BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

COOPERATIVE GAME WITH SPACE-TIME DUALITY

KOOPERATIVNÍ HRA S DUALITOU ČASU A PROSTORU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **VOJTĚCH CZAKAN**
AUTOR PRÁCE

SUPERVISOR **Ing. TOMÁŠ POLÁŠEK**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

156985

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Czakan Vojtěch**
Programme:	Information Technology
Title:	**Cooperative Game with Space-Time Duality**
Category:	Computer Graphics
Academic year:	2023/24

Assignment:

1. Survey the current state of cooperative games.
2. Design a cooperative game with elements of Space-Time duality and describe it in a Game Design Document.
3. Implement the game by means of your choice.
4. Iterate implementation with continuous testing and feedback integration.
5. Evaluate the space-time elements of your game in a user study.
6. Present your results using a poster and a short video.

Literature:

- Koster, Raph. Theory of fun for game design. O'Reilly Media, Inc., 2013.
- Schell, Jesse. The Art of Game Design: A book of lenses. CRC press, 2008.
- Yao, Richard et al. Oculus VR Best Practices Guide. Online, 2014.
- Leap Motion, VR Best Practices Guidelines. Online, 2015
- Unity Learn. Unity, https://learn.unity.com/.
- Further sources according to the supervisor.

Requirements for the semestral defence:
Goals 1, 2 and a working game prototype.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Polášek Tomáš, Ing.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	9.11.2023

Abstract

This thesis details the development of a cooperative video game where two players, each in a separate dimension, manipulate time to progress through levels while contending with dangerous obstacles. The game was developed using the Unity engine, enhanced by Netcode for GameObjects and Unity Relay. The solution offers unique multiplayer experience that emphasizes communication, strategy, and timely coordination between players. The implementation demonstrated the feasibility of combining complex time manipulation mechanics with robust multiplayer support to enhance interactive gameplay.

Abstrakt

Tato práce detailně popisuje vývoj kooperativní videohry, ve které dva hráči, každý ve své dimenzi, manipulují časem k postupu úrovněmi a současně se potýkají s nebezpečnými překážkami. Hra byla vyvinuta pomocí enginu Unity, rozšířeného o Netcode for GameObjects a Unity Relay. Řešení nabízí jedinečný multiplayerový zážitek, který klade důraz na komunikaci, strategii a koordinaci mezi hráči. Implementace prokázala proveditelnost kombinace složitých mechanik manipulace časem s robustní podporou multiplayeru pro zlepšení herního zážitku.

Keywords

Unity, game, cooperative, multiplayer, time control, game design, Netcode for GameObjects, Unity Relay.

Klíčová slova

Unity, hra, kooperativní, pro více hráčů, ovládání času, herní design, Netcode for GameObjects, Unity Relay

Reference

CZAKAN, Vojtěch. *Cooperative Game with Space-Time Duality.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Tomáš Polášek

Cooperative Game with Space-Time Duality

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Tomáš Polášek. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Vojtěch Czakan
May 8, 2024

Acknowledgements

I would like to sincerely thank my supervisor, Ing. Tomáš Polášek, for all his advice and insightful comments. I also want to express my gratitude to all the testing participants for their valuable feedback.

Contents

Chapter 1

Introduction

This thesis explores the development of a cooperative game designed for two players using the Unity engine, emphasizing the essential role of communication for successful level completion. Each player exists in a distinct dimension that mirrors the other in layout but features unique obstacles. The primary challenge involves asynchronous movement mechanics: Only one player can move at a time. This is combined with dynamic time manipulation: As time progresses in one dimension, it pauses in the other. This setup increases the urgency and complexity of navigating obstacles and necessitates effective communication and strategic collaboration between players.

From years of gameplay and research in the cooperative gaming domain, I have recognized the potential benefits of integrating a subtle competitive element among players. While cooperation is critical for success, introducing competition adds an interesting layer of intrigue. To this end, a points system was devised to encourage players to achieve individually while still collaborating. However, this system also introduces a delicate balance in which deception and manipulation might occur, challenging players to manage trust and the risks of misjudgment that could lead to collective failure.

As the game progresses, it becomes more complex with the introduction of game modifiers that allow players to adversely affect each other's gameplay. These strategic choices increase the competitive tension further. Additionally, end-of-level guessing mechanics promote deception and strategic thinking, challenging players to outsmart each other.

This game concept was inspired by a personal passion for cooperative gaming and the identification of a market gap for two-player cooperative experiences that incorporate elements of sabotage and deception. By exploring this niche, this thesis aims to enrich the diversity and innovation in the gaming industry.

In Chapter 2, I will explore the complexities of cooperative and multiplayer dynamics, offering a detailed analysis of these key aspects. Chapter 3 will focus on game design specifics, including mechanics, settings, and level design. Chapter 4 will outline the implementation strategies and discuss the coding challenges encountered throughout the development process. Chapter 5 will present the results of the user study, analyzing player interactions and feedback to gauge the impact of the game. The thesis will conclude with Chapter 6, which will offer a comprehensive synthesis of the completed work and outline the future directions.

Chapter 2

Cooperative Games

The following chapter aims to provide an overview of the current state of cooperative games, focusing on their challenges, existing solutions, and the theoretical background necessary to understand the subject matter. This summary is not an exhaustive encyclopedia of cooperative games, but aims to give readers a comprehensive understanding of the field relevant to this thesis.

2.1 Challenges in Cooperative Games

Cooperative games, in which players work together towards a common goal, present unique challenges compared to competitive games. Some of the key challenges include the following.

2.1.1 Game Design

Game design stands as the foundational pillar of cooperative games, shaping their mechanics, dynamics, aesthetics, and overall player experience. It involves the intricate process of conceptualizing, planning, and executing the various elements and systems that constitute a game. Effective game design is paramount to crafting engaging, immersive and rewarding cooperative game experiences that resonate with players and foster long-term engagement.

Understanding Game Design Principles

Game design principles cover a broad spectrum of considerations [2]. **Core Mechanics** form the fundamental rules, actions, and interactions that define gameplay and drive player engagement. These mechanics are complemented by **Narrative Design**, which includes the integration of narrative elements such as characters, plot, and world-building, thereby enriching the game's narrative and immersing players in its universe.

Aesthetic Design of the game includes visual and auditory elements such as graphics, sound design, music, and art style, all contributing to the game's atmosphere and aesthetics. Additionally, **User Experience (UX)** Design focuses on enhancing the player experience through intuitive controls, clear feedback, seamless navigation, and an overall user-friendly design.

Lastly, **Balancing and Tuning** are essential to ensure that the game mechanics, difficulty levels, rewards, and challenges are well-adjusted to offer a challenging yet achievable and enjoyable experience. A particularly crucial aspect is the balance between player skills and game difficulty; when this ratio is well-maintained, players can achieve a flow state.

The flow state, as depicted in Figure 2.1, describes a condition where players become fully immersed in their activities under optimal conditions [7]. This immersive experience is crucial for engaging and retaining players, as it enhances their enjoyment and interaction with the game.

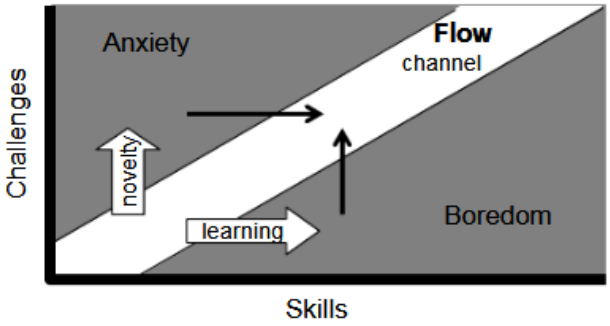

Figure 2.1: Three dimensions of experience (anxiety, flow, boredom): flow channel diagram. Taken from [6]

Challenges in Cooperative Game Design

The design of cooperative games presents unique challenges that require careful consideration and creativity to overcome. One significant challenge is **Team Dynamics**; this involves creating mechanics and systems that support effective collaboration, communication, and coordination between players [21]. These systems are crucial to ensure that all players can contribute and enjoy the game.

Another area of focus is the development of **Reward Systems**. These must be balanced and motivating, incentivizing cooperative play while fairly recognizing the contributions of each player [20]. This approach promotes a sense of achievement and satisfaction among all participants.

Narrative Cohesion is also vital [12]. It is important to ensure that the cooperative gameplay mechanics and the narrative align seamlessly. This alignment enhances immersion and keeps players engaged with the story and each other, providing a richer game experience.

Lastly, **Accessibility** is a key consideration. Games must be designed to be enjoyable and playable for a diverse audience [1]. This includes catering to various skill levels, preferences, and play styles to accommodate as many players as possible. Ensuring accessibility makes games inclusive and enjoyable for everyone, which is particularly important in cooperative gaming scenarios.

Key Considerations in Cooperative Game Design

In the realm of cooperative game design, several key considerations play a crucial role in crafting an engaging experience. At the forefront is **Player-Centric Design**, which emphasizes prioritizing player experience, engagement, and satisfaction throughout the design process [3]. This approach ensures that the game remains enjoyable and compelling for all players.

An **Iterative Design Process** is another critical aspect , where game development involves incorporating player feedback and continually refining the mechanics of the game

to enhance the overall experience [3]. This method allows designers to adjust and improve the game dynamically based on actual player interactions. The Iterative Design Process is visualized in Figure 2.3.

Community Engagement also holds significant importance [16]. By fostering a supportive and engaged gaming community through effective communication, updates, and community-driven initiatives, developers can create a more immersive and inclusive environment.

Furthermore, encouraging **Innovation and Creativity** in game design is essential [13]. Exploring new ideas, mechanics, and concepts not only differentiates the game from others, but also helps to captivate players, keeping them interested and involved in the gameplay.

Addressing Game Design Challenges

Addressing the complexities of game design requires a strategic and thoughtful approach. **Collaborative Design** plays a central role here, involving designers, developers, artists, writers, and other stakeholders in a process that leverages diverse expertise and perspectives [13].

Player Testing and Feedback are integral to this process, involving thorough testing sessions to gather feedback, identify potential issues, and refine game design elements [3]. This feedback loop is vital for ensuring that the game meets the high standards and expectations of its players.

Lastly, **Continuous Learning and Adaptation** are essential for staying informed about emerging trends, technologies, and best practices in game design [3]. This ongoing process of learning and adapting helps designers continuously improve and innovate, ensuring that their games remain relevant and engaging in a fast-evolving industry.

Conclusion

Game design serves as the cornerstone of cooperative games, influencing their mechanics, dynamics, aesthetics, and overall player experience. Embracing the principles of effective game design, addressing unique challenges, and adopting a player-centric approach are essential to create engaging, immersive, and rewarding cooperative game experiences that resonate with players, foster long-term engagement, and stand the test of time.

2.1.2 Communication and Coordination in Cooperative Games

One of the most critical aspects of cooperative games is the need for effective communication and coordination between players [21]. Unlike competitive games, where players often act independently to outperform others, cooperative games require players to collaborate closely to achieve shared objectives.

Importance of Communication

Effective communication is the cornerstone of successful cooperation in games. It enables players to share information, strategize, and coordinate their actions toward achieving common goals. In cooperative games, constant communication about intentions, actions, and observations is essential to ensure alignment and avoid misunderstandings.

Challenges in Communication

Effective communication is paramount in cooperative games, yet it faces several significant challenges that can impede successful collaboration.

Ambiguity in Communication: One of the primary issues is ambiguity. In-game situations often arise that require players to interpret and convey information accurately, which isn't always straightforward. This ambiguity can lead to misunderstandings and misalignment of strategies among players, potentially disrupting gameplay.

Noise Interference: Another significant hurdle is noise, especially in multiplayer online environments. External factors such as background noise or poor audio quality can severely hinder players' ability to hear and understand each other. Clear communication is essential for coordinating strategies and actions effectively, and these obstacles can dramatically impair that process.

Limited Time for Strategic Communication: Moreover, players often face constraints on the time and opportunities available for strategizing and communicating during gameplay. This scarcity of time can pressure players, leading to rushed decisions and inadequate communication. Highlighting the importance of game designs that incorporate more opportunities for players to discuss tactics and align their approaches seamlessly is crucial.

These challenges underscore the need for carefully considered game design elements that improve communication effectiveness and reduce potential barriers to collaboration among players.

Coordination Challenges

Coordination is as crucial as communication in cooperative gameplay, involving the intricate alignment of players' actions and strategies to effectively achieve shared objectives. This complex task encompasses several key challenges:

Role Specialization is a primary aspect, where players often assume specific roles or responsibilities within a game. Successful coordination demands that each player not only understands but also effectively executes their designated role. This execution must complement the actions and roles of other players, requiring a high level of mutual understanding and teamwork.

Timing and Synchronization are equally important, especially in real-time cooperative games. Precise synchronization of actions and timing is essential, as players must make split-second decisions that significantly influence the outcome of the game. This necessity calls for seamless coordination of movements and strategies.

Lastly, **Adaptability** is vital. As game dynamics change, players must quickly adapt and re-coordinate their strategies. This flexibility and rapid decision making are crucial to maintaining effective coordination despite new challenges or unexpected changes in the game scenario.

Conclusion

Communication and coordination are integral to the success of cooperative games. Despite their importance, these aspects pose significant challenges due to factors such as ambiguity, noise, cultural differences, role specialization, timing, and adaptability. Innovative game design approaches and technologies are required to effectively address these challenges and facilitate communication and coordination among players. Further exploration and research

in this area are crucial to enhancing the cooperative gaming experience and fostering better collaboration among players.

2.1.3 Skill Differences

Skill differences among players can significantly impact the dynamics and success of cooperative gameplay. When players have varying levels of skill, it can create imbalances within teams, affecting collaboration, communication, and overall gaming experience.

Understanding Skill Differences

Skill differences among players can significantly affect the dynamics of cooperative games, manifesting in various key aspects that impact gameplay in distinct ways.

Game Proficiency: One critical area where skill differences are evident is in game proficiency. Players who possess higher proficiency levels typically have a deeper understanding of the game mechanics, strategies, and tactics. This superior knowledge allows them to make more informed decisions and contribute more effectively to achieving team objectives.

Execution Skills: Another important aspect is execution skills, which encompass reflexes, accuracy, and coordination. Variations in these skills can greatly influence a player's ability to successfully perform specific tasks or actions within the game. For example, players with better execution skills are often able to handle complex maneuvers and respond quicker to changing game conditions, attributes that are crucial in fast-paced scenarios.

Strategic Thinking: Strategic thinking also plays a vital role in distinguishing skill levels among players. Those with advanced strategic thinking skills can anticipate opponents' moves, plan ahead, and coordinate with teammates more efficiently. Such abilities not only contribute directly to the team's success but also enhance the overall strategic harmony of the group.

Causes of Skill Differences

Several factors significantly contribute to the skill differences observed among players in cooperative games. Understanding these can help in designing more balanced and engaging game experiences.

Experience Level: Experience plays a pivotal role in determining a player's proficiency. Generally, players with more experience possess superior game knowledge and skills, honed through continuous gameplay and practice. This accumulated experience typically results in better performance and more effective contributions to team objectives.

Learning Curve: The complexity of the game also affects skill levels. Some games are inherently more challenging and require more time and effort to master than others. This variability in the learning curve can create disparities in skill levels among players, as some may advance more quickly than others, depending on their ability to learn and adapt to the game's challenges.

Accessibility and Resources: Accessibility to high-quality gaming equipment, as well as tutorials and guides, is another crucial factor. Players who have access to better resources can often develop their skills more rapidly and achieve higher levels of proficiency than those without such advantages.

Motivation and Commitment: The degree of motivation and commitment also significantly influences skill development. Players who are more motivated and committed

to improving their skills tend to invest more time and effort into their gameplay, which can lead to higher skill levels and a greater ability to contribute positively to cooperative gameplay.

Consequences of Skill Differences

Skill differences among players can have several significant implications for cooperative gameplay, impacting everything from team dynamics to individual player experiences.

Imbalanced Teams: One major consequence is the formation of imbalanced teams [21]. Significant skill disparities among players often result in less-skilled players relying heavily on their more competent teammates. This dependency can disrupt overall team cohesion and performance, as the balance of contribution becomes skewed.

Communication Challenges: Varying skill levels can also lead to communication challenges [21]. Less-skilled players might struggle to understand or execute strategies proposed by their teammates, leading to breakdowns in communication. In some cases, more skilled players may dominate the strategic planning and execution, effectively marginalizing less skilled team members and deteriorating team interactions.

Frustration and Disengagement: Furthermore, frustration and disengagement can become prevalent, especially among players who feel they are less skilled. These players may view themselves as burdens, leading to feelings of inadequacy and a diminished interest in continuing the game. Such emotional responses can adversely affect not only their own game experience, but also the morale and unity of the entire team.

Addressing Skill Differences

Addressing skill differences in cooperative games involves a multifaceted approach that includes game design innovations, effective community management, and targeted player education. Examples of games with addressing this problems well can be found in Figure 2.2.

Implementing Matchmaking Systems: A foundational strategy is the implementation of Matchmaking Systems. By considering players' skill levels, these systems can form balanced teams, which enhances the overall gaming experience by ensuring that teams are evenly matched, thus reducing the likelihood of imbalanced games.

Skill-Based Training: Skill-based training also plays a crucial role. Providing players with tutorials, guides, and training sessions tailored to their specific skill levels can significantly bridge the skill gap [3]. These resources enable less experienced players to improve their skills and better understand game mechanics, promoting a more inclusive gaming environment.

Community Support: Another key element is fostering a supportive gaming community where players can learn from each other, share experiences, and collaborate effectively [16]. This type of community encourages ongoing learning and improvement at all skill levels, enhancing cooperative gameplay.

Cooperative Game Mechanics: Designing game mechanics that necessitate cooperative play can also help mitigate skill disparities [21]. By making it impossible for a player to succeed alone, these mechanics encourage constant communication and teamwork, fostering a more balanced team dynamic where every player's contribution is essential.

Adaptive Difficulty Settings: The introduction of adaptive difficulty settings can align the level of challenge of the game with the player's proficiency [21]. This system can dynamically adjust based on various factors, such as player performance, strategies used,

or even the cumulative score. Adjustments may include changes to the game environment, obstacles, or even altering the capabilities of the character of the player, helping to maintain a balanced and engaging experience for all players.

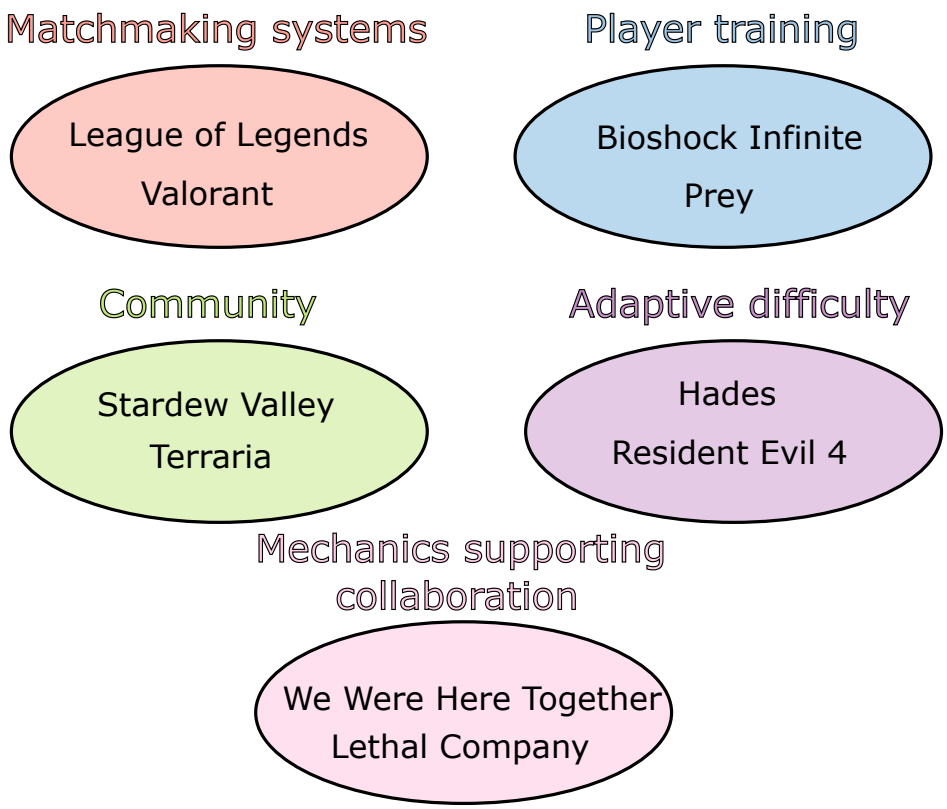

Figure 2.2: Examples of games representing different mechanics.

Conclusion

Skill differences between players present significant challenges in cooperative games, affecting team dynamics, communication, and the overall gaming experience. Understanding the underlying causes and consequences of skill differences is crucial for developing effective strategies to address this problem and promote a more collaborative and enjoyable gaming experience for all players.

2.1.4 Reward Disbalance

Reward disbalance represents a significant challenge in cooperative games, which affects the motivation, engagement, and overall gaming experience of players [20]. This issue arises when players perceive an unequal or unfair distribution of rewards for their contributions and efforts within the game. Understanding the complexities of reward imbalance is crucial for game designers, developers, and researchers who want to enhance the cooperative gameplay experience.

Understanding Reward Disbalance

Reward disbalance can significantly impact player experience and satisfaction in cooperative games, manifesting in several detrimental ways.

Unequal Distribution of Rewards: A common issue is the unequal distribution of rewards. Players may feel that their efforts are not appropriately valued, especially if they perceive that they are receiving fewer rewards compared to teammates who may have contributed less. This can create a sense of injustice and resentment within the team, negatively affecting team dynamics and overall game enjoyment.

Mismatched Effort and Reward: Another significant concern is mismatched effort and reward. Players who invest considerable effort and time to achieve game objectives might find that the rewards they receive do not match their input. This discrepancy can cause feelings of frustration and dissatisfaction, potentially discouraging players from investing similar efforts in future games.

Lack of Incentives for Cooperative Gameplay: Additionally, insufficient incentives for cooperative gameplay can undermine team collaboration [21]. Without appropriate rewards to encourage cooperative efforts, players might not feel motivated to participate in team activities. This lack of motivation can decrease teamwork and overall engagement in the game, diminishing the cooperative aspect crucial for the game's success.

Causes of Reward Disbalance

Several key factors can contribute to the occurrence of reward disbalance in cooperative games, affecting player satisfaction and overall game dynamics.

Poor Reward Systems: One major issue is the existence of ineffective or poorly designed reward systems. When these systems do not adequately recognize and reward players' contributions, significant disbalance can occur. This often leads to diminished player engagement and motivation, as players feel that their efforts are not being fairly rewarded.

Mismatched Skill Levels: Differences in player skill levels also play a critical role in contributing to the disbalance of rewards. These disparities can lead to uneven contributions during gameplay, which may result in perceptions of unfair reward distribution. Such perceptions can foster feelings of resentment and injustice within the team, affecting the overall harmony and effectiveness.

Lack of Transparency: Additionally, a lack of transparency about how rewards are determined can further aggravate disbalance issues. Without a clear understanding of the criteria or mechanisms behind the reward distribution, players can develop misconceptions and dissatisfaction, which undermines trust in the fairness of the game.

Economic Factors: Economic factors related to in-game economies and monetization strategies also play an important role. When these strategies prioritize revenue generation over player satisfaction, they can distort reward distributions and exacerbate the sense of disbalance among players. This prioritization can lead to a gaming environment in which the monetary investment outweighs the skill or effort as a determinant of reward, further eroding the balance and fairness of the game.

Consequences of Reward Disbalance

Reward disbalance can lead to several negative effects on cooperative gameplay, affecting player involvement and overall game health.

Reduced Motivation: One significant consequence is reduced motivation. Players who perceive a disbalance in rewards may feel demotivated, becoming less inclined to participate in cooperative activities. This lack of motivation can sap the energy and enthusiasm essential for engaging gameplay, potentially leading to a passive or disinterested player base.

Erosion of Trust: Furthermore, erosion of trust is a critical issue caused by perceived unfairness and unequal treatment. When players feel that rewards are not distributed equitably, trust among team members can erode. This deterioration undermines team cohesion and collaboration, which are crucial elements for successful cooperative gameplay.

Decreased Engagement: Additionally, decreased engagement is another serious outcome of the disbalance of rewards. Dissatisfaction with how rewards are distributed may lead players to reduce their playing time or even abandon the game altogether. This decline in engagement and retention can severely affect the game's community and longevity, as fewer players remain active and invested.

Addressing Reward Disbalance

Successfully addressing reward disbalance in cooperative games involves a combination of strategic game design, clear communication, and player-centric approaches to ensure fairness and maintain player engagement.

Implementing Balanced Reward Systems: A fundamental strategy is the design of reward systems that equitably recognize and compensate players' contributions. By ensuring fairness in rewards, games can significantly mitigate issues of imbalance. This approach helps maintain motivation and encourages all players to participate actively.

Clear Communication: Clear communication plays a critical role in managing expectations and increasing satisfaction. Providing players with transparent information about how rewards are determined, the criteria used, and what outcomes they can expect helps to build trust and understanding [3]. This transparency can prevent misconceptions and dissatisfaction related to the distribution of rewards.

Incorporating Player Feedback: Actively seeking and incorporating player feedback is another vital strategy [3]. By engaging with the player community and listening to their experiences and suggestions regarding rewards and gameplay, developers can identify potential problems early. This proactive approach is invaluable for making informed adjustments that enhance the game's reward dynamics.

Adaptive Reward Mechanisms: Lastly, implementing adaptive reward mechanisms that adjust rewards based on individual contributions, skill levels, and overall engagement can further promote fairness. These mechanisms ensure that all players, regardless of their initial skill or level of experience, have the opportunity to earn rewards that are commensurate with their efforts and dedication to the game.

Conclusion

Reward disbalance represents a significant challenge in cooperative games, affecting players' motivation, engagement, and overall gaming experience. Understanding the underlying causes and consequences of reward disbalance is crucial for developing effective strategies to address this issue and promote a more collaborative, engaging, and rewarding gaming experience for all players.

2.1.5 Strategy Complexity

Strategy complexity is a key element in cooperative games, profoundly influencing gameplay dynamics, player engagement, and overall gaming experience. This dimension refers to the complexity and depth of strategies that players must employ to achieve goals, collaborate effectively with teammates, and navigate challenges within the game environment. Understanding the nuances of strategy complexity is essential for game designers, developers, and researchers aiming to craft compelling, immersive, and strategically rich cooperative gameplay experiences.

Understanding Strategy Complexity

Strategy complexity in cooperative games involves several key facets that enrich the gameplay and challenge players to think critically and collaborate effectively.

Tactical Depth: At the core of strategy complexity is tactical depth. This refers to the range and intricacy of strategies that players can employ to outmaneuver opponents, complete objectives, and secure advantages. A high level of tactical depth allows for a more engaging and dynamic gameplay experience as players explore various tactics and approaches to achieve victory.

Collaborative Strategies: Effective coordination and collaboration among teammates are crucial for executing complex strategies and achieving shared goals [21]. This facet of strategy complexity requires players to communicate clearly and work together closely, significantly enhancing the cooperative aspect of the game.

Decision-making Complexity: Another important component is the complexity involved in making decisions. The decision-making process in these games includes multiple variables, uncertainties, and potential outcomes that players must consider. This complexity not only tests players' problem-solving skills but also their ability to anticipate and react to the changing dynamics of the game.

Adaptive Strategies: Lastly, adaptive strategies are essential for success in dynamic game environments. Players must continuously adapt and evolve their strategies in response to changing game conditions, opponents' actions, and unforeseen challenges. This adaptability adds a layer of depth to the gameplay, forcing players to rethink and modify their strategies as the game progresses.

Causes of Strategy Complexity

The level of strategy complexity in cooperative games is influenced by multiple interconnected factors, each adding depth and challenge to the gameplay.

Game Design Philosophies: These are fundamental in shaping how strategy complexity is approached. Designers' philosophies about crafting gameplay experiences directly impact the depth, challenge, and degree of player agency within the game. A focus on intricate gameplay ensures that strategies are not only complex but also rewarding and engaging for players.

Game Mechanics: This aspect also plays a crucial role. The design and implementation of mechanics, features, and systems are intended to encourage strategic thinking and decision-making. Effective mechanics challenge players to assess various situations and make strategic choices that significantly influence the game's outcome.

Team Dynamics: The dynamics among teammates are another significant factor [21]. The interplay and coordination required among team members necessitate effective com-

munication and collaboration, which can either enhance or complicate strategic decisions based on the team's cohesion.

Narrative Elements: Elements such as lore and world-building add additional layers of complexity to strategies and player interactions. By enriching the game's context, these narrative elements can profoundly influence strategic decisions, integrating deeply with gameplay mechanics and objectives.

Consequences of Strategy Complexity

The complexity of strategies in cooperative games can profoundly influence the gaming experience, shaping how players interact with the game and each other.

Enhanced Engagement: One positive effect of well-designed strategy complexity is enhanced engagement. When strategies are complex, they foster deeper investment, engagement, and satisfaction among players. This is because players are rewarded for their strategic thinking and creativity, which encourages them to delve deeper into the game mechanics and collaborate more effectively with their teammates.

Increased Learning Curve: However, high levels of complexity may also introduce a steep learning curve. This can be daunting for casual or inexperienced players, potentially leading to frustration or alienation among those who find the game too challenging to enjoy initially.

Dynamic Gameplay: Strategy complexity contributes to dynamic gameplay. It allows for evolving gameplay experiences that can change and adapt with each play session, promoting replayability and longevity [21]. This dynamism keeps the game fresh and exciting for players, encouraging them to return and explore new strategic possibilities.

Challenges in Collaboration: On the other hand, complex strategies can lead to challenges in collaboration [21]. These strategies often require high levels of coordination and clear communication among teammates. Without these, collaboration can become strained, potentially leading to misunderstandings or conflicts within the team.

Addressing Strategy Complexity

Successfully managing and designing strategy complexity in cooperative games requires a multifaceted approach that balances depth with accessibility and provides adequate support for players.

Balanced Design: Achieving the right balance between complexity and accessibility is crucial. It is essential to design game mechanics that are both challenging and approachable, ensuring that the game remains engaging for experienced players without being overly daunting for newcomers. This balance helps prevent frustration among less experienced players and keeps them actively engaged.

Tutorials and Guidance: Providing comprehensive tutorials, clear guidance, and accessible resources plays an essential role [3]. These supports are vital for helping players understand, learn, and master complex strategies. With adequate guidance, players of all skill levels can reach a competent level of play where they can fully engage with and enjoy the game.

Iterative Design: Continuously incorporating player feedback into game design is another important strategy [3]. This iterative process involves refining gameplay mechanics to enhance strategy complexity while also improving clarity and playability. By listening to the community and adapting the game based on their input, developers can ensure that the

game evolves in a way that maintains strategic depth. This process is visualized in Figure 2.3.

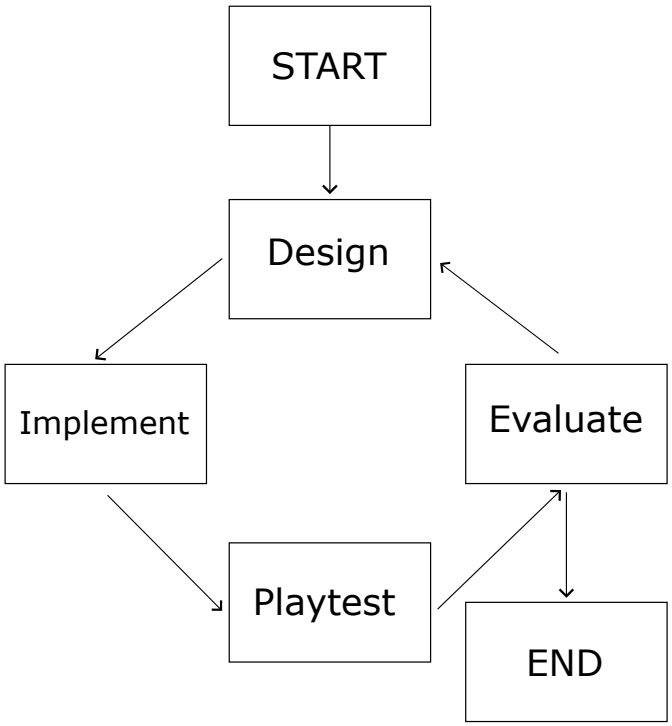

Figure 2.3: Iterative game design

Community Support: Lastly, fostering a supportive and collaborative gaming community is crucial. A community where players can share ideas, learn from each other, and collaborate on strategies significantly enhances the gameplay experience [16]. Such a community not only helps players feel more connected but also deepens their understanding and appreciation of the game's strategic elements.

Conclusion

The complexity of the strategy is a key aspect of cooperative games, influencing gameplay dynamics, player engagement, and overall gaming experience. Understanding the intricacies of strategy complexity and its impact on cooperative gameplay is essential to crafting compelling, immersive, and strategically rich gaming experiences that cater to diverse player preferences and skill levels.

2.2 Multiplayer Challenges in Games

Multiplayer games are a vital aspect of the gaming industry, offering players unique experiences by enabling interactive environments where individuals can play together, compete, or collaborate. The concept of multiplayer gaming encompasses various types and forms, from massively multiplayer online role-playing games (MMORPGs) to co-located console games, each offering unique benefits and challenges.

2.2.1 Types and Forms

Multiplayer games vary widely in their setup and gameplay. MMORPGs, such as „World of Warcraft,“ are expansive virtual worlds where players create avatars and participate in both social and combative activities [5]. Cooperative multiplayer games, on the other hand, offer a more accessible experience, typically focusing on time-flexibility and social relationships.

2.2.2 Social Dynamics

The social aspects of multiplayer games are a key component of their success. Players form strong friendships and emotional bonds in these environments, which can enhance their enjoyment [5]. Multiplayer games serve as online communication tools, facilitating the formation of communities and teamwork. Cooperative learning and collaborative gameplay are also prominent features, enhancing educational aspects.

2.2.3 Technical Challenges

In addition to managing network delays, multiplayer games face several other technical challenges. Ensuring real-time challenge balancing across players of varying skills without affecting individual player experiences is complex. Also, network delays can lead to discrepancies in player actions, causing gameplay synchronization issues.

2.2.4 Educational Potential

Multiplayer games are increasingly recognized as powerful educational tools. They foster collaborative learning and have been used in various learning environments. Serious games combine entertainment with educational content, enhancing motivation and engagement.

2.2.5 Conclusion

Multiplayer games are multifaceted, combining entertainment, social interaction, technical innovation, and educational potential. Their diverse forms and applications make them a unique and influential sector in the gaming industry and beyond.

2.3 Existing Solutions and Tools

Several approaches have been proposed to address the challenges in cooperative games:

2.3.1 Communication Tools

Effective communication stands at the core of successful cooperative gameplay, facilitating coordination, strategy planning, and team cohesion. As cooperative games continue to evolve and gain popularity, a plethora of tools and solutions have emerged to support and enhance communication among players. These tools range from in-game communication systems to third-party applications, each offering unique features, functionalities, and benefits tailored to the diverse needs of cooperative gaming communities.

In-Game Communication Systems

In-game communication systems are crucial for facilitating direct and effective interaction among players in cooperative games. These systems are designed to cater to various communication needs and preferences, ensuring that players can easily share information and strategize together in real time.

Voice Chat: One of the most common forms of in-game communication is voice chat. Many cooperative games include built-in voice chat functionality, which allows players to communicate verbally. This real-time communication tool is particularly useful for coordinating actions swiftly and effectively. Voice chat systems often come equipped with features like private channels, player muting, and audio settings adjustments, enhancing both clarity and immersion.

Text Chat: Another popular method is text chat. This system allows players to send written messages to each other, facilitating detailed strategic discussions and information sharing. Text chat can be particularly valuable in scenarios where voice chat might be impractical or disruptive. To enrich the communication experience, text chat systems typically include support for emoticons, text formatting, and a history of past messages, allowing players to express emotions and refer back to earlier conversations easily.

Ping and Marker Systems: Additionally, many games feature ping and marker systems. These tools enable players to place markers on the game world or highlight specific objects, locations, or enemies. Pings and markers serve as non-verbal cues that can convey essential tactical information and help coordinate team actions without the need for spoken or written communication. This type of system is incredibly useful for quick and clear communication, especially in fast-paced gaming environments.

Third-Party Communication Tools

While in-game communication systems are integral to player interaction in cooperative games, many players also turn to third-party tools to enhance their gaming experience further. These external platforms offer additional features and functionalities that improve communication and foster a stronger sense of community among players.

Discord: A standout choice for gamers, Discord is renowned for its robust communication options, including voice chat, text chat, and community engagement features. Players can create dedicated Discord servers with various channels for different games or topics, which enhances organization and simplifies the management of large groups.

TeamSpeak: Known for its high-quality voice communication capabilities, TeamSpeak is favored in large-scale cooperative games. It offers customizable servers, intricate channel structures, and detailed permission settings, allowing players to tailor their communication experience to their group's specific needs.

Mumble: As an open-source alternative, Mumble provides low-latency voice communication ideal for real-time strategic gameplay. Its unique feature, positional audio, simulates sound from various directions, enhancing spatial awareness and improving coordination and immersion in multiplayer environments.

General-Purpose Platforms: Platforms like Skype and Zoom, though not specifically designed for gaming, are sometimes used by gamers for their video conferencing and screen sharing capabilities. These features are invaluable for visual communication and collaboration on strategies, particularly useful during planning phases or when discussing complex scenarios.

Conclusion

Communication tools play a pivotal role in facilitating coordination, strategy planning, and team cohesion in cooperative games. As cooperative gaming continues to evolve, the landscape of communication tools and solutions is expected to diversify, innovate, and adapt to meet the evolving needs of players, fostering collaboration, enhancing engagement, and enriching the cooperative gaming experience.

2.3.2 Voice Chat Tools for Unity

Unity, one of the leading game development platforms, supports several robust voice chat plugins tailored to various needs and scenarios [18]. We will delve into more specifics of the engine in Section 4.1 Here is an overview of three popular voice chat systems: Photon Voice, Vivox, and Dissonance Voice. Each of these tools integrates seamlessly with Unity and offers unique features to developers.

Photon Voice

Photon Voice is a popular choice among developers for its efficient and scalable real-time voice chat service, specifically designed for multiplayer games and applications [14]. It operates on the Photon Realtime platform and is optimized for high-quality audio communication with minimal bandwidth usage and low latency, ideal for fast-paced multiplayer environments.

Features of Photon Voice include:

- Seamless integration with Photon servers, known for their stability and scalability.
- Support for positional audio, which creates realistic sound effects based on the player's location within the game world.
- Automatic detection of voice activity and cancelation of the echo to enhance the audio experience.

Vivox

Vivox is known for providing robust voice and text chat solutions on all major gaming platforms, ensuring versatility for cross-platform development [19]. Its reliability and high scalability have made it the choice for leading games in the industry, such as Fortnite and League of Legends.

Features of Vivox include:

- Capability to offer both 2D and 3D audio channels, allowing for positional sound effects and distance-based volume attenuation.
- Advanced audio features such as echo cancelation, noise reduction, and automatic gain control provide superior voice quality.
- Support for hundreds of simultaneous users in a single voice channel, suitable for large-scale multiplayer games.

Dissonance Voice Chat

Dissonance Voice is a comprehensive real-time voice chat plugin for Unity games and applications [8]. It is praised for its ease of integration and reliability, providing clear voice communication across various platforms and network conditions.

Features of Dissonance voice chat include:

- High-quality voice reproduction with low latency, making it ideal for interactive applications such as virtual reality.
- Compatibility with both Unity networking and various third-party networking solutions, offering flexibility in multiplayer game development.
- A wide range of supported audio codecs to optimize performance or quality based on the developer's preferences.

2.3.3 Multiplayer Tools for Unity

Unity offers a robust set of tools designed to facilitate the development of multiplayer games. These tools allow developers to implement a variety of multiplayer features, from matchmaking and communication to synchronization of complex game states across multiple players. Here is an overview of some of the key multiplayer tools available for Unity, which can help developers build immersive and interactive multiplayer experiences.

Netcode for Game Objects

Unity Netcode for GameObjects is the successor to UNet and is designed to provide a reliable and scalable framework for the development of multiplayer games [10]. It focuses on delivering high-performance networking specifically optimized for Unity games.

Performance and Scalability: The framework ensures smooth gameplay through real-time synchronization, crucial for maintaining the flow of fast-paced action. It also efficiently manages network traffic and is capable of supporting large numbers of players simultaneously, making it an excellent choice for developers looking to scale their games.

Developer Support: Unity Netcode for GameObjects is developer-friendly, offering comprehensive documentation and tutorials that help new users get started quickly. This support is vital for developers who need to troubleshoot issues or learn how to make the most of the framework's capabilities efficiently.

Photon Unity Networking (PUN)

Photon Unity Networking (PUN) is one of the most popular networking frameworks for Unity, known for its ability to simplify the process of adding multiplayer capabilities to games [15]. PUN stands out due to its scalability and ease of use, making it an excellent choice for developers across all skill levels.

Ease of Integration: PUN is celebrated for its easy setup, which allows developers to quickly integrate multiplayer features into their games. This ease of use is a significant advantage for developers looking to accelerate the development process without compromising quality.

Cross-Platform Compatibility: The framework supports all major platforms, ensuring that players can connect and play together regardless of the device they are using.

This cross-platform capability is crucial to creating an inclusive gaming environment in which everyone can participate.

Scalability: PUN excels in managing varying player counts efficiently, from small groups to very large communities without difficulty. This scalability makes it versatile and capable of supporting different types of multiplayer games, from intimate sessions to massive arenas.

Mirror Networking

Mirror is a high-level networking API for Unity, offering a powerful yet simple alternative to Unity's own low-level API [9]. It simplifies many aspects of networked game development, making it accessible even to those new to game programming.

Streamlined Simplicity: Mirror's simplicity helps developers quickly get up to speed with networked game development without sacrificing the power or flexibility needed for more complex projects. This approach ensures that even those new to networking can implement advanced features with ease, streamlining the development process while maintaining high-quality results.

Community-Driven Development: The API is actively maintained by a vibrant community of developers. This community-driven approach ensures that Mirror receives regular updates and improvements, reflecting the latest in networked game development practices.

Optimized Performance: Mirror is optimized for low bandwidth usage and fast performance, making it an excellent choice for developers concerned with maintaining high performance while minimizing resource consumption.

Chapter 3

Game Design Document

The objective of this thesis is to outline a comprehensive game design and subsequently implement a vertical slice of the game's mechanics. This chapter details the complete game design, covering various aspects such as mechanics, user interface, and objectives, providing a thorough view of the intended gameplay experience.

3.1 Introduction

This cooperative video game, developed as part of a Bachelor's thesis, challenges players to collaborate, communicate, and conquer time to successfully complete all levels. As players progress, they discover that the main adversary is not time itself but rather their own teammate. Throughout the game, tensions rise, and strategies shift, leading to a dramatic conclusion where only one player can emerge victorious.

3.2 Target System

This game is being developed primarily for PCs running the Windows operating system. Upon completion of the initial development phase, consideration will be given to creating ports for the latest Xbox and PlayStation consoles. However, this document will focus exclusively on the PC version of the game.

3.3 Development System

3.3.1 Software

This game is being developed using the Unity game engine 4.1, specifically version 2022. The primary programming language utilized within the Unity engine for this project will be C#.

3.4 Specification

3.4.1 Concept

The core objective of this thesis is to develop a team-based game that not only emphasizes essential skills such as communication and teamwork, but also introduces elements of

sabotage and deception among friends. This combination aims to create a challenging yet enjoyable gameplay experience.

3.4.2 Story

Although the story is not the central focus, it serves as an essential backdrop that enhances player understanding and engagement by explaining the events occurring within the game.

Introduction and Setup

The game begins with a mysterious scenario in which the player awakens in a room that appears identical to that of their character. Upon rising from the bed and sitting on a computer, the character inadvertently plays a video that installs a virus in their brain. This virus initiates a startling transformation where the player's body splits, creating a materialized shadow that acts as the second player. Each player exists in separate dimensions, perceiving the other as a shadow in their world.

Conflict and Quest

The two characters, driven by the need to eliminate the virus, decide to search for a cure initially believed to be in a pharmacy in the house. However, they soon discover that the necessary device is not there, prompting a journey to a hospital. It is on this journey that they learn a harrowing truth: Only one can survive after the virus is removed, transforming their cooperative efforts into a competitive struggle for survival.

Climax and Conclusion

The narrative reaches its climax in the hospital, the setting for the final showdown. Here, a player manages to get cured, emerging as the sole survivor and victor of the game, marking a dramatic end to the adventure.

Setting

The game is set in the present day but within an alternative reality that is technologically more advanced than our own. This setting allows for the integration of futuristic elements and technologies that are not available in our current timeline, providing a unique backdrop that enriches the gameplay experience.

3.4.3 Game Structure

The game is strategically divided into three main locations, each featuring distinct visual environments that enrich the player experience. These settings include:

1. **Apartment Building:** This is where the player character resides. It serves as the starting point of the game, introducing players to the basic mechanics and initial challenges.
2. **Outdoor Area (The City):** After navigating through the apartment, the players will venture into the city. This area expands the gameplay with more complex challenges and interactions set against an urban backdrop.

3. **Hospital:** The final location is the hospital. This climactic setting hosts the most difficult challenges and serves as the game's conclusion where the narrative and player competition come to a head.

In each of these locations, players will progress through several levels. Although the physical and visual design of these levels remains consistent for both players, the placement of obstacles will differ. This design choice ensures that the experience and strategy of each player will vary, focusing on the themes of cooperation and competition of the game.

3.4.4 Players

This game is exclusively designed for two players, each using separate computers. It cannot be played solo or on a single computer using split-screen mode. This design ensures that each player has a full view of the game environment from their respective perspectives, enhancing the gameplay experience and strategic depth.

To facilitate effective communication and collaboration, players are equipped with voice chat capabilities. This feature allows for real-time coordination and strategy sharing, which are essential for navigating the challenges and obstacles presented in the game.

3.4.5 Action

Throughout the game, players will face numerous moving obstacles that pose a constant threat to their progress. These obstacles are designed to test the players' avoidance skills and teamwork capabilities.

If a player collides with any obstacle, it results in both players losing and having to restart the level. This rule underscores the game's emphasis on cooperative gameplay and the need for synchronized team strategies.

Although the obstacles encountered are consistent in type for both players, their placement within the level varies. This variation ensures that each player's experience and challenges are unique, yet equally demanding. The obstacles are placed manually and are not generated dynamically. This careful placement ensures that there are no „blind spots“ — areas free from obstacles that might otherwise allow players an easy passage. Such a strategy guarantees that each section of the game consistently presents a challenge, maintaining a uniform level of difficulty and requiring continuous vigilance and strategic thinking from the players.

3.4.6 Objective

The objective of the entire game differs from the objectives of individual levels.

3.4.7 Level Objective

The objective of each level involves both players working collaboratively to navigate through various obstacles and reach a predefined goal. This goal could manifest itself in several forms, such as a door, an elevator, or similar functional elements within the game environment. Successful navigation and teamwork are crucial to advance to subsequent levels.

3.4.8 Game Objective

The overarching objective of the entire game is for players to compete to accumulate more points than their counterpart at the end of the game. Points can be earned in multiple ways, such as completing levels efficiently - often measured by being the first to finish a particular level. Additional methods for scoring points are detailed extensively in the Gameplay section.

3.4.9 Graphics

Although graphics are not the main focus for the development of this video game, a distinctive visual style is planned to enrich the gameplay experience. Initially, placeholder assets will be used during the early stages of development. The game will feature a 3D environment, utilizing vivid and abundant colors to create a sense of unreality. This vibrant color palette is intended to reflect the influence of the virus on the player's perception, contributing to the game's thematic depth.

Visual Details

The models used in the game will be low polygonal and minimalistic, aligned with the overall artistic direction. The visual style is inspired by games such as Sludge Life and High on Life, which are known for their striking and unconventional aesthetics. This approach is designed to enhance the surreal feel of the game, making the gameplay environment engaging and visually unique.

3.5 Gameplay

3.5.1 Levels

The game levels are categorized into three distinct environments, each offering unique challenges and aesthetics: the apartment building, the outdoor spaces (city), and the hospital. These categories are designed to provide varying gameplay experiences as players progress through the game.

Level 1

Name: Tutorial - Player's Room
Location: Apartment Building
Decription:
This initial level acts as a tutorial to familiarize players with the primary mechanics of the game, specifically time manipulation. It is set in the player character's room and aims to be concise and instructive.

The level is designed to be short and straightforward to ensure that players can grasp the basic concepts without feeling overwhelmed.

It features one or two simple obstacles to teach players how to navigate and use the time manipulation mechanic effectively. The sketch of the first level is illustrated in Figure 3.1.

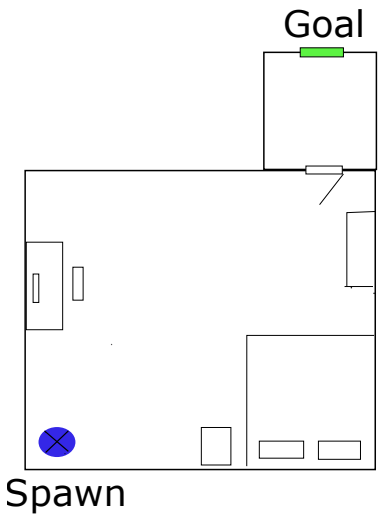

Figure 3.1: First level sketch.

Level 2

Name: Corridor
Location: Apartment Building
Decription:
This level is also straightforward, but more advanced than Level 1. It serves as a practice stage, allowing players to refine their understanding and application of the time manipulation mechanic introduced in Level 1. The sketch of the first level is illustrated in Figure 3.2.

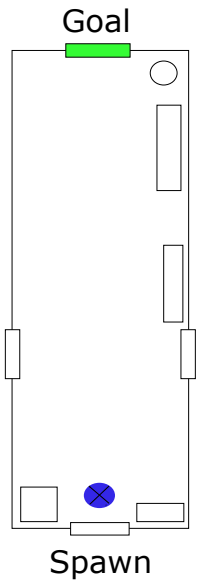

Figure 3.2: Second level sketch.

3.5.2 Time Manipulation

Time manipulation is the core gameplay mechanic of the game, intricately designed to enhance strategic collaboration between players. This mechanic operates on the principle that when both players remain stationary, the time within both realities slows down, allowing for a moment for strategic planning and decision making.

However, the moment one player initiates movement, time resumes its normal pace in the reality of the other player, and previously static obstacles begin to move, introducing immediate threats. This interplay requires players to alternate motion and stillness strategically.

The game restricts simultaneous movement, permitting only one player to move at a time. This mechanic passes control back and forth between players as they navigate through obstacles. Effective coordination and precise timing are critical, as players must synchronize their actions to avoid obstacles and advance through the levels successfully.

3.5.3 Cards with Effects

As the game progresses, players are introduced to card mechanics, which significantly improve gameplay through strategic decision making. This mechanic is divided into two distinct parts, each contributing to the competitive and cooperative elements of the game.The sketch of this mechanic user interface can be found in Figure 3.4.

Initial Selection and Effects Application:
At the beginning of each round, both players are presented with three cards drawn from a randomized card database. Each card possesses either a positive or negative effect, which will be secretly applied to the opposing player. Players must select one card at the start of the round, and the chosen effect is then applied to their co-player. Importantly, the effects are designed to be subtle, preventing the other player from immediately discerning the impact. Notably, negative effects award more points than positive effects, incentivizing strategic deception.

End-of-Level Guessing Game:
The second part of this mechanic unfolds at the end of each level, after both players have completed their respective challenges. Here, each player is shown the same three cards that their co-player viewed at the start of the round. The players must then attempt to guess which effect was originally chosen and applied to them. Correct guesses result in bonus points, while incorrect guesses yield no additional points. This element of the game fosters a deeper level of strategy, as players must balance the use of deception and manipulation throughout the round, aiming to mislead their co-player about their chosen effect.

Cards Effects

These are not all the card effects. More will be gradually added.

Negative	Positive
• More obstacles	• Less obstacles
• Faster obstacles	• Slower obstacles
• Decreased movement speed	• Increased movement speed
• Less energy	• More energy
• Slower energy regeneration	• Faster energy regeneration

3.5.4 Energy

Each player is equipped with an individual energy meter that depletes as they move. The rate of energy depletion accelerates during more energy-intensive actions such as sprinting or jumping. Should a player's energy meter fall to zero, they will „die“ and consequently must restart the level.

Energy Management and Gameplay Dynamics:
This energy mechanic is strategically implemented to encourage active participation from both players. It prevents a scenario where one player might remain in a safe spot devoid of obstacles, allowing the other player to complete the level unilaterally. Instead, players must alternate their movements to progress, as the only way to replenish energy is by allowing the other player to advance. This introduces a critical need to manage one's energy carefully and strategically coordinate movements with the co-player.

Strategic Implications:
The requirement for energy management adds a layer of strategic depth to the game. Players must not only navigate obstacles and coordinate with each other, but also continuously plan their energy usage to ensure that both can sustain their progress without exhausting their resources prematurely.

3.5.5 Points

As players progress through the game, they will discover the importance of not only collecting points but also ensuring that they collect more points than their opponent to avoid losing. The game integrates several methods through which players can accumulate points, enhancing both competitive tension and strategic depth.

Methods for Earning Points:

1. **Level Completion:** Players can earn points by being the first to complete a level. This method rewards quick thinking and fast execution, encouraging players to balance speed with accuracy.

2. **Card Mechanics:** Utilizing the card mechanics strategically plays a crucial role in point accumulation. Players who select negative effect cards receive more points,

reflecting the risk and strategic consideration involved in choosing such effects. Additionally, at the end of each level, if a player correctly guesses the card chosen by their co-player, they earn bonus points. This aspect not only adds a layer of strategic deception but also rewards players for their perceptiveness and memory.

Strategic Implications:
The point system is designed to foster a competitive environment in which players must continually make strategic decisions that could affect their position in the game. This system compels players to engage deeply with the game mechanics, from choosing their moves in each level to selecting and remembering cards, thus enriching the overall gameplay experience.

3.5.6 Controls

The game will be controlled by mouse and keyboard.

- Move forward
- Move backward
- Move left
- Move right
- Jump
- Sprint
- Crouch
- Voice chat

3.5.7 Obstacles

Obstacles within the game will vary significantly depending on the location, with each type designed to challenge players in unique ways. In addition, the speed and size of these obstacles will differ, increasing the complexity and requiring adaptive strategies from players. This variability ensures that each level presents distinct challenges, keeping the gameplay dynamic and engaging.

Apartment Building

- Paper planes
- Plates

Outdoors (city)

- People
- Car
- Motorbike
- Bike
- Skateboard

Hospital

- Hospital stretchers
- Needles

3.6 Front End

3.6.1 Intro

The game begins with an animated intro that sets the stage for the narrative and introduces the main gameplay mechanics. In this sequence, the playable character awakens and rises from their bed, walks over to a computer, and plays a video. This video inadvertently installs a virus into the character's brain chip, visually represented by a distorted effect on the screen, symbolizing the virus's immediate impact.

Following this, the character experiences a dramatic split, resulting in the creation of a materialized shadow version of themselves. The character attempts to interact with this shadow, reaching out to touch it, but finds it impossible to make physical contact. At this moment, moving obstacles begin to materialize, and time comes to a standstill, setting an eerie and suspenseful atmosphere.

This introductory sequence concludes with the onset of the tutorial, where players are gradually introduced to the game's controls and mechanics as they navigate through the newly formed challenges.

3.6.2 Outro

The game concludes with an animated outro that determines and reveals the winner. After navigating through numerous challenges, the players reach the pharmacy, the final destination, where they are each required to take a mechanical pill. This moment serves as the climax of the game.

Upon swallowing the pill, the winning player observes that the shadow of his co-player begins to gradually fade away, symbolizing his victory. Conversely, the losing player experiences their environment, and eventually themselves, slowly disappearing, culminating in a „Game Over“ screen. This visual representation not only emphasizes the finality of their loss, but also enhances the dramatic impact of the game's conclusion.

For the winning player, the game ends with a celebratory „You Won“ screen, acknowledging their success. This is immediately followed by the closing credits, providing a resolution to the game's narrative and acknowledging the contributors to the game's development.

3.6.3 Menus

The game features a streamlined menu system designed to facilitate easy navigation and minimal disruption to gameplay. The various components of this system include:

- **Main Menu:** The primary interface where players can start the game. This menu serves as the gateway to all the major game functions and settings. The screenshot of the main menu is shown in Figure 3.3.
- **Settings Menu:** Accessible from the main menu, this interface allows players to adjust various game settings according to their preferences, enhancing the user experience and accessibility.

- **Pause Menu:** Available during gameplay, this menu enables players to pause the game and make adjustments or access options without leaving the game environment.
- **Lobby System:**
  - **Host Interface:** The lobby where the host player can initiate a new game session. This interface displays a join code prominently, which is necessary for other players to enter the game session.
  - **Join Menu:** For players who are joining a game, this menu provides a simple interface to enter the join code provided by the host player and connect to the game session.

These menus are designed to be intuitive and user-friendly, ensuring that players of all skill levels can navigate the game settings and connections seamlessly.

Figure 3.3: In-game screenshot from main menu.

3.6.4 User Interface

The user interface in the game is designed to be minimalistic, focusing the player's attention primarily on the gameplay itself. The sole UI element is an energy indicator, which plays a critical role in conveying the player's remaining energy.

Energy Indicator Details:
The energy indicator is uniquely represented by a beating heart icon located in one of the upper corners of the screen. This choice symbolizes the vitality and urgency of the gameplay. The heart's behavior is dynamically linked to the player's energy consumption: as energy usage increases, the heart beats faster, enhancing the visual feedback for the player. In addition, when energy consumption reaches a critical level, the screen will subtly begin to turn red, simulating an increased sense of danger and urgency. This visual cue helps players gauge their remaining energy intuitively, allowing them to make strategic decisions based on their current status.

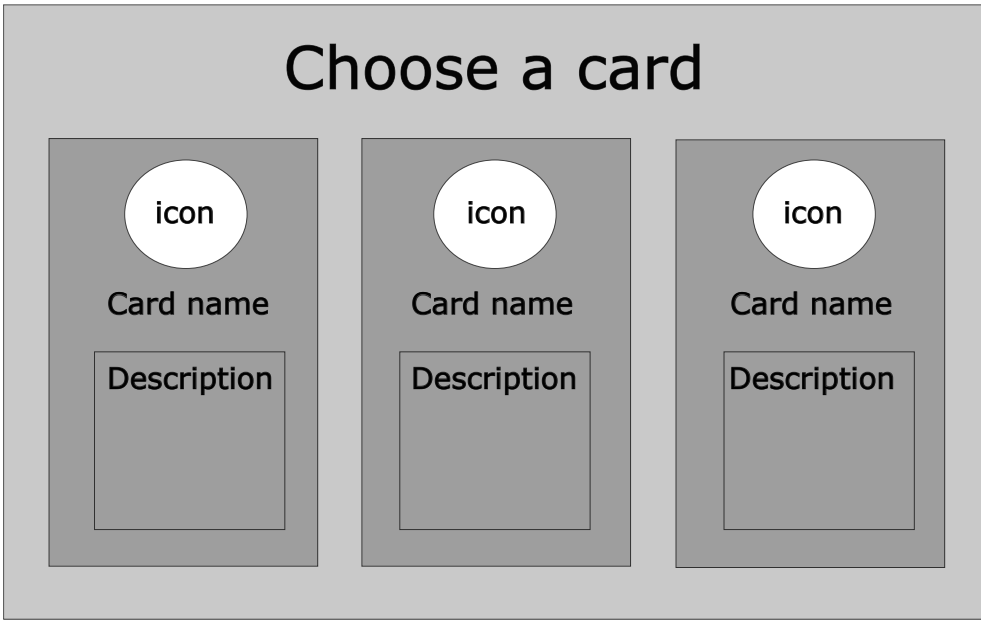

Figure 3.4: Sketch of the 'Choose a Card' user interface.

3.6.5 Player Shadow

The Player Shadow is a crucial feature designed to indicate the spatial presence of a player within their respective dimension. It provides a visual cue that informs each player of the other's location, despite the dimensional separation. The presence of the Player Shadow enriches the gameplay by enabling players to strategize based on the positional awareness of their counterpart in the alternate dimension. It also adds a layer of depth to the game, highlighting the duality of the players' existence - separate yet interconnected. By integrating Player Shadow into the game's mechanics, it becomes not just a feature for orientation but a fundamental component that bridges the gap between two parallel gameplay experiences.

3.7 Multiplayer

The multiplayer functionality of the game is designed to support two players. The hosting model is structured such that the game session is initiated and hosted directly on the computer of the player who starts the game.

3.7.1 Networking Framework

The game utilizes the Relay Unity package for its networking capabilities [18]. This choice is strategic, as it allows the game to be hosted on Relay servers without the need for dedicated servers. The diagram of a connection using a relay server is shown in Figure 3.5. Players can join a game session by entering a unique code generated by the host. This setup not only facilitates ease of connectivity, but also enhances the security and integrity of the game environment.

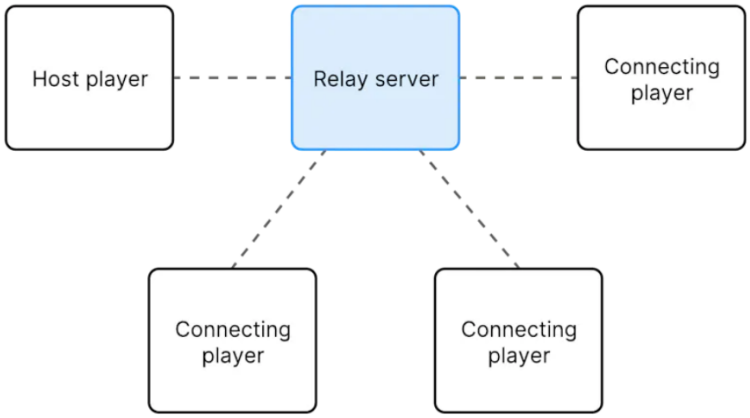

Figure 3.5: Diagram illustrating a multiplayer game network with a relay server connecting one host player to multiple connecting players. Diagram taken from [18].

3.7.2 Game State Management

Given that the game is primarily played among friends, there is an inherent level of trust among players regarding the modification of the game state. This reduces the need for stringent security measures typically required to prevent unauthorized client-side changes. The expectation that friends will not cheat allows for a more streamlined development process, eliminating the need for Remote Procedure Calls (RPCs). This simplification reduces the complexity of network coding and minimizes potential points of failure, making the game more robust and responsive.

3.7.3 Netcode for Game Objects

Netcode for Game Objects has been selected as the preferred networking solution for this project due to its seamless integration with Unity Relay [10]. This integration simplifies the networking process, particularly in setting up and managing multiplayer game sessions within Unity environments.

3.7.4 Voice Chat

For the voice communication component of the game, Vivox has been selected as the preferred solution due to its widespread popularity and ease of use [19]. Vivox is renowned for providing robust, high-quality voice chat services that are seamlessly integrated into gaming environments.

3.8 Testing

Testing is a critical component of the game development process, and it is structured in phases to ensure thorough evaluation and refinement of the game's mechanics and usability.

3.8.1 Initial Developer Testing

In the early stages of development, testing will be carried out primarily by the developer. This phase focuses on preliminary assessments of core game functionalities, such as the basic user interface and single-player elements, before integrating more complex features such as multiplayer capabilities and time manipulation mechanics.

3.8.2 Player Testing Phase

Once the foundational mechanics are implemented, the testing will expand to include external players. This phase is crucial for understanding the player experience in a real-world scenario. Key elements of the player testing phase include:

Setup and Initial Interaction:

Testers will receive the game with no prior instructions to assess the intuitiveness of the game setup and basic mechanics. This approach helps identify any initial hurdles or misunderstandings that could impede player engagement.

Observational Studies:

The behavior of the testers will be closely observed during gameplay to pinpoint areas of confusion or difficulty. This observation is vital for refining gameplay flow and ensuring that the game mechanics are understandable and accessible.

Feedback Collection:

After testing, feedback will be collected systematically using a questionnaire. This feedback is instrumental in gauging player satisfaction and identifying aspects of the game that require improvement.

Chapter 4

Implementation

This chapter presents an organized overview of the implementation sequence, detailing the step-by-step process of the project's development. The implementation was carried out using the Unity engine, which utilizes C# as its primary scripting language.

4.1 Unity Engine

Unity is a versatile game engine renowned for its comprehensive capabilities in developing both 2D and 3D video games, simulations, and interactive experiences. Its core strength lies in its flexibility and the extensive ecosystem of tools and assets available through the Unity Asset Store. This richness makes Unity an ideal platform for the efficient creation of diverse gaming projects.

4.1.1 Implementation Using Unity

The decision to use Unity Engine for this project was primarily driven by its all-encompassing development environment, which supports C# scripting, an extensive array of libraries, and built-in functionalities that streamline development processes. Unity's real-time rendering capabilities, combined with its sophisticated physics engine, facilitate the creation of realistic and interactive experiences, elements that are pivotal for the immersive aspects of the project.

In this project, Unity provided a cohesive environment in which various components such as 3D models, animations, and user interface elements could be seamlessly integrated. The engine's robust character controller and advanced networking library were particularly instrumental in developing the game's interactive and multiplayer features.

For this project, Unity provided a cohesive environment where components such as 3D models, animations, and user interface elements could be integrated seamlessly. The engine's characters controller and networking library.

4.2 Camera

The implementation of the first-person camera was prioritized as it is a fundamental aspect of player interaction and immersion in the game. The code provided controls both the positioning and orientation of the camera, responding dynamically to player input. This section will explore the structure and functionality of the camera control system.

4.2.1 Camera Positioning

```
1 cameraHolder.transform.position = transform.position + new Vector3(0f,
      1.9f, 0f);
```

Listing 4.1: Setting the camera to eye level in a game environment

This line positions the camera within the game world, specifically at the player's location, offset by 1.9 units vertically to simulate the height of the human eyes. This positioning is crucial to ensure that the camera provides a perspective that feels natural to the player, enhancing the realism of the game environment.

4.2.2 Input Handling

```
1 float mouse_x = Input.GetAxisRaw("Mouse X") * Time.unscaledDeltaTime *
      sensitivity;
2 float mouse_y = Input.GetAxisRaw("Mouse Y") * Time.unscaledDeltaTime *
      sensitivity;
```

Listing 4.2: Mouse input handling for camera control

These lines are integral to the camera's responsiveness, capturing horizontal and vertical mouse movements. The input is scaled by a sensitivity factor, which adjusts the camera's responsiveness to mouse movements, allowing for customization based on player preference or gameplay requirements.

The use of `Time.unscaledDeltaTime` is particularly noteworthy. Unlike `deltaTime`, which varies with changes in the game's time scale, `Time.unscaledDeltaTime` remains consistent even when the game's time scale is altered or set to zero. This consistency ensures that camera movements stay smooth and responsive, crucial for maintaining player control and immersion in scenarios where the game's time scale frequently changes, such as during slow-motion effects or gameplay pauses.

Furthermore, the use of `Input.GetAxisRaw` to capture mouse movements is critical for maintaining high precision. This method fetches the raw input from the mouse without any smoothing or filtering applied, which is vital for fast-paced gaming scenarios where instantaneous camera adjustments are necessary to maintain an immersive and competitive player experience.

4.2.3 Camera Orientation

```
1 _rotation_y += mouse_x;
2 _rotation_x -= mouse_y;
3 _rotation_x = Mathf.Clamp(_rotation_x, -90f, 90f);
4 cameraHolder.transform.rotation = Quaternion.Euler(_rotation_x,
      _rotation_y, 0f);
5 orientation.rotation = Quaternion.Euler(0f, _rotation_y, 0f);
```

Listing 4.3: Updating camera orientation based on mouse input

These lines update the camera's orientation based on the processed input data. The horizontal rotation (`_rotation_y`) affects both the camera and the player's orientation, enabling

yaw movements that are essential for navigation. The vertical rotation (`_rotation_x`) is clamped to prevent the camera from overturning, maintaining a realistic human field of view. This clamping also prevents disorientation and enhances the user experience by keeping the horizon level constant.

4.3 Movement

4.3.1 Initial Approach Using Unity's Physics Engine

Initially, movement in the game was facilitated through Unity's built-in 3D physics engine [18]. This method involved applying forces to rigid bodies using the `AddForce` function, which inherently relies on `Time.deltaTime` to calculate movement. This dependence on the game's time scale posed a challenge; `Time.deltaTime` varies with changes in time scale, compromising movement control especially when the time scale is reduced to zero.

4.3.2 Transition to Unity's CharacterController

Due to the need for consistent player movement regardless of time scale adjustments, I switched to using Unity's `CharacterController` [18]. This component simplifies movement mechanics by allowing movement that is constrained by collisions without the complexities of managing a rigid body. The `CharacterController` is not influenced by forces and is driven by the `Move` function, which can effectively utilize `Time.unscaledDeltaTime`. This approach ensures that player movement remains fluid and responsive, even when the game's time scale is altered or set to zero, thereby maintaining consistent gameplay experience and control.

4.3.3 Ground Check

```
1 _isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance,
      groundMask);
```

Listing 4.4: Ground check for player's position in game environment

Before applying movement or jump logic, the code checks if the player is grounded by using a sphere cast positioned at groundCheck.position and with a radius of groundDistance. This check determines whether the sphere intersects with any colliders on the groundMask layer, effectively ensuring that the player is on the ground

4.3.4 Movement Input and Calculation

```
1 _currentMovementInput = new Vector2(Input.GetAxisRaw("Horizontal"),
      Input.GetAxisRaw("Vertical"));
2 Vector3 moveDirection = _ctx.orientation.forward *
      _ctx.CurrentMovementInput.y + _ctx.orientation.right *
      _ctx.CurrentMovementInput.x;
3
4 _ctx.controller.Move(moveDirection * _ctx.speed * Time.unscaledDeltaTime);
```

Listing 4.5: Calculating and applying player movement input

Players provide input through the vertical and horizontal axes, which are captured raw for more precise control responses. The move direction is computed using the player's orientation, allowing movement relative to where the player is facing. The movement is then applied using `Time.unscaledDeltaTime` to ensure that the player's movement speed remains consistent, independent of any variations in the game's time scale.

4.3.5 Jump Mechanics and Formula

```
1 _ctx.Velocity = new Vector3(_ctx.Velocity.x, Mathf.Sqrt(_ctx.jumpHeight *
      -2f * _ctx.gravity), _ctx.Velocity.z);
```

Listing 4.6: Velocity calculation formula

The jump mechanic is activated when the player presses the jump button and is confirmed to be on the ground. The key to the mechanic is the calculation of the initial vertical velocity (v_0) that the character needs to achieve a specific jump height (h). The formula used is the following.

$$v_0 = \sqrt{h \cdot (2) \cdot g} \tag{4.1}$$

In this context, v_0 represents the initial velocity necessary for the jump, h is the desired jump height, and g is the acceleration due to gravity, which on Earth is approximately $9.81 m/s^2$. This value is negative in Unity as gravity pulls objects downward. The formula is derived from the kinematic equations that describe the motion of objects in free fall: [4]

$$v^2 = v_0^2 + 2gh \tag{4.2}$$

In the game's simulation, where v (the final velocity) is zero at the peak of the jump, the formula simplifies to calculate the necessary initial velocity for the ascent. By using the square root of the product of twice the jump height and the gravitational acceleration, we obtain the precise initial speed required to reach the desired jump height.

Because Unity uses a coordinate system where gravity is defined with negative values to represent the downward force, the gravity variable is set to -9.81. Moreover, to reflect this in the code, instead of using a positive 2 as the multiplier for gravity in the equation, a negative value is used, hence the „$-2f$“ seen in the code. This ensures that the jump mechanic functions correctly within the game's physics system, allowing for a predictable and realistic jump experience.

4.3.6 Gravity Application

```
1 _velocity += new Vector3(Velocity.x, gravity * Time.unscaledDeltaTime,
      Velocity.z);
2
3 controller.Move(Velocity * Time.unscaledDeltaTime);
```

Listing 4.7: Applying gravity to player's vertical velocity

Gravity is continuously applied to the player's vertical velocity, affecting the player's jump and fall. This application uses Time.unscaledDeltaTime to maintain consistent behavior under variable time scales, simulating realistic gravitational effects.

4.4 Hierarchical State Machine

Implementing a hierarchical state machine in a game can greatly enhance the control and management of different state levels of the game [11]. Here is an overview of how the hierarchical state machine is structured in this game, including its three levels: „time management," „grounded" and „movement states," along with an outline of the essential scripts that define the state machine. State machine diagram can be seen in Figure 4.1.

4.4.1 Time Management (root level)

The root level of the state machine, 'Time Management', oversees all time-related controls within the game, comprising four primary states:

1. **Time Slowed:** When both players are present, allowing full movement capabilities.
2. **Time Stopped:** Player can move while the game time is halted.
3. **Time Moving:** Limited to camera movement, no player motion allowed unless transitioning to „Time Slowed."
4. **Game Paused:** Complete game pause initiated by a player, typically by pressing Esc, stopping all movement and enabling menu interaction.

   The transitions between these states are controlled via network variables. For example, when one player moves, the state switches to „Time Stopped" for the mover and to „Time Moving" for the other. When the moving player stops, both players' state switches to „Time Slowed". „Game Paused" state can be entered independently.

4.4.2 Grounded and Jumping

These are direct substates under „Time Management." The transition to „Jumping" is triggered by the player pressing the spacebar, and it persists until the character lands and reverts to the „Grounded" state.

4.4.3 Movement States

Under „Grounded" and „Jumping," substates govern detailed player movements. There are currently two:

1. **Idle:** For camera navigation and standing still.
2. **Moving:** Activated when the player initiates movement.

Both „Grounded" and „Jumping" states can transition into any of the „Movement States," enabling a full range of actions even while in the air.

4.4.4 State Machine Scripts

The state machine is built upon three fundamental scripts:

1. **PlayerStateMachine:** The `PlayerStateMachine` is the central controller for player states within the game, inheriting from `NetworkBehaviour` to manage state transitions and behaviors across a networked environment. It stores all necessary player

variables and initializes a state factory to create different player states as needed. This setup allows for dynamic state management based on gameplay conditions. Unlike typical `MonoBehaviour` classes, concrete states in the state machine inherit from `PlayerBaseState` and do not directly invoke `MonoBehaviour`'s `Update` method. Instead, the state machine manually calls the `Update` method for the current state and any substates, ensuring that all state-specific logic is processed efficiently and encapsulated properly.

2. **PlayerStateFactory:** This class functions as a factory for creating player state instances, holding a reference to `PlayerStateMachine` to provide context for each new state instance it generates.

3. **PlayerBaseState:** An abstract base class defining the core structure for all player states in the game. Concrete states like „Idle," „Walk," „Jump," etc., inherit from this class and implement its abstract methods to define their behavior. The code for `PlayerBaseState` can be found in the Appendix A of this document.

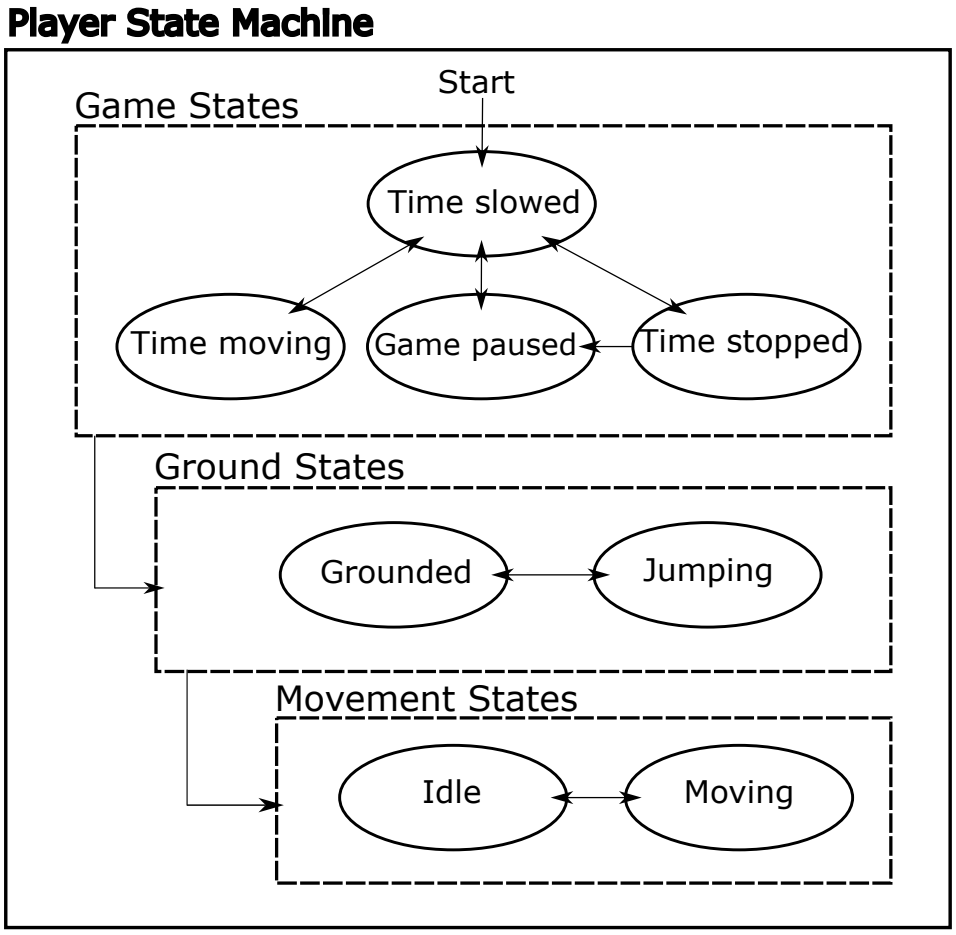

Figure 4.1: Hierarchical state machine diagram.

4.5 Multiplayer

Multiplayer functionality is implemented using Netcode for GameObjects. The initial package installed is Relay (version 1.0.5). The Unity project needs to be linked to Unity Cloud, which can be done through Project Settings under Services. Next, go to Unity Cloud, select the project, navigate to the Multiplayer tab, and enable Relay [17].

4.5.1 Relay

Relay is implemented in the `CreateRelay` and `JoinRelay` scripts. The first step is to initialize Unity services and authenticate the user, achieved using Unity Authentication [18]. Unity Authentication allows for anonymous user sign-in. In the `JoinRelay` script, users are signed in through the Start method, which is called from the main menu. The code snippet for this process looks as follows:

```
1 await UnityServices.InitializeAsync();
2 if (!AuthenticationService.Instance.IsSignedIn)
3 {
4     await AuthenticationService.Instance.SignInAnonymouslyAsync();
5 }
```

Listing 4.8: Initializing Unity services and handling user authentication for relay setup

The code includes a condition to check if the user is already signed in, accounting for cases where the user returns to the main menu from the game while still signed in.

The next step is to create the actual relay. This involves creating an allocation on an available relay server:

```
1 Allocation allocation = await
    RelayService.Instance.CreateAllocationAsync(1);
```

Listing 4.9: Creating a relay allocation

Here, the allocation sets the number of players who can join the server, specified in the first parameter, which is set to 1 in this example.

Subsequently, you need to retrieve the join code and display it to the player:

```
1 string joinCode = await
    RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);
```

Listing 4.10: Retrieving the join code for player connection

After this, in the `JoinRelay` script, the player can join the server using:

```
1 JoinAllocation joinAllocation = await
    RelayService.Instance.JoinAllocationAsync(joinCode);
```

Listing 4.11: Joining the relay server using the retrieved join code

4.5.2 Netcode for GameObjects

After setting up the relay allocation, the next step is to open a connection using Netcode for GameObjects and its underlying protocol, Unity Transport [10]. This involves installing

the necessary packages: Netcode for GameObjects, Multiplayer Tools, and Unity Transport. The `NetworkManager` object is created and set to `DontDestroyOnLoad`, making it persistent throughout the game. More about persistent objects can be found in Section 4.7. Subsequently, it is necessary to add the `NetworkManager` component to this object.

In the `NetworkManager` inspector, the network transport layer is set to Unity Transport. All important elements of the inspector are highlighted in Figure 4.2. The Unity Transport component is configured to use the Relay Unity Transport protocol in inspector. The script then connects Relay with Transport using:

```
1 RelayServerData serverData = new RelayServerData(allocation, "dtls");
2 NetworkManager.Singleton.GetComponent<UnityTransport>()
      .SetRelayServerData(serverData);
3 NetworkManager.Singleton.StartHost();
```

Listing 4.12: Configuring Netcode for GameObjects with Unity Transport for relay-based multiplayer setup

Here, Relay Server Data is initialized with the relay server allocation and configured for DTLS, which adds an extra layer of security. The `StartHost` method initiates the host, while `StartClient` can be used to start a client.

To prepare the player object prefab, add the `NetworkObject` component and create a network prefab list, adding the player prefab to it. Next, assign the player prefab to `NetworkManager`'s Player Prefab element in the inspector, enabling automatic player spawning. Lastly, add the created network prefab list to the `NetworkManager`'s Network Prefab Lists element in inspector. This allows the object to be shared across multiple `NetworkManagers`.

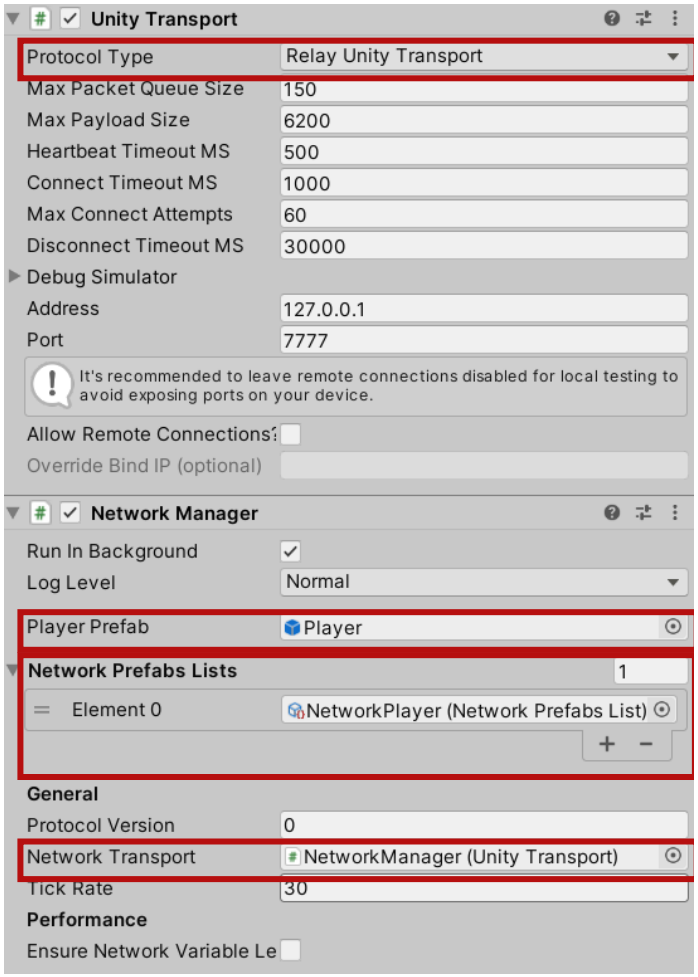

Figure 4.2: Screenshot of `NetworkManager` inspector with important elements highlited in red.

4.5.3 Synchronization

To manage player movement synchronization, add a check to ensure only the player's input is processed:

```
1 if (!IsOwner) return;
```

Listing 4.13: Skipping script execution if not the player owner

Next, add `ClientNetworkTransform` to player prefab, as described in section 4.6.3, to synchronize player movements across the network.

4.6 Player Shadow

In the game, players are situated in separate but parallel dimensions, represented as different scenes within Unity's environment. The scenes, labeled as `level1_1D` and `level1_2D`, are identical in layout but diverge in obstacle placements. A screenshot of the player's shadow can be seen in Figure 4.3.

4.6.1 Scene Segregation and Player Spawning

Each player is spawned into a designated scene - the first player in '1D' and the second in '2D'. Despite being in separate scenes, it's vital for players to be aware of each other's positions to maintain the cooperative and competitive elements of the game.

4.6.2 Visual Representation Through Shadows

To bridge the dimensional gap, I employ a „shadow" representation system. The „shadow" is a non-interactive placeholder within each scene that represents the other player. It is a networked object that mirrors the player's movements without possessing any physical properties, such as colliders, that would allow for interaction with scene elements.

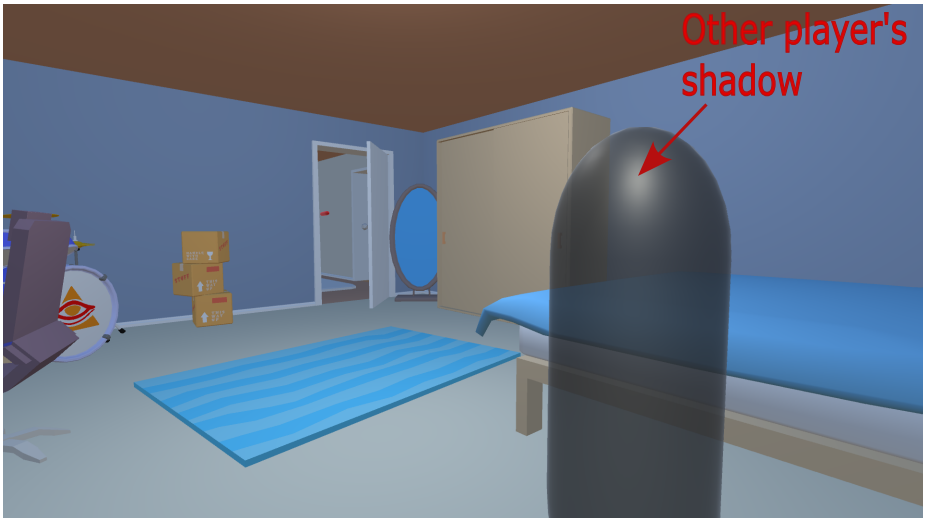

Figure 4.3: Screenshot of player's shadow.

4.6.3 Coordinate Sharing and Synchronization

Synchronization between player and shadow coordinates in parallel dimensions is achieved using the `ClientNetworkTransform` script from the Netcode for GameObjects package [10]. This script facilitates client-side game state changes, including position updates, reducing the dependency on the host for synchronization tasks. The player object is equipped with this script, configured to synchronize position data exclusively.

4.7 Scene Management

The inherent design of the Netcode for GameObjects' scene management expects players to reside within the same Unity scene. However, my game's design involves multiple dimensions, each represented as a separate scene ('1D' and '2D'), necessitating a different approach to scene management.

```
1 UnityEngine.SceneManagement.SceneManager.LoadScene("LevelX_XD");
```

Listing 4.14: Loading a new scene based on player

This line of code is responsible for placing players in their respective scenes and varies depending on whether a player is joining or hosting:

1. **Joining Players:** Upon joining, a player is transitioned to the appropriate scene using the `MovePlayerToScene()` method, which is used in `JoinRelay.cs` script.

2. **Hosting Players:** The player hosting the game is immediately transferred to the '1D' scene upon selecting „Host game“ in the Main Menu.

Persistent Objects Across Scene Transitions

To ensure that critical objects (such as player objects, Vivox for voice communication, and NetworkManager) persist across scene loads and reloads, `Instance` scripts are used. These scripts create instances of objects and employ Unity's `DontDestroyOnLoad` method [18] to prevent these objects from being destroyed during scene transitions.:

```
1 DontDestroyOnLoad(gameObject);
```

Listing 4.15: Preserving objects between scene transitions

This approach ensures the continuity of crucial gameplay components and uninterrupted network functionality throughout the game.

4.8 Vivox

First, it was necessary to integrate Vivox into the project using Unity Cloud, followed by installation through the Unity Package Manager [17]. To properly utilize the Vivox SDK, initialization is required with the following steps:

```
1 await UnityServices.InitializeAsync();
2 await AuthenticationService.Instance.SignInAnonymouslyAsync();
3 await VivoxService.Instance.InitializeAsync();
```

Listing 4.16: Initializing Vivox SDK for voice communication

When using Unity Game Services, the Vivox package automatically manages credentials and tokens [18]. Subsequently, it is essential for the player to log in using:

```
1 LoginOptions options = new LoginOptions();
2 options.DisplayName = UserDisplayName;
3 options.EnableTTS = true;
4 await VivoxService.Instance.LoginAsync(options);
```

Listing 4.17: Logging in to Vivox for voice communication

Once the player is logged in, the `JoinGroupChannelAsync` method can be utilized to join a channel [19], with `channelName` serving as the unique identifier for the channel:

```
1 await VivoxService.Instance.JoinGroupChannelAsync(channelName,
     ChatCapability.AudioOnly);
```

Listing 4.18: Joining a group channel for voice communication in Vivox

Chapter 5

Testing, and Suggestions for Improvements

This chapter will illustrate the methodologies employed to validate the game's functionality, including user studies. The aim is to provide a comprehensive overview of the verification process and its outcomes.

5.1 Approach to Testing

To ensure a robust evaluation of the game, two different testing approaches were used. These methods were designed to cover both the technical functionality of the game and its usability from a player's perspective.

5.1.1 Preliminary Testing

The initial phase of testing was carried out by the developer, focusing on each new feature as it was integrated into the game. Given that the game requires two players, certain features could not be thoroughly tested by a single individual. Consequently, another tester was occasionally brought in to help. This phase of testing was crucial for identifying and addressing fundamental issues such as movement discrepancies and collision detection failures.

5.1.2 Comprehensive Player Testing

Following preliminary testing, a more extensive testing phase was implemented. For this, early versions of the game were distributed to selected players. Accompanying these distributions was a brief guide, serving as a provisional tutorial to aid players in understanding the game mechanics. This guide is intended to be replaced in future versions by a short tutorial animation, which was not the focus of this thesis due to its emphasis on other developmental aspects.

Players were encouraged to engage with the game naturally, without further guidance or intervention. After completing their play sessions, participants were asked to complete a questionnaire. This questionnaire was designed to collect feedback on your experience, report any bugs encountered, and provide suggestions for improvements. This method of unguided testing was invaluable for observing authentic player reactions and gathering data on the game's intuitiveness and overall user experience.

5.2 User Study

The user study is structured into five distinct sections, each designed to collect specific types of data about players' experiences and interactions with the game.

5.2.1 Player Experience

The first section assesses the gaming experience of the players to tailor the complexity and interface of the game accordingly. Questions include:

- Do you play video games?
- How experienced are you with playing video games?
- On average, how many hours a day do you play video games?

This section helps to understand the player's familiarity with video games, which can influence their interaction with the game's mechanics and user interface.

5.2.2 User Interface

This section evaluates how intuitive and user-friendly the game's interface is, crucial to ensuring that players can navigate and utilize game features effectively. Questions asked are:

- How would you rate the clarity of the main menu?
- How easy was it to start playing the game?
- How clear was the location of your teammate?
- How easy was it to recognize that your movement is blocked?
- How visible were the obstacles?
- How clear was the goal of the level?
- Did you find any bugs in the user interface? If so, please describe.
- Comments and suggestions for improvements to the user interface.

Feedback from this section is intended to refine the user interface to ensure that it is not only aesthetically pleasing but also functional and straightforward.

5.2.3 Understanding the Main Mechanic: Time Manipulation

This section investigates the players' comprehension and engagement with the central game mechanic of time manipulation. Questions include:

- How well explained was the time control mechanism?
- How easy was it to understand the time control?
- How interesting did you find the time control mechanic?

- Was the first level sufficient for you to understand how to control time?
- Did you find any bugs in the time manipulation? If so, please describe.
- Comments and suggestions for improving the time control mechanics.

Feedback gathered here will help in assessing whether the time manipulation mechanic is accessible and engaging, and how it might be improved.

5.2.4 Game Balancing

The fourth section gathers information on the difficulty levels of the game and the effectiveness of in-game communication between players. Questions include:

- How challenging was it to communicate with your teammate?
- How difficult was the first level?
- How difficult was the second level?
- How difficult was the third level?
- Was the character's speed comfortable for you?
- Was the speed of the obstacles comfortable for you?
- Did you have enough time to plan you approach?
- Comments and suggestions on game balance.

This section aims to adjust the game's challenge to suit a broad range of players, ensuring it is neither too easy nor prohibitively difficult.

5.2.5 General Comments

The final section allows players to provide open-ended feedback on any aspect of the game:

- Did you encounter any additional bugs?
- Do you have any suggestions for improvements?

This open-ended feedback is crucial to identify issues that structured questions might not cover and to gather innovative ideas from players.

5.3 Results

The results of the questionnaire are presented in the following sections. Data were collected from seven different participants.

5.3.1 Player Experience

The experience levels of participants were rated on a scale from 1 to 5, where 1 signifies 'inexperienced' and 5 denotes 'very experienced'.

Player:	1	2	3	4	5	6	7
Do you play video games?	Y	Y	Y	Y	Y	Y	Y
How experienced are you with playing video games?	4	5	4	4	5	5	5
How many hours per day do you play video games?	5+	5+	1	1	5+	2	2

Table 5.1: Player Experience results

5.3.2 User Interface

The user interface was evaluated on a scale from 1 to 5, where 1 indicates 'not clear/not easy' and 5 represents 'very clear/very easy'.

Player:	1	2	3	4	5	6	7
How would you rate the clarity of the main menu?	5	5	5	5	5	5	5
How easy was it to start playing the game?	4	5	3	5	2	3	5
How clear was the location of your teammate?	3	4	5	4	5	3	4
How easy was it to recognize that your movement is blocked?	5	5	4	5	4	3	3
How visible were the obstacles?	5	5	5	5	5	4	3
How clear was the goal of the level?	5	5	5	5	5	5	4

Table 5.2: User Interface results

5.3.3 Understanding the Main Mechanic: Time Manipulation

The understanding of the main game mechanic, time manipulation, was evaluated using a scale from 1 to 5. On this scale, a rating of 1 corresponds to 'very hard/not interesting/not well explained' and 5 represents 'very easy/very interesting/very well explained'.

Player:	1	2	3	4	5	6	7
How well explained was the time control mechanism?	5	5	4	5	4	3	4
How easy was it to understand the time control?	4	5	3	1	1	3	4
How interesting did you find the time control mechanic?	5	5	5	5	5	4	5
Was the first level sufficient for you to understand how to control time?	Y	Y	Y	Y	Y	Y	Y

Table 5.3: Understanding time manipulation results

5.3.4 Game Balancing

The game balancing was evaluated using a scale from 1 to 5. On this scale, a rating of 1 corresponds to 'very hard/too slow' and 5 represents 'very easy/too fast'.

Player:	1	2	3	4	5	6	7
How easy was it to communicate with your teammate?	5	5	3	3	4	4	3
How easy was the first level?	5	5	5	5	5	4	5
How easy was the second level?	1	1	1	2	4	2	3
How easy was the third level?	4	1	4	1	2	2	1
Was the character's speed comfortable for you?	3	3	3	2	4	2	3
Was the speed of the obstacles comfortable for you?	3	3	4	5	3	3	5
Did you have enough time to plan you approach?	4	2	2	2	4	2	1

Table 5.4: Game balancing results

5.4 Improvements

Feedback collected from the user study highlighted several valuable improvement ideas. The participants suggested enhancing the time control mechanics by adding a visual indication to show when the other player stops moving and implementing a stamina feature to limit indefinite movement. In addition, there were calls to increase the number of levels and obstacles, improve the visibility of obstacles, and show their trajectories.

Most of these improvements align with planned future developments, such as adding stamina restrictions and expanding the game with more levels and obstacles. The suggestions to visualize obstacle trajectories and provide visual cues for player movement cessation are particularly intriguing and are being considered for integration in later stages of the game's development. These enhancements aim to enrich the gameplay and user interaction, ensuring a more engaging and balanced experience.

5.5 Conclusion

In general, the participants expressed general satisfaction with the game, particularly highlighting the time manipulation mechanic as the most enjoyable and innovative feature. Many found this aspect unique and expressed interest in revisiting the game in the future. However, there were notable concerns about the balancing of the game. Specifically, many participants found the obstacles too fast, limiting their ability to react effectively. In response, adjustments were made to the obstacle speeds immediately after the feedback sessions.

Despite positive reception, participants also encountered several bugs during the game, particularly in levels 2 and 3. Although some of these issues were resolved immediately after testing, others remain and are slated for future correction. Addressing these bugs is a priority as the game continues to develop, ensuring that the gameplay experience is both enjoyable and polished for all users.

Chapter 6

Conclusion

The goal of this thesis was to develop a cooperative game in which two players, each in a different dimension, manipulate time to navigate through levels and avoid dangerous obstacles. This objective was achieved successfully using the Unity engine, complemented by networking tools such as Netcode for GameObjects and Unity Relay, which streamlined development and facilitated an enjoyable multiplayer experience.

The thesis began with an analysis of the challenges in creating cooperative games, proposing potential solutions, and culminating in the creation of a detailed game design document. This document, which outlines the final state envisioned of the game, was continually refined throughout the development process. Initially, I utilized Unity's physics library to manage movement. However, this approach proved problematic due to its dependency on the time scale of the game, which conflicted with the time stopping mechanic. This necessitated a transition to Unity's Character Controller for movement handling.

During development, I also faced challenges with poorly readable code that was difficult to build on. To address this issue, I implemented a state machine. While this solution required substantial code refactoring, it ultimately streamlined development, significantly improving code modularity and readability. The greatest challenge of development was synchronizing player actions, such as scene transitions and menu interactions, particularly synchronizing the state machines of both players. Although I managed to achieve synchronization, there remains room for improvement.

From this project, I gained valuable experience working with multiplayer systems and learned the importance of employing robust coding patterns from the outset. Had I known the significance of this earlier, I would have prioritized establishing a sound coding framework at the beginning of the implementation.

Moving forward, my goal is to continue developing this game. My short-term objectives include resolving the remaining bugs identified during testing, adding more levels, and introducing a greater variety of obstacles. In the long term, I plan to implement the remaining game mechanics described in the game design document that were not included due to the time constraints of this thesis. This continued work will not only complete the game as originally envisioned but also enhance its depth and playability.

Bibliography

[1] Baltzar, P.; Hassan, L. and Turunen, M. Social accessibility in multiplayer games: Theory and praxis. *Entertainment Computing*, 2023, vol. 47, p. 100592. ISSN 1875-9521.

[2] Bryan, R. Understanding the Fundamentals of Game Design. *Medium* online, 2023. Available at: https://medium.com/@richardbryan2242/understanding-the-fundamental-of-game-design-8ced9daaaa5b. [cit. 2024-04-19].

[3] Charles, D.; McNeill, M.; Mcalister, M.; Black, M.; Moore, A. et al. Player-centred game design: Player modelling and adaptive digital games. *Proceedings of DiGRA 2005 Conference: Changing Views - Worlds in Play*, january 2005.

[4] cnxuniphysics. *University Physics Volume 1* online. 2016. Available at: https://pressbooks.bccampus.ca/universityphysicssandbox/. [cit. 2024-04-24].

[5] Cole, H. and Griffiths, M. Social Interactions in Massively Multiplayer Online Role-Playing Gamers. *Cyberpsychology behavior : the impact of the Internet, multimedia and virtual reality on behavior and society*, 2007, 10 4, p. 575–83.

[6] Cowley, B.; Charles, D.; Black, M. and Hickey, R. Toward an understanding of flow in video games. *Computers in Entertainment*, 2008, vol. 6, no. 2, p. 1–27.

[7] Csikszentmihalyi, M. and Lebuda, I. A Window Into the Bright Side of Psychology: Interview With Mihaly Csikszentmihalyi. *Eur J Psychol*, 2017, vol. 13, no. 4, p. 810–821.

[8] Dissonance Voice Chat. online, 2023. Available at: https://placeholder-software.co.uk/dissonance/docs/. [cit. 2024-04-23].

[9] *Mirror Networking Documentation* online. 2023. Available at: https://mirror-networking.gitbook.io/docs. [cit. 2024-05-01].

[10] Netcode for GameObjects. online, 2024. Available at: https://docs-multiplayer.unity3d.com/netcode/current/about/. [cit. 2024-04-24].

[11] Nystrom, R. *Game Programming Patterns*. Genever Benning, 2014.

[12] Ojeda, C. M. *In The Game: An Exploration of the Concept of Immersion in Video-Games and its Usage in Game Design*. 2007. Bachelor's thesis. Edith Cowan University. Available at: https://ro.ecu.edu.au/theses_hons/1298.

[13] Peter Zackariasson, M. W. and Wilson, T. L. Management of Creativity in Video Game Development. *Services Marketing Quarterly*. Routledge, 2006, vol. 27, no. 4, p. 73–97.

[14] Photon Voice Intro. online, 2023. Available at: https://doc.photonengine.com/voice/current/getting-started/voice-intro. [cit. 2024-04-23].

[15] *Introduction to Photon Unity Networking* online. 2023. Available at: https://doc.photonengine.com/pun/current/getting-started/pun-intro. [cit. 2024-05-01].

[16] Ruggles, C.; Wadley, G. and Gibbs, M. Online Community Building Techniques Used by Video Game Developers, 2005, vol. 3711, p. 114–125.

[17] *Unity Cloud* online. Available at: https://cloud.unity.com/. [cit. 2024-05-01].

[18] Unity Documentation - Unity Manual. online, 2024. Available at: https://docs.unity3d.com/Manual/index.html. [cit. 2024-04-24].

[19] Vivox Developer Documentation. online, 2023. Available at: https://docs.vivox.com/v5/general/core/5_23_0/en-us/Default.htm. [cit. 2024-04-23].

[20] Wang, H. and Sun, C.-T. Game Reward Systems: Gaming Experiences and Social Meanings. online, 2012. Available at: https://www.researchgate.net/publication/268351726_Game_Reward_Systems_Gaming_Experiences_and_Social_Meanings. [cit. 2024-05-02].

[21] Zagal, J. and Rick, J. Collaborative games: Lessons learned from board games. *Simulation Gaming - Simulat Gaming*, 2006, vol. 37, p. 24–40.

Appendix A

Player Base State Abstract Class

```
1  public abstract class PlayerBaseState
2  {
3      protected bool _isRootState = false;
4      protected PlayerStateMachine _ctx;
5      protected PlayerStateFactory _factory;
6      protected PlayerBaseState _currentSuperState;
7      protected PlayerBaseState _currentSubState;
8
9      public PlayerBaseState(PlayerStateMachine currentContext,
           PlayerStateFactory stateFactory)
10     {
11         _ctx = currentContext;
12         _factory = stateFactory;
13     }
14
15     public abstract void EnterState();
16
17     public abstract void UpdateState();
18
19     public abstract void ExitState();
20
21     public abstract void CheckSwitchState();
22
23     public abstract void InitializeSubStates();
24
25     public void UpdateStates()
26     {
27         UpdateState();
28         if (_currentSubState != null)
29         {
30             _currentSubState.UpdateState();
31             if (_currentSubState._currentSubState != null)
32             {
33                 _currentSubState._currentSubState.UpdateState();
```

```
34              }
35          }
36      }
37
38      protected void SwitchState(PlayerBaseState newState)
39      {
40          ExitState();
41          if (_isRootState)
42          {
43              if(_currentSubState != null) _currentSubState.ExitState();
44
45              newState.EnterState();
46              _ctx.CurrentState = newState;
47          }
48          else if (_currentSuperState != null)
49          {
50              _currentSuperState.SetSubState(newState);
51          }
52      }
53
54      protected void SetSuperState(PlayerBaseState superState)
55      {
56          _currentSuperState = superState;
57      }
58
59      protected void SetSubState(PlayerBaseState subState)
60      {
61          _currentSubState = subState;
62          subState.SetSuperState(this);
63
64          _currentSubState.EnterState();
65      }
66  }
```

Listing A.1: Abstract base class for player states