiceResult = dice1Result + dice2Result;
             turnManager.GetDiceResult(GetPlayerIndex(), totalDiceResult);
          }
          else
          {
             RollDiceAnimation();
          }
       }
    }

    private void RollDiceAnimation()
```

```
    {
        int randomSide1 = Random.Range(0, 6);
        int randomSide2 = Random.Range(0, 6);

        Sprite sprite1 = diceSideSprites[randomSide1];
        Sprite sprite2 = diceSideSprites[randomSide2];

        diceImages[0].sprite = sprite1;
        diceImages[1].sprite = sprite2;
    }

    private void UpdateDiceVisuals(int roll1, int roll2)
    {
        diceImages[0].sprite = diceSideSprites[roll1 - 1];
        diceImages[1].sprite = diceSideSprites[roll2 - 1];
    }

    private void DisplayResult(int diceResult)
    {
        diceResultText.text = diceResult.ToString();
    }

    private int GetPlayerIndex()
    {
        return transform.GetSiblingIndex();
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class PlayerProfile : MonoBehaviour
{
    public Image playerTokenImage;
    public TMP_Text playerNameText;
    public TMP_Text playerMoneyText;
    public TMP_Text sitesOwnedText;
    public TMP_Text housesOwnedText;
    public TMP_Text utilitiesOwnedText;
    public Image routeImage;

    private string playerName;
    private int playerMoney;
    private int sitesOwned;
    private int housesOwned;
    private int utilitiesOwned;
    private string route;
    private Vector3 tokenPosition;

    public void SetPlayerProfile(string playerName, Sprite playerTokenSprite, int
playerMoney, int sitesOwned, int housesOwned, int utilitiesOwned, string route)
    {
        this.playerName = playerName;
        this.playerMoney = playerMoney;
        this.sitesOwned = sitesOwned;
        this.housesOwned = housesOwned;
        this.utilitiesOwned = utilitiesOwned;
        this.route = route;

        playerNameText.text = playerName;
        playerTokenImage.sprite = playerTokenSprite;
        playerMoneyText.text = playerMoney.ToString();
        sitesOwnedText.text = sitesOwned.ToString();
        housesOwnedText.text = housesOwned.ToString();
        utilitiesOwnedText.text = utilitiesOwned.ToString();
        SetRouteImage(route);
    }

    public void SetPlayerToken(GameObject playerToken)
    {
        playerTokenImage.gameObject.SetActive(true);
        playerTokenImage.sprite =
playerToken.GetComponent<SpriteRenderer>().sprite;
    }

    public string GetPlayerName()
    {
        return playerName;
    }

    public int GetPlayerMoney()
    {
        return playerMoney;
    }
    public Vector3 GetPlayerTokenPosition()
    {
        return tokenPosition;
    }

    public void SetPlayerTokenPosition(Vector3 newPosition)
    {
        tokenPosition = newPosition;
        // Update the position of the player token using the newPosition value
        // Replace this comment with the actual code to update the player token's
position
    }
```

```
public void AddMoney(int amount)
{
   playerMoney += amount;
   playerMoneyText.text = playerMoney.ToString();
}

public void DeductMoney(int amount)
{
   playerMoney -= amount;
   playerMoneyText.text = playerMoney.ToString();
}

public void AddSiteOwned()
{
   sitesOwned++;
   sitesOwnedText.text = "Sites Owned: " + sitesOwned.ToString();
}

public void AddHouseOwned()
{
   housesOwned++;
   housesOwnedText.text = "Houses Owned: " + housesOwned.ToString();
}

public void AddUtilityOwned()
{
   utilitiesOwned++;
   utilitiesOwnedText.text = "Utilities Owned: " + utilitiesOwned.ToString();
}

public void SetRoute(string route)
{
   this.route = route;
   SetRouteImage(route);
}

public string GetRoute()
{
   return route;
}

public void SetTokenPosition(Vector3 position)
{
   tokenPosition = position;
   transform.position = position;
}

private void SetRouteImage(string route)
{
        switch (route)
        {
            case "Air":
                routeImage.sprite = airRouteSprite;
                break;
            case "Water":
                routeImage.sprite = waterRouteSprite;
                break;
            case "Land":
                routeImage.sprite = landRouteSprite;
                break;
            case "noRoute":
                routeImage.sprite = noRouteSprite;
                break;
            default:
                routeImage.sprite = null;
                break;
        }
    }

    public Sprite airRouteSprite;
    public Sprite waterRouteSprite;
    public Sprite landRouteSprite;
    public Sprite noRouteSprite;


}



using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class PlayerProfileManager : MonoBehaviour
{
    public GameObject playerProfilePrefab;
    public RectTransform playerProfileContainer;
    public PlayerTokensSprite playerTokensSprite;
    public Transform playerTokensContainer;
    public PlayerTokensSpawner playerTokensSpawner;

    private PlayerProfile[] playerProfiles;

    public void CreatePlayerProfileUI()
    {
        int numberOfPlayers = GameManager.Instance.GetNumberOfPlayers();

        playerProfiles = new PlayerProfile[numberOfPlayers]; // Store player profiles
        // Retrieve the spawned player tokens from PlayerTokensSpawner
        GameObject[] playerTokens =
playerTokensSpawner.GetSpawnedPlayerTokens();

        for (int i = 0; i < numberOfPlayers; i++)
        {
            GameObject playerProfileGO = Instantiate(playerProfilePrefab,
playerProfileContainer);

            Vector3 position = GetPlayerProfilePosition(i);
            playerProfileGO.GetComponent<RectTransform>().anchoredPosition =
position;

            PlayerProfile playerProfile =
playerProfileGO.GetComponent<PlayerProfile>();

            string playerName = GetPlayerName(i);
            Sprite playerTokenSprite = GetPlayerTokenSprite(i);
            int playerMoney = GetPlayerMoney(i);
            int sitesOwned = GetSitesOwned(i);
            int housesOwned = GetHousesOwned(i);
            int utilitiesOwned = GetUtilitiesOwned(i);
            string route = GetRoute(i);

            playerProfile.SetPlayerProfile(playerName, playerTokenSprite, playerMoney,
sitesOwned, housesOwned, utilitiesOwned, route);

            // Assign player tokens to PlayerProfile
            playerProfile.SetPlayerToken(playerTokens[i]);

        }

        Debug.Log("Player Profile UI Created");
    }

    private Vector3 GetPlayerProfilePosition(int playerIndex)
    {
        int numberOfPlayers = PlayerPrefs.GetInt("NumberOfPlayers");

        if (numberOfPlayers == 2)
        {
            if (playerIndex == 0)
            {
                return new Vector3(0f, 240f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(0f, 120f, 0f);
            }
        }
        else if (numberOfPlayers == 3)
        {
            if (playerIndex == 0)
            {
                return new Vector3(0f, 240f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(0f, 120f, 0f);
            }
            else if (playerIndex == 2)
            {
                return new Vector3(0f, 0f, 0f);
            }
        }
        else if (numberOfPlayers == 4)
        {
            if (playerIndex == 0)
            {
                return new Vector3(0f, 240f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(0f, 120f, 0f);
            }
            else if (playerIndex == 2)
            {
                return new Vector3(0f, 0f, 0f);
            }
            else if (playerIndex == 3)
            {
                return new Vector3(0f, -120f, 0f);
            }
        }

        return Vector3.zero;
    }

    private string GetPlayerName(int playerIndex)
    {
        switch (playerIndex)
        {
            case 0:
                return "Red";
            case 1:
                return "Blue";
            case 2:
                return "Green";
            case 3:
                return "Yellow";
            default:
                return "";
        }
    }

    private Sprite GetPlayerTokenSprite(int playerIndex)
    {
        switch (playerIndex)
        {
            case 0:
                return playerTokensSprite.redSprite;
            case 1:
                return playerTokensSprite.blueSprite;
            case 2:
                return playerTokensSprite.greenSprite;
            case 3:
                return playerTokensSprite.yellowSprite;
            default:
                return null;
        }
    }

    private int GetPlayerMoney(int playerIndex)
    {
        return 25000;
    }

    private int GetSitesOwned(int playerIndex)
    {
        return 0;
    }

    private int GetHousesOwned(int playerIndex)
    {
        return 0;
    }

    private int GetUtilitiesOwned(int playerIndex)
    {

        return 0;
    }

    private string GetRoute(int playerIndex)
    {

        return "noRoute";
    }

    public void SetPlayerToken(int index, GameObject playerToken)
```

```
    {
        PlayerProfile playerProfile =
playerProfileContainer.GetChild(index).GetComponent<PlayerProfile>();
        playerProfile.SetPlayerToken(playerToken);
    }


    public void UpdatePlayerProfileUI(int[] playerTurnOrder)
    {
        Debug.Log("Received Player Turn Order in Player Profile Manager: " +
string.Join(", ", playerTurnOrder));

        for (int i = 0; i < playerTurnOrder.Length; i++)
        {
            GameObject playerProfileGO =
playerProfileContainer.GetChild(playerTurnOrder[i]).gameObject;

            Vector3 position = GetPlayerProfilePosition(i);
            playerProfileGO.GetComponent<RectTransform>().anchoredPosition =
position;
        }
    }


    public PlayerProfile[] GetPlayerProfiles()
    {
        return playerProfileContainer.GetComponentsInChildren<PlayerProfile>();
    }
}


using UnityEngine;

public class PlayerTokensSpawner : MonoBehaviour
{
    public GameObject playerPrefab;
    public Transform playerTokenContainer;
    public PlayerProfileManager profileManager;
    private GameObject[] playerTokens; // Declare playerTokens as a class-level
variable

    public void SpawnPlayerTokens()
    {
        int numberOfPlayers = GameManager.Instance.GetNumberOfPlayers();
        Debug.Log("Number of Players: " + numberOfPlayers);

        playerTokens = new GameObject[numberOfPlayers]; // Initialize the
playerTokens array

        for (int i = 0; i < numberOfPlayers; i++)
        {
            Vector3 startingPosition = GetStartingPosition(i);

            GameObject currentPlayer = Instantiate(playerPrefab, startingPosition,
Quaternion.identity, playerTokenContainer);

            Sprite playerTokenSprite = GetPlayerTokenSprite(i);
            if (playerTokenSprite != null)
            {
                SpriteRenderer playerSpriteRenderer =
currentPlayer.GetComponent<SpriteRenderer>();
                if (playerSpriteRenderer != null)
                {
                    playerSpriteRenderer.sprite = playerTokenSprite;
                }
            }

            currentPlayer.transform.rotation = Quaternion.Euler(-60f, 0f, 0f);

            playerTokens[i] = currentPlayer;

        }

        Debug.Log("Player Tokens Spawned");
    }

    public GameObject[] GetSpawnedPlayerTokens()
    {
        return playerTokens;
    }


    private Vector3 GetStartingPosition(int playerIndex)
    {
        int numberOfPlayers = PlayerPrefs.GetInt("NumberOfPlayers");

        if (numberOfPlayers == 2)
        {
            if (playerIndex == 0)
            {
                return new Vector3(-2.2f, 0f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(2.2f, 0f, 0f);
            }
        }
        else if (numberOfPlayers == 3)
        {
            if (playerIndex == 0)
            {
                return new Vector3(-4.4f, 0f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(0f, 0f, 0f);
            }
            else if (playerIndex == 2)
            {
                return new Vector3(4.4f, 0f, 0f);
            }
        }
        else if (numberOfPlayers == 4)
        {
            if (playerIndex == 0)
            {
                return new Vector3(-6.6f, 0f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(-2.2f, 0f, 0f);
            }
            else if (playerIndex == 2)
            {
                return new Vector3(2.2f, 0f, 0f);
            }
            else if (playerIndex == 3)
            {
                return new Vector3(6.6f, 0f, 0f);
            }
        }

        return Vector3.zero;
    }

    private Sprite GetPlayerTokenSprite(int playerIndex)
    {
        PlayerTokensSprite playerTokensSprite =
FindObjectOfType<PlayerTokensSprite>();

        switch (playerIndex)
        {
            case 0:
                return playerTokensSprite.redSprite;
            case 1:
                return playerTokensSprite.blueSprite;
            case 2:
                return playerTokensSprite.greenSprite;
            case 3:
                return playerTokensSprite.yellowSprite;
            default:
                return null;
        }
    }

    public void UpdatePlayerTokens(int[] playerTurnOrder)
    {
        Debug.Log("Received Player Turn Order in Player Tokens Spawner: " +
string.Join(", ", playerTurnOrder));

        for (int i = 0; i < playerTurnOrder.Length; i++)
        {
            GameObject currentPlayer =
playerTokenContainer.GetChild(playerTurnOrder[i]).gameObject;

            Vector3 position = GetStartingPosition(i);
            currentPlayer.transform.position = position;
        }
    }
}


using UnityEngine;

public class PlayerTokensSprite : MonoBehaviour
{
    public Sprite redSprite;
    public Sprite blueSprite;
    public Sprite greenSprite;
    public Sprite yellowSprite;

}
```

```
using UnityEngine;
using System;
using System.Collections.Generic;

public class PlayerData
{
    public int playerIndex;
    public int diceResult;

    public PlayerData(int index, int result)
    {
        playerIndex = index;
        diceResult = result;
    }
}

public class PlayerTurnManager : MonoBehaviour
{
    public GameObject dicePrefab;
    public Transform diceContainer;

    private List<PlayerData> playerDataList;
    private int[] diceResults;
    private int[] sortedPlayerIndex;

    public GameManager gameManager;

    public void CreateDiceInstantiate()
    {
        int numberOfPlayers = GameManager.Instance.GetNumberOfPlayers();
        diceResults = new int[numberOfPlayers];
        playerDataList = new List<PlayerData>();


        for (int i = 0; i < numberOfPlayers; i++)
        {
            GameObject diceObject = Instantiate(dicePrefab, diceContainer);

            PlayerDiceRoller diceRoller =
diceObject.GetComponent<PlayerDiceRoller>();
            diceRoller.Initialize(this);

            Vector3 position = GetDicePosition(i);
            diceObject.GetComponent<RectTransform>().anchoredPosition = position;
            diceRoller.RollDice();
        }
    }

    private Vector3 GetDicePosition(int playerIndex)
    {
        int numberOfPlayers = PlayerPrefs.GetInt("NumberOfPlayers");

        if (numberOfPlayers == 2)
        {
            if (playerIndex == 0)
            {
                return new Vector3(270f, 215f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(270f, 105f, 0f);
            }
        }
        else if (numberOfPlayers == 3)
        {
            if (playerIndex == 0)
            {
                return new Vector3(270f, 225f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(270f, 105f, 0f);
            }
            else if (playerIndex == 2)
            {
                return new Vector3(270f, -15f, 0f);
            }
        }
        else if (numberOfPlayers == 4)
        {
            if (playerIndex == 0)
            {
                return new Vector3(270f, 225f, 0f);
            }
            else if (playerIndex == 1)
            {
                return new Vector3(270f, 105f, 0f);
            }
            else if (playerIndex == 2)
            {
                return new Vector3(270f, -15f, 0f);
            }
            else if (playerIndex == 3)
            {
                return new Vector3(270f, -135f, 0f);
            }
        }

        return Vector3.zero;
    }



    public int GetDiceResult(int playerIndex, int diceResult)
    {
        Debug.Log("Player " + playerIndex);
        Debug.Log("Dice Result: " + diceResult);

        diceResults[playerIndex] = diceResult;

        if (IsArrayFull())
```

```
        {
            CheckArrayCondition();
        }

        return diceResults[playerIndex];
    }

    private bool IsArrayFull()
    {
        for (int i = 0; i < diceResults.Length; i++)
        {
            if (diceResults[i] == 0)
            {
                return false;
            }
        }

        return true;
    }

    private void CheckArrayCondition()
    {
        bool hasSameResult = false;

        for (int i = 0; i < diceResults.Length - 1; i++)
        {
            for (int j = i + 1; j < diceResults.Length; j++)
            {
                if (diceResults[i] == diceResults[j])
                {
                    hasSameResult = true;
                    break;
                }
            }

            if (hasSameResult)
                break;
        }

        if (hasSameResult)
        {
            UpdateGame();
        }
        else
        {
            for (int i = 0; i < diceResults.Length; i++)
            {
                AssignDiceResult(i, diceResults[i]);
            }
            SortDiceResults();
            ExtractPlayerIndexes();
        }
    }

    private void AssignDiceResult(int playerIndex, int diceResult)
    {
        PlayerData playerData = new PlayerData(playerIndex, diceResult);
        playerDataList.Add(playerData);
        Debug.Log("Player " + playerData.playerIndex + " dice result: " +
playerData.diceResult);
    }

    private void SortDiceResults()
    {
        playerDataList.Sort((a, b) => b.diceResult.CompareTo(a.diceResult));

        Debug.Log("Sorted dice results:");
        foreach (PlayerData playerData in playerDataList)
        {
            Debug.Log("Player " + playerData.playerIndex + " dice result: " +
playerData.diceResult);
        }
    }

    private void ExtractPlayerIndexes()
    {
        sortedPlayerIndex = new int[playerDataList.Count];

        for (int i = 0; i < playerDataList.Count; i++)
        {
            sortedPlayerIndex[i] = playerDataList[i].playerIndex;
        }

        gameManager.UpdateTurnOrder(sortedPlayerIndex);
    }

    public void DestroyDiceObjects()
    {
        foreach (Transform child in diceContainer)
        {
            Destroy(child.gameObject);
        }
    }

    private void UpdateGame()
    {
        Debug.Log("Updating game...");

            foreach (Transform child in diceContainer)
            {
                Destroy(child.gameObject);
            }

        playerDataList.Clear();
        Array.Clear(diceResults, 0, diceResults.Length);

        CreateDiceInstantiate();
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;

public class AudioManager : MonoBehaviour
{
    public static AudioManager instance;

    public Sound[] musicSounds;
    public Sound[] sfxSounds;

    public AudioSource musicSource;
    public AudioSource sfxSource;

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }

        GameObject musicSliderObject = GameObject.FindWithTag("MusicSlider");
        if (musicSliderObject != null)
        {
            Slider musicSlider = musicSliderObject.GetComponent<Slider>();
            if (musicSlider != null)
            {
                musicSource.volume = musicSlider.value;
            }
        }
        GameObject sfxSliderObject = GameObject.FindWithTag("SfxSlider");
        if (sfxSliderObject != null)
        {
            Slider sfxSlider = sfxSliderObject.GetComponent<Slider>();
            if (sfxSlider != null)
            {
                sfxSource.volume = sfxSlider.value;
            }
        }

    }

    public void PlayMusic(string soundName)
    {
        Sound sound = GetSound(soundName, musicSounds);
        if (sound != null)
        {
            musicSource.clip = sound.clip;
            musicSource.loop = sound.loop;
            musicSource.volume = sound.volume;
            musicSource.pitch = sound.pitch;
            musicSource.Play();
        }
    }

    public void PlaySFX(string soundName)
    {
        Sound sound = GetSound(soundName, sfxSounds);
        if (sound != null)
        {
            sfxSource.PlayOneShot(sound.clip, sound.volume);
            sfxSource.pitch = sound.pitch;
        }
    }

    private Sound GetSound(string soundName, Sound[] sounds)
    {
        foreach (Sound sound in sounds)
        {
            if (sound.name == soundName)
            {
                return sound;
            }
        }

        Debug.LogWarning("Sound not found: " + soundName);
        return null;
    }

    public void SetMusicVolume(float volume)
```

```
    {
        musicSource.volume = volume;
    }

    public void SetSFXVolume(float volume)
    {
        sfxSource.volume = volume;
    }

    public void StopMusic()
    {
        musicSource.Stop();
    }

    public void StopSFX()
    {
        sfxSource.Stop();
    }

}


using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System;

public class BoardSetupManager : MonoBehaviour
{
    public GameObject board;
    public GameObject boardVertice;
    public GameObject boardShadow;
    public GameObject boardInterior;
    public GameObject[] tiles;
    public GameObject[] tile2s;
    public GameObject[] tile3s;

    public float boardSetupDelay = 1f;
    public float boardInteriorSetupDelay = 1f;
    private float fallDelay = 0.2f;

    public event Action OnBoardBaseSetupComplete;
    private bool skipPressed = false;

    private Vector3 boardTargetPosition;
    private Vector3 boardInteriorTargetPosition;

    public void StartSetupBoard()
    {
        board.gameObject.SetActive(true);
        StartCoroutine(SetupBoard());
    }

    private IEnumerator SetupBoard()
    {
        board.transform.position = new Vector3(board.transform.position.x,
board.transform.position.y, -30f);
        yield return new WaitForSeconds(boardSetupDelay);

        if (AudioManager.instance != null)
        {
            AudioManager.instance.PlaySFX("Board Drop");
        }

        float startTime = Time.time;
        float journeyDuration = .25f;
        Vector3 startPosition = board.transform.position;
        Vector3 boardTargetPosition = new Vector3(board.transform.position.x,
board.transform.position.y, 0f);

        while (Time.time < startTime + journeyDuration)
        {
            if (skipPressed)
            {
                board.transform.position = boardTargetPosition;
                yield break;
            }

            float t = (Time.time - startTime) / journeyDuration;
            board.transform.position = Vector3.Lerp(startPosition, boardTargetPosition, t);

            yield return null;
        }
        board.transform.position = boardTargetPosition;
        boardVertice.gameObject.SetActive(true);
        boardShadow.gameObject.SetActive(true);
        boardInterior.gameObject.SetActive(true);
        StartCoroutine(SetupBoardInterior());
    }

    private IEnumerator SetupBoardInterior()
    {
        boardInterior.transform.position = new
Vector3(boardInterior.transform.position.x, boardInterior.transform.position.y, -30f);
        yield return new WaitForSeconds(boardInteriorSetupDelay);

        if (AudioManager.instance != null)
        {
            AudioManager.instance.StopSFX();
            AudioManager.instance.PlaySFX("Board Drop");
        }

        float startTime = Time.time;
        float journeyDuration = .25f;
        Vector3 startPosition = boardInterior.transform.position;
        Vector3 boardInteriorTargetPosition = new
Vector3(boardInterior.transform.position.x, boardInterior.transform.position.y, 0f);

        while (Time.time < startTime + journeyDuration)
        {
            if (skipPressed)
            {
                boardInterior.transform.position = boardInteriorTargetPosition;
                yield break;
            }

            float t = (Time.time - startTime) / journeyDuration;
            boardInterior.transform.position = Vector3.Lerp(startPosition,
boardInteriorTargetPosition, t);

                yield return null;
        }

        boardInterior.transform.position = boardInteriorTargetPosition;
        OnBoardBaseSetupComplete?.Invoke();
    }

    public void StartSetupTiles()
    {
        StartCoroutine(SetupTiles());
    }

    private IEnumerator SetupTiles()
    {
        foreach (GameObject tile in tiles)
        {
            tile.SetActive(false);
        }

        for (int i = 0; i < tiles.Length; i++)
        {
            GameObject tile = tiles[i];
            tile.SetActive(true);

            Vector3 startPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, -30f);
            tile.transform.position = startPosition;

            Vector3 targetPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, 0f);
```

```
        if (skipPressed)
        {
            tile.transform.position = targetPosition;
            break;
        }
        else
        {
            yield return new WaitForSeconds(fallDelay);
            float startTime = Time.time;
            float duration = 0.225f;

            while (Time.time < startTime + duration)
            {
                float t = (Time.time - startTime) / duration;
                tile.transform.position = Vector3.Lerp(startPosition, targetPosition, t);

                if (t >= 0.8f )
                {
                    if (AudioManager.instance != null)
                    {
                        AudioManager.instance.StopSFX();
                        AudioManager.instance.PlaySFX("Tile Drop");
                    }
                }

                yield return null;
            }

            tile.transform.position = targetPosition;
        }

        if (i == 1 && tile2s != null)
        {
            StartCoroutine(SetupTile2s());
        }

        if (i == 3 && tile3s != null)
        {
            StartCoroutine(SetupTile3s());
        }
    }
}

private IEnumerator SetupTile2s()
{
    foreach (GameObject tile in tile2s)
    {
        tile.SetActive(false);
    }
```

```
        for (int i = 0; i < tile2s.Length; i++)
        {
            GameObject tile = tile2s[i];
            tile.SetActive(true);

            Vector3 startPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, -30f);
            tile.transform.position = startPosition;

            Vector3 targetPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, 0f);

            if (skipPressed)
            {
                tile.transform.position = targetPosition;
                break;
            }
            else
            {
                yield return new WaitForSeconds(fallDelay);
                float startTime = Time.time;
                float duration = 0.225f;

                while (Time.time < startTime + duration)
                {
                    float t = (Time.time - startTime) / duration;
                    tile.transform.position = Vector3.Lerp(startPosition, targetPosition, t);
                    yield return null;
                }

                tile.transform.position = targetPosition;
            }
        }
    }

    private IEnumerator SetupTile3s()
    {
        foreach (GameObject tile in tile3s)
        {
            tile.SetActive(false);
        }

        for (int i = 0; i < tile3s.Length; i++)
        {
            GameObject tile = tile3s[i];
            tile.SetActive(true);

            Vector3 startPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, -30f);
```

```
            tile.transform.position = startPosition;

            Vector3 targetPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, 0f);

            if (skipPressed)
            {
                tile.transform.position = targetPosition;
                break;
            }
            else
            {
                yield return new WaitForSeconds(fallDelay);
                float startTime = Time.time;
                float duration = 0.225f;

                while (Time.time < startTime + duration)
                {
                    float t = (Time.time - startTime) / duration;
                    tile.transform.position = Vector3.Lerp(startPosition, targetPosition, t);
                    yield return null;
                }

                tile.transform.position = targetPosition;
            }
        }
    }


    public void OnSkipButtonPress()
    {
        AudioManager.instance.StopSFX();
        skipPressed = true;
        board.gameObject.SetActive(true);
        boardVertice.gameObject.SetActive(true);
        boardShadow.gameObject.SetActive(true);
        boardInterior.gameObject.SetActive(true);
        ActiveTiles();
        ActiveTile2s();
        ActiveTile3s();
    }

    private void ActiveTiles()
    {
        foreach (GameObject tile in tiles)
        {
            tile.SetActive(true);
        }

        for (int i = 0; i < tiles.Length; i++)
        {
            GameObject tile = tiles[i];
            tile.SetActive(true);

            Vector3 startPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, -30f);
            tile.transform.position = startPosition;

            Vector3 targetPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, 0f);

            tile.transform.position = targetPosition;

        }
    }

    private void ActiveTile2s()
    {
        foreach (GameObject tile in tile2s)
        {
            tile.SetActive(true);
        }

        for (int i = 0; i < tile2s.Length; i++)
        {
            GameObject tile = tile2s[i];
            tile.SetActive(true);

            Vector3 startPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, -30f);
            tile.transform.position = startPosition;

            Vector3 targetPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, 0f);

            tile.transform.position = targetPosition;

        }
    }

    private void ActiveTile3s()
    {
        foreach (GameObject tile in tile3s)
        {
            tile.SetActive(true);
        }

        for (int i = 0; i < tile3s.Length; i++)
        {
            GameObject tile = tile3s[i];
            tile.SetActive(true);

            Vector3 startPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, -30f);
            tile.transform.position = startPosition;

            Vector3 targetPosition = new Vector3(tile.transform.position.x,
tile.transform.position.y, 0f);

            tile.transform.position = targetPosition;

        }
    }
}


using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class CameraManager : MonoBehaviour
{
    public GameObject entryCameraTopView;
    public GameObject entryCameraBottomLeft;
    public GameObject mainCamera;

    public event Action OnCameraSettingDone;
    private bool skipPressed = false;

    private float transitionDuration = 2f;

    public TransitionManager transitionManager;

    public void ActivateEntryCameraTopView()
    {
        if (entryCameraTopView != null)
        {
            entryCameraTopView.SetActive(true);
        }
    }

    public void StartEntryCameraTopViewZoomIn()
    {
        float delay = 0.5f;
        StartCoroutine(DelayedEntryCameraTopViewZoomIn(delay));
    }

    private IEnumerator DelayedEntryCameraTopViewZoomIn(float delay)
    {
```

```
    yield return new WaitForSeconds(delay);
    StartCoroutine(EntryCameraTopViewZoomIn());
}

private IEnumerator EntryCameraTopViewZoomIn()
{
    if (entryCameraTopView != null)
    {
        Camera camera = entryCameraTopView.GetComponent<Camera>();
        if (camera != null)
        {
            float startFOV = camera.fieldOfView;
            float endFOV = 58f;
            float startTime = Time.time;
            float duration = 1.5f;

            while (Time.time < startTime + duration)
            {
                if (skipPressed)
                {
                    yield break;
                }

                float t = (Time.time - startTime) / duration;
                if (camera != null)
                    camera.fieldOfView = Mathf.Lerp(startFOV, endFOV, t);

                if (duration >= 0.5f && transitionManager != null)
                {
                    transitionManager.StartFadeInBlack();
                }

                yield return null;
            }

            if (camera != null)
                camera.fieldOfView = endFOV;
        }

        if (OnCameraSettingDone != null)
            OnCameraSettingDone.Invoke();
    }
}

public void SwitchToEntryCameraBottomLeft()
{
    DeactivateEntryCameraTopView();
    ActivateEntryCameraBottomLeft();
}
```

```
public void DeactivateEntryCameraTopView()
{
   if (entryCameraTopView != null)
   {
      entryCameraTopView.SetActive(false);
   }
}

public void ActivateEntryCameraBottomLeft()
{
   if (entryCameraBottomLeft != null)
   {
      entryCameraBottomLeft.SetActive(true);
   }
}

public void DestroyEntryCameraTopView()
{
   if (entryCameraTopView != null)
   {
      Destroy(entryCameraTopView);
   }
}

public void StartEntryCameraBottomLeftZoomIn()
{
   StartCoroutine(EntryCameraBottomLeftZoomIn());
}

private IEnumerator EntryCameraBottomLeftZoomIn()
{
   if (entryCameraBottomLeft != null)
   {
      Camera camera = entryCameraBottomLeft.GetComponent<Camera>();
      if (camera != null)
      {
         float startFOV = camera.fieldOfView;
         float endFOV = 23.5f;
         float startTime = Time.time;
         float duration = 1.5f;

         if (transitionManager != null)
            transitionManager.StartFadeOutBlack();

         while (Time.time < startTime + duration)
         {
            if (skipPressed)
            {
               yield break;
            }
```

```
                float t = (Time.time - startTime) / duration;
                if (camera != null)
                    camera.fieldOfView = Mathf.Lerp(startFOV, endFOV, t);

                yield return null;
            }

            if (camera != null)
                camera.fieldOfView = endFOV;
        }

        if (OnCameraSettingDone != null)
            OnCameraSettingDone.Invoke();
    }
}

public void StartRevolveEntryCamera()
{
    StartCoroutine(RevolveEntryCamera());
}

private IEnumerator RevolveEntryCamera()
{
    if (entryCameraBottomLeft != null)
    {
        Camera camera = entryCameraBottomLeft.GetComponent<Camera>();
        if (camera != null)
        {
            Vector3 originalPosition = entryCameraBottomLeft.transform.position;
            float startTime;
            float duration;

            Vector3 targetPosition = new Vector3(150f, originalPosition.y,
originalPosition.z);
            startTime = Time.time;
            duration = 6.921f;

            while (Time.time < startTime + duration)
            {
                if (skipPressed)
                {
                    yield break;
                }

                float t = (Time.time - startTime) / duration;
                if (entryCameraBottomLeft != null)
                    entryCameraBottomLeft.transform.position =
Vector3.Lerp(originalPosition, targetPosition, t);
                yield return null;
            }

            if (entryCameraBottomLeft != null)
               entryCameraBottomLeft.transform.position = targetPosition;

            yield return new WaitForSeconds(0.3f);

            originalPosition = entryCameraBottomLeft.transform.position;
            targetPosition = new Vector3(150f, 42.5f, originalPosition.z);
            startTime = Time.time;
            duration = 3.75f;

            while (Time.time < startTime + duration)
            {
               if (skipPressed)
               {
                  yield break;
               }

               float t = (Time.time - startTime) / duration;
               if (entryCameraBottomLeft != null)
                  entryCameraBottomLeft.transform.position =
Vector3.Lerp(originalPosition, targetPosition, t);
               yield return null;
            }

            if (entryCameraBottomLeft != null)
               entryCameraBottomLeft.transform.position = targetPosition;

            yield return new WaitForSeconds(0.3f);

            originalPosition = entryCameraBottomLeft.transform.position;
            targetPosition = new Vector3(0f, 42.5f, originalPosition.z);
            startTime = Time.time;
            duration = 6.921f;

            while (Time.time < startTime + duration)
            {
               if (skipPressed)
               {
                  yield break;
               }

               float t = (Time.time - startTime) / duration;
               if (entryCameraBottomLeft != null)
                  entryCameraBottomLeft.transform.position =
Vector3.Lerp(originalPosition, targetPosition, t);
               yield return null;
            }
            if (entryCameraBottomLeft != null)
                entryCameraBottomLeft.transform.position = targetPosition;

            yield return new WaitForSeconds(0.3f);

            originalPosition = entryCameraBottomLeft.transform.position;
            targetPosition = new Vector3(0f, -52.5f, originalPosition.z);
            startTime = Time.time;
            duration = 3.75f;

            while (Time.time < startTime + duration)
            {
                if (skipPressed)
                {
                    yield break;
                }

                float t = (Time.time - startTime) / duration;
                if (entryCameraBottomLeft != null)
                    entryCameraBottomLeft.transform.position =
Vector3.Lerp(originalPosition, targetPosition, t);
                yield return null;
            }

            if (entryCameraBottomLeft != null)
                entryCameraBottomLeft.transform.position = targetPosition;

            yield return new WaitForSeconds(.85f);

            if (OnCameraSettingDone != null)
                OnCameraSettingDone.Invoke();
        }
    }
}

public void StartSmoothSwitchToMainCamera()
{
    StartCoroutine(SmoothSwitchToMainCamera());
}

private IEnumerator SmoothSwitchToMainCamera()
{
    if (entryCameraBottomLeft != null && mainCamera != null)
    {
        float startTime = Time.time;
        Vector3 startPosition = entryCameraBottomLeft.transform.position;
        Vector3 startRotation =
entryCameraBottomLeft.transform.rotation.eulerAngles;
        float startFOV =
entryCameraBottomLeft.GetComponent<Camera>().fieldOfView;
```

```
        Vector3 targetPosition = mainCamera.transform.position;
        Vector3 targetRotation = mainCamera.transform.rotation.eulerAngles;
        float targetFOV = mainCamera.GetComponent<Camera>().fieldOfView;

        while (Time.time < startTime + transitionDuration)
        {
            if (skipPressed)
            {
                yield break;
            }

            float t = (Time.time - startTime) / transitionDuration;

            if (entryCameraBottomLeft != null)
                entryCameraBottomLeft.transform.position = Vector3.Lerp(startPosition,
targetPosition, t);

            if (entryCameraBottomLeft != null)
                entryCameraBottomLeft.transform.rotation =
Quaternion.Euler(Vector3.Lerp(startRotation, targetRotation, t));

            if (entryCameraBottomLeft != null)
                entryCameraBottomLeft.GetComponent<Camera>().fieldOfView =
Mathf.Lerp(startFOV, targetFOV, t);

            yield return null;
        }

        if (entryCameraBottomLeft != null)
        {
            entryCameraBottomLeft.transform.position = targetPosition;
            entryCameraBottomLeft.transform.rotation =
Quaternion.Euler(targetRotation);
            entryCameraBottomLeft.GetComponent<Camera>().fieldOfView =
targetFOV;
        }

        if (OnCameraSettingDone != null)
            OnCameraSettingDone.Invoke();
    }
  }

  public void SwitchToMainCamera()
  {
    DeactivateEntryCameraBottomLeft();
    ActivateMainCamera();
  }

  public void DeactivateEntryCameraBottomLeft()
```

```
    {
        if (entryCameraBottomLeft != null)
        {
            entryCameraBottomLeft.SetActive(false);
        }
    }
    public void ActivateMainCamera()
    {
        if (mainCamera != null)
        {
            mainCamera.SetActive(true);
        }
    }

    public void DestroyEntryCameraBottomLeft()
    {
        if (entryCameraBottomLeft != null)
        {
            Destroy(entryCameraBottomLeft);
        }
    }

    public void DestroyEntryCameras()
    {
        DestroyEntryCameraTopView();
        DestroyEntryCameraBottomLeft();
    }

    public void OnSkipButtonPress()
    {
        skipPressed = true;
    }
}



using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System;

public class MessageManager : MonoBehaviour
{
    public TextMeshProUGUI messageText;
    public Image messageBG;
    public Button messageButton;

    public event Action MessageClicked;
    public void DisplayClickableMessage(string message, Vector2 bgSize = default,
Color? bgColor = null, int? fontSize = null, Color? fontColor = null, Vector2 position
= default)
    {
        EnableInputDetection();
        messageButton.gameObject.SetActive(true);
        messageText.text = message;

        messageBG.rectTransform.sizeDelta = bgSize == default ?
messageBG.rectTransform.sizeDelta : bgSize;

        messageBG.color = bgColor ?? messageBG.color;

        messageText.fontSize = fontSize ?? messageText.fontSize;

        messageText.color = fontColor ?? messageText.color;

        messageBG.rectTransform.anchoredPosition = position == default ?
messageBG.rectTransform.anchoredPosition : position;

    }

    private void EnableInputDetection()
    {
        messageButton.onClick.AddListener(HandleMessageClick);
    }

    private void HandleMessageClick()
    {
        HideClickableMessage();

        MessageClicked?.Invoke();
    }

    private void HideClickableMessage()
    {
        messageText.text = "";
        if (messageButton != null)
        {
            messageButton.gameObject.SetActive(false);
        }
    }

    public void DisplayMessage(string message, Vector2 bgSize = default, Color?
bgColor = null, int? fontSize = null, Color? fontColor = null, Vector2 position =
default)
    {
        messageButton.gameObject.SetActive(true);
        messageText.text = message;
        messageBG.rectTransform.sizeDelta = bgSize == default ?
messageBG.rectTransform.sizeDelta : bgSize;

        messageBG.color = bgColor ?? messageBG.color;

        messageText.fontSize = fontSize ?? messageText.fontSize;

        messageText.color = fontColor ?? messageText.color;

        messageBG.rectTransform.anchoredPosition = position == default ?
messageBG.rectTransform.anchoredPosition : position;

        DisableInputDetection(); // Remove the listener to disable message click
        UnregisterMessageClicked(); // Unregister the MessageClicked event
    }

    private void DisableInputDetection()
    {
        messageButton.onClick.RemoveListener(HandleMessageClick);
    }

    private void UnregisterMessageClicked()
    {
        MessageClicked = null;
    }

    public void HideMessage()
    {
        messageText.text = "";
        if (messageButton != null)
        {
            messageButton.gameObject.SetActive(false);
        }
    }
}
```

```
using UnityEngine;
using System.Collections;

public class PlayerTokenMovement : MonoBehaviour
{
    public Transform airWaypointContainer;
    public Transform waterWaypointContainer;
    public Transform landWaypointContainer;
    public GameObject diceRollerObject;
    public int playerSpeed = 5 ;
    private DiceRoller diceRoller;
    private string waypointType;
    private int currentPosition = 0;

    private void Start()
    {
        diceRoller = diceRollerObject.GetComponent<DiceRoller>();
        waypointType = null;
    }

    public void RollDice()
    {
        if (waypointType != null)
        {
            // Already rolled the dice for this turn
            return;
        }

        // Roll the dice using the DiceRoller script
        int[] diceResults = diceRoller.GetDiceResult();
        int diceResult1 = diceResults[0];
        int diceResult2 = diceResults[1];
        int sum = diceResult1 + diceResult2;

        if (sum == 2 || sum == 3 || sum == 5)
        {
            waypointType = "Air";
        }
        else if (sum == 4 || sum == 6 || sum == 11 || sum == 12)
        {
            waypointType = "Water";
        }
        else if (sum == 7 || sum == 8 || sum == 9 || sum == 10)
        {
            waypointType = "Land";
        }

        // Move the player token based on the dice roll
        MovePlayer(sum);
    }

    private void MovePlayer(int diceSum)
    {
        // Hide the dice
        diceRollerObject.SetActive(false);

        Transform targetWaypoint = null;

        if (waypointType == "Air")
        {
            targetWaypoint = airWaypointContainer.GetChild(currentPosition + diceSum);
        }
        else if (waypointType == "Water")
        {
            targetWaypoint = waterWaypointContainer.GetChild(currentPosition +
diceSum);
        }
        else if (waypointType == "Land")
        {
            targetWaypoint = landWaypointContainer.GetChild(currentPosition +
diceSum);
        }

        float travelTime = Vector3.Distance(transform.position, targetWaypoint.position)
/ playerSpeed;

        StartCoroutine(MoveTowardsWaypoint(targetWaypoint.position, travelTime));

        currentPosition += diceSum;

        if (currentPosition >= targetWaypoint.parent.childCount)
        {
            // Reached the end, turn ends
            EndTurn();
        }
    }

    private IEnumerator MoveTowardsWaypoint(Vector3 targetPosition, float
travelTime)
    {
        float elapsedTime = 0f;
        Vector3 startingPosition = transform.position;

        while (elapsedTime < travelTime)
        {
            float normalizedTime = elapsedTime / travelTime;
            transform.position = Vector3.Lerp(startingPosition, targetPosition,
normalizedTime);
            elapsedTime += Time.deltaTime;
            yield return null;
        }

        transform.position = targetPosition;
    }

    public void EndTurn()
    {
        waypointType = null;
        diceRollerObject.SetActive(true);
        // Activate end turn button or perform other end turn actions
    }
```

```
}




using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class Sound
{
    public string name;
    public AudioClip clip;
    public bool loop;
    public float volume;
    public float pitch;
}
```

Chapter: 5

Result and Discussions

After finalizing the game designs and creating the necessary assets, I encountered a challenge during the development phase. Specifically, I faced difficulties in implementing the functionality to move the player token in the game.

Moving the player token is a crucial aspect of the game mechanics, as it determines the progress and interaction of the players within the game. However, due to unforeseen technical limitations or complexities, I was unable to successfully implement this feature.

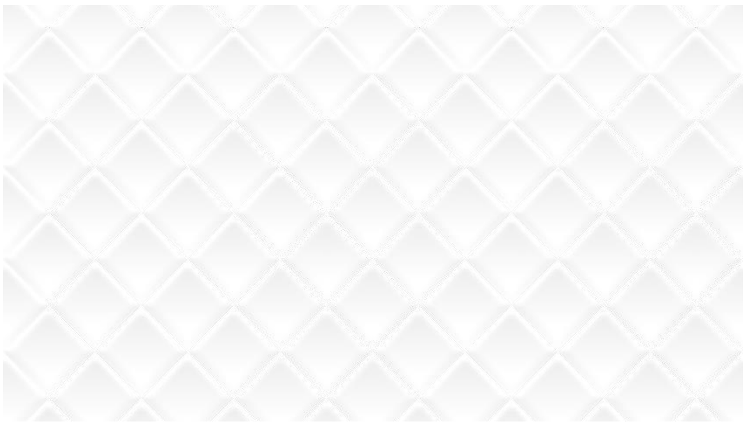
This hurdle impacted the overall progress of the game development, as the movement of the player token is a fundamental component that directly affects gameplay. Despite my efforts to overcome this obstacle, I was unable to find a suitable solution within the given timeframe.

As a result, further development and completion of the game were hindered, and the project could not reach its desired outcome. Moving forward, it would be necessary to seek additional resources, assistance, or expertise in order to overcome this challenge and continue the development process.

Although the inability to develop the movement of the player token was a setback, it is important to acknowledge the progress made in terms of game design, asset creation, and the overall planning and implementation within the Unity framework. These accomplishments provide a solid foundation for future iterations of the game or similar projects.

In conclusion, while I encountered difficulties in the development of the player token movement, the project highlights the importance of thorough planning, clear requirements, and potential collaboration or support from others to address technical challenges and ensure the successful completion of a game development project.

Here are some finalize game designs :



START
& GO

₹
BONUS

INDIA
₹ 7,500

USA
₹ 11,000

NORWAY
₹ 5,000

SINGAPORE
₹ 4,500

Chapter: 6

Conclusions and Future Scope

In conclusion, the game development project for "Traders" reached a certain level of progress, encompassing various aspects such as game design, asset creation, UI development, audio design, performance optimization, and menu functionality. Despite encountering challenges during the development process, valuable experience and knowledge were gained in the field of game development.

The project showcased the importance of thorough requirement analysis, system specification, and careful consideration of software and hardware requirements. It also highlighted the significance of user interface design, game mechanics, and audio elements in creating an engaging and immersive gaming experience.

Future Scope:

Although the project faced limitations and was unable to be fully completed, there is potential for future scope and enhancements. Some areas for further exploration and development include:

1. Implementing the missing functionality: Addressing the specific challenges faced during the project, such as the movement of player tokens, would be a priority for future development. Seeking assistance or exploring alternative approaches may help overcome these hurdles.

2. Expanding game features: Adding additional game modes, levels, and challenges could enhance the overall gameplay experience. This could involve introducing new trading mechanics, strategic elements, or multiplayer functionality to provide more depth and variety.

3. Enhancing visual and audio elements: Improving the visual quality of assets, incorporating dynamic lighting or special effects, and refining the audio design can contribute to a more immersive and visually appealing game environment.

4. Playtesting and feedback incorporation: Conducting playtesting sessions with a diverse group of players can provide valuable insights and feedback for refining gameplay, balancing mechanics, and identifying areas for improvement.

5. Platform expansion: Considering the possibility of porting the game to different platforms, such as mobile devices or consoles, can broaden its reach and accessibility to a wider audience.

6. Community engagement and updates: Establishing a community around the game, actively engaging with players, and providing regular updates and content expansions can help foster a dedicated user base and extend the longevity of the game.

By pursuing these future enhancements and addressing the challenges faced, "Traders" has the potential to evolve into a polished and engaging game that meets player expectations and offers an enjoyable gaming experience.