Faculty of Computer Science and Technology

DISASTER AWARENESS AND PREPAREDNESS-BASED GAME FOR COMMUNITY

Tiew Chuan Hong

Bachelor of Computer Science with Honours (Multimedia Computing)
2025

DISASTER AWARENESS AND PREPAREDNESS-BASED GAME FOR COMMUNITY

TIEW CHUAN HONG

This project is submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Computer Science with Honours (Multimedia Computing)

Faculty of Computer Science and Technology
UNIVERSITI MALAYSIA SARAWAK
2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE: **DISASTER AWARENESS AND PREPAREDNESS-BASED GAME FOR COMMUNITY**

ACADEMIC SESSION: SEM 2 24/25

_____

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☑	UNRESTRICTED	

Validated by

_____ _____

(AUTHOR'S SIGNATURE) (SUPERVISOR'S SIGNATURE)

DR MOHAMAD IMRAN BIN HJ BANDAN
Senior Lecturer
Network Computing Programme
Faculty of Cognitive Sciences and Human Development
Universiti Malaysia Sarawak

Permanent Address

No.34, Jalan Sungai Keramat 29,
Taman Klang Utama,
42100 Klang, Selangor.

Date: 23 June 2025 Date: 23 June 2025

Note * Thesis refers to PhD, Master, and Bachelor Degree
** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

DECLARATION

I hereby declare that this project and its contents are the result of my own original work. Any information, data, or material obtained from other sources has been properly cited or acknowledged where applicable. This thesis has not been submitted to any other institution for any academic qualification.

I further confirm that the work presented here was prepared independently, and any guidance or assistance received during the process has been duly acknowledged.

Signed,

……

Tiew Chuan Hong (82204)

Bachelor of Computer Science with Honours (Multimedia Computing) Faculty of Computer Science and Information Technology (FCSIT)

Universiti Malaysia Sarawak (UNIMAS)

23 June 2025

ACKNOWLEDGEMENT

I would like to express my gratitude to everyone who has contributed to the completion of this study.

Special thanks to my supervisor, Dr Mohamad Imran bin Hj Bandan, Senior Lecturer, Faculty of Computer Science and Information Technology (FCSIT) for the guidance and support throughout my studies. I am also thankful to Ts. Syahrul Nizam Junaini, for the comment and improvement for the studies.

Finally, I truly appreciate the management of Universiti Malaysia Sarawak for providing me with the opportunity and resources to complete my study. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

NGO	Non-governmental Organization
CBDRR	Community-Based Disaster Risk Reduction
VR	Virtual Reality
AR	Augmented Reality
ERD	Entity Relationship Diagram

ABSTRACT

This study focus on the design and development of a serious game titled Disaster Wise. The game aims to enhance disaster preparedness and awareness while promoting engagement and knowledge retention. It addresses the need for an engaging and educational tool to help users, especially younger individuals, understand disaster preparedness through interactive gameplay. The game was developed using Unity and incorporates core gameplay mechanics such as a turn-based system, card system, and action system. The development process followed Agile practices to create the game prototype. Educational content was integrated through card descriptions, in-game scenarios, and background visuals to support learning outcomes. Testing was conducted through functional testing, responsive UI testing, and user evaluation using a survey form. Feedback from 10 testers indicated above-average satisfaction, with positive responses regarding usability, gameplay experience, and educational value. While the prototype successfully met its core objectives, some limitations were identified, such as limited device responsiveness and content variety, which can be addressed in future improvements. Overall, the project demonstrates the effectiveness of serious games in delivering educational content and providing engaging user experiences.

Permainan Video Berasaskan Kesedaran dan Kesiapsiagaan Bencana untuk Komuniti

ABSTRAK

Kajian ini memberi tumpuan kepada reka bentuk dan pembangunan sebuah permainan video serius yang bertajuk Disaster Wise. Permainan ini bertujuan untuk meningkatkan tahap kesiapsiagaan dan kesedaran terhadap bencana sambil menggalakkan penglibatan serta pengekalan pengetahuan. Ini menangani keperluan terhadap alat pendidikan yang menarik dan interaktif bagi membantu pengguna, terutamanya golongan muda, memahami kesiapsiagaan bencana melalui permainan video. Permainan ini dibangunkan menggunakan Unity dan menggabungkan mekanik permainan teras seperti sistem berasaskan giliran, sistem kad, dan sistem tindakan. Proses pembangunan mengikut amalan pembangunan Agile bagi menghasilkan prototaip permainan. Kandungan pendidikan disepadukan melalui penerangan pada kad, senario dalam permainan, dan visual latar belakang untuk menyampaikan hasil pembelajaran. Pengujian dilaksanakan melalui pengujian fungsional, pengujian antara muka pengguna responsif, serta penilaian pengguna melalui borang soal selidik. Maklum balas daripada 10 penguji menunjukkan tahap kepuasan yang tinggi, dengan respon positif terhadap kebolehgunaan, pengalaman permainan, dan nilai pendidikan. Walaupun prototaip ini berjaya mencapai objektif utamanya, beberapa kekangan telah dikenalpasti seperti sokongan peranti yang terhad dan kekurangan variasi kandungan, ini boleh ditambahbaikan dalam versi akan datang. Secara keseluruhan, projek ini menunjukkan keberkesanan permainan serius dalam menyampaikan kandungan pendidikan dan menyediakan pengalaman yang menarik kepada pengguna.

CHAPTER 1

INTRODUCTION

1.1 Background

During the last decade, natural disasters have become more of a threat to communities worldwide, with incidents such as flooding, earthquakes, and hurricanes claiming numerous lives along with causing destruction and long-term socioeconomic implications. In 2023, a total of 399 natural hazard-related disasters were recorded in the Emergency Events Database, leading to 86,473 deaths and affecting 93.1 million people, with economic losses estimated at US$202.7 billion (Centre for Research on the Epidemiology of Disasters, 2024). The results indicate that improvements are needed in strategies and preparedness efforts to effectively mitigate the effects of disasters on both human lives and economies. Preparedness involves creating a comprehensive plan to ensure that all aspects of early and initial risk assessments are properly identified, and mechanisms for minimizing risks, as well as safeguarding lives and property, are effectively established and coordinated (Stikova, 2016).

Disaster preparedness in Malaysia remains limited and insufficient to address the range of natural hazards the country faces. Recent data from the INFORM Risk Country Profile 2025 further emphasizes the need for improved preparedness. Malaysia faces potential threats beyond just floods, including droughts, earthquakes, tsunamis, and epidemics, with an overall risk index score of 2.9 out of 10.0. This includes a hazards and exposure score of 2.6, a vulnerability score of 3.4, and a coping capacity score of 2.9, indicating that while the likelihood of major crises is low, the country's ability to respond

effectively is limited (INFORM, 2024). These scores reflect a lack of resilience, particularly in communities with limited resources, which leaves the population vulnerable to disaster impacts. The consequences of inadequate preparedness will lead to casualties and economic damage during such events. In 2021, floods in Malaysia resulted in 69 deaths, and in 2022, 27 individuals lost their lives to floods, with an additional 33 fatalities from landslides (Hannah Ritchie and Pablo Rosado, 2022).

The current state of disaster preparedness highlights significant knowledge gaps, particularly among younger generations, which compromises the ability to respond effectively to natural hazards. However, many traditional methods of learning about disaster preparedness and related knowledge, such as articles, speeches, and lectures, often fail to engage people, especially youth, due to a lack of motivation. This project introduces a game-based learning approach to learn disaster preparedness knowledge in a more accessible and engaging way. Shaffer, Halverson, Squire, and Gee (2005) define game-based learning as a type of gameplay with defined learning outcomes, a concept further explored by Plass, Homer, and Kinzer (2015). By combining entertainment with education, this game-based approach makes disaster preparedness knowledge both appealing and memorable, enhancing understanding among younger audiences.

1.2 Problem Statement

Disaster preparedness is crucial for minimizing the impact of natural hazards on communities and local economies. However, many people, especially younger generations, lack awareness and preparedness when it comes to disasters. This issue is particularly evident in Malaysia, where the relatively infrequent occurrence of major disasters creates a false sense of security. While Malaysia experiences fewer natural disasters compared to other

countries, events such as floods and landslides still occur, yet many individuals remain uninformed or unprepared to effectively handle these situations. A significant knowledge gap exists when it comes to understanding the causes of disasters and how to prepare for them. Traditional methods of disaster education, such as articles, speeches, and lectures, often fail to capture and retain younger audiences' interest and attention. These methods tend to be text-heavy and complex, making it difficult for youth to retain essential information. Additionally, the lack of interactive learning tools further widens the gap, as younger people are less likely to engage with static, text-heavy resources.

This project introduces a game-based learning approach specifically designed to make disaster preparedness education more accessible and engaging for younger audiences. Through interactive gameplay, the game will present essential preparedness knowledge in a more digestible and engaging way. By making disaster education more relevant and approachable, the project seeks to build a more informed and resilient community, helping to bridge the existing knowledge gap and raising awareness of disaster preparedness.

1.3 Aims and Objectives

Objectives

1. To design a serious game for disaster preparedness and awareness initiative.
2. To develop a functionality game using game development engine, incorporating components including game mechanics, player interactions, user interface and multimedia elements.
3. To evaluate the functionality of the game by testing all features to ensure they operate as intended and are free of technical issues.

1.4 Scope

This project focuses on developing a game to raise awareness and improve understanding of disaster-related knowledge. It will provide essential knowledge on disaster preparedness, response, and recovery strategies for various types of disasters. The educational content will cover topics such as early warning systems, community response teams, and the roles of government, organizations, and communities in disaster management. The game is designed for a wide audience, including students and the public, especially for younger generations who may lack adequate knowledge about disasters.

The conceptual game design for this project is in a turn-based strategy card format. Player will encounter different disaster scenarios and must strategically select from a set of preparedness cards that represent actions such as emergency response plans, resource allocation, and public education. They need to think carefully about their decisions in each round and minimize the impacts of the disaster. The game is designed in 2D for single-player gameplay and will feature a turn-based system along with card deck management.

This project will utilize Unity as the game engine and C# programming language to develop the game. The game will be developed to run on mobile devices to reach a wider audience. The development process will cover various aspects, including game mechanics implementation, level design, graphics design, sound design, and user interface design. The game will use the assets from Unity or online platforms for its graphics, animation, and sound elements. Additionally, this project may include the creation of graphics, animations, and sound effects that need to be used in the game to enhance the user experience. If it aligns with game mechanics and design, the augmented reality feature will be considered to further enrich the overall player experience.

1.5 Brief Methodology

This project will adopt the Agile model as the guiding methodology for the development process. Agile is an iterative, incremental approach that emphasizes continuous feedback, adaptation, and improvement, which is aligning with the project's goal. The development process will be divided into small, manageable phases known as sprints, typically lasting one to two weeks. Each sprint will focus on developing specific game features, such as gameplay mechanics, user interface design, and multimedia components. This method allows for flexible refinement, enabling the project to adapt and evolve based on insights gained throughout development.

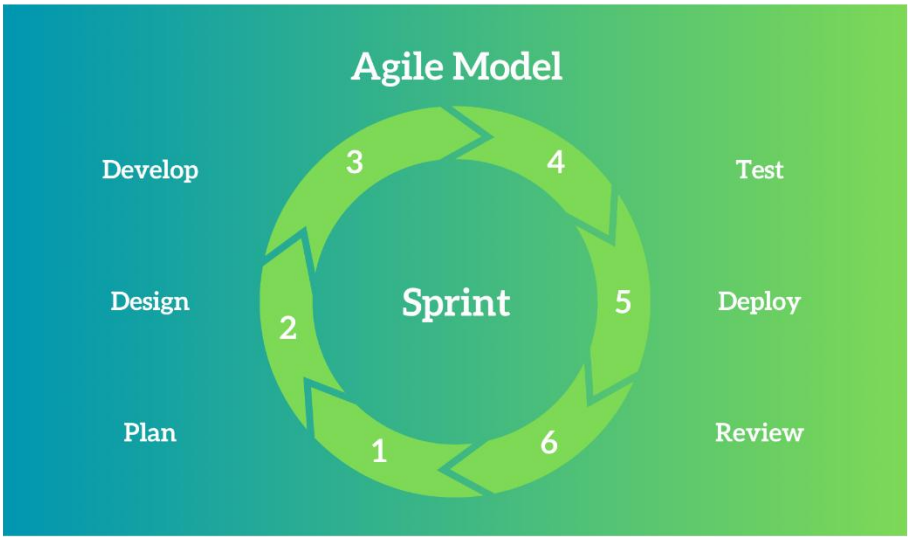


Figure 1.1: Agile Model.

Agile's cyclical nature enables continuous testing and improvement with each sprint cycle. The sprint begins with a planning phase, where short-term objectives are outlined, such as creating a particular set of game features. Following the planning stage, the development phase focuses on coding, designing, and implementing the intended features. Once the development phase is complete, a testing and review phase assesses functionality and checks for bugs. In this phase, testing will be conducted to evaluate how the newly

implemented features integrate with the existing ones to ensure that all functionalities meet the requirements and align with the project's objectives. Through the lessons learned from each sprint, the adjustments can be made for the next cycle and achieve continuous improvement.

By following this method, the game's components and user experience can be fine-tuned progressively. Agile model emphasis on feedback and adaptation ensures that each game component, from user interface design to sound effects, can be refined over time, ultimately leading to a well-rounded and engaging final prototype. This approach helps ensure that the project stays aligned with its disaster awareness objectives and adapts to any emerging ideas or necessary changes throughout development.

1.6 Significance of Project

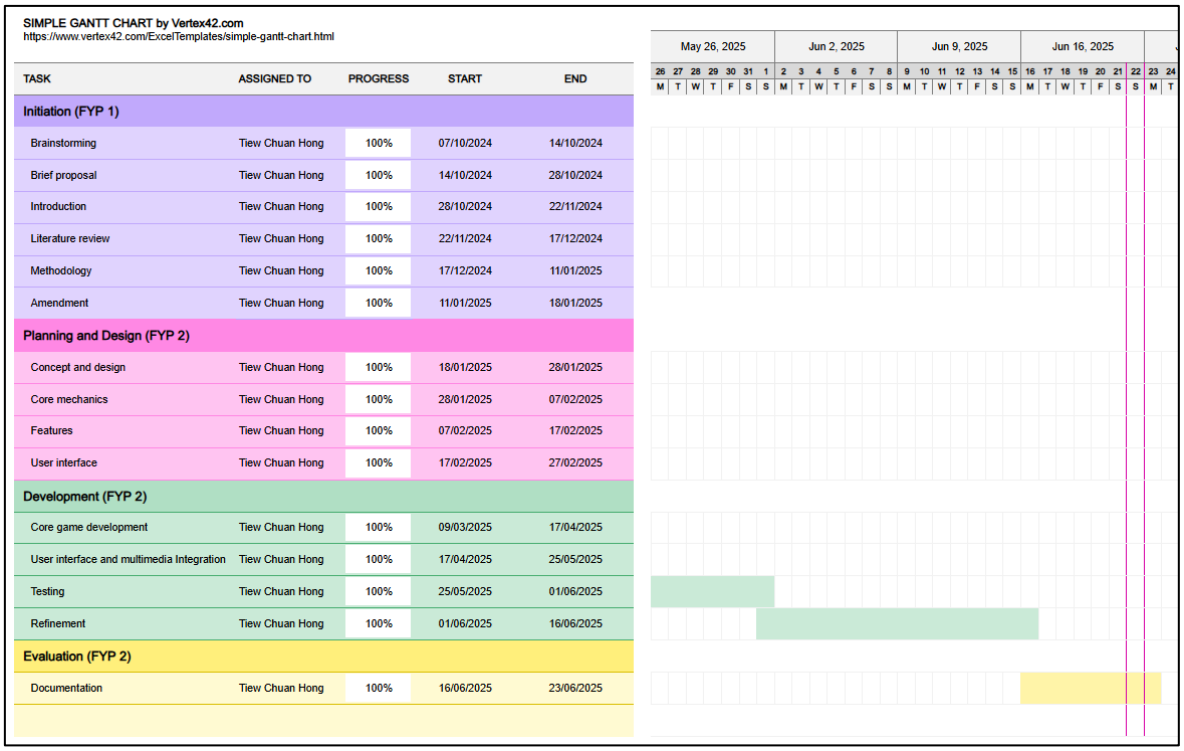
The significance of this project lies in its potential to enhance disaster preparedness education by introducing an interactive, game-based learning tool that is based on existing knowledge in disaster management and educational technology. This approach revises traditional methods of disaster education by utilizing interactive gameplay. This method encourages engagement and retention by involving player in interactive learning processes, which provide a deeper understanding of disaster preparedness knowledge.

The anticipated results of this project will contribute to understanding game-based learning as an effective method for educational interventions, specifically in disaster preparedness. If successful, this game could serve as a model for how educational games can impart critical knowledge while retaining user engagement, something that traditional methods may lack, especially for younger audiences. This project has significant implications across multiple areas. In educational practice, it could inspire new disaster

preparedness programs that incorporate game-based learning into school curricula or community training. It provides a practical and scalable solution for educational outreach. On a practical level, the game's engaging approach to disaster education has the potential to reach a broader audience, especially young people, and build a more disaster-resilient population and community. The use of mobile devices as a platform ensures accessibility, aligning with modern technology trends and expanding the reach of preparedness education beyond traditional settings. This approach may lead to greater acceptance and implementation of game-based learning as a valuable approach to public education. This project provides an innovative approach that not only bridges knowledge gaps but also extends the practical applications of interactive learning in different educational settings.

1.7 Project Schedule

This project is estimated to be completed within 9 months. A detailed timeline can be accessed through the following link to the Gantt chart: https://unimas-my.sharepoint.com/:x:/g/personal/82204_siswa_unimas_my/EXgyhb9MzMlDtKFm9lrQADwBFfCsKLmc5AMKmWWY114OSA



SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

TASK	ASSIGNED TO	PROGRESS	START	END
Initiation (FYP 1)				
Brainstorming	Tiew Chuan Hong	100%	07/10/2024	14/10/2024
Brief proposal	Tiew Chuan Hong	100%	14/10/2024	28/10/2024
Introduction	Tiew Chuan Hong	100%	28/10/2024	22/11/2024
Literature review	Tiew Chuan Hong	100%	22/11/2024	17/12/2024
Methodology	Tiew Chuan Hong	100%	17/12/2024	11/01/2025
Amendment	Tiew Chuan Hong	100%	11/01/2025	18/01/2025
Planning and Design (FYP 2)				
Concept and design	Tiew Chuan Hong	100%	18/01/2025	28/01/2025
Core mechanics	Tiew Chuan Hong	100%	28/01/2025	07/02/2025
Features	Tiew Chuan Hong	100%	07/02/2025	17/02/2025
User interface	Tiew Chuan Hong	100%	17/02/2025	27/02/2025
Development (FYP 2)				
Core game development	Tiew Chuan Hong	100%	09/03/2025	17/04/2025
User interface and multimedia Integration	Tiew Chuan Hong	100%	17/04/2025	25/05/2025
Testing	Tiew Chuan Hong	100%	25/05/2025	01/06/2025
Refinement	Tiew Chuan Hong	100%	01/06/2025	16/06/2025
Evaluation (FYP 2)				
Documentation	Tiew Chuan Hong	100%	16/06/2025	23/06/2025

Figure 1.2: Timeline Gnatt Chart.

1.8 Expected Outcome

The expected outcome of this project is the development of a functional game prototype. This prototype will integrate components, including the single player mode, turn-based system, card management system, player interactions, user interface, and multimedia elements.

Player enables to take turns during gameplay, drawing and playing a limited number of cards in each round, alternating with event phases that present new challenges. The card management system includes a randomized drawing mechanism and a comprehensive and diverse set of cards that introduce various effects and abilities. The designed cards will be categorized into different types, such as action cards, resource cards, and event cards. Player will be able to drag and drop cards to play them during their turns. They will also have access to game features and options through intuitive navigation within the user interface. The user

interface design will include a homepage, settings, gameplay interface, and dedicated screens for victory and defeat. The graphical elements will reflect the game's theme, including backgrounds, card illustrations, visual effects, and user interface components that enhance the overall aesthetic. The soundtrack will consist of background music that aligns with the game's themes and scenarios, while audio effects will respond dynamically to player interactions. The most important is that the game can run smoothly and have functionality on mobile devices.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Natural disasters have a significant impact on individuals and communities in various aspects, such as health and safety, economic stability, infrastructure damage, displacement, and environmental degradation (Sharrieff, 2023). Disaster preparedness and awareness are important to mitigate these impacts. The current ways of conducting disaster preparedness and awareness involve a combination of community-based education, simulation and training exercises, advanced technologies, and coordinated government efforts. Communities are educated and trained through campaigns, trainings, preparedness plans, drills, and volunteer programs. The integration of the technologies, including early warning systems, mobile apps, and social media, helps in enhancing real-time communication and response. There are also geographic information systems (GIS), and drones used in assessing risks and damage. The uses of artificial intelligence and big data help in optimizing disaster predictions and resource management. The collaboration with NGOs, private sectors, and international partners strengthens response capabilities, emphasizing both physical readiness and mental health support for resilient recovery.

2.2 Review on Disaster Preparedness and Awareness Program

Disaster preparedness and awareness programs cover a wide range of areas and operate at different levels to ensure comprehensive readiness. These areas include public awareness, education, skills training, simulation exercises, disaster risk management, and the integration of advanced technologies such as early warning systems. At different levels,

such as community, organizational, and governmental, these programs target diverse audiences with tailored strategies. Community-level initiatives focus on grassroots engagement, while organizational programs emphasize workplace preparedness. Community-based disaster preparedness involves disaster awareness programs, victim handling exercises, evacuation drills, early warning communication training, leadership development, resource management training, and disaster preparedness simulations (Purnomo et al., 2024). Meanwhile, national and governmental programs aim to implement policies, mobilize resources, infrastructure development and provide large-scale disaster response frameworks. Together, these efforts contribute to building a culture of preparedness and resilience.

2.2.1 Public Awareness Campaigns

Public awareness campaigns for disaster preparedness use a variety of methods to educate the public about disaster risks and preparedness strategies. These campaigns typically involve the dissemination of information through different types of mediums such as social media, television, and community outreach programs. The purpose is to inform the public on disaster-related knowledge, including emergency procedures, safety protocols, and the significance of preparedness for disasters. It commonly uses several effective methods, including workshops, infographics, short videos, articles, mobile apps, events, and seminars to maximize reach.

The previous campaign in the United States, National Preparedness Month, organized by the Federal Emergency Management Agency (FEMA), is an annual initiative observed every September to promote disaster preparedness among individuals, families, and communities (Federal Emergency Management Agency [FEMA], 2024). This campaign

utilizes a wide range of methods and existing tools to disseminate information, including posters, videos, mobile apps, events, articles, and other reading materials to raise awareness and encourage proactive disaster preparedness behaviours.

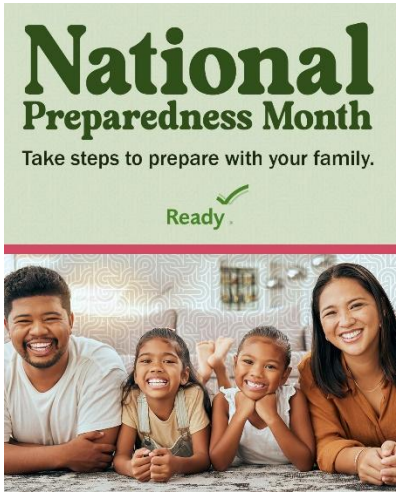


Figure 2.1: National Preparedness Month Poster (Federal Emergency Management Agency [FEMA], 2024).

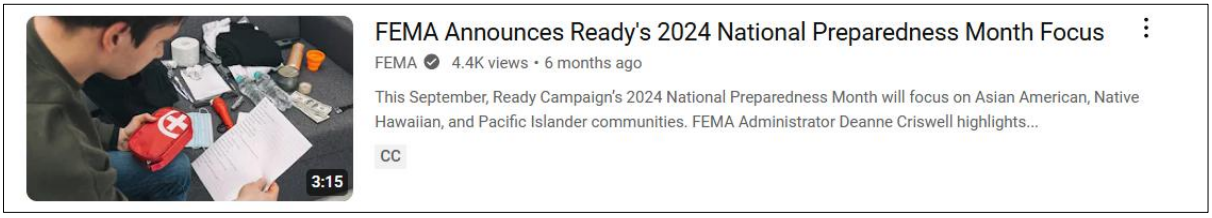


Figure 2.2: National Preparedness Month Video (FEMA, 2024).

In Malaysia, many workshops are organized to promote public awareness and disaster preparedness. These workshops are designed to educate communities on disaster risk reduction and provide essential knowledge on how to respond during emergencies. The SeDAR program has conducted community workshops for residents in areas like Ulu Klang and Kg. Tok Muda. The workshops include content modules such as “Understanding Hazards and Risks in Our Community”, which help participants identify local risks, including floods, landslides, and fires. Other modules, such as “Why Do We Need CBDRR” and “What’s Next: Planning & Taking Action,” emphasize the importance of disaster

preparedness and encourage participants to take proactive steps in planning and implementing safety measures within their communities (JPP, 2022).

The SeDAR program has utilized a variety of mediums across different activities to disseminate information on disaster preparedness and awareness. These include posters, videos, fridge magnets, brochures, and other informative reading materials.

Figure 2.3: Learning About & Taking Action Against Disaster Risk Mural Poster (SeDAR Malaysia-Japan, n.d.).

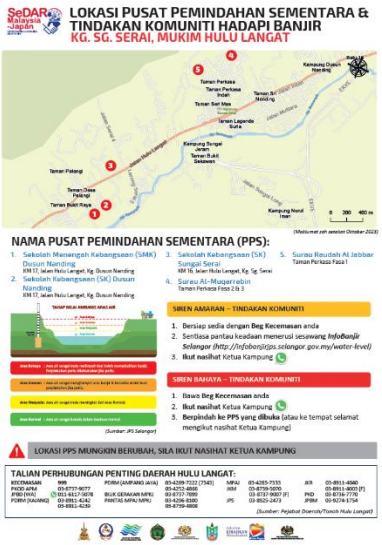


Figure 2.4: Evacuation Centre Location Map & Public Warning System Procedures (Hulu Langat Sub-District) Poster (SeDAR Malaysia-Japan, n.d.).

Public awareness campaigns for disaster preparedness can significantly improve community resilience and safety. It not only raises immediate awareness but also contributes to fostering a long-term culture of preparedness. Consistent reinforcement of key messages ensures that disaster preparedness remains a priority for the public.

2.2.2 Training and Simulation Programs

Training and simulation programs are comprehensive initiatives designed to equip individuals, organizations, and communities with essential skills and knowledge to respond effectively during disasters and emergencies. These programs typically combine education, hands-on training, simulations, and the integration of advanced technologies like virtual reality (VR) and augmented reality (AR) to create more immersive and impactful learning experiences.

One of the primary components of these programs is skills-based training, which includes first aid and basic emergency response techniques. These skills enable individuals and communities to respond during crises. For professionals, advanced training programs emphasize areas such as disaster risk management, contingency planning, and strategic response. These programs are designed to enhance decision-making, improve coordination, and prepare professionals to manage complex emergency scenarios.

Simulations are another part of disaster preparedness programs. These simulations range from tabletop drills to full-scale simulations. Tabletop drills can be known as tabletop exercises, which are discussion-based sessions in an informal way for team members to discuss their roles and responses to the emergency (Federal Emergency Management Agency [FEMA], 2024). AR/VR simulations use advanced technology to replicate real-life disaster scenarios in virtual or augmented environments. These simulations offer highly immersive training experiences in different settings, allowing participants to engage in realistic disaster response exercises with risks. Full-scale simulations are highly realistic operational exercises that replicate disaster scenarios with physical settings, equipment, and personnel. These exercises involve all relevant stakeholders, such as emergency responders and

community members, and test real-world preparedness by mimicking the complexities of disasters like floods, earthquakes, or hazardous material spills.

Different methods of training and simulation programs offer distinct benefits. Tabletop drills are cost-effective and provide a foundation for understanding roles, decision-making, and communication during disasters. AR/VR simulations offer more immersive and better formulation of disaster preparedness plans for trainees compared with tabletop drills (Alshowair et al., 2024). Full-scale simulations are the most comprehensive, testing real-world operational readiness by involving physical resources and actors. While more effective in certain contexts, all these methods contribute significantly to raising awareness and equipping individuals and communities with the necessary skills to respond to disasters and minimize risks. According to Darmawan & Kamaluddin (2021), tabletop exercises can effectively improve disaster preparedness by increasing confidence, knowledge, and understanding of roles in disaster preparedness. In contrast, research by Schumacher et al. (2022) shows that full-scale simulations enhance institutional preparedness and increase staff awareness in hospital pharmacies. The study by Panggabean (2023) shows that a VR-based Flood Simulator Game can enhance preparedness for dealing with flood disasters.

2.3 Computer Games

Computer games, also known as video games, are defined as any form of computer-based entertainment software, whether textual or image-based, that operates on electronic platforms such as personal computers or consoles and can involve one or multiple players in a physical or networked environment (Frasca, 2001). As the video game industry evolves, there is a wide range of genres and types of video games. Common genres include action, role-playing, sandbox, and open-world games, among others. There is also a category of

video games defined by their purpose, which includes educational games, exergames, serious games, etc. Video games can improve learning in specific topic when they are thoughtfully designed and purposefully focused on the topic (Taheri & Javeid, 2021). Serious games, a subset of video games, are the games that specifically designed to facilitate learning and education.

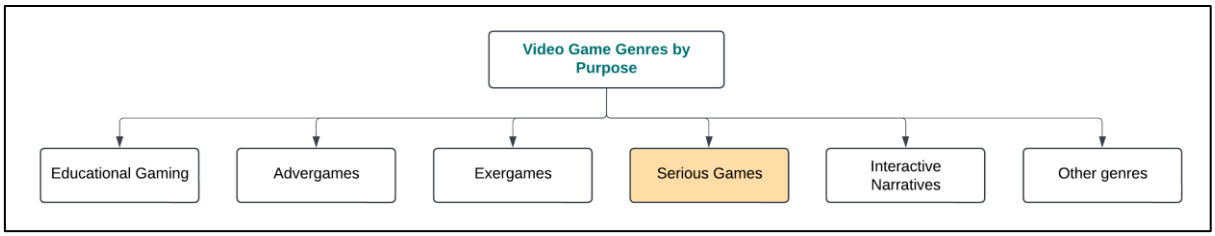


Figure 2.5: Video Game Genres by Purpose (Nguyen, 2024).

In the context of this project, serious games are an important aspect as they are designed with a primary focus on education, training, and raising awareness, rather than pure entertainment. Serious games are interactive digital applications that go beyond entertainment and incorporate educational, informational, or training objectives as the goal of the game (Lamb, 2024). Examples of serious games include Microsoft Flight Simulator, which provides a realistic simulation for flight training and enhances understanding of aerodynamics and navigation systems, and Bleached Az, a game aimed at promoting ocean health awareness and conservation by engaging player in activities related to protecting marine ecosystems.

The primary distinction between serious and other games lies in their purpose-driven design. Serious games are developed with objectives beyond entertainment, often aimed at training, education, health improvement, or raising awareness about social or environmental issues. These games focus on skill development, problem-solving, and knowledge acquisition, frequently incorporating realistic elements such as authentic scenarios and real-

world data to simulate real-life challenges. They integrate educational content within engaging narratives or mechanics to motivate player to actively participate and learn.

2.4 Review of Related Works

2.4.1 Disaster Mind

Disaster Mind is an educational web-based simulation game created by FEMA Region 8 and iThrive Games. The game is designed to teach high school students about disaster preparedness. Player needs to make quick decisions in the face of three natural disasters, which are blizzard, wildfire, and flood. The game mechanics involve selecting actions that could help ensure safety, building skills related to disaster response. The decision-making process involves assessing risk and consequences. The game features interactive storytelling, with each decision affecting the outcome. Its primary goal is to raise awareness about emergency preparedness in an engaging way.

2.4.1.1 Process Flowchart

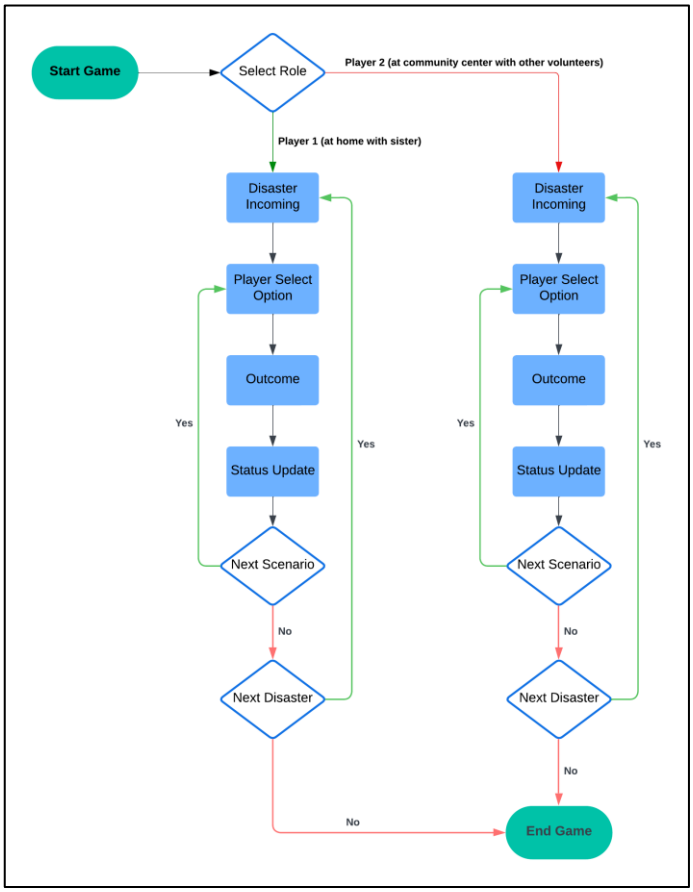


Figure 2.6: Disaster Mind Process Flowchart.

The game begins by selecting different roles for the main characters to simulate individuals in different situations. Player 1 takes on the role of a character at home with their sister, while Player 2 assumes the role of a character at a community centre with other volunteers. Each role will encounter different scenarios, but the overall process flow remains the same.

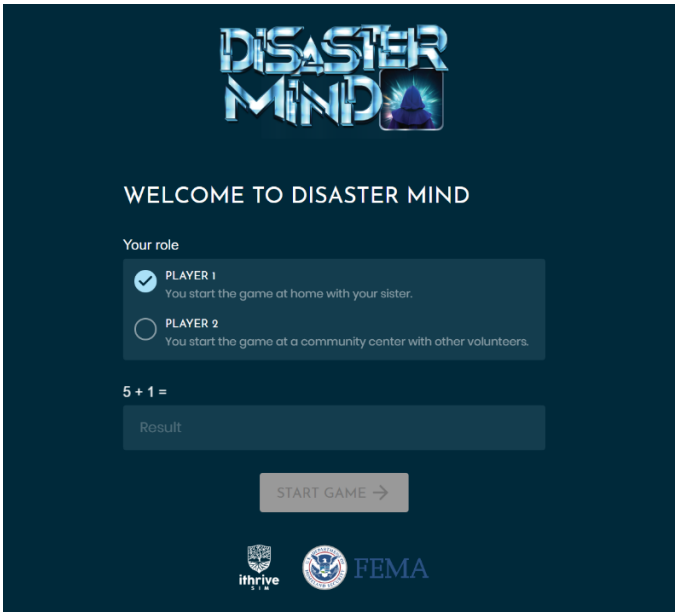


Figure 2.7: Roles of the Main Character.

The game features a time system and a health system. As time progresses, new scenarios will emerge, and player must make decisions within a limited timeframe. Player's choices will impact the health of their characters. The time system introduces urgency and realism, which enables player to think quickly and adapt their strategies in the situations. Meanwhile, the health system adds a layer of strategy and consequence to the gameplay.

Figure 2.8: Time System and Health System.

In the gameplay, the game presents information through a mix of short videos, audio, and images, rather than relying solely on text. This approach simulates real-life environments such as news broadcasts, voice messages, and information from social media.

Figure 2.9: Short Video for Breaking News.

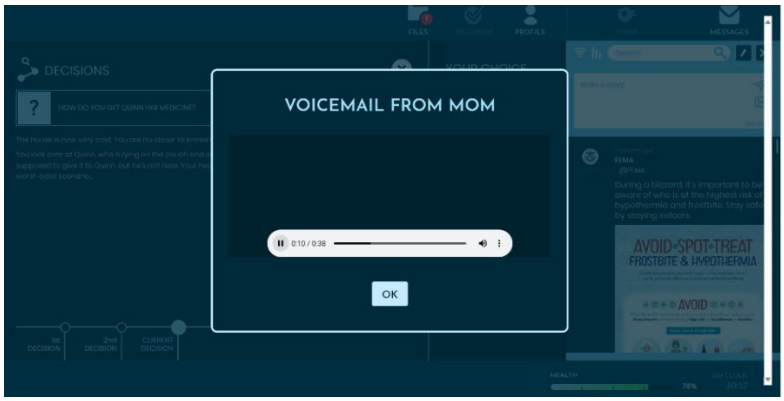


Figure 2.10: Audio Message.

Figure 2.11: Image of How to Treat Frostbite or Hypothermia if the Need Arises.

The game features a virtual social media design that allows player to gather important information and updates from other characters. This feature simulates real-life crisis communication, where people receive information through their phones and the internet. By

collecting this information, player can make informed decisions to address the challenges during the disaster.

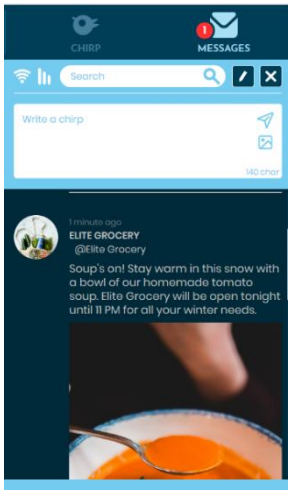


Figure 2.12: Virtual Social Media Interface.

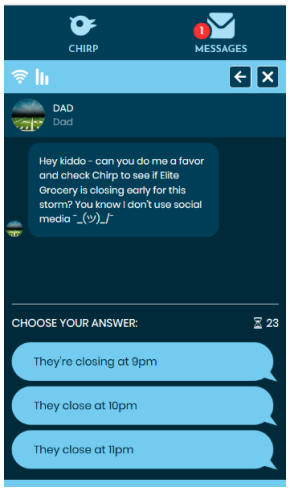


Figure 2.13: Message from Other Characters and Decision-Making for Replying to Messages.

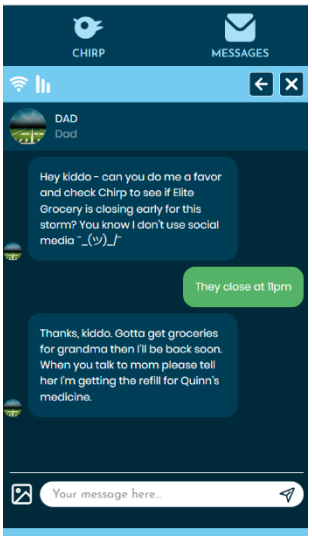


Figure 2.14: Outcome of Message Replies.

The game incorporates decision-making elements as its core mechanics. As simulator clock time progresses, new scenarios and decisions will arise, prompting player to make quick choices within a limited timeframe. The game presents several options for player to choose from, and once a selection is made, the outcome is displayed, along with its impact on the character's health. The scene will repeat until the simulator's clock reaches the end of its designated time. The uses of decision-making elements can help player develops skills that are directly applicable to disaster preparedness, such as critical thinking, problem solving, and risk management, while also making the game more interactive and educational.

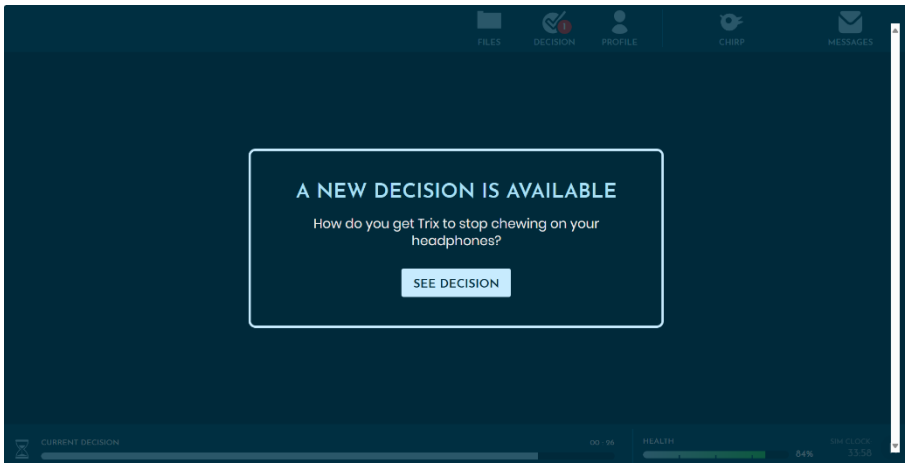


Figure 2.15: New Decision Scenario.

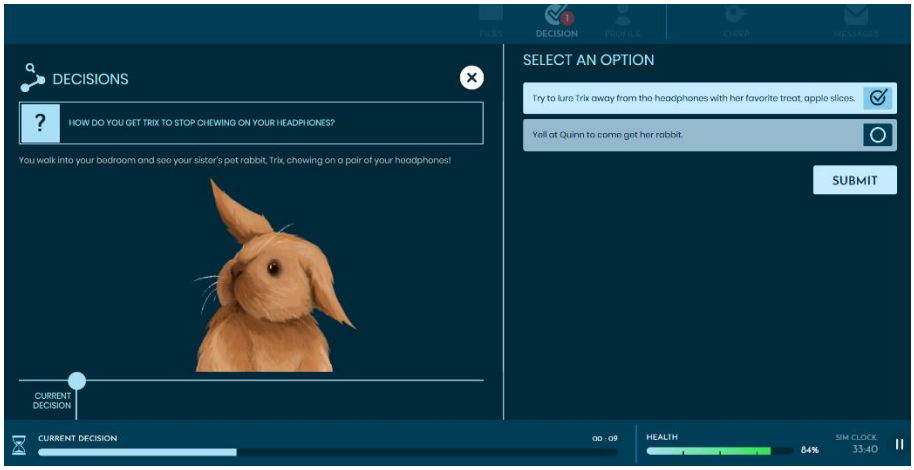


Figure 2.16: Time-Limited Decision-Making.

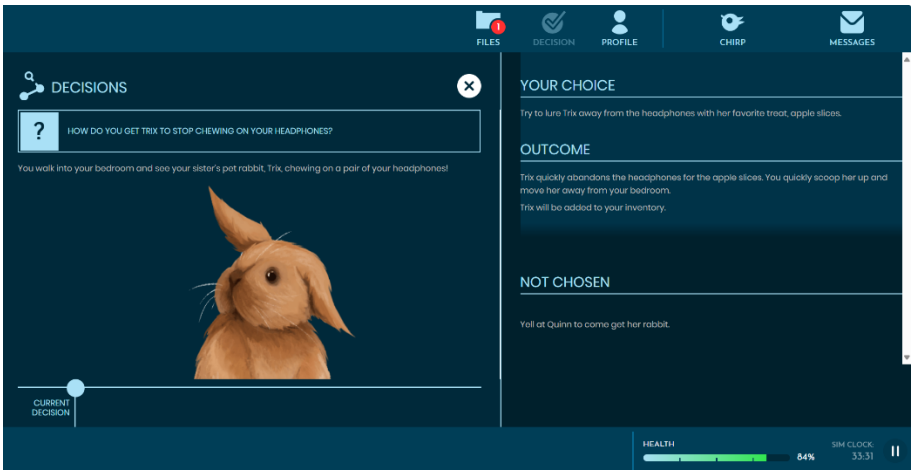


Figure 2.17: Decision Outcome.

2.4.1.2 Strengths and Weaknesses

The game is well-designed and interactive, with several strengths and a few weaknesses identified through observations and gameplay. The game provides different scenarios relevant to disaster preparedness and awareness by including three disaster types: blizzards, floods, and wildfires. Each scenario is designed to simulate situations individuals might face in real life, such as evacuations, communication breakdowns, and emergency kit preparation. These scenarios provide player an opportunity to experience and respond to emergencies while enhancing their awareness of disaster-specific challenges.

The educational content in the game is comprehensive and provides player with detailed, practical information on disaster preparedness and response. It covers topics such as preparedness strategies, emergency kit preparation, and response planning. Additionally, the game provides player with feedback on their decisions through outcomes and the health system to demonstrate how certain actions can either mitigate or worsen the impact of a disaster. By integrating these lessons into the gameplay, the game ensures that player can apply the knowledge gained to real-world disaster situations, making it both informative and impactful.

Figure 2.18: Information of How to Prepare for a Blizzard.

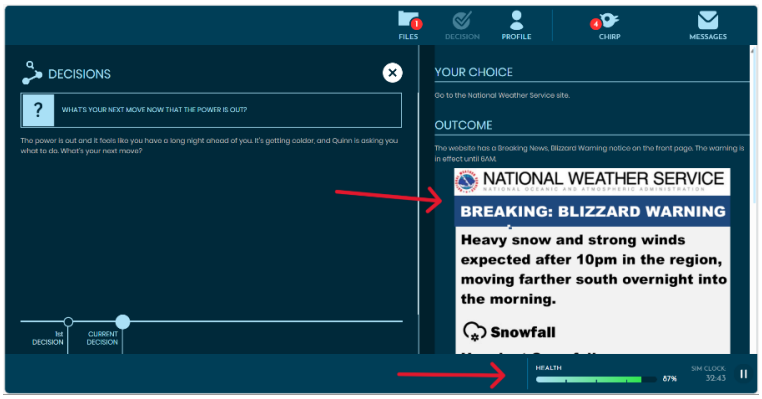


Figure 2.19: Feedback on Player Decisions.

One of the weaknesses of the game is that, although feedback is provided, it is not clear and detailed. For example, while health is affected by the player's choices, the feedback

is not obvious. If player do not pay attention to the health, they will not know the impact of their decision. Additionally, the outcomes of unselected choices are not shown, leaving player unaware of what could have happened if they had made a better or worse choice.

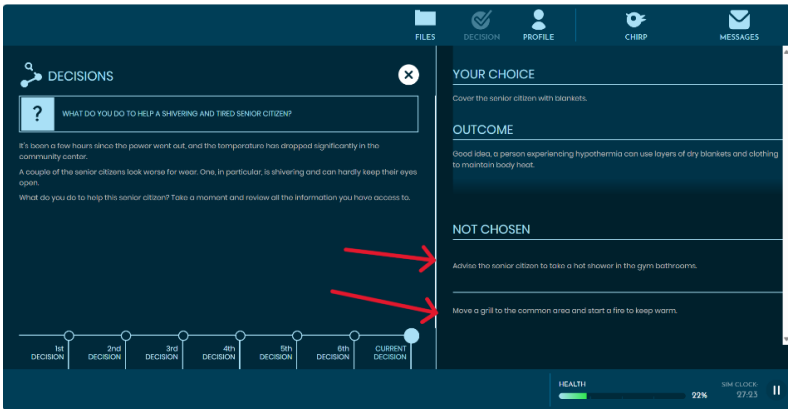


Figure 2.20: Not Chosen Choices Without Outcomes.

Overall, the game combines realistic disaster scenarios, educational content, and engaging gameplay elements. It serves as an effective tool to provide valuable lessons in disaster preparedness.

2.4.2 Stop Disasters!

"Stop Disasters!" is a web-based simulation game developed by playerthree, with contributions from experts and organizations, under the United Nations Office for Disaster Risk Reduction (UNDRR). The game educates both adults and children how to build safer communities against disasters.

2.4.2.1 Process Flowchart

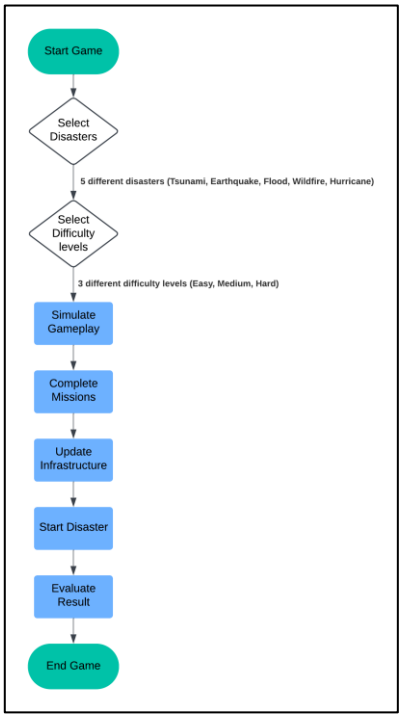


Figure 2.21: Stop Disaster Process Flowchart.

The game begins with the selection of one of five disasters: tsunami, earthquake, flood, wildfire, or hurricane. Each disaster occurs in a different area and requires specific strategies and preparedness measures before it strikes. After selecting a disaster, player choose a difficulty level with varying resources and conditions.

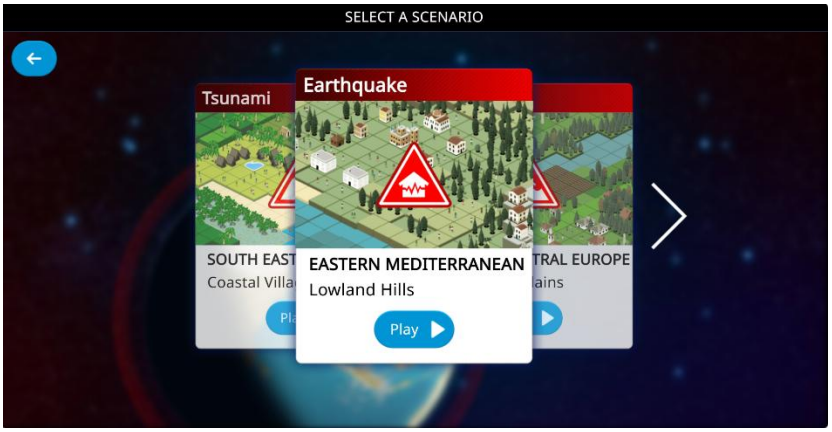


Figure 2.22: Select Types of Disasters.

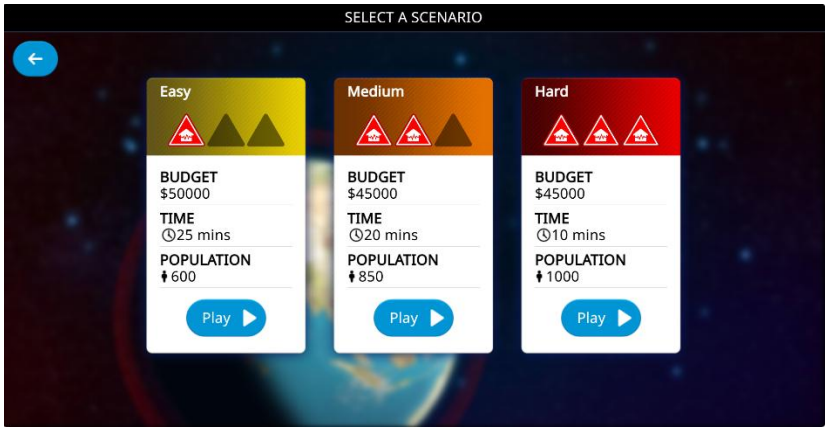


Figure 2.23: Select Difficulty.

The core mechanics of the game focus around construction, requiring player to build and upgrade structures to protect against disasters. The game features a grid system, enabling player to strategically place buildings within designated grid spaces.

Figure 2.24: Grid System.

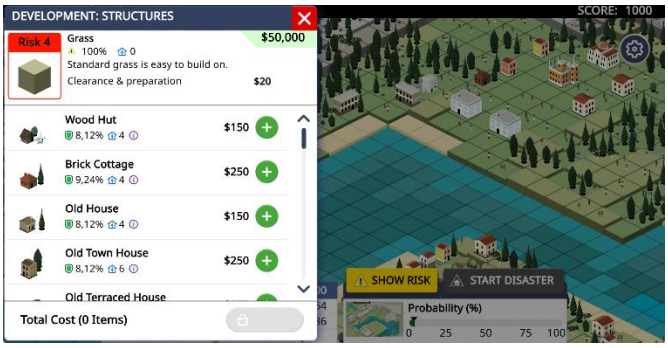


Figure 2.25: Build a Structure.

Player must complete four minimum requirements before the disaster strikes, all within a limited budget and a restricted timeframe.

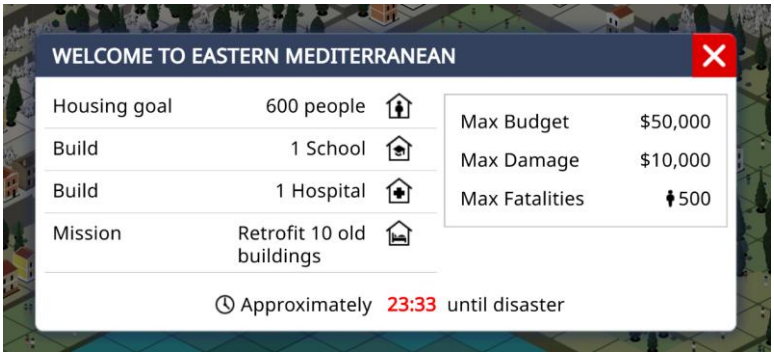


Figure 2.26: Requirements, Budget and Timeframe.

During the simulation, player must build and enhance infrastructure using various strategies. For example, upgrading a community centre may involve implementing measures like evacuation signs, training programs, local alarm systems, or radio networks to improve protection against earthquakes.

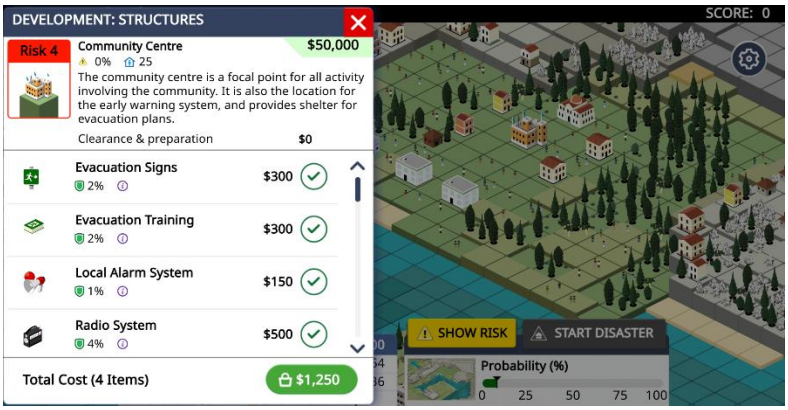


Figure 2.27: Preparedness Measures.

The game includes an additional feature that highlights risky areas on the grid using color-coded borders across five levels. Level 5, marked in purple, represents the highest risk during a disaster, followed by level 4 in red, level 3 in orange, level 2 in yellow, and level 1 in green, indicating the lowest risk. This feature enables player to think whether an area is suitable for building and make informed decisions accordingly.

Figure 2.28: Risky Areas.

Once the objectives are met or time runs out, the disaster strikes. The game will then show the outcome of the disaster and provide a result evaluation based on the player's preparedness and decisions.

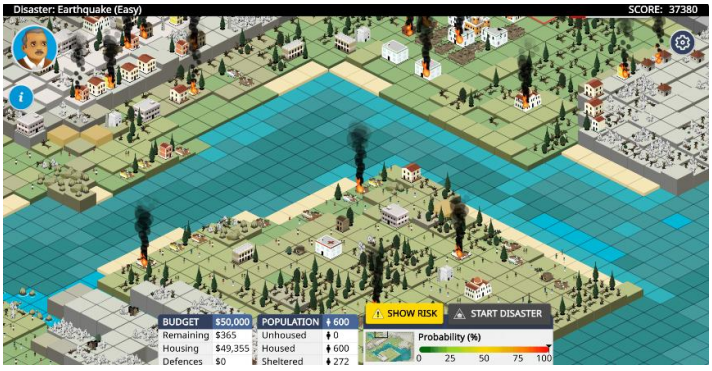


Figure 2.29: Depiction of the Disaster Outcome.

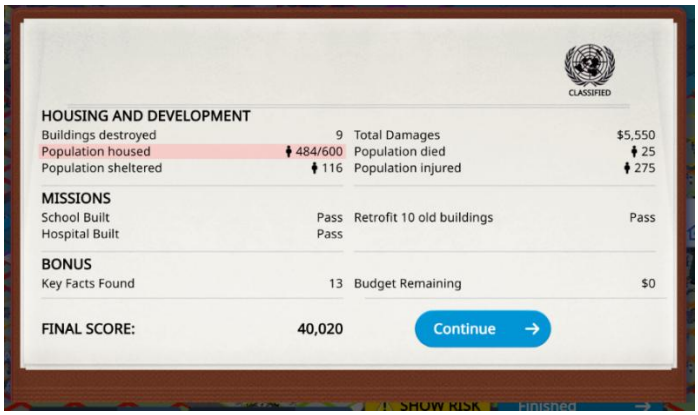


Figure 2.30: Result Evaluation.

2.4.2.2 Strengths and Weaknesses

The graphic design of the game plays a pivotal role in immersing player and enhancing their understanding of disaster scenarios. The game's visual style is highly interactive, combining a clean, user-friendly interface, color-coded elements that draw attention to key areas of the map, such as high-risk zones and strategic locations for building infrastructure. These visual cues allow player to easily assess the current state of their environment and make informed decisions in real-time. The depiction of disaster outcomes is particularly impactful. The game uses dynamic animations and graphics to simulate the real-time progression of disasters, such as floods, earthquakes, and wildfires, helping player visualize the scale and severity of the disaster as it strikes. This is enhanced by detailed visual effects, such as shifting landscapes, rising water levels, or crumbling buildings, that represent the disaster's progression and its impact on the community.

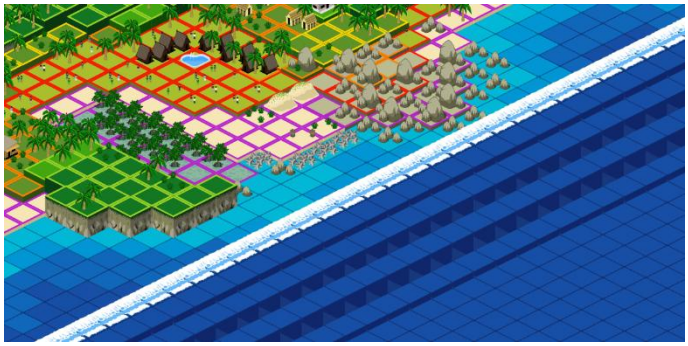


Figure 2.31: Depiction of Tsunami.

The implementation of preparedness measures, such as upgrading buildings with evacuation signs, local alarm systems, and training programs, adds significant depth to the gameplay. For example, upgrading evacuation signs can directly impact the survival rate during a disaster by reducing the number of casualties, making the game more realistic and educational. This system ties player actions to tangible results, which reinforces the importance of disaster preparedness.

One of the weaknesses of the game is that the instructions are not clear enough, especially for first-time player. New player may struggle to understand the game mechanics initially and may need additional time to become familiar with the controls and objectives. Another weakness is the lack of one-click or group upgrade options. In the game, player need to upgrade buildings individually, which can be tedious when there are many grids and buildings on the map. This limitation can slow down the gameplay and affect the overall user experience.

Overall, the game is an interactive tool for disaster preparedness and awareness. It provides player with the means to understand and navigate the complexities of disaster preparedness with an engaging and educational experience.

2.4.3 CoronaQuest

CoronaQuest is a project created by the DFJC (Department of Education and Training, Youth and Culture) of the Canton of Vaud (Switzerland), directed by Councillor of State Cesla Amarelle. It is a video game developed in collaboration with DNS-Studios and other experts. The game is designed to help children feel calmer and safer when returning to school and encourages kindness. It also teaches player of actions to protect their health and live safely with the coronavirus, both in school and in other places.

2.4.3.1 Process Flowchart

The game is designed as a turn-based strategy card game, where player take turns drawing and playing cards to perform actions, such as attacking, defending, or using special abilities. Each card has unique effects, and player must carefully plan their moves to outsmart their opponents, manage resources, and achieve victory.

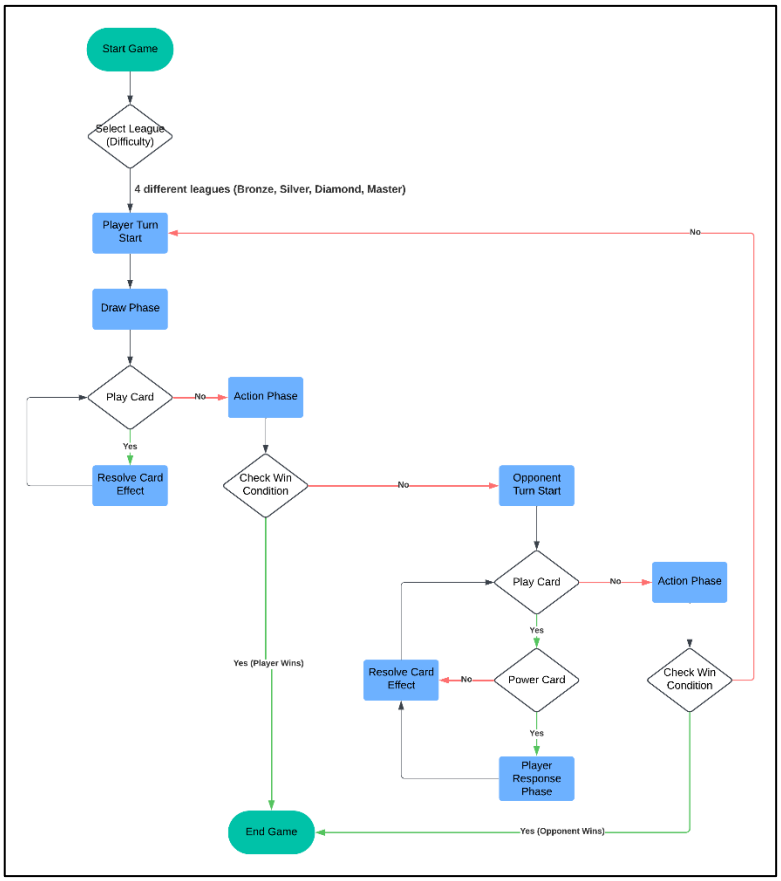


Figure 2.32: CoronaQuest Process Flowchart.

The game begins with choosing a league, which determines the difficulty level. Player can choose from four leagues: Bronze, Silver, Diamond, and Master. As the leagues progress, the difficulty increases, and new cards will be added during gameplay. To unlock the next league, the player must achieve a specific number of wins in the current league.

Figure 2.33: CoronaQuest Menu Page.

Figure 2.34: Choose Difficulty.

After choosing the difficulty, the match starts with player's turn first, player needs to play cards to defeat the opponent, the coronavirus. The game includes three types of cards: Character Cards, which represent the player's units; Power Cards, which provide bonuses or disrupt the opponent; and Defense Cards, which counter the virus's powers.

Figure 2.35: Playfield of the Game.

Figure 2.36: Card Types of the Game.

After the player plays their cards and ends the round, the characters placed on the field will take action to attack the minions or coronavirus, dealing damage based on their attack stats and reducing the opponent's health. Once the player's turn ends, the opponent takes their turn, possibly using power cards to lower the stats of the player's characters. The player must then respond by using related defense cards to counteract these effects and protect their characters.

Figure 2.37: Characters with Their Respective Attack and Health Stats.

Figure 2.38: Player Needs Response to Opponent's Power Card with Related Defense Card.

The game alternates turns until either the player's or opponent's health reaches zero. The player must think strategically and manage their resources to win the match. Through the game, player can gain valuable knowledge about how to protect themselves and others from the coronavirus. The game also promotes critical thinking, decision-making, and strategic planning as player manage resources and respond to challenges.

Figure 2.39: Player and Opponent's Health.

2.4.3.2 Strengths and Weaknesses

The game provides strong replay value due to the dynamic nature of its mechanics. Each match feels unique because of the variety in card draws and the different strategies player can adopt. This unpredictability ensures that no two matches are exactly the same, keeping the gameplay fresh and encouraging player to return to the game. The turn-based card game mechanics are interactive and allow player to think critically and plan their moves. The combination of attacking, defending, and using special abilities creates an engaging loop that requires strategy and decision-making. This aspect is particularly appealing to player who enjoy games that combine skill and tactical thinking.

One of the main weaknesses of the game is the absence of detailed descriptions for card abilities. Player may find it difficult to understand the effects or strategic value of certain cards, especially if they are new to the game. This can lead to confusion and missed opportunities for optimal play. Clearer, more detailed descriptions would enhance the overall experience and make the game more accessible to a wider audience.

Overall, the game is a serious game that features interactive mechanics designed to engage player while offering strong replayability. It effectively teaches player how to protect themselves from the coronavirus.

2.4.4 Comparison Between Related Works

The table below shows the strengths and weaknesses of three related games. These insights help identify mechanisms and features to enhance the developed game while avoiding weaknesses that could impact its quality.

Table 2.1: Comparison Between Three Related Games.

Games	Game mechanics	Strengths	Weaknesses
Disaster Mind	• Time-constrained decision-making • Interactive storytelling	• Immersive storytelling • Realistic multimedia simulation	• Low replayability • Vague feedback on decisions • No further explanation for unselected choices
Stop Disaster!	• Grid-based construction • Simulation • Strategy	• Gradual difficulty levels • Good visuals and animations	• Unclear instructions • Ineffective upgrade
CoronaQuest	• Turn-based card play • Strategy	• Gradual difficulty levels • High replayability • Engaging gameplay	• Lacks detailed card descriptions

2.5 Discussion

The reviewed disaster preparedness and awareness initiatives provide valuable insights into disaster-related knowledge, covering areas such as preparedness, risk management, and effective response strategies. These initiatives emphasize the importance of equipping individuals and communities with the skills and information needed to mitigate disaster risks and strengthen resilience. Disaster preparedness and awareness can be achieved through various methods tailored to specific needs and contexts. These methods include public awareness campaigns to disseminate critical information, training and simulation exercises to develop practical skills, and advancements in technology that further

enhance preparedness and response efforts. The insights gained from these initiatives can be utilized in this project as foundational content for the development of a game-based tool.

Serious games such as “Disaster Mind”, “Stop Disasters!”, and “CoronaQuest” effectively integrate educational content with interactive gameplay, making them valuable tools for educational purposes. The review identified strengths and weaknesses that can guide the creation of a game-based tool for disaster preparedness and awareness. Based on the results, a common limitation among these games is the lack of detailed educational content and insufficient gameplay instructions, which may affect the learning and user experience. However, they also showcase notable strengths that can be leveraged in serious game design. High-replayability game design, such as different difficulty levels, can encourage continuous engagement, allowing player to revisit and reinforce their learning. Additionally, aesthetic graphics, visuals, and animation designs help capture and maintain player attention, enhancing the overall experience.

2.6 Summary

The review of disaster preparedness and awareness initiatives highlights various strategies, including public awareness campaigns, training programs, simulation exercises, and advancements in technology, all aimed at building community resilience and equipping individuals with essential knowledge and skills. Serious games like “Disaster Mind”, “Stop Disasters!”, and “CoronaQuest” demonstrate the potential of interactive tools in disaster education, effectively combining educational content with engaging gameplay. However, the review also identified limitations, such as insufficient educational content and unclear gameplay instructions, which may affect user experience. At the same time, strengths like high replayability and visually appealing designs provide valuable insights for developing

future game-based tools. These findings will guide the creation of a serious game in this project, ensuring it effectively promotes disaster preparedness and awareness.

CHAPTER 3

METHODOLOGY, REQUIREMENT ANALYSIS AND DESIGN

3.1 Introduction

This chapter outlines the methodology, requirement analysis, and design specifications used in the development of the game. It provides a comprehensive overview of the approach adopted to gather requirements, analyze user needs, and design the system. The chapter focuses on the selection of the Agile Model as the development methodology, which emphasizes iterative progress, flexibility, and continuous feedback. The requirements gathered from target users are analysed to identify key features and functionalities, which are then translated into clear design specifications. This process ensures that the game aligns with user expectations and addresses the key concerns related to disaster preparedness. The design section includes visual representations of the user interface to provide a detailed view of the game.

3.2 Methodology

This project adopts the Agile Model as the methodology for game development. The Agile approach allows for iterative progress and flexibility. By using this approach, the game's components and user experience can be developed and refined progressively throughout the development process.

3.2.1 Agile Model

The Agile Model is an iterative and incremental development methodology that emphasizes flexibility and continuous feedback. It divides the project into smaller iterations

or sprints, with each delivering a functional piece of the system. This model was selected for its adaptability, which aligns with the dynamic nature of game development and allows for frequent testing and refinement.

The game mechanics and systems are divided into smaller, manageable sprints to streamline the development process. Each sprint is prioritized based on the complexity of its tasks and their overall impact on gameplay. For example, the initial sprint focuses on basic mechanics, such as implementing card decks and cardholder functionality. Features will be systematically designed and developed within their respective sprints. At the end of each sprint, testing will be conducted to identify bugs and ensure that the implemented features function as intended. Based on the result, adjustments and refinements will be made to improve the game's quality and user experience.

3.2.2 Sprint Breakdown

This project is divided into several sprints, each lasting between two to four weeks. Each sprint is focused on specific features or mechanics to ensure that smaller deliverables are completed incrementally.

Sprint 1: Core Mechanics and System Setup

The first sprint focuses on establishing the basic game mechanics. This includes the card deck system, the cardholder mechanic, and defining the card abilities that will be used throughout the game.

Sprint 2: Turn-Based System and Health System

The second sprint focuses on developing the turn-based gameplay system and the player health system. The turn-based system will allow the player and enemy to alternate their turns, while the health system will track the player's health and calculate the impact of each round.

Sprint 3: Visual Design and multimedia element

The third sprint focuses on the visual design of the game. This includes creating and refining the appearance of the cards, user interface, and animations, and integrating multimedia elements such as sound effects and background music to enhance the player experience.

Sprint 4: Polishing visuals and refining gameplay.

The fourth sprint focuses on polishing and refining the game's overall gameplay. This includes fixing any bugs, optimizing performance, adjusting game balance, and enhancing visual and audio elements.

3.3 Requirement Analysis

This project utilized a questionnaire to identify and collect the needs and expectations of users. A Google Form was specifically designed and distributed to gather data from the target audience. The primary audience for this survey consisted of younger individuals, predominantly university students. The data collected offered valuable insights into user awareness and preparedness regarding disasters, as well as their expectations, preferences, and the key functionalities required for the proposed game system.

3.3.1 Questionnaire Design

The questionnaire was structured to collect both qualitative and quantitative data for a comprehensive understanding of user needs. It consisted of five sections:

- Demographic Information: Gathers basic details such as age group, gender, and geographic location of the respondents.
- General Awareness of Disasters: Assesses respondents' knowledge and understanding of disasters.
- Preparedness Knowledge and Actions: Evaluates respondents' familiarity with disaster preparedness practices and their current efforts to mitigate risks.
- Perception and Attitude Toward Disaster Preparedness and Awareness: Explores respondents' attitudes toward the importance of disaster awareness and their willingness to engage in preparedness activities.
- Disaster Preparedness and Awareness Through Games: Investigates the feasibility and potential of using games as a medium to enhance disaster awareness and preparedness.

3.3.2 Data Collection

A total of 20 responses are received through the questionnaire. The responses are collected anonymously to encourage honest and unbiased feedback. However, a challenge arises due to incomplete responses, primarily caused by the addition of questions after the form was distributed.

3.3.3 Analysis of Requirements

Based on the data collected through the questionnaire, the following aspects were analysed to define the functional and non-functional requirements. Out of the 20 responses, the gender distribution is balanced, with half of the respondents identifying as female and half as male. The age group of respondents predominantly falls between 18 to 24 years, followed by a small proportion under 18.

The most experienced disasters identified by respondents include floods, landslides, and wildfires. Since floods, landslides, and wildfires are the most common disasters reported by respondents, the game will focus on these types of disasters. This will make the game more relevant to users and help address their main concerns about disaster preparedness.

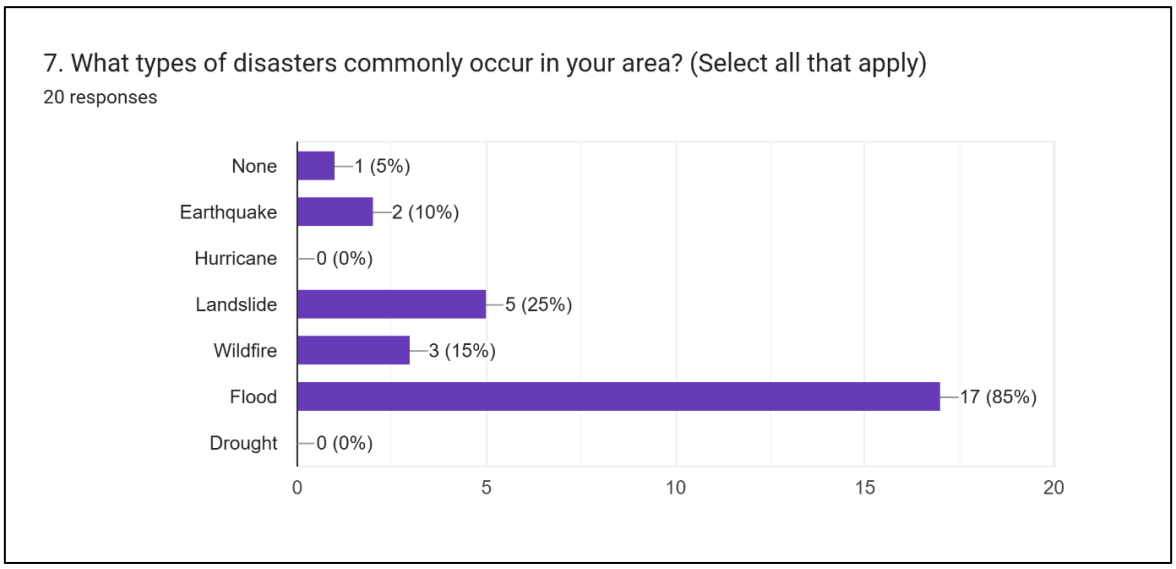


Figure 3.1: Commonly Occurring Disasters in the Area.

In terms of awareness, 65% of respondents are aware of common natural disasters in their area, and 75% understand the impact of disasters, such as loss of livelihoods, property damage, and environmental degradation. This indicates that most respondents are aware of the risks associated with disasters.

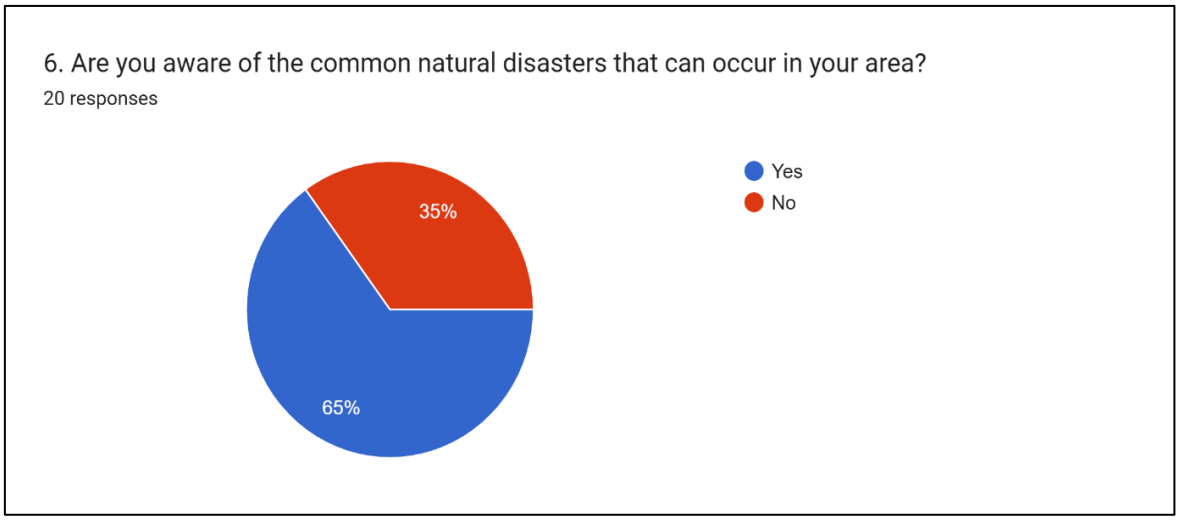


Figure 3.2: Awareness of Common Natural Disasters in the Area.

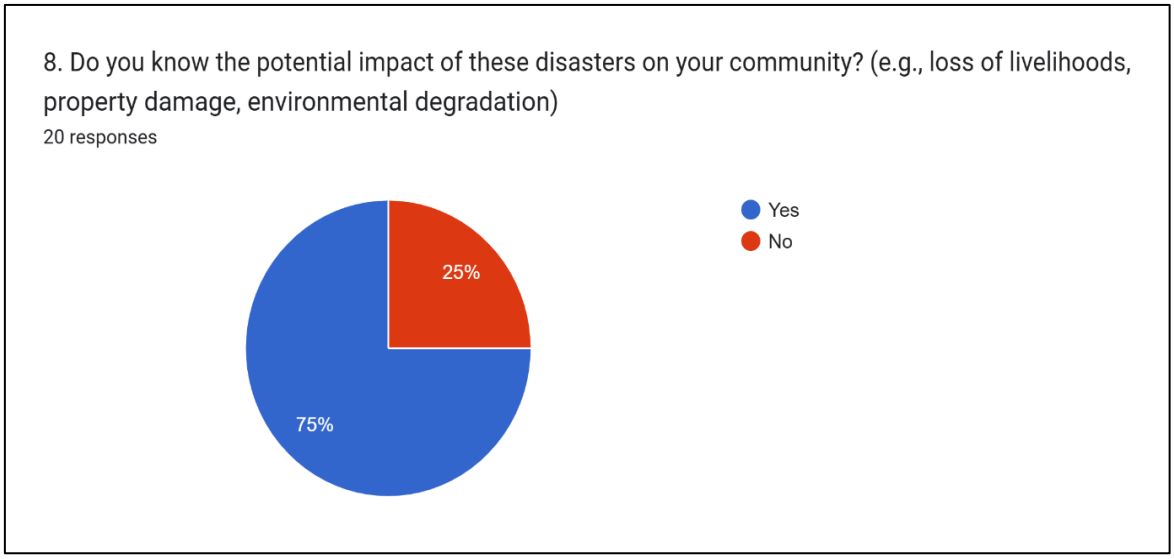


Figure 3.3: Awareness of the Potential Impacts of Disasters on the Community.

However, only 20% are familiar with evacuation routes or preparedness plans. Additionally, 30% know the emergency contact numbers for their area, and only 5% have a disaster preparedness kit at home. These findings highlight low preparedness levels and a significant gap in disaster preparedness knowledge.

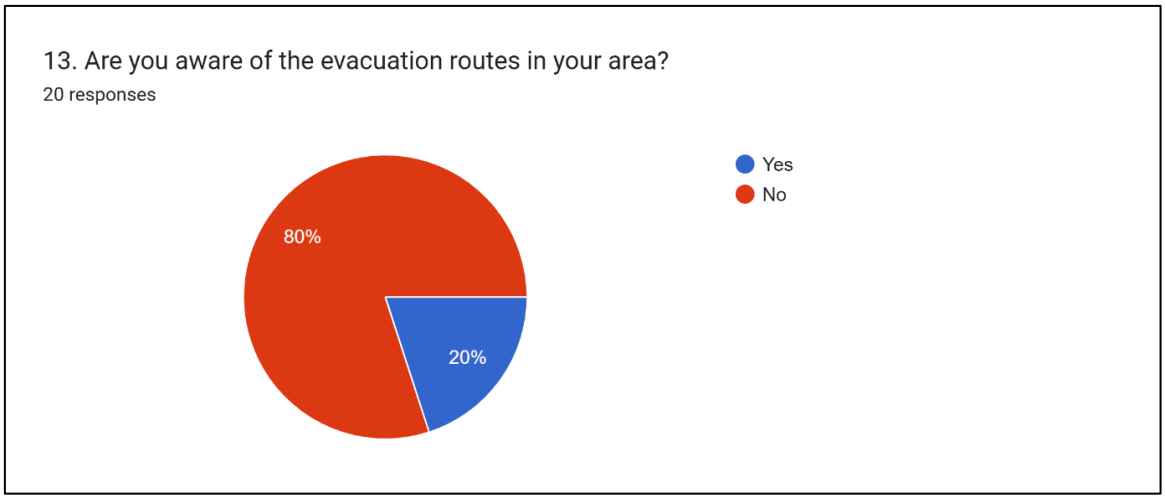


Figure 3.4: Awareness of Evacuation Routes in the Area.

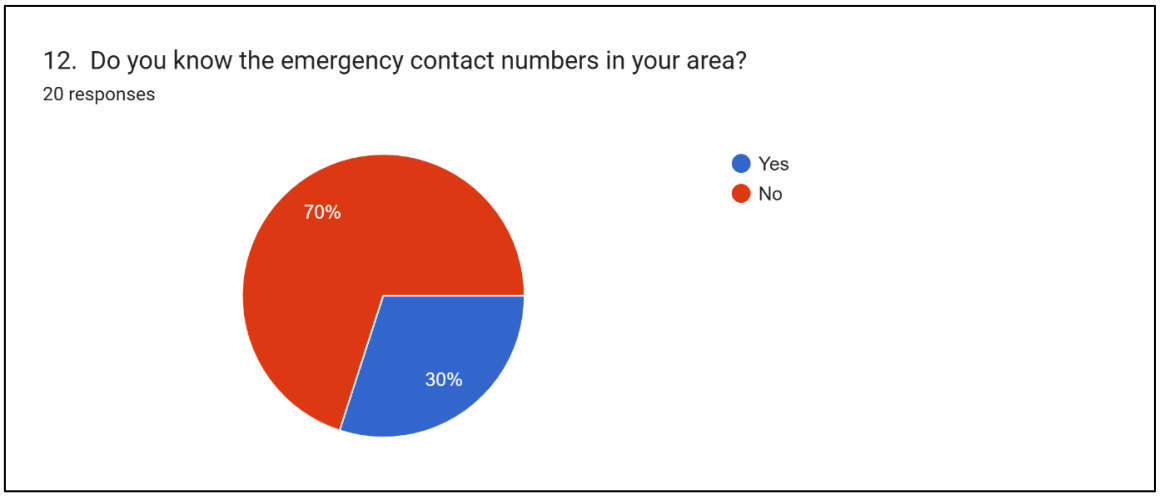


Figure 3.5: Awareness of Emergency Contact Numbers in the Area.

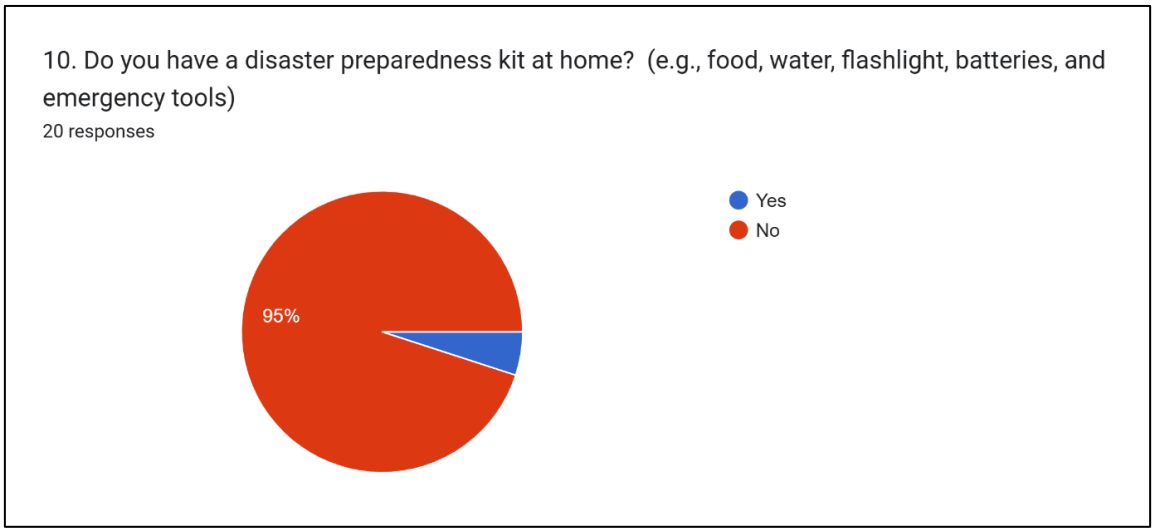


Figure 3.6: Presence of Disaster Preparedness Kits at Home.

In terms of perception and attitude, 90% of respondents believe that disaster preparedness and awareness can significantly reduce the impact of a disaster. Additionally, 80% of respondents think that a lack of knowledge or training is a barrier to better disaster preparedness, while 75% believe that inadequate information or guidance is a key obstacle. Regarding responsibilities, respondents feel that local governments should bear the highest responsibility, followed by national governments, and then individuals and families.

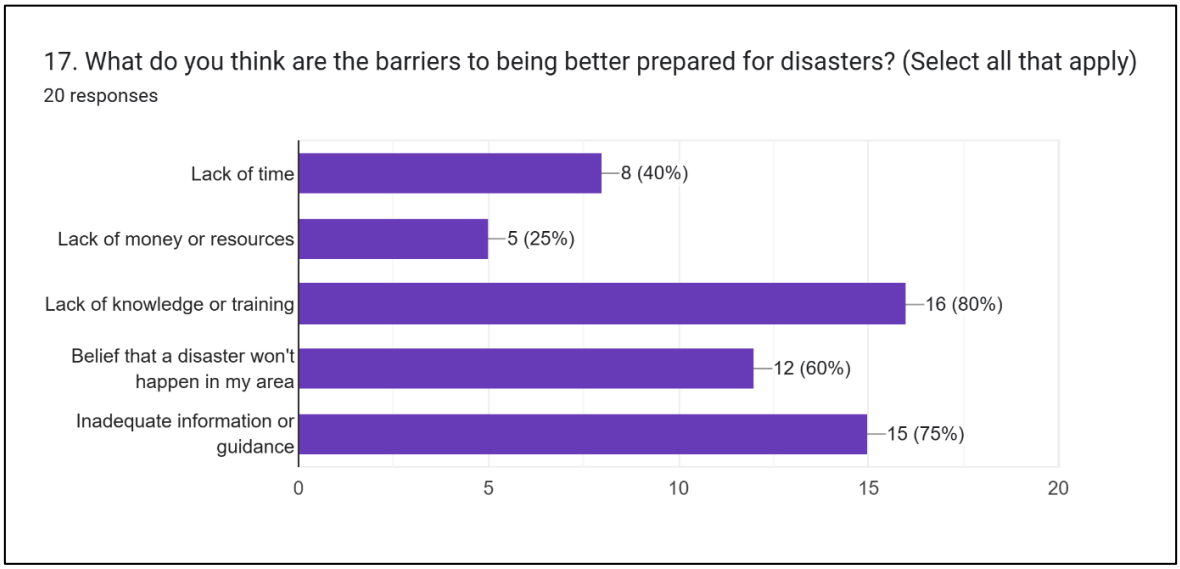


Figure 3.7: Barriers to Better Disaster Preparedness.

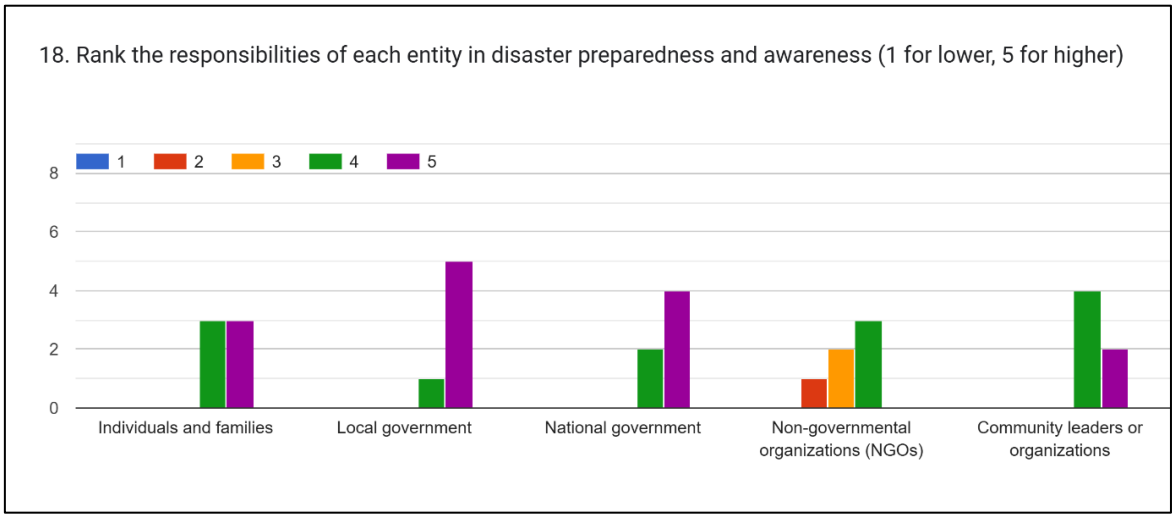


Figure 3.8: Ranking of Responsibilities in Disaster Preparedness and Awareness.

80% of respondents prefer hands-on training to improve disaster preparedness and awareness, while 75% prefer interactive games or simulations. This shows that games can be a tool for enhancing disaster preparedness and awareness. Furthermore, all respondents are willing to use a game-based tool for this purpose.

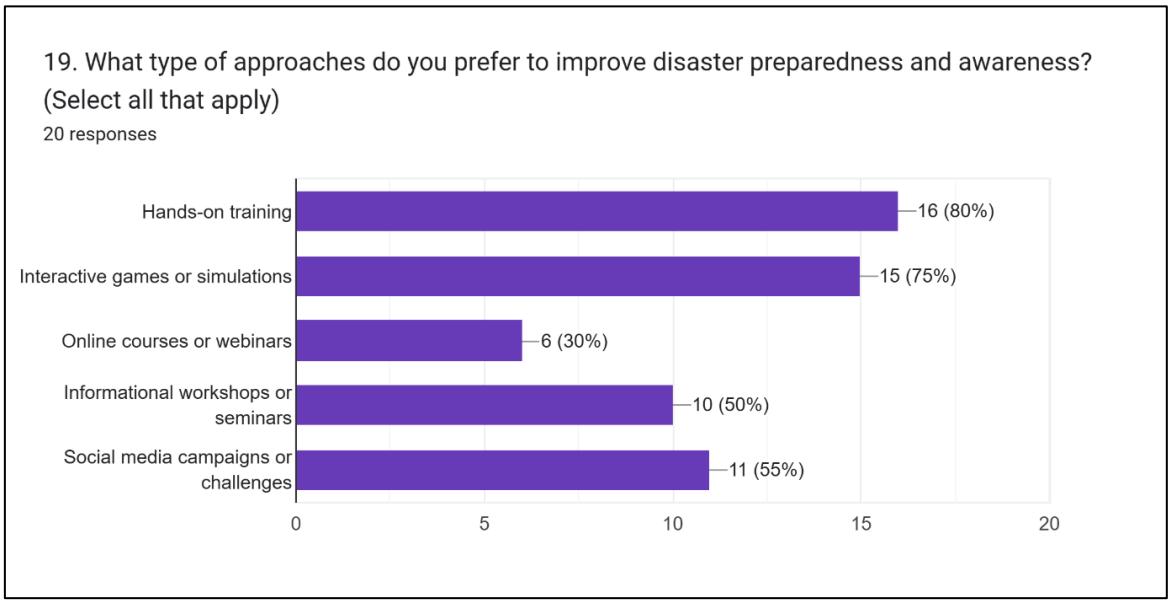


Figure 3.9: Preferred Approaches to Improve Disaster Preparedness and Awareness.

In terms of game preferences, 75% of respondents prefer using mobile devices or tablets to play games. Therefore, the game will be designed with mobile and tablet users in mind, given their higher preference compared to other platforms.

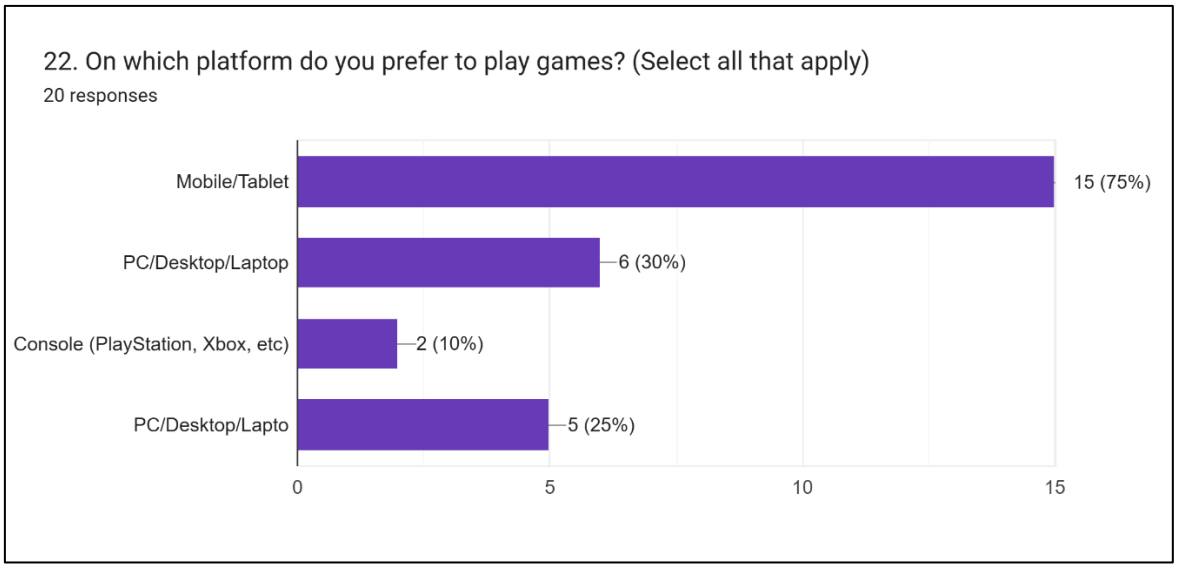


Figure 3.10: Preferred Gaming Platform.

For the disaster preparedness and awareness game, 50% of respondents preferred simulated real-life disaster situations as the most engaging feature. Meanwhile, timed challenges for quick decision-making and strategy-based planning for disaster response were each chosen by 25% of respondents. Notably, problem-solving puzzles or riddles received no selections, indicating a lack of interest in incorporating puzzles into disaster preparedness and awareness games. These insights suggest that a combination of simulations, strategic elements, and time-sensitive challenges could create an engaging and effective learning environment for disaster preparedness education.

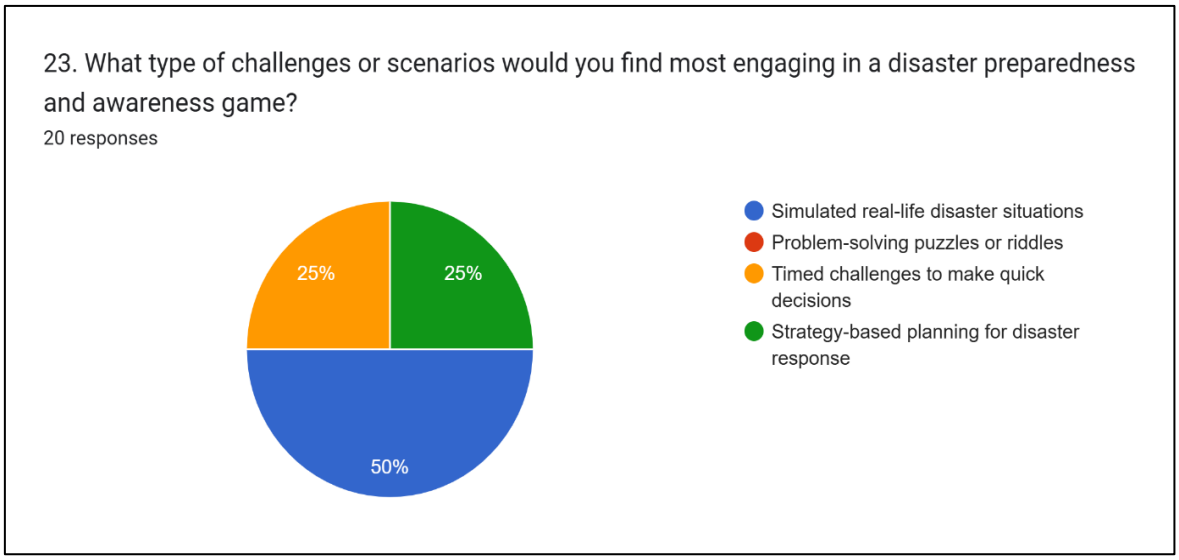


Figure 3.11: Engaging Challenges or Scenarios in a Disaster Preparedness Game.

For the game features, 80% of respondents think that realistic disaster scenarios can make the game engaging. Storyline or role-playing elements were chosen by 70% of respondents. Additionally, 60% of respondents showed interest in time-sensitive decisions. Interactive quizzes or challenges received 40% of respondents' interest, while only 25% selected a rewards or points system. These results assist in identifying the key features and sub-features of the game, prioritizing them based on respondent interest. Additionally, the learning content that users are willing to engage with in a disaster preparedness and awareness game was also collected from respondents, providing input for the game development.

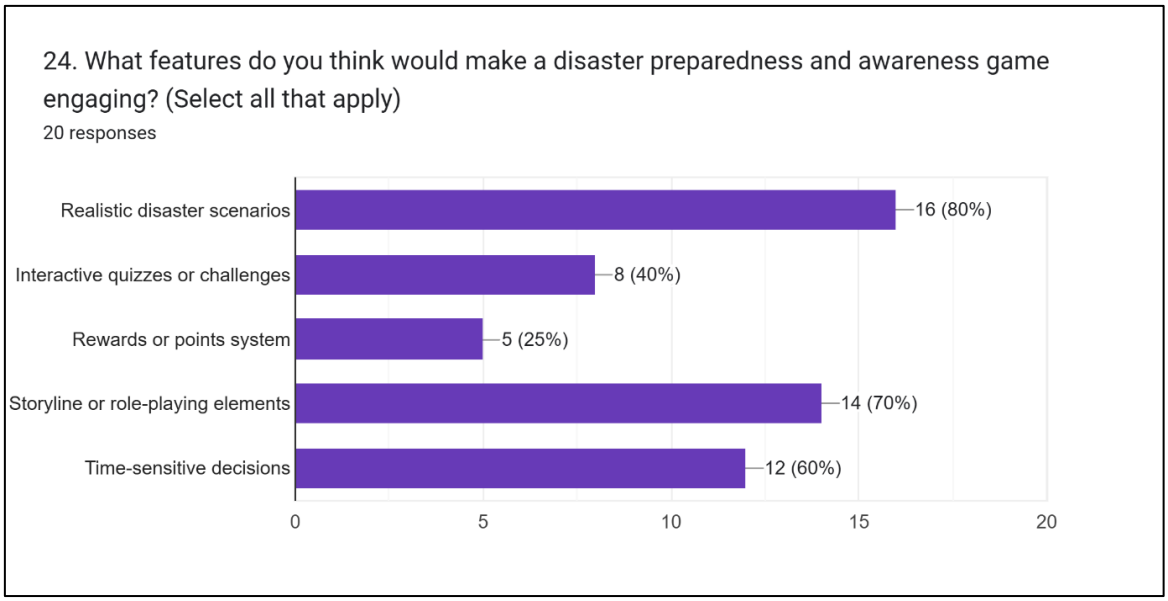


Figure 3.12: Features that Make a Disaster Preparedness Game Engaging.

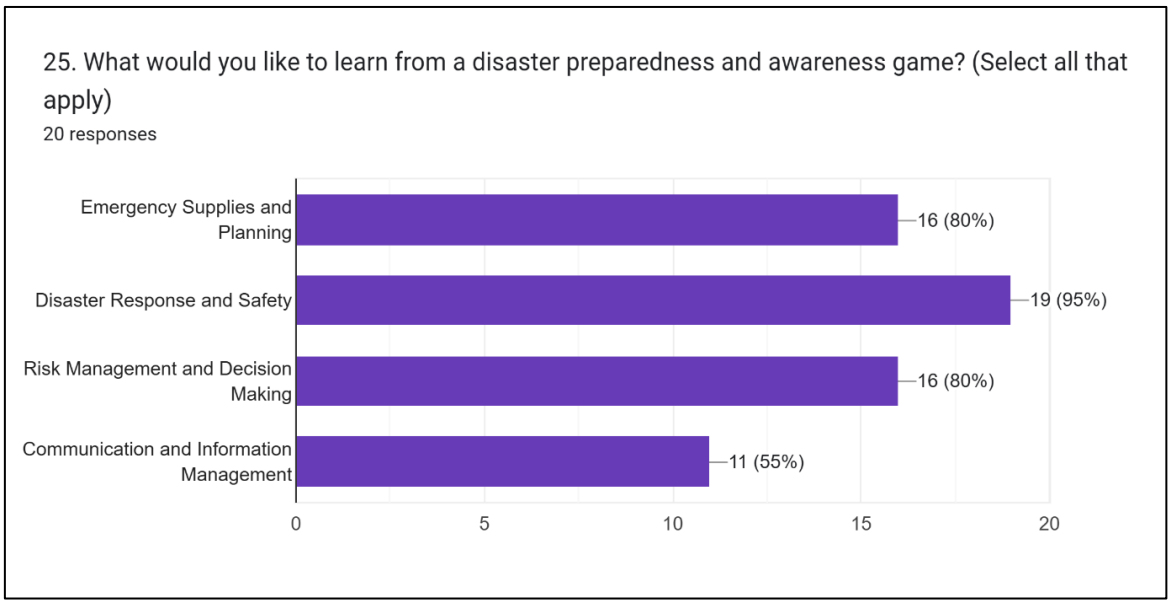


Figure 3.13: Learning Preferences from a Disaster Preparedness Game.

3.4 Design Specifications

This section provides an overview of the key design specifications for the game, which include the system's architecture, gameplay flow, and user interface elements. It covers various diagrams, such as the use case diagram, sequence diagram, activity diagram, and entity relationship diagram (ERD), as well as the user interface wireframes. These

specifications ensure that the game's design aligns with the functional and non-functional requirements derived from the user research.

3.4.1 Use Case Diagram

The use case diagram is used to describe the functionality of the game system. The use case diagram below shows the interactions between the Player and the Game System. It includes user-related functionalities such as Register and Login. Basic functionalities of the game include Start Game, Restart Game, Exit Game, and Adjust Sound Settings, which provide controls for gameplay and customization. Advanced game mechanics and feature-specific functionalities include Make Action, End Turn, Select Level, and View Achievements, which enable player to engage with the game content, track progress, and interact with core gameplay elements.

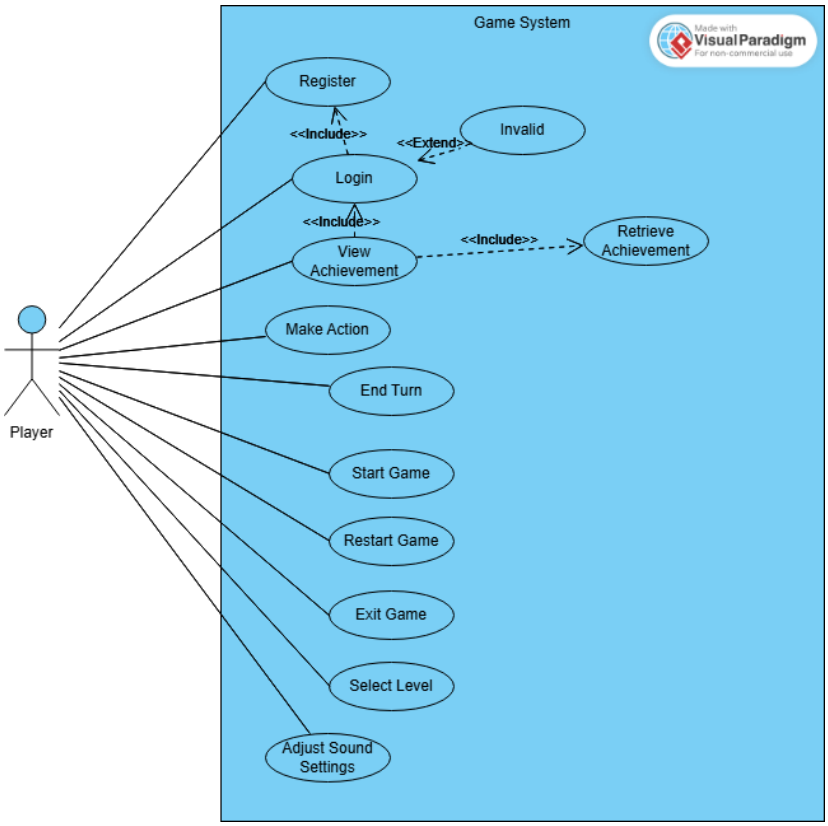


Figure 3.14: Use Case Diagram of the Game System.

3.4.2 Use Case Specifications

Use case specification is a textual detail for the use case to illustrate the sequence of events in the game system. The use case specifications show as below:

(1) Register

Table 3.1: Register Use Case Specification.

Use Case	Register
Actors	Player
Description	Allows a new player to create an account in the game system.
Pre-conditions	• The game system is online and accessible. • The player has not already registered an account with the same email or username.
Post-conditions	• The player's account details are saved in the database. • The player is redirected to the login screen or logged in automatically.
Basic Flow	1. The player selects the "Register" option on the login screen. 2. The game system displays a registration form. 3. The player enters the required details. 4. The game system validates the entered information. [E1] [E2] 5. The game system creates a new account with the provided details and stores it in the database. [E3] 6. The game system confirms successful registration to the player.
Alternate Flow	None

Table 3.1 continued

Exception Flow	[E1] Username or email already taken. The player is prompted to choose a different username or email and resubmit. [E2] Password does not meet requirements. The player is prompted to re-enter a password. [E3] Network or database error. The game system displays an error message and advises the player to try again later.

(2) Login

Table 3.2: Login Use Case Specification.

Use Case	Login
Actors	Player
Description	Allows the player to access their account by entering valid credentials.
Pre-conditions	• The game system is online and connected to the database. • The player has a registered account.
Post-conditions	• The player is authenticated and redirected to the game interface. • The system retrieves and loads the player's game data.

Table 3.2 continued

Basic Flow	1. The player selects the "Login" option on the game's main screen. 2. The system displays the login form. 3. The player enters their email or username and password. [A1] [E3] 4. The system validates the credentials against the database. [E1] [E2] 5. The player gains access to the game with their account data. [E3]
Alternate Flow	[A1] Third-party login. The game system redirects to the respective platform.
Exception Flow	[E1] Invalid credentials. The player is prompted to re-enter their credentials. [E2] Account not found. The player is prompted to register a new account or retry. [E3] Forgot password. The game system redirects to a password recovery process. [E4] Network or database error. The game system displays an error message and advises the player to try again later.

(3) View Achievement

Table 3.3: View Achievement Use Case Specification.

Use Case	View achievement

Table 3.3 continued

Actors	Player
Description	Allows the player to view their unlocked and locked achievements in the game.
Pre-conditions	• The player must be logged in to view achievements.
Post-conditions	• The player views the list of achievements
Basic Flow	1. The player navigates to the "Achievements" section from the main menu. 2. The game system retrieve achievement from the database. 3. The game system displays the list of achievements to the player. [E1]
Alternate Flow	None
Exception Flow	[E1] Network and database error. The game system displays an error message and gives the option to retry or return to the main menu.

(4) Start Game

Table 3.4: Start Game Use Case Specification.

Use Case	Start game
Actors	Player
Description	The player initiates a new game session, triggering the system to set up the game data, shuffle the deck, and prepare for the player's first turn.

Table 3.4 continued

Pre-conditions	• The game is installed and successfully launched. • The player has selected the necessary game settings (if applicable).
Post-conditions	• A new game session is successfully created. • The game deck is prepared and shuffled. • The game data is initialized. • The game is ready for the player to take their first turn or enter the initial game phase.
Basic Flow	1. The player selects the "Start Game" option from the main menu. 2. The game system initializes the game data. [E1] 3. The game system prepares the player's deck, shuffles it, and allocates the initial hand of cards. [E1] 4. The game data is successfully initialized, and all necessary elements are initialized. 5. The system transitions to the player's first turn, displaying the game interface and available actions.
Alternate Flow	None
Exception Flow	[E1] Setup error. The game system displays an error message and gives the option to retry the setup or return to the main menu.

(5) Select Level

Table 3.5: Select Level Use Case Specification.

Use Case	Select level
Actors	Player
Description	The player selects a game level to play. The selected level determines the game difficulty, rules, and challenges.
Pre-conditions	• The game is installed and successfully launched. • The player has navigated to the level selection screen.
Post-conditions	• The selected level is confirmed and saved. • The game is ready to transition to the Start Game phase, where the game data is initialized based on the selected level.
Basic Flow	1. The player navigates to the level selection screen. 2. The player browses the available levels and selects a desired level. 3. The game system validates the player's selection to ensure it is available. [E1] 4. The selected level is confirmed, and the game transitions to the Start Game phase.
Alternate Flow	None
Exception Flow	[E1] Invalid level selection. The game system displays a message indicating the reason.

(6) Make Action

Table 3.6: Make Action Use Case Specification.

Use Case	Make action
Actors	Player
Description	The player performs an in-game action during their turn, such as playing a card, activating an ability, or interacting with the game board. This action modifies the game state.
Pre-conditions	• The game starts. • The player's turn starts. • The player has actions available.
Post-conditions	• The selected action is executed, and the game state is updated. • The player can continue performing additional actions, provided conditions are met.
Basic Flow	1. The player selects an available action. 2. The game system validates the action for eligibility. [E1] 3. The game system processes the action. 4. The game state is updated. 5. The system displays the outcome of the action to the player. 6. The player may continue making further actions or decide to end their turn.
Alternate Flow	None
Exception Flow	[E1] Invalid action. The game system displays an error message indicating the reason.

(7) End Turn

Table 3.7: End Turn Use Case Specification.

Use Case	End turn
Actors	Player
Description	The player ends their turn, triggering the system to transition to the next phase of the game. This may include resolving end-of-turn effects, replenishing resources, or initiating the opponent's turn.
Pre-conditions	• The player has completed their desired actions for the turn or has no remaining actions. • The game state allows the turn to end.
Post-conditions	• The player's turn is ended. • Any end-of-turn effects are resolved. • The game transitions to the next phase.
Basic Flow	1. The player clicks the "End Turn" button. [A1] 2. The game system checks for pending actions or unresolved effects. 3. The game system processes any end-of-turn effects. 4. Resources are replenished or reset. 5. The game transitions to the next phase. 6. The game system displays a notification confirming the end of the turn. [E1]
Alternate Flow	[A1] Confirm end turn. The player can either confirm to proceed or cancel to resume their turn.

Table 3.7 continued

Exception Flow	[E1] End turn error. The game system displays an error message. The game state is restored to the last valid state, and the player is prompted to retry ending the turn.

(8) Exit Game

Table 3.8: Exit Game Use Case Specification.

Use Case	Exit game
Actors	Player
Description	The player exits the current game session, returning to the main menu.
Pre-conditions	• The game is in progress. • The game is in a state that allows exiting.
Post-conditions	• The current game session is terminated (if applicable). • The player is returned to the main menu.
Basic Flow	1. The player selects the "Exit Game" option from the in-game menu. 2. The game system prompts the player for confirmation to exit. [A1] 3. The game system terminates the currrent game session. 4. Returns to the main menu.
Alternate Flow	[A1] Cancel exit. The current game session continues uninterrupted.

Table 3.8 continued

Exception Flow	[E1] Error while exiting. The game system displays an error message and prompts the player to retry.

(9) Restart Game

Table 3.9: Restart Game Use Case Specification.

Use Case	Restart game
Actors	Player
Description	The player restarts the current game session, resetting all game elements to their initial states.
Pre-conditions	• A game session is currently in progress. • The game is in a state that allows restarting.
Post-conditions	• The current game session is terminated. • A new game session is created with all elements reset to their initial states.
Basic Flow	1. The player selects the "Restart Game" option from the result menu. 2. Reinitializes the game, deck, and resources. 3. Starts a new game session.
Exception Flow	[E1] Setup error. The game system displays an error message and gives the option to retry the setup or return to the main menu.

(10) Adjust Sound Settings

Table 3.10: Adjust Sound Settings Use Case Specification.

Use Case	Adjust sound settings
Actors	Player
Description	The player adjusts the game's sound settings, such as volume levels for background music, or sound effects.
Pre-conditions	• The game is in progress or at a menu screen where settings can be accessed. • The sound settings menu is available.
Post-conditions	• The new sound settings are applied immediately or saved for future sessions. • The audio experience reflects the updated settings.
Basic Flow	1. The player navigates to the "Settings" menu and selects "Sound Settings." 2. The sound settings menu is displayed, showing adjustable sliders or options for master volume, background volume and sound effect volume. 3. The player adjusts the desired settings using sliders or toggles. [A1] 4. The system updates the sound settings in real-time to reflect the changes. 5. The player confirms and saves the settings (if applicable) or exits the menu, applying the changes.

Table 3.10 continued

Alternate Flow	[A1] Reset to default. The game system applies the default settings immediately.
Exception Flow	[E1] Setting error. The system reverts to the last valid configuration and displays an error message.

3.4.3 Sequence Diagram

This section presents the sequence diagrams that illustrate the interactions between the player and the game system throughout various stages of gameplay. The three key sequences covered are: start game, make action, and end turn. These diagrams provide a detailed view of how the game progresses and how the system responds to the player's actions in each phase.

The sequence diagram below illustrates the flow of interactions within the game system during the process of starting a game. The Player initiates the sequence by invoking the start game function on the Game Manager, which orchestrates the setup process. The Game Manager sets the game state and resets the resources by calling the Cost Manager, Defense Manager, and Discard Manager to reset all resource values to their default. Subsequently, the Game Manager interacts with the Enemy Manager to set up the enemy card. Once the enemy card is ready, it triggers the Turn Manager to start the turn. After the turn begins, the game is ready for the player’s turn.

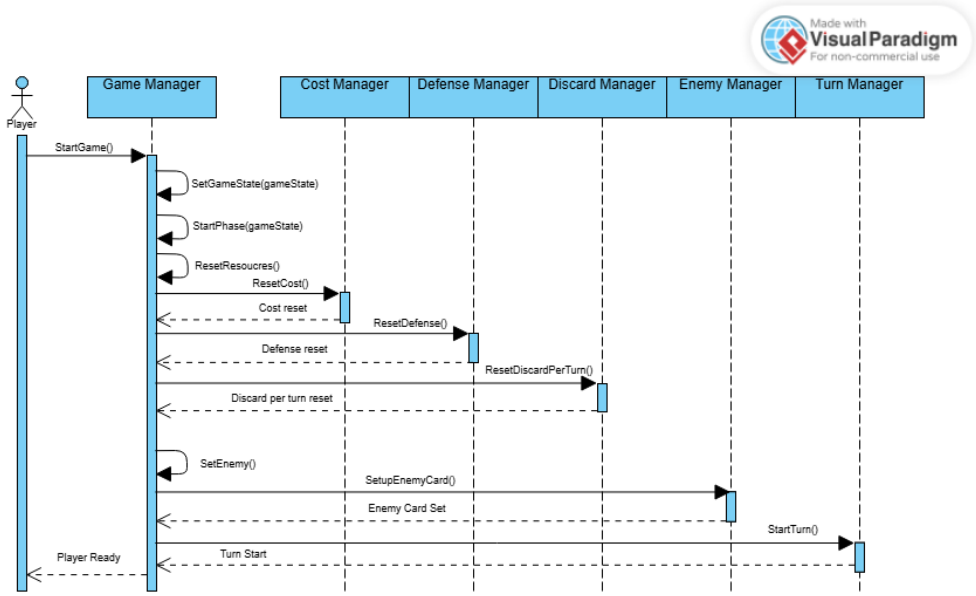


Figure 3.15: Start Game Sequence Diagram.

The sequence diagram below illustrates the process of a player taking actions which is playing a card in the game. The player plays a card onto the Drop Area. The Drop Area then calls the Action System to resolve the card's effects. After the effects are applied, the Defense Manager updates the player's defense accordingly. This marks the completion of the player's action.

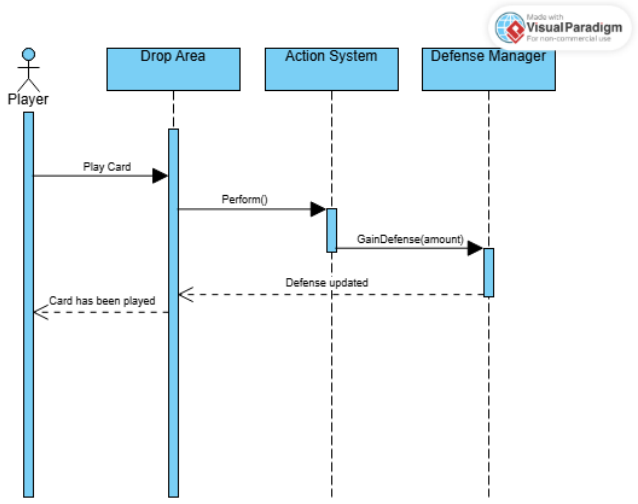


Figure 3.16: Make Action Sequence Diagram.

The sequence diagram below illustrates the process that occurs when a player ends their turn. The Game Manager updates the game state, and the Action System resolves the opponent's actions and updates the player's health through the Health Manager. Finally, the Game Manager changes the game state back to the start of the turn.

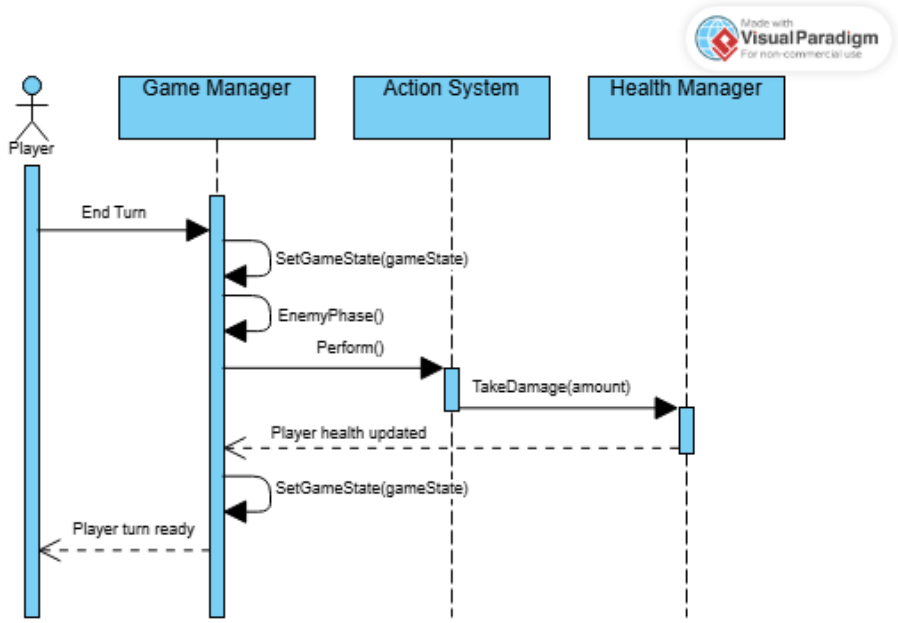


Figure 3.17: End Turn Sequence Diagram.

3.4.4 Activity Diagram

The section presents an activity diagram that outlines the flow of actions within the turn-based gameplay. It showcases how the player takes an action, followed by the disaster applying its effects. This cycle repeats until a specified condition is met and ends the game.

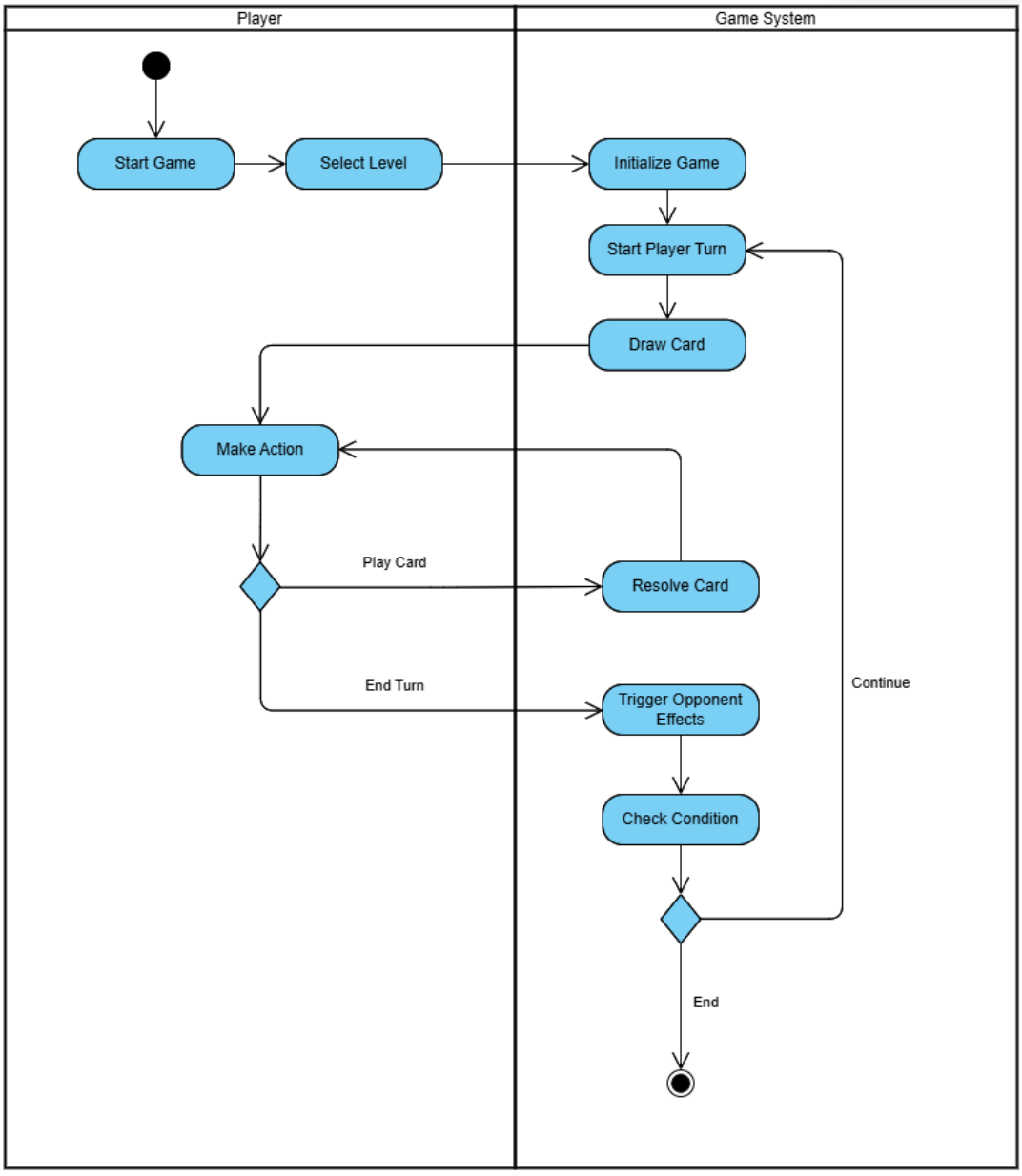


Figure 3.18: Game Activity Diagram.

3.4.5 Entity Relationship Diagram (ERD)

This section presents the Entity Relationship Diagram (ERD) that models the relationships between key entities in the game system. It includes entities related to achievements, such as player and their progress in unlocking achievements. The ERD below

illustrates how these entities interact and how achievement data is stored and managed within the system.

There are two main entities in this diagram, which are Player and Achievement. The Player entity tracks information about the player, such as player ID and other relevant details. The Achievement entity represents the player's achievements in the game, such as the achievement ID and its description. The relationship between the Player and Achievement is that a player can unlock multiple achievements throughout the game. This relationship is shown in the diagram with a one-to-many relationship, meaning each player can have multiple achievements.

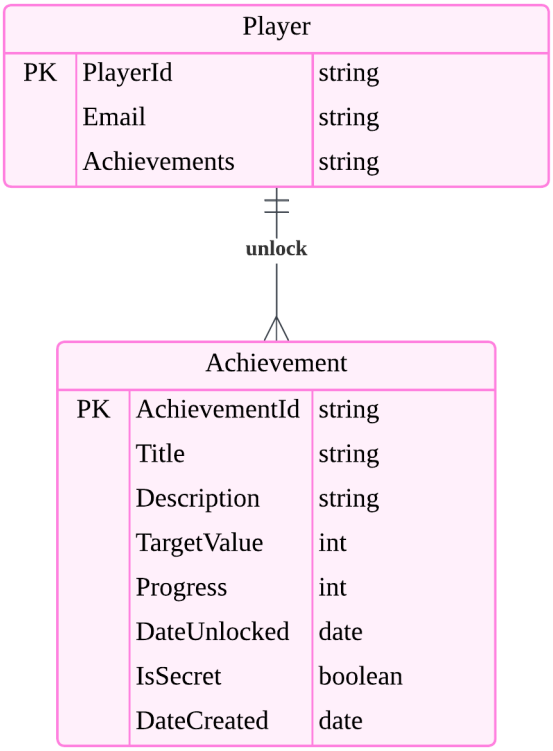


Figure 3.19: Entity Relationship Diagram for Achievement.

3.5 User Interface Wireframes

This section presents low-fidelity wireframes designed using Figma to outline the basic structure and layout of the game's user interface. The wireframes include key screens such as the homepage, difficulty selection, gameplay, victory and defeat, and achievements. These designs provide a visual guide for the game's user interface and ensure clarity and usability in the final implementation. The layout and concept design may evolve during the development process to accommodate technical requirements or feedback.

The main menu layout is designed to provide a clean and intuitive user experience. At the center, the game logo and title highlight the game’s identity and branding. Below it is the Play button to start the game. In the top-left corner is the Quit button, while the top-right corner contains the Tutorial and Settings buttons. On the right side are the Profile and Achievement buttons. The layout balances simplicity and clarity, ensuring that players can easily navigate through the menu.

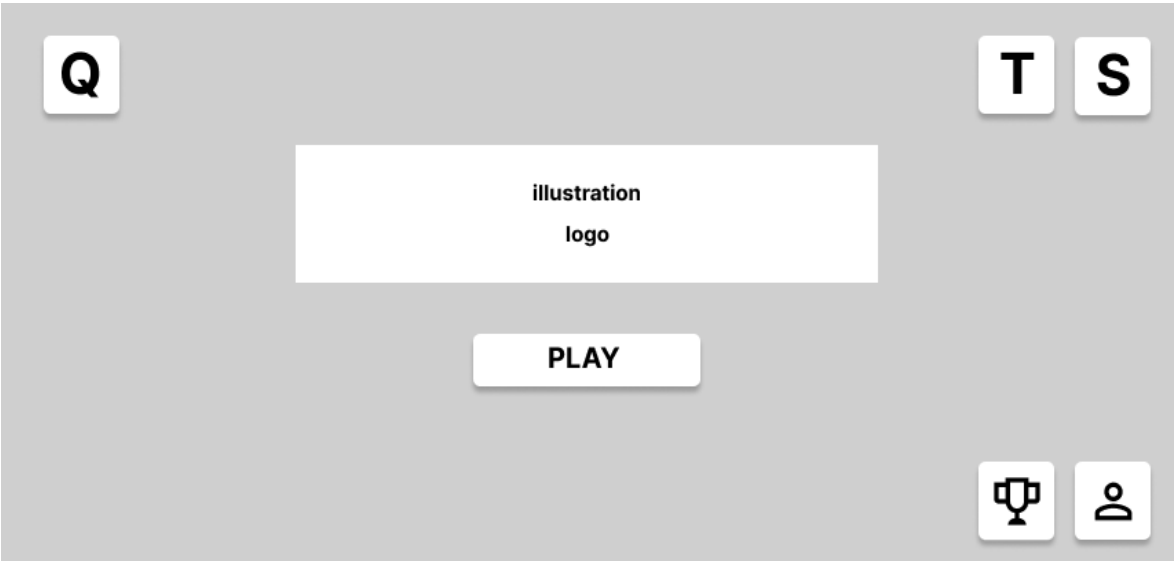


Figure 3.20: Homepage of the Game.

The difficulty selection screen layout includes a visual representation of each stage at the top. Below the visuals, the player can select the desired difficulty by clicking on them, with a highlighted border indicating the current selection. A Start button is positioned at the bottom to begin the game, while a Back arrow in the top-left corner allows the player to return to the previous menu.

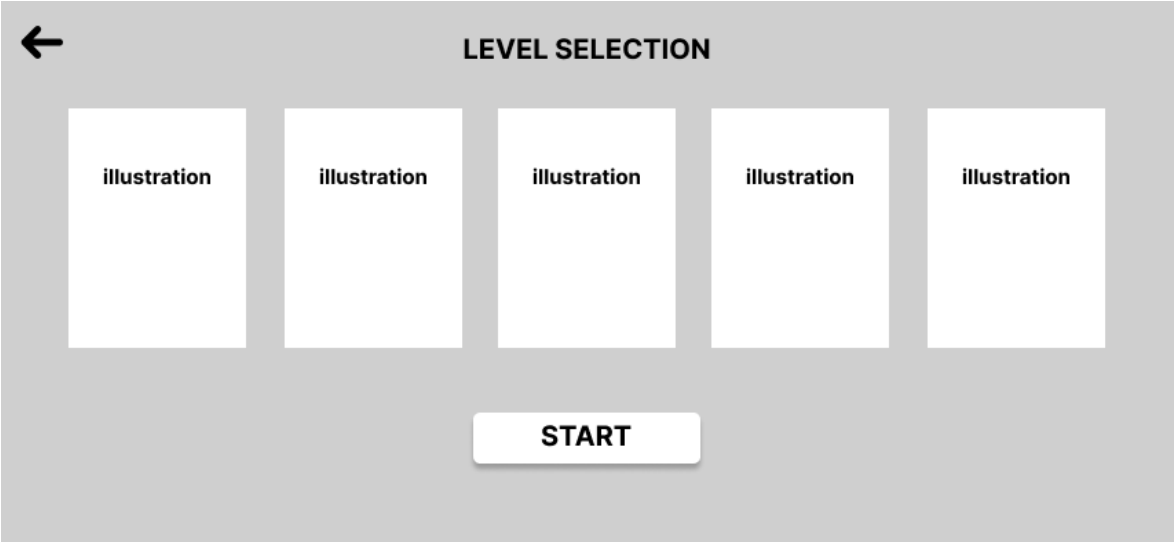


Figure 3.21: Difficulty Selection of the Game.

The gameplay interface is designed around an organized and centered playfield. At the center is the main area where cards are placed, with dedicated spaces for both the player's cards and the opponent's disaster cards. On the left, a used card area displays discarded cards, while resource and effect icons below it help track the game status. The right side features the card deck area for drawing new cards. Above the card deck are the End Turn button and Cost Indicator. A Settings icon is placed in the top-left corner for quick access to configuration options, while the Card Collection and Tutorial buttons are in the top-right corner. The layout ensures functionality and clear visibility of all essential game components.

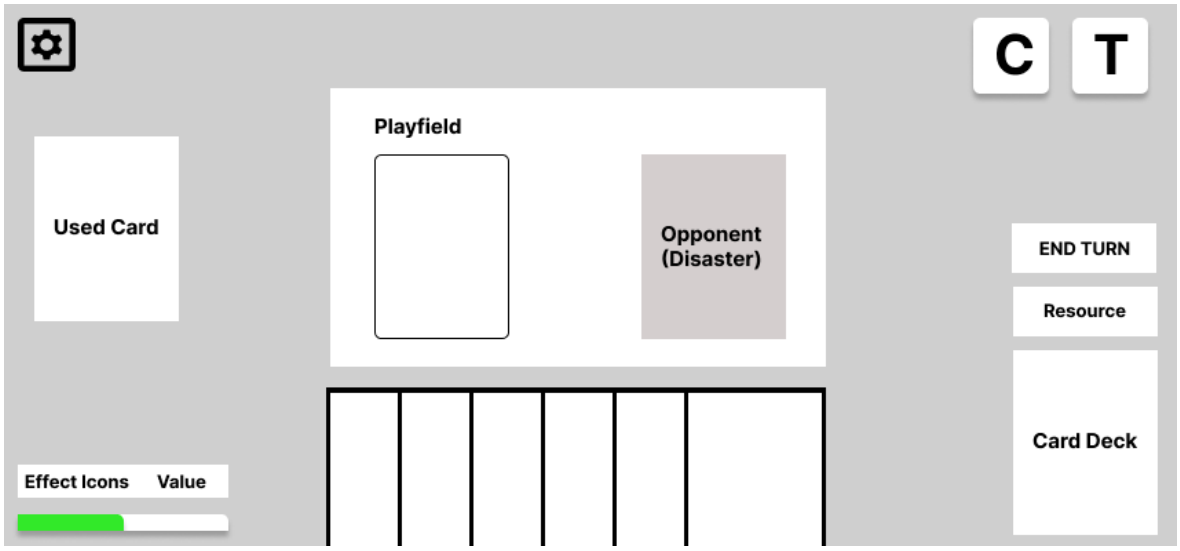


Figure 3.22: Gameplay of the Game.

The results screen layout is designed to display the outcome of the game in a clear and visually engaging manner. At the center, an illustration conveys victory or defeat, accompanied by star icons that visually represent the player's performance. Below the illustration, the total points earned are prominently displayed. Three buttons are positioned at the bottom, a retry option to replay the game, an "OK" button to confirm and exit the results, and a menu button to navigate to additional options.

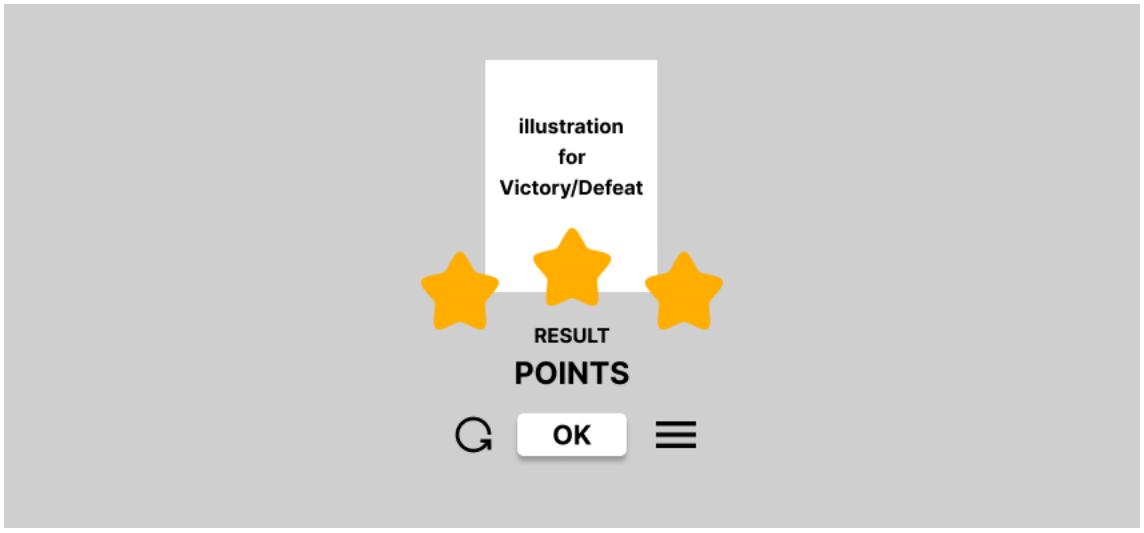


Figure 3.23: Victory and Defeat of the Game.

The achievements screen provides a structured view of the player's accomplishments, each displayed with a title, description, and an icon for visual recognition. An icon is placed beside the completed achievements. A back button at the top allows navigation to previous menus. The achievements screen enables player to track their progress and milestones in the game.

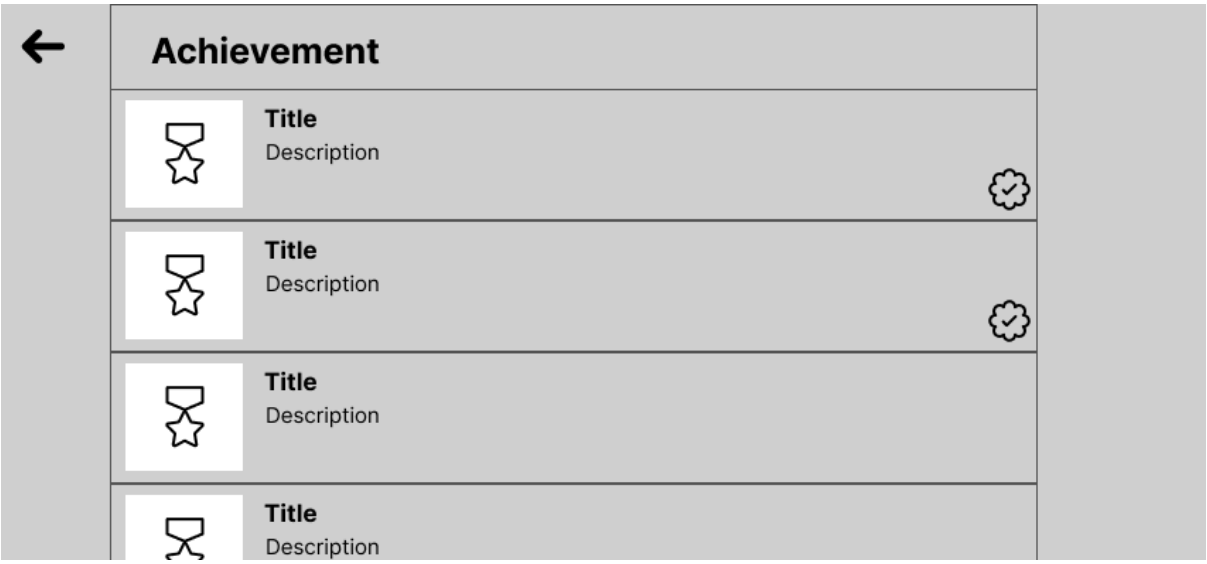


Figure 3.24: Achievement of the Game.

3.6 Summary

This project adopts the Agile Model by dividing game development into iterative sprints to ensure flexibility, continuous feedback, and good quality outcomes. User requirements were gathered through a questionnaire targeting university students. The results collected reveal significant gaps in disaster preparedness knowledge, such as low awareness of evacuation routes and emergency resources. The findings provide valuable insights and ideas for the game development, including core mechanics, features, and content. The detailed use case scenarios, system diagrams, and low-fidelity wireframes created by Figma guide the implementation of core mechanics, gameplay, and user interface design.

CHAPTER 4

IMPLEMENTATION

4.1 Hardware and Software Requirements

4.1.1 Playing Requirements

This section specifies the minimum and recommended specifications needed for users to play the game on Android devices.

Table 4.1: Playing Requirements.

Components	
Minimum Software Requirements	Android 7.0 “Nougat” (API Level 24)
Recommended Software Requirements	Android 10 or higher for better performance and security updates.
Game Size	The final APK size is approximately 130 MB, including all assets, scripts, and resources.
Storage Requirements	It is recommended that users have at least 200 MB of free storage space to accommodate the game installation and gameplay data.
Network Requirements	Internet connection: For users to access the login and sign-up features and synchronize achievements. Data Usage: The game uses minimal data during authentication and achievement syncing.

4.1.2 Development Requirements

This section specifies the hardware and software used for developing the game.

Hardware Requirements

1. **Development Machine**
    - CPU: Intel Core i5 9th Gen
    - RAM: 16 GB
    - Storage: SSD with at least 70 GB of free space available for Unity projects, build files, and software
    - Display: Full HD (1920x1080) resolution or higher
2. **Android Device**

These devices are used for testing and debugging:

- POCO F5 (Android 14)
- Black Shark 4 (Android 11)

Software Requirements

1. **Game Engine and Build Tools**

The game was developed using the Unity Game Engine and built through the Unity platform. The development and build environment include the following tools:

- Game Engine: Unity 2022.3.5f1
- Android Build Tools (managed via Unity):
    - OpenJDK
    - Android SDK & NDK Tools

2. **Integrated Development Environment (IDE)**

The game's scripts were developed and debugged using Visual Studio 2022, which supports C# programming for Unity.

3. Firebase Integration

The game uses the Firebase Unity SDK packages version 12.10.0, imported via Unity's Package Manager. Firebase services were integrated into the game to handle user authentication and data storage. The following Firebase modules were used:

- Firebase Authentication:

  Used to implement a secure login and sign-up system. It supports user registration, sign-in, and session management through email and password credentials.

- Firebase Realtime Database:

  Used to store and manage player achievement data. Achievements are synchronized in real-time with the cloud. It allows users to access their progress across multiple sessions and devices.

4. Other Libraries

The game utilizes several libraries to enhance development efficiency and implement specific features:

- DOTween**:**

  A tweening library used for smooth animations and transitions, such as card movements and UI effects.

- SerializeReference**:**

  A Unity attribute that enables polymorphic serialization for modular data structures within the game.

5. Game Assets and Sprites

- The game uses 2D sprites for cards, backgrounds, buttons, and UI elements.

- Audio assets include sound effects for card interactions, button clicks, and background music to enhance gameplay experience.
- Assets were sourced from:
    - Canva for designing and editing custom graphics.
    - AI generated assets
    - Unity Asset Store
    - Various open-source websites
- All sprite assets were used under open licenses with appropriate attribution in the game.

4.2 Implementation

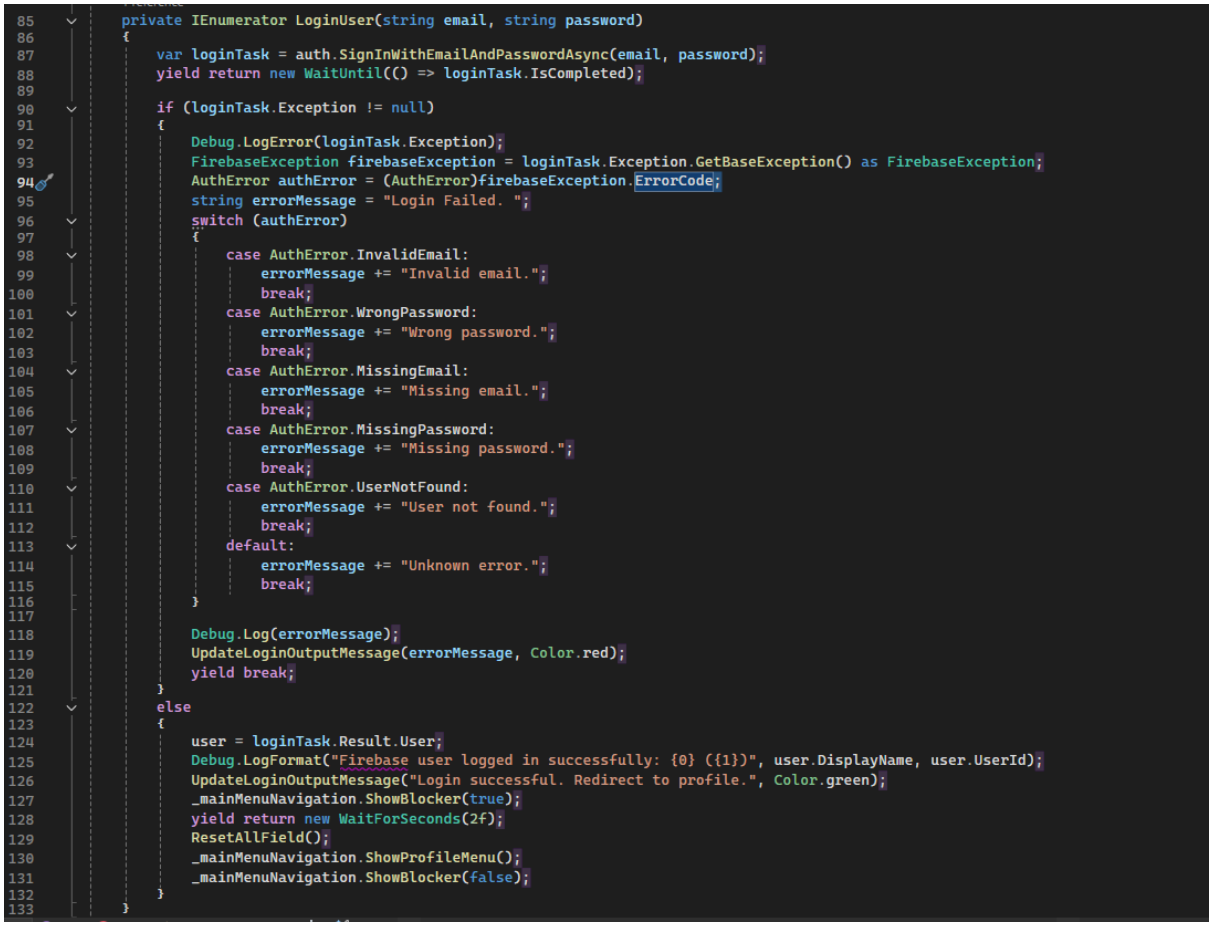
4.2.1 System Architecture

The system architecture of the game is built using modular components to efficiently manage various gameplay features. The core systems include the Action System, Turn System, Card System, Health and Defense System, and Cost System. Each of these systems is managed by dedicated managers responsible for handling specific features. These systems are coordinated through centralized control by the Game Manager, which handles the overall game flow such as turn progression, card play, and health updates.

4.2.2 Key Features and Functions

4.2.2.1 Login and Signup System

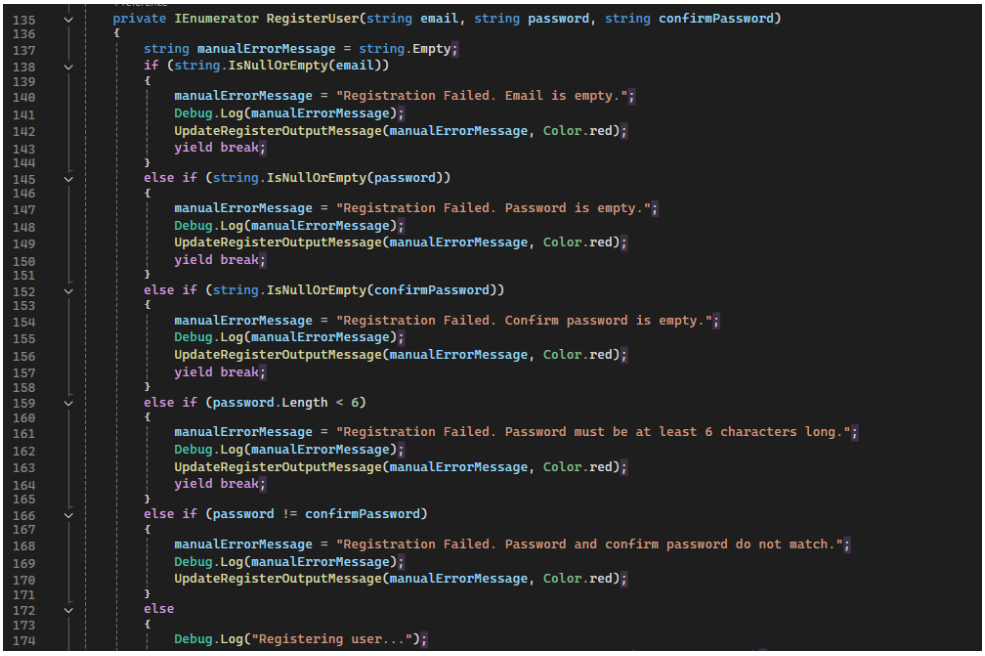
The game implements a secure login and signup system using Firebase Authentication to manage user accounts. The figure 4.1 below shows the `LoginUser` function.



```
85      private IEnumerator LoginUser(string email, string password)
86      {
87          var loginTask = auth.SignInWithEmailAndPasswordAsync(email, password);
88          yield return new WaitUntil(() => loginTask.IsCompleted);
89
90          if (loginTask.Exception != null)
91          {
92              Debug.LogError(loginTask.Exception);
93              FirebaseException firebaseException = loginTask.Exception.GetBaseException() as FirebaseException;
94              AuthError authError = (AuthError)firebaseException.ErrorCode;
95              string errorMessage = "Login Failed. ";
96              switch (authError)
97              {
98                  case AuthError.InvalidEmail:
99                      errorMessage += "Invalid email.";
100                     break;
101                 case AuthError.WrongPassword:
102                     errorMessage += "Wrong password.";
103                     break;
104                 case AuthError.MissingEmail:
105                     errorMessage += "Missing email.";
106                     break;
107                 case AuthError.MissingPassword:
108                     errorMessage += "Missing password.";
109                     break;
110                 case AuthError.UserNotFound:
111                     errorMessage += "User not found.";
112                     break;
113                 default:
114                     errorMessage += "Unknown error.";
115                     break;
116             }
117
118             Debug.Log(errorMessage);
119             UpdateLoginOutputMessage(errorMessage, Color.red);
120             yield break;
121         }
122         else
123         {
124             user = loginTask.Result.User;
125             Debug.LogFormat("Firebase user logged in successfully: {0} ({1})", user.DisplayName, user.UserId);
126             UpdateLoginOutputMessage("Login successful. Redirect to profile.", Color.green);
127             _mainMenuNavigation.ShowBlocker(true);
128             yield return new WaitForSeconds(2f);
129             ResetAllField();
130             _mainMenuNavigation.ShowProfileMenu();
131             _mainMenuNavigation.ShowBlocker(false);
132         }
133     }
```

Figure 4.1: Implementation of LoginUser Function.

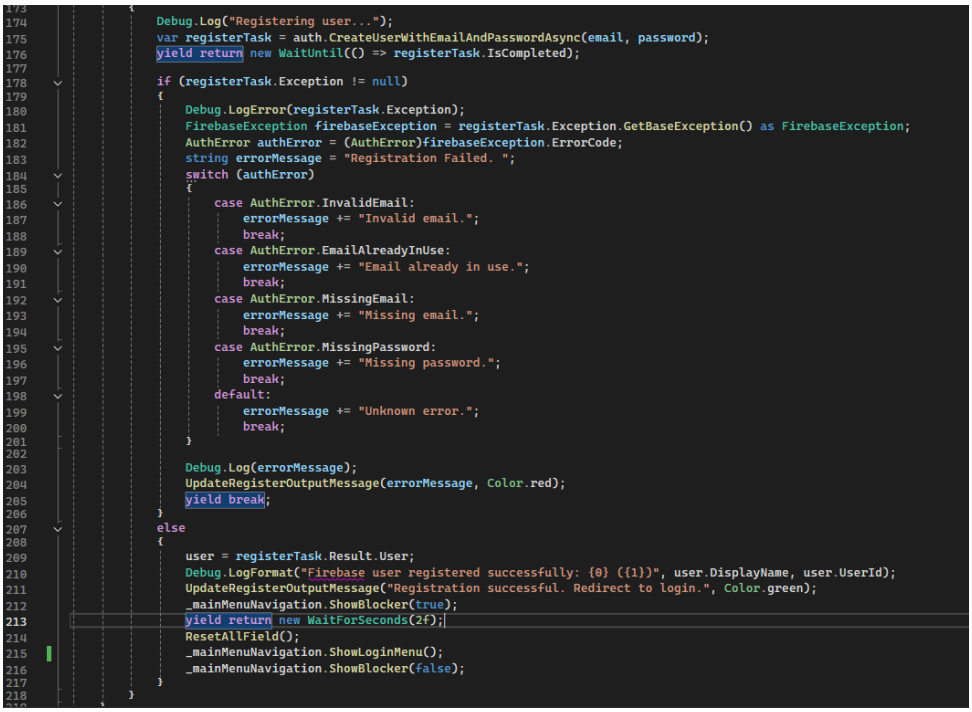
The function collects the user's email and password from input fields. Then it uses Firebase Authentication's `SignInWithEmailAndPasswordAsync` method to validate the credentials. If the login is successful, the user is directed to the profile menu; if it fails, an error message is displayed to guide the user. Line 90 – 121 handle the display of error messages. Lines 124 – 131 manage the navigation to the profile menu upon successful login. The user registration function follows a similar structure to the login function. The `RegisterUser` function is responsible for creating a new user account using Firebase Authentication. The figure 4.2 and 4.3 below show the `RegisterUser` function.



```
private IEnumerator RegisterUser(string email, string password, string confirmPassword)
{
    string manualErrorMessage = string.Empty;
    if (string.IsNullOrEmpty(email))
    {
        manualErrorMessage = "Registration Failed. Email is empty.";
        Debug.Log(manualErrorMessage);
        UpdateRegisterOutputMessage(manualErrorMessage, Color.red);
        yield break;
    }
    else if (string.IsNullOrEmpty(password))
    {
        manualErrorMessage = "Registration Failed. Password is empty.";
        Debug.Log(manualErrorMessage);
        UpdateRegisterOutputMessage(manualErrorMessage, Color.red);
        yield break;
    }
    else if (string.IsNullOrEmpty(confirmPassword))
    {
        manualErrorMessage = "Registration Failed. Confirm password is empty.";
        Debug.Log(manualErrorMessage);
        UpdateRegisterOutputMessage(manualErrorMessage, Color.red);
        yield break;
    }
    else if (password.Length < 6)
    {
        manualErrorMessage = "Registration Failed. Password must be at least 6 characters long.";
        Debug.Log(manualErrorMessage);
        UpdateRegisterOutputMessage(manualErrorMessage, Color.red);
        yield break;
    }
    else if (password != confirmPassword)
    {
        manualErrorMessage = "Registration Failed. Password and confirm password do not match.";
        Debug.Log(manualErrorMessage);
        UpdateRegisterOutputMessage(manualErrorMessage, Color.red);
    }
    else
    {
        Debug.Log("Registering user...");
```

Figure 4.2: Implementation of RegisterUser Function (Part 1).

In the `RegisterUser` function, manual input checks are included before calling the authentication method. These checks ensure that the user has entered valid input values, such as matching passwords and a properly formatted email.



```
        Debug.Log("Registering user...");
        var registerTask = auth.CreateUserWithEmailAndPasswordAsync(email, password);
        yield return new WaitUntil(() => registerTask.IsCompleted);

        if (registerTask.Exception != null)
        {
            Debug.LogError(registerTask.Exception);
            FirebaseException firebaseException = registerTask.Exception.GetBaseException() as FirebaseException;
            AuthError authError = (AuthError)firebaseException.ErrorCode;
            string errorMessage = "Registration Failed. ";
            switch (authError)
            {
                case AuthError.InvalidEmail:
                    errorMessage += "Invalid email.";
                    break;
                case AuthError.EmailAlreadyInUse:
                    errorMessage += "Email already in use.";
                    break;
                case AuthError.MissingEmail:
                    errorMessage += "Missing email.";
                    break;
                case AuthError.MissingPassword:
                    errorMessage += "Missing password.";
                    break;
                default:
                    errorMessage += "Unknown error.";
                    break;
            }

            Debug.Log(errorMessage);
            UpdateRegisterOutputMessage(errorMessage, Color.red);
            yield break;
        }
        else
        {
            user = registerTask.Result.User;
            Debug.LogFormat("Firebase user registered successfully: {0} ({1})", user.DisplayName, user.UserId);
            UpdateRegisterOutputMessage("Registration successful. Redirect to login.", Color.green);
            _mainMenuNavigation.ShowBlocker(true);
            yield return new WaitForSeconds(2f);
            ResetAllField();
            _mainMenuNavigation.ShowLoginMenu();
            _mainMenuNavigation.ShowBlocker(false);
        }
    }
```

Figure 4.3: Implementation of RegisterUser Function (Part 2).

For registration, the script uses Firebase Authentication's method to create a new user account with the `CreateUserWithEmailAndPasswordAsync` method. Similar to the `LoginUser` function, if the registration is successful, the user is redirected to the login menu. If it fails, an appropriate error message is displayed to inform the user.

In addition to login and registration, the game also includes password reset and logout functionalities to enhance account management and user experience. The figure 4.4 and 4.5 below show the `PasswordResetEmail` and `Logout` functions.

```
1 reference
private IEnumerator PasswordResetEmail(string email)
{
    var resetTask = auth.SendPasswordResetEmailAsync(email);
    yield return new WaitUntil(() => resetTask.IsCompleted);
    if (resetTask.Exception != null)
    {
        Debug.LogError(resetTask.Exception);
        FirebaseException firebaseException = resetTask.Exception.GetBaseException() as FirebaseException;
        AuthError authError = (AuthError)firebaseException.ErrorCode;
        string errorMessage = "Failed to send email. ";
        switch (authError)
        {
            case AuthError.InvalidEmail:
                errorMessage += "Invalid email.";
                break;
            case AuthError.MissingEmail:
                errorMessage += "Missing email.";
                break;
            default:
                errorMessage += "Unknown error.";
                break;
        }

        Debug.Log(errorMessage);
        UpdateLoginOutputMessage(errorMessage, Color.red);
        yield break;
    }
    else
    {
        Debug.Log("Password reset email sent successfully.");
        UpdatePasswordResetOutputMessage("Password reset email sent successfully. Redirect to login.", Color.green);
        _mainMenuNavigation.ShowBlocker(true);
        yield return new WaitForSeconds(2f);
        ResetAllField();
        _mainMenuNavigation.ShowProfileMenu();
        _mainMenuNavigation.ShowBlocker(false);
    }
}
```

Figure 4.4: Implementation of PasswordResetEmail Function.

The password reset feature allows users to reset their password via email. This is implemented using Firebase Authentication's `SendPasswordResetEmailAsync` method. When a user provides a valid registered email, a reset link is sent to their inbox.

```
0 references
public void Logout()
{
    if (auth != null && user != null)
    {
        auth.SignOut();
        Debug.Log("User logged out.");
        _mainMenuNavigation.ShowProfileMenu();
    }
}
```

Figure 4.5: Implementation of Logout Function.

The logout function allows users to securely sign out of their account. It uses Firebase Authentication's `SignOut` method to terminate the current session. Upon logout, the user is returned to the profile menu.

4.2.2.2 Card System

The card system is a core component of the game and involves multiple interconnected scripts and components to handle card behaviour, interaction, and game logic.

1. Card Data (ScriptableObject)

The game uses Unity's `ScriptableObject` to store and manage card data such as name, cost, preparedness value, and disaster type. Each card is represented as a `ScriptableObject` asset and is used to initialize the `CardUI` during gameplay. Figures 4.6 and 4.7 below show the variables used to define each card's data.

```
1   using UnityEngine;
2
    Unity Script | 6 references
3   public abstract class BaseCard : ScriptableObject
4   {
        4 references
5       [field: SerializeField] public string CardName { get; private set; }
        2 references
6       [field: SerializeField] public Sprite CardImage { get; private set; }
        2 references
7       [field: SerializeField] public string CardDescription { get; private set; }
8   }
9
```

Figure 4.6: Variables of BaseCard Class.

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

[CreateAssetMenu(menuName = "Player Card Data")]
Unity Script | 7 references
public class PlayerCardData : BaseCard
{
    0 references
    [field: SerializeField] public string CardDescriptionTitle { get; private set; }
    3 references
    [field: SerializeField] public int CardPlayCost { get; private set; }
    3 references
    [field: SerializeField] public CardType CardType { get; private set; }
    [SerializeReference][SR] public List<CardEffect> CardEffects;
}

```

Figure 4.7: Variables of PlayerCardData Class.

There are multiple `ScriptableObject` assets, each representing different card data. These individual card assets are grouped together to form a complete deck used in the game.

2. Card UI

`CardUI` is responsible for setting up the card's visual elements, including the card image, name, cost, and preparedness value. It initializes the UI based on the data provided by the associated `ScriptableObject` asset. Figures 4.8, 4.9, and 4.10 show the `PlayerCard` and `CardUI` scripts used to implement this functionality.

```
using UnityEngine;

[RequireComponent(typeof(PlayerCardUI))]
[RequireComponent(typeof(CardMovement))]
Unity Script (2 asset references) | 41 references
public class PlayerCard : MonoBehaviour
{
    17 references
    [field: SerializeField] public PlayerCardData PlayerCardData { get; private set; }

    5 references
    public void Setup(PlayerCardData playerCardData)
    {
        PlayerCardData = playerCardData;
        GetComponent<PlayerCardUI>().SetCardUI();
    }
}

```

Figure 4.8: PlayerCard Class for Card.

`PlayerCard` is the main class that represents an in-game card and is used by other scripts to access card-related data and behaviour.

```
1 reference
public void SetCardUI()
{
    if (_playerCard != null && _playerCard.PlayerCardData != null)
    {
        SetCardImage();
        SetCardTypeIcon();
        SetCardTextAreaImages();
        SetCardTexts();
        Debug.Log("Player card UI set");
    }
}
```

Figure 4.9: Implementation of CardUI Script (Part 1).

```
1 reference
private void SetCardImage()
{
    _cardImage.sprite = _playerCard.PlayerCardData.CardImage;
}

1 reference
private void SetCardTextAreaImages()
{
    _cardNameBackground.sprite = _cardNameBackgroundSprite;
    _cardDescriptionBackground.sprite = _cardDescriptionBackgroundSprite;
    _cardPlayCostBackground.sprite = _cardPlayCostBackgroundSprite;
    _cardDefenseBackground.sprite = _cardDefenseBackgroundSprite;
}

1 reference
private void SetCardTypeIcon()
{
    _cardTypeBackground.sprite = _cardTypeBackgroundSprite;
    switch (_playerCard.PlayerCardData.CardType)
    {
        case CardType.All:
            _cardTypeIcon.sprite = _allTypeIcon;
            break;
        case CardType.Earthquake:
            _cardTypeIcon.sprite = _earthquakeTypeIcon;
            break;
        case CardType.Flood:
            _cardTypeIcon.sprite = _floodTypeIcon;
            break;
        case CardType.Landslide:
            _cardTypeIcon.sprite = _landslideTypeIcon;
            break;
        case CardType.Wildfire:
            _cardTypeIcon.sprite = _wildfireTypeIcon;
            break;
    }
}

1 reference
private void SetCardTexts()
{
    _cardName.text = _playerCard.PlayerCardData.CardName;
    _cardDescription.text = _playerCard.PlayerCardData.CardDescription;
    _cardPlayCost.text = _playerCard.PlayerCardData.CardPlayCost.ToString();
    foreach (Defense effect in _playerCard.PlayerCardData.CardEffects)
    {
        _cardDefense.text = effect.DefenseAmount.ToString();
    }
}
```

Figure 4.10: Implementation of CardUI Script (Part 2).

The `CardUI` script sets the text, image, and sprite elements of the card by retrieving data from the associated card data and assigning it to the corresponding UI components on the game screen. This includes updating the card's name, cost, icon, preparedness value, and image to match the values defined in the `ScriptableObject`.

3. Card Movement

The `CardMovement` script is responsible for enabling drag-and-drop interactions for the cards during gameplay. It implements Unity's `IBeginDragHandler`, `IDragHandler`, and `IEndDragHandler` interfaces to detect and manage user input when a card is picked up, dragged, and released. This allows player to interact with cards intuitively, such as moving them to different play areas like the hand, deck, or discard zones. `IPointerEnterHandler` and `IPointerExitHandler` are used to detect when the pointer hovers over or exits a card.

Figure 4.11: CardMovement that Implements Interfaces.

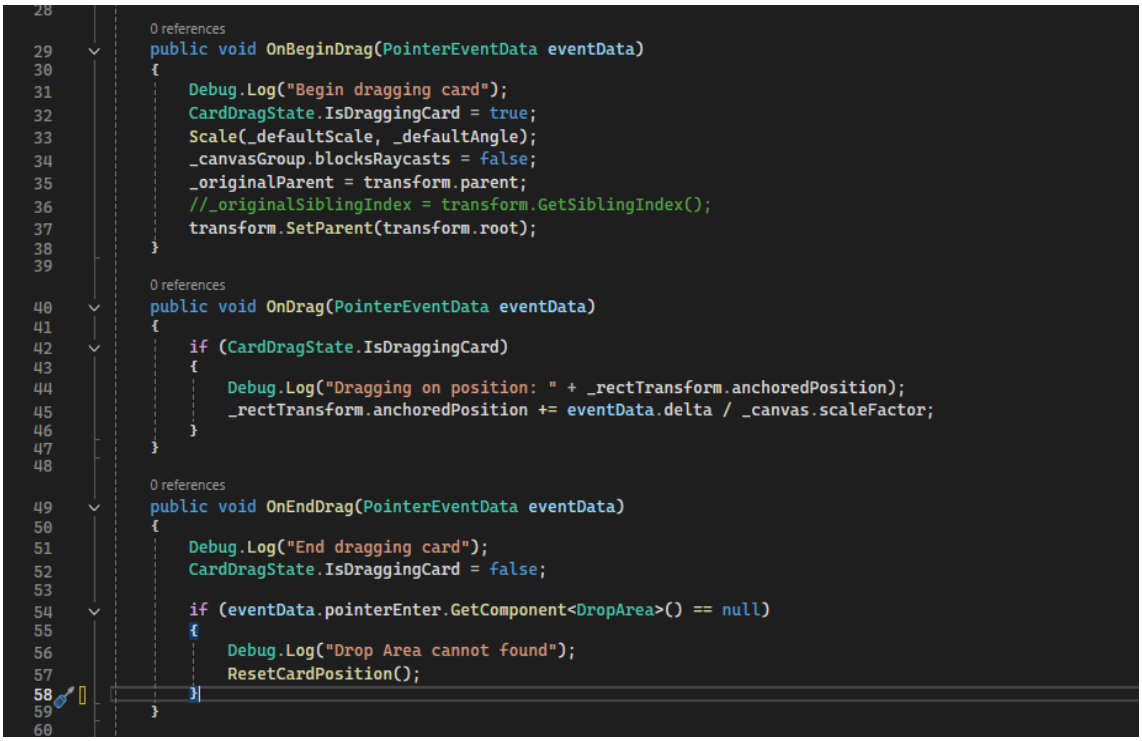


Figure 4.12: Implementation of CardMovement Script (Part 2).

The figure 4.12 above shows the card movement implementation using the `OnBeginDrag`, `OnDrag`, and `OnEndDrag` methods from Unity's drag handler interfaces. The core movement logic is in line 45, where the card's position is updated based on the pointer

movement. Before dragging begins, the script checks and stores the card's original position and order within the hand. During dragging, the card follows the pointer. Upon releasing the card, the script checks whether it has been dropped into a valid drop area. If not, it resets the card to its original position using the stored values from `OnBeginDrag`.

```
0 references
public void OnPointerEnter(PointerEventData eventData)
{
    if (CardDragState.IsDraggingCard)
    {
        return;
    }

    _originalSiblingIndex = transform.GetSiblingIndex();
    transform.SetAsLastSibling();
    _originalCardAngle = transform.eulerAngles;
    _originalPosition = _rectTransform.anchoredPosition;
    Scale(new Vector3(1.1f, 1.1f, 1.1f), _defaultAngle);
    _rectTransform.anchoredPosition = new Vector2(_originalPosition.x, 51f);
    Debug.Log("PlayerCard pointer enter");
}

0 references
public void OnPointerExit(PointerEventData eventData)
{
    if (CardDragState.IsDraggingCard)
    {
        return;
    }

    transform.SetSiblingIndex(_originalSiblingIndex);
    Scale(_defaultScale, _originalCardAngle);
    _rectTransform.anchoredPosition = _originalPosition;
    Debug.Log("PlayerCard pointer exit");
}
```

Figure 4.13: Implementation of CardMovement Script (Part 2).

Figure 4.13 above shows the card enlarge feature. When the pointer enters the card area, the card is scaled up to highlight it. When the pointer exits, the card returns to its original size and rotation.

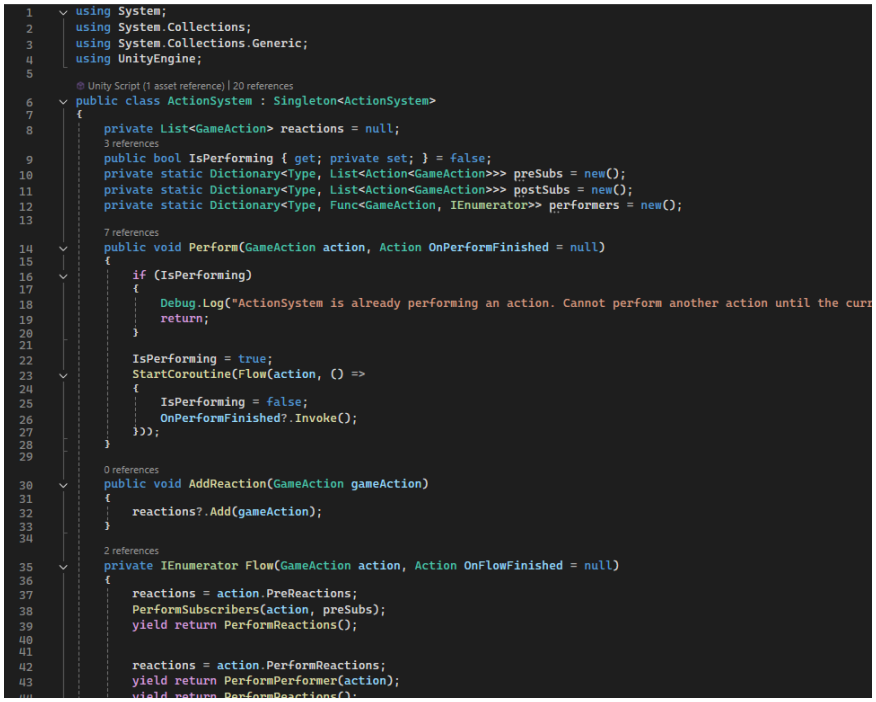
4.2.2.3 Action System

The game's `ActionSystem` manages the execution of game actions through a structured flow consisting of pre-reactions, a main performer, and post-reactions. It ensures that only one action runs at a time and supports asynchronous execution using coroutines. This system allows custom performers to be defined for specific actions and enables reactions to be

subscribed to run either before or after an action. For example, in a draw card action, a healing reaction can be added as a pre-reaction. When the draw action is triggered, it will first heal the player's health and then proceed to draw the card.

This system includes following functions:

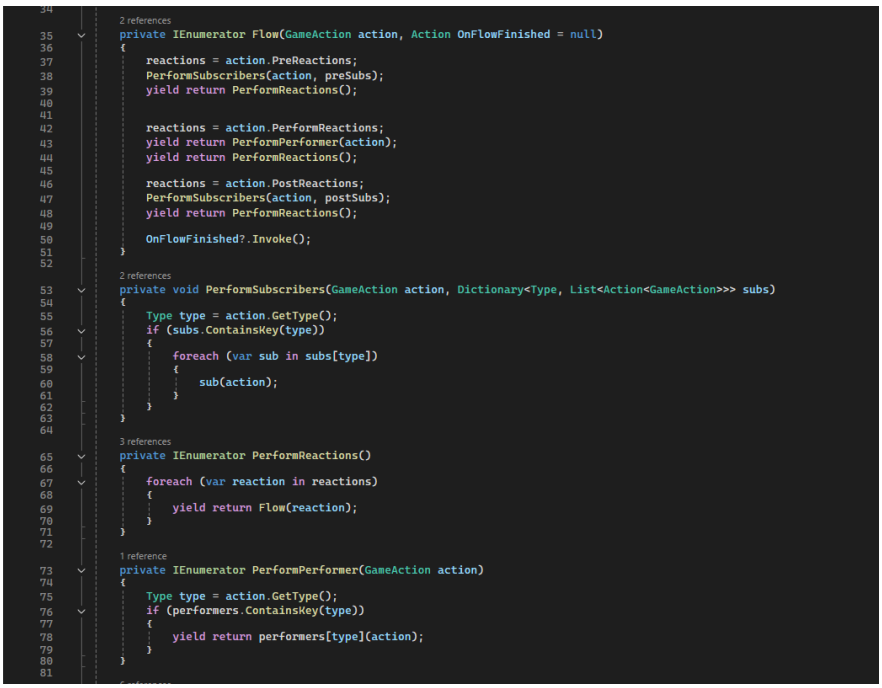
- `Perform` – The main entry point of the system; it allows other scripts to execute a game action.
- `AddReaction` – Adds a reaction to be processed as part of a game action.
- Flow – Handles the full execution cycle of a game action, including pre-reactions, the main performer, and post-reactions.
- `PerformSubscribers` – Filters and gets subscribed reactions for a given game action based on its type.
- `PerformReactions` – Executes all queued reactions related to the current game action.
- `PerformPerformer` – Executes the main logic which is performer associated with the game action.
- `AttachPerformer` – Registers a performer for a specific type of game action.
- `DetachPerformer` – Removes a registered performer for a specific game action type.
- `SubscribeReaction` – Subscribes a function to be triggered before or after a specific game action, based on reaction timing, pre or post.
- `UnsubscribeReaction` – Removes a subscribed reaction from the system.



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

Unity Script (1 asset reference) | 20 references
public class ActionSystem : Singleton<ActionSystem>
{
    private List<GameAction> reactions = null;
    3 references
    public bool IsPerforming { get; private set; } = false;
    private static Dictionary<Type, List<Action<GameAction>>> preSubs = new();
    private static Dictionary<Type, List<Action<GameAction>>> postSubs = new();
    private static Dictionary<Type, Func<GameAction, IEnumerator>> performers = new();

    7 references
    public void Perform(GameAction action, Action OnPerformFinished = null)
    {
        if (IsPerforming)
        {
            Debug.Log("ActionSystem is already performing an action. Cannot perform another action until the curr
            return;
        }

        IsPerforming = true;
        StartCoroutine(Flow(action, () =>
        {
            IsPerforming = false;
            OnPerformFinished?.Invoke();
        }));
    }

    0 references
    public void AddReaction(GameAction gameAction)
    {
        reactions?.Add(gameAction);
    }

    2 references
    private IEnumerator Flow(GameAction action, Action OnFlowFinished = null)
    {
        reactions = action.PreReactions;
        PerformSubscribers(action, preSubs);
        yield return PerformReactions();


        reactions = action.PerformReactions;
        yield return PerformPerformer(action);
```

Figure 4.14: Implementation of ActionSystem Script (Part 1).

In the Perform function, the system first checks whether an action is currently being performed. If not, it passes the game action to the Flow function, which handles the full execution sequence of that game action.



```
    2 references
    private IEnumerator Flow(GameAction action, Action OnFlowFinished = null)
    {
        reactions = action.PreReactions;
        PerformSubscribers(action, preSubs);
        yield return PerformReactions();


        reactions = action.PerformReactions;
        yield return PerformPerformer(action);
        yield return PerformReactions();

        reactions = action.PostReactions;
        PerformSubscribers(action, postSubs);
        yield return PerformReactions();

        OnFlowFinished?.Invoke();
    }

    2 references
    private void PerformSubscribers(GameAction action, Dictionary<Type, List<Action<GameAction>>> subs)
    {
        Type type = action.GetType();
        if (subs.ContainsKey(type))
        {
            foreach (var sub in subs[type])
            {
                sub(action);
            }
        }
    }

    3 references
    private IEnumerator PerformReactions()
    {
        foreach (var reaction in reactions)
        {
            yield return Flow(reaction);
        }
    }

    1 reference
    private IEnumerator PerformPerformer(GameAction action)
    {
        Type type = action.GetType();
        if (performers.ContainsKey(type))
        {
            yield return performers[type](action);
        }
    }
```

Figure 4.15: Implementation of ActionSystem Script (Part 2).

The Flow function executes the pre-reactions for the game action, then the game action itself through the performer, and finally the post-reactions. This process includes filtering and finding the corresponding performer and reactions.

```
6 references
public static void AttachPerformer<T>(Func<T, IEnumerator> performer) where T : GameAction
{
    Type type = typeof(T);
    IEnumerator wrappedPerformer(GameAction action)
    {
        return performer((T)action);
    }

    if (performers.ContainsKey(type))
    {
        performers[type] = wrappedPerformer;
    }
    else
    {
        performers.Add(type, wrappedPerformer);
    }
}

6 references
public static void DetachPerformer<T>() where T : GameAction
{
    Type type = typeof(T);
    if (performers.ContainsKey(type))
    {
        performers.Remove(type);
    }
}

0 references
```

Figure 4.16: Implementation of ActionSystem Script (Part 3).

Since there are many game actions, `GameAction` serves as the base class. The `ActionSystem` works with the `GameAction` type, so in the `AttachPerformer` function, performer is wrapped as a `GameAction` delegate before being added to the list. The `DetachPerformer` function removes the corresponding game action performer from the system.

```
0 references
public static void SubscribeReaction<T>(Action<T> reaction, ReactionTiming timing) where T : GameAction
{
    Dictionary<Type, List<Action<GameAction>>> subs;
    if (timing == ReactionTiming.PRE)
    {
        subs = preSubs;
    }
    else
    {
        subs = postSubs;
    }

    void wrappedReaction(GameAction action)
    {
        reaction((T)action);
    }

    if (subs.ContainsKey(typeof(T)))
    {
        subs[typeof(T)].Add(wrappedReaction);
    }
    else
    {
        subs.Add(typeof(T), new());
        subs[typeof(T)].Add(wrappedReaction);
    }
}

0 references
public static void UnsubscribeReaction<T>(Action<T> reaction, ReactionTiming timing) where T : GameAction
{
    Dictionary<Type, List<Action<GameAction>>> subs;
    if (timing == ReactionTiming.PRE)
    {
        subs = preSubs;
    }
    else
    {
        subs = postSubs;
    }

    if (subs.ContainsKey(typeof(T)))
    {
        void wrappedReaction(GameAction action)
        {
            reaction((T)action);
        }

        subs[typeof(T)].Remove(wrappedReaction);
    }
}
```

Figure 4.17: Implementation of ActionSystem Script (Part 4).

Similar to `AttachPerformer`, the `SubscribeReaction` function adds reactions to different lists based on their timing (pre or post) without needing to wrap them. The `UnsubscribeReaction` function removes the specified reaction from the corresponding list. One of the key use cases of the `ActionSystem` is the Draw Card game action. This is defined through a `DrawCardGA` class, which specifies the number of cards to draw.

```
sembly-CSharp                                   DrawCardGA
        10 references
1       public class DrawCardGA : GameAction
2       {
3           public int Amount;
4
            3 references
5           public DrawCardGA(int amount)
6           {
7               Amount = amount;
8           }
9       }
10
```

Figure 4.18: DrawCardGA Script.

```
// Performer (GameAction)
ActionSystem.AttachPerformer<DrawCardGA>(DrawCardPerformer);
```

Figure 4.19: Attach Performer Code.

The code in Figure 4.19 above shows how the draw card performer is registered in the `ActionSystem`.

```
1 reference
private IEnumerator DrawCardPerformer(DrawCardGA drawCardGA)
{
    int drawAmount = drawCardGA.Amount;
    yield return StartCoroutine(_turnManager.ActivateBlocker());
    for (int i = 0; i < drawAmount; i++)
    {
        if (_deck.Count == 0)
        {
            Debug.Log("Deck is empty, cannot draw card");
            yield break;
        }

        PlayerCardData playerCardData = _deck[Random.Range(0, _deck.Count)];
        _deck.Remove(playerCardData);
        PlayerCard playerCard = Instantiate(_cardPrefab, _spawnPoint.transform);
        playerCard.Setup(playerCardData);
        Tween tween = playerCard.transform.DOMove(_handArea.transform.position, 0.5f);
        tween.OnComplete(() => playerCard.transform.SetParent(_handArea.transform));
        AudioManager.Instance.PlaySFX(AudioManager.Instance._drawCard);
        yield return tween.WaitForCompletion();
        _handManager.AddCardToHand(playerCard);
    }

    yield return StartCoroutine(_turnManager.DeactivateBlocker());
}
```

Figure 4.20: Implementation of DrawCardPerformer.

Figure 4.20 above shows the implementation of the draw card game logic within the performer. It retrieves card data from the deck, instantiates the card on the screen at the correct position, sets up the UI, plays the draw animation and sound effect, and finally adds the card to the player's hand.

```
1 reference
90  private IEnumerator DrawCard()
91  {
92      int drawAmount = 5 - _handManager.GetHandSize();
93      if (_handManager.GetHandSize() < 5)
94      {
95          DrawCardGA drawCardGA = new(drawAmount);
96          ActionSystem.Instance.Perform(drawCardGA);
97      }
98
99      yield return null;
100 }
101
```

Figure 4.21: Use ActionSystem in Code.

Figure 4.21 above shows how the `DrawCardGA` is used in other scripts. First, a new instance of `DrawCardGA` is created with the desired number of cards to draw. Then, `ActionSystem.Instance.Perform` is called with that instance. The `ActionSystem` will handle the execution of the draw card action using its registered performer, which includes the core game logic, animations, and any related effects.

4.2.2.4 Game Manager

The Game Manager is the primary controller responsible for managing the overall flow of the game. It keeps track of the current game state and determines the appropriate actions based on that state. The Game Manager also coordinates with other feature-specific managers during gameplay. Figure 4.22 below shows the various managers controlled by the Game Manager.

```
3
      Unity Script (1 asset reference) | 4 references
4     public class GameManager : Singleton<GameManager>
5     {
          3 references
6         public GameState CurrentState { get; private set; }
          11 references
7         public GameLevel CurrentLevel { get; private set; }
8
9         [SerializeField] private TurnManager _turnManager;
10        [SerializeField] private CostManager _costManager;
11        [SerializeField] private HealthManager _healthManager;
12        [SerializeField] private DefenseManager _defenseManager;
13        [SerializeField] private HandManager _handManager;
14        [SerializeField] private DiscardManager _discardManager;
15        [SerializeField] private EnemyManager _enemyManager;
16        [SerializeField] private ScoreManager _scoreManager;
17
          Unity Message | 0 references
```

Figure 4.22: Managers that Control by Game Manager.

```
6 references
public void SetGameState(GameState gameState)
{
    CurrentState = gameState;
    switch (CurrentState)
    {
        case GameState.StartPhase:
            StartCoroutine(StartPhase());
            break;
        case GameState.PlayerPhase:
            StartCoroutine(PlayerPhase());
            break;
        case GameState.EnemyPhase:
            StartCoroutine(EnemyPhase());
            break;
        case GameState.EndPhase:
            StartCoroutine(EndPhase());
            break;
    }
}
```

Figure 4.23: Game State.

The game consists of four states: Start, Player, Enemy, and End. Figure 4.24 below shows the implementation of the `StartPhase` method. In this phase, the game resets all resources and sets up the enemy, which represents a disaster. After the setup, the game proceeds by calling `StartTurn`, which initiates a new turn and transitions the game state to the Player phase.

```
#region Start Phase
1 reference
private IEnumerator StartPhase()
{
    yield return StartCoroutine(ResetResources());
    yield return StartCoroutine(SetEnemy());
    yield return StartCoroutine(_turnManager.StartTurn());
    SetGameState(GameState.PlayerPhase);
}

1 reference
private IEnumerator ResetResources()
{
    _costManager.ResetCost();
    _defenseManager.ResetDefense();
    _discardManager.ResetDiscardPerTurn();
    yield return null;
}

1 reference
private IEnumerator SetEnemy()
{
    _enemyManager.SetupEnemyCard();
    yield return null;
}
```

Figure 4.24: Start Phase.

In the `PlayerPhase` method, the game automatically draws cards until the player's hand reaches the maximum limit of five cards. The game then waits for the player's actions until the player ends their turn. Once the player ends the turn, the state transitions to the `EnemyPhase`. In this phase, the game evaluates the disaster's damage and uses the `ActionSystem` to perform the corresponding damage-dealing actions. After the enemy phase, the cycle repeats until either the maximum number of turns is reached or the player's health is depleted, at which point the game transitions to the End phase.

```
92
93      #region Player Phase
        1 reference
94      private IEnumerator PlayerPhase()
95      {
96          yield return StartCoroutine(DrawCard());
97      }
98
99
        1 reference
100     private IEnumerator DrawCard()
101     {
102         int drawAmount = 5 - _handManager.GetHandSize();
103         if (_handManager.GetHandSize() < 5)
104         {
105             DrawCardGA drawCardGA = new(drawAmount);
106             ActionSystem.Instance.Perform(drawCardGA);
107         }
108
109         yield return null;
110     }
111
112     #endregion
113
114     #region Enemy Phase
        1 reference
115     private IEnumerator EnemyPhase()
116     {
117         EnemyCard enemyCard = _enemyManager.CurrentEnemyCard;
118         DealDamageGA dealDamageGA = new(enemyCard._enemyCardData.CardDamage);
119         ActionSystem.Instance.Perform(dealDamageGA);
120         yield return new WaitForSeconds(1f);
121
122         if (_turnManager.GetCurrentTurn() >= _turnManager.GetMaxTurns() || _healthManager.GetCurrentHealth() == 0)
123         {
124             SetGameState(GameState.EndPhase);
125         }
126         else
127         {
128             SetGameState(GameState.StartPhase);
129         }
130     }
131
```

Figure 4.25: Player Phase and Enemy Phase.

```
133
        1 reference
134     private IEnumerator EndPhase()
135     {
136         _scoreManager.GetResult();
137         Debug.Log("Ending Phase");
138         yield return null;
139     }
140
```

Figure 4.26: End Phase.

In the End phase, the game triggers the Score Manager to calculate the result and display the score menu to the player. The overall game flow begins with the Start phase, followed by the Player phase and then the Enemy phase. After the Enemy phase, the game checks for end conditions. If the conditions are not met, the cycle restarts from the Start phase. If the conditions are met, the game transitions to the End phase.

4.2.2.5 Level Manager

The Level Manager is responsible for configuring the data used in each level. When player select Level 1 with easy difficulty, it sets up the appropriate player and enemy cards specific to that level. As shown in Figure 4.27, each level contains different data configurations, including the number of enemy cards, the number of player cards, the maximum number of turns, and the assigned game level.

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

Unity Script (1 asset reference) | 6 references
public class LevelManager : PersistentSingleton<LevelManager>
{
    [SerializeField] private CardCollection _enemyCardCollection;
    [SerializeField] private CardCollection _playerCardCollection;

    private List<EnemyCardData> _currentEnemyCards;
    private List<PlayerCardData> _currentPlayerCards;
    private int _currentMaxTurns;
    private GameLevel _currentGameLevel;

    1 reference
    public void SetLevelData(GameLevel gameLevel)
    {
        if (gameLevel == GameLevel.Level1)
        {
            _currentEnemyCards = SelectEnemyCardCollection(2);
            _currentPlayerCards = SelectPlayerCardCollection(_currentEnemyCards);
            _currentMaxTurns = 3;
            _currentGameLevel = gameLevel;
        }
        else if (gameLevel == GameLevel.Level2)
        {
            _currentEnemyCards = SelectEnemyCardCollection(3);
            _currentPlayerCards = SelectPlayerCardCollection(_currentEnemyCards);
            _currentMaxTurns = 4;
            _currentGameLevel = gameLevel;
        }
        else if (gameLevel == GameLevel.Level3)
        {
            _currentEnemyCards = SelectEnemyCardCollection(4);
            _currentPlayerCards = SelectPlayerCardCollection(_currentEnemyCards);
            _currentMaxTurns = 5;
            _currentGameLevel = gameLevel;
        }
        else
        {
            _currentEnemyCards = SelectEnemyCardCollection(4);
            _currentPlayerCards = SelectPlayerCardCollection(_currentEnemyCards);
            _currentMaxTurns = 5;
            _currentGameLevel = GameLevel.Level3;
        }
    }
```

Figure 4.27: Set Level Data in Level Manager.

```
4 references
private List<EnemyCardData> SelectEnemyCardCollection(int enemyCardNumber)
{
    List<EnemyCardData> enemyCards = new(_enemyCardCollection.EnemyCardsInCollection);
    List<EnemyCardData> selectedEnemyCards = new List<EnemyCardData>();

    for (int i = 0; i < enemyCardNumber; i++)
    {
        int randomIndex = Random.Range(0, enemyCards.Count);
        selectedEnemyCards.Add(enemyCards[randomIndex]);
        enemyCards.RemoveAt(randomIndex);
    }

    return selectedEnemyCards;
}

4 references
private List<PlayerCardData> SelectPlayerCardCollection(List<EnemyCardData> selectedEnemyCards)
{
    List<PlayerCardData> playerCards = new(_playerCardCollection.PlayerCardsInCollection);
    List<PlayerCardData> selectedPlayerCards = new List<PlayerCardData>();
    for (int i = 0; i < playerCards.Count; i++)
    {
        if (playerCards[i].CardType == CardType.All)
        {
            selectedPlayerCards.Add(playerCards[i]);
        }
        else
        {
            for (int j = 0; j < selectedEnemyCards.Count; j++)
            {
                if (playerCards[i].CardType == selectedEnemyCards[j].CardType)
                {
                    selectedPlayerCards.Add(playerCards[i]);
                }
            }
        }
    }

    return selectedPlayerCards;
}
}
```

Figure 4.28: Functions used to Filter Cards.

Figure 4.28 above shows the functions used to select cards for different levels. These functions retrieve cards from a predefined collection that contains all disaster and player cards. The system then filters and selects specific cards from this collection according to the requirements of each level.

4.2.2.6 Turn System

The Turn System manages the flow of gameplay rounds and player turns. It tracks the current round and turns count, acting as a Turn Manager that controls the start and end of each turn. This system is used by the main Game Manager to coordinate and control the overall game flow.

```
1 reference
public IEnumerator StartTurn()
{
    _currentTurn++;
    yield return StartCoroutine(ShowTurnLabelRoutine());
    UpdateTurnText();
    Debug.Log("Player Turn: " + _currentTurn);
}

0 references
public void EndTurn()
{
    Debug.Log("Turn " + _currentTurn + " ended");
    GameManager.Instance.SetGameState(GameState.EnemyPhase);
}
```

Figure 4.29: Implementation of Turn Manager Script (Part 1).

```
68      private IEnumerator ShowTurnLabelRoutine()
69      {
70          StartCoroutine(ActivateBlocker());
71
72          if (_isGameStarting == true)
73          {
74              _turnLabelText.text = "Game Start!";
75              _turnLabel.gameObject.SetActive(true);
76              _isGameStarting = false;
77              yield return new WaitForSeconds(1.5f);
78              _turnLabel.gameObject.SetActive(false);
79              yield return new WaitForSeconds(0.5f);
80          }
81
82          _turnLabelText.text = "Turn " + _currentTurn;
83          _turnLabel.gameObject.SetActive(true);
84          yield return new WaitForSeconds(1.5f);
85          _turnLabel.gameObject.SetActive(false);
86          StartCoroutine(DeactivateBlocker());
87      }
88
89      1 reference
90      private void UpdateTurnText()
91      {
92          if (_turnNumberText != null)
93          {
94              _turnNumberText.text = "Turn: " + _currentTurn + " / " + _maxTurns;
95              Debug.Log("Turn Text Updated: " + _currentTurn + " / " + _maxTurns);
96          }
97          else
98          {
99              Debug.LogError("Turn Counter Text is not assigned in the inspector.");
00          }
01      }
```

Figure 4.30: Implementation of Turn Manager Script (Part 2).

Figures 4.22 and 4.23 show the `StartTurn` function, which triggers the display of the start turn label and updates the turn text on the UI. The `EndTurn` function changes the game state to the enemy's turn, prompting the game to execute the enemy phase game logic.

4.2.2.7 Health and Defense System

The Health and Defense System manages the player's health points (HP) and defense values during gameplay. It tracks damage taken, applies defense modifiers to reduce incoming

damage, and updates the health UI accordingly. The system ensures that damage calculation and UI updates are handled consistently to provide clear player feedback throughout each turn.

This system consists of Health Manager and Defense Manager. When the player plays card, it will trigger Defense Manager function, `GainDefense` and add the current defense value as well as update the defense UI. The figure 4.24 below shows the implementation of Defense Manager.

```
Unity Script (1 asset reference) | 3 references
public class DefenseManager : MonoBehaviour
{
    [SerializeField] private int _currentDefense = 0;
    [SerializeField] private TextMeshProUGUI _defenseNumberText;

    1 reference
    public void ResetDefense()
    {
        _currentDefense = 0;
        UpdateDefenseUI();
    }

    1 reference
    public void GainDefense(int amount)
    {
        if (amount < 0)
        {
            Debug.Log("Defense amount cannot be negative. Setting to 0.");
            amount = 0;
        }

        Debug.Log($"Gaining defense in defense mangerrrrrrrrrrrrrrrrrrrrrr: {amount}");
        _currentDefense += amount;
        UpdateDefenseUI();
    }

    1 reference
    public int GetCurrentDefense()
    {
        return _currentDefense;
    }

    2 references
    private void UpdateDefenseUI()
    {
        if (_defenseNumberText != null)
        {
            _defenseNumberText.text = $"{_currentDefense}";
            Debug.Log($"Preparedness updated: {_currentDefense}");
        }
        else
        {
            Debug.LogError("Defense text UI element is not assigned.");
        }
    }
}
```

Figure 4.31: Implementation of Defense Manager Script.

After that, the deal damage performer calculates the damage dealt and passes the final amount to the Health Manager for further execution.

```
1 reference
private IEnumerator DealDamagePerformer(DealDamageGA dealDamageGA)
{
    int damageAmount = dealDamageGA.Amount;

    if (_healthManager == null)
    {
        Debug.LogWarning("Target does not have a HealthManager component.");
        yield break;
    }

    Debug.Log("Damage amount here: " + damageAmount);
    int finalDamage = Mathf.Max(0, damageAmount - _defenseManager.GetCurrentDefense());
    Debug.Log("Final damage after applying here: " + finalDamage);
    _healthManager.TakeDamage(finalDamage);
    Debug.Log("Performing DealDamageGA. Here is Performer");
    yield return new WaitForSeconds(0.5f);
}
}
```

Figure 4.32: Implementation of DealDamagePerformer.

```
1 reference
public void TakeDamage(int amount)
{
    if (amount < 0)
    {
        amount = 0;
        Debug.Log("Damage amount cannot be negative. Setting to 0.");
    }

    _currentHealth -= amount;
    if (_currentHealth < 0)
    {
        _currentHealth = 0;
        Debug.Log("Health cannot be negative. Setting to 0.");
        GameManager.Instance.SetGameState(GameState.EndPhase);
    }

    UpdateHealthUI();
}
```

Figure 4.33: Implementation of Health Manager Script (Part 1).

```
2 references
private void UpdateHealthUI()
{
    if (_healthText != null)
    {
        _healthText.text = $"{_currentHealth}";
        Debug.Log($"Health updated: {_currentHealth} / {_maxHealth}");
    }
    else
    {
        Debug.LogError("Health text UI element is not assigned.");
    }
}
}
```

Figure 4.34: Implementation of Health Manager Script (Part 2).

The figure 4.26 and 4.27 above show how the Health Manager manage the player health and update the UI when receive damage amount from deal damage performer.

4.2.2.8 Cost System

The Cost System is managed by a Cost Manager, which controls the player's available cost when playing cards. Each time a card is played, the system reduces the current cost accordingly and updates the UI to reflect the change. At the start of a new turn, the cost is reset to the maximum value. The figure 4.28 below shows the implementation of Cost Manager.

```
Unity Script (1 asset reference) | 3 references
public class CostManager : MonoBehaviour
{
    [SerializeField] private int _currentCost = 0;
    [SerializeField] private int _maxCost = 3;
    [SerializeField] private TextMeshProUGUI _costNumberText;

    1 reference
    public void ResetCost()
    {
        _currentCost = _maxCost;
        UpdateCostUI();
    }

    1 reference
    public void SpendCost(int amount)
    {
        if (amount < 0)
        {
            Debug.Log("Cost amount cannot be negative. Setting to 0.");
            amount = 0;
        }

        if (_currentCost >= amount)
        {
            _currentCost -= amount;
            UpdateCostUI();
        }
        else
        {
            Debug.Log("Not enough cost available. Current cost: " + _currentCost);
        }
    }

    2 references
    public int GetCurrentCost()
    {
        return _currentCost;
    }

    2 references
    private void UpdateCostUI()
    {
        if (_costNumberText != null)
        {
            _costNumberText.text = $"Cost: {_currentCost} / {_maxCost}";
            Debug.Log($"Cost updated: {_currentCost} / {_maxCost}");
        }
        else
        {
            Debug.LogError("Cost text UI element is not assigned.");
        }
    }
}
No issues found
```

Figure 4.35: Implementation of Cost Manager Script.

These functions are used in the Game Manager to control the game flow. When a new turn starts, the `ResetCost` function is called to restore the player's cost to the maximum value.

When the player plays a card, the `SpendCost` function is used to deduct the corresponding cost.

4.2.2.9 Audio Manager

The game uses music and sound effects to enhance the gameplay experience. Figure 4.36 shows the implementation of the Audio Manager, which is responsible for controlling background music and sound effects throughout the game.

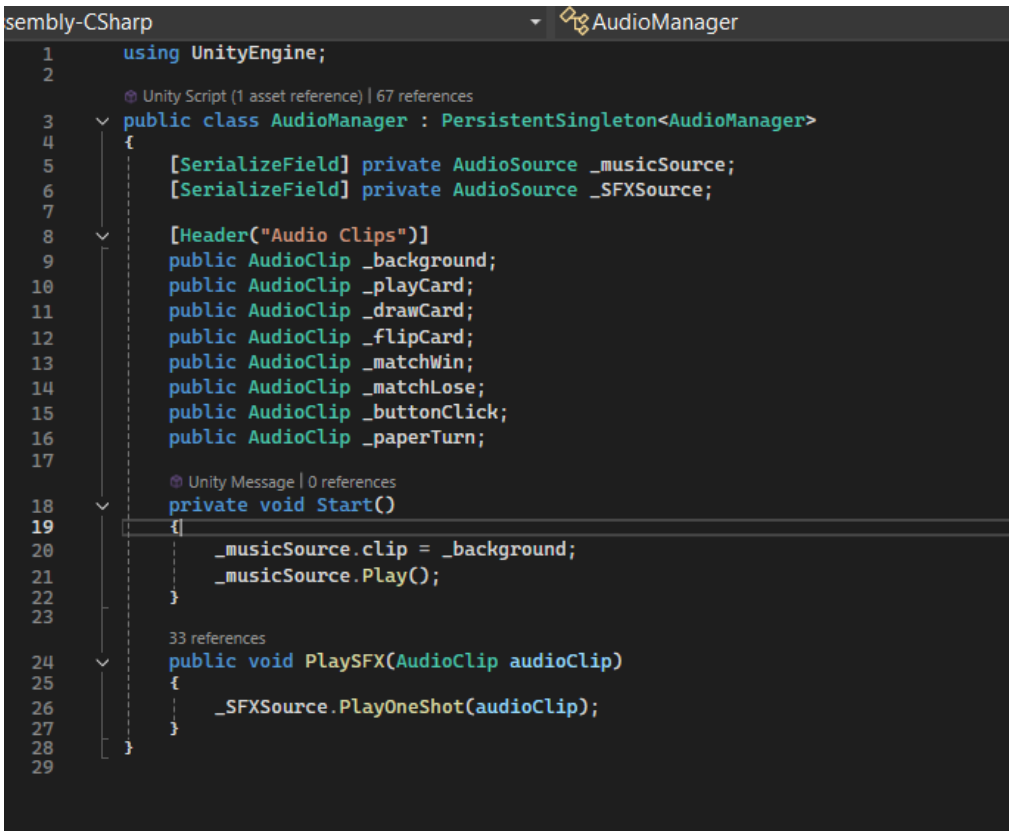


Figure 4.36: Implementation of Audio Manager.

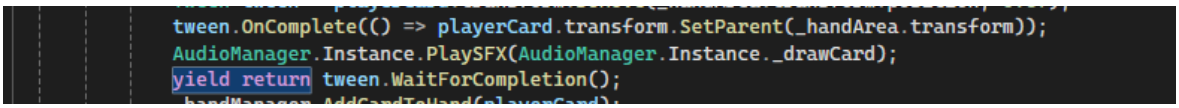


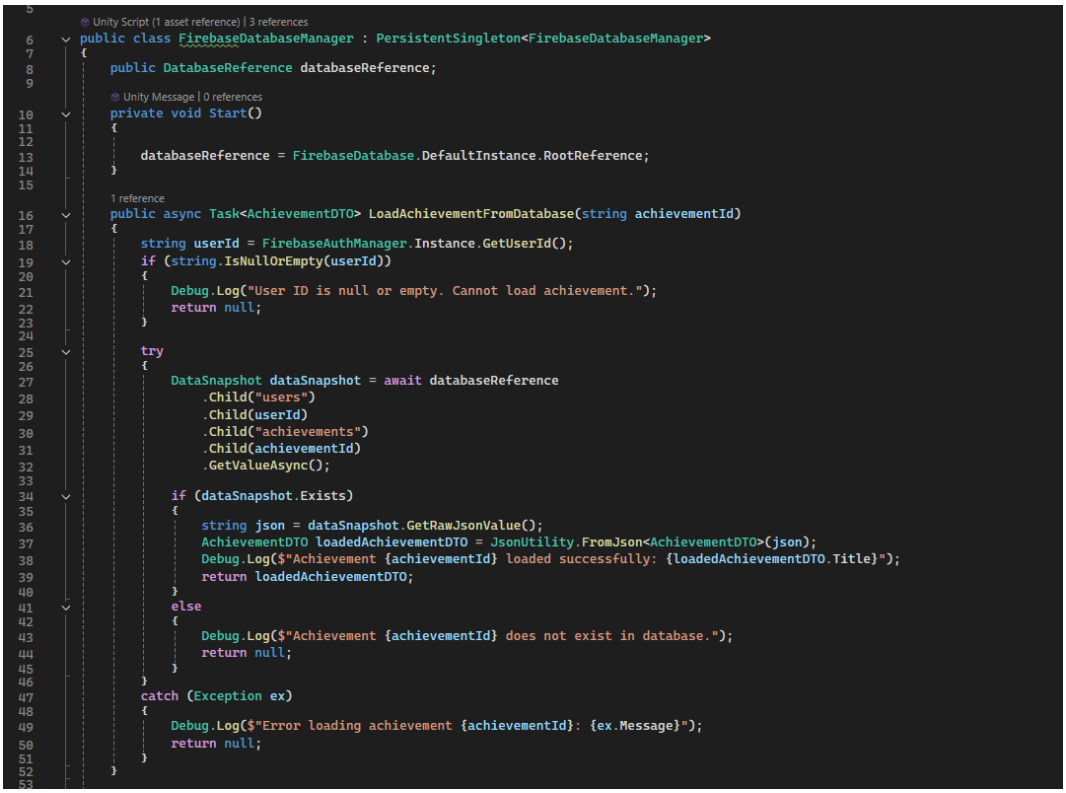
Figure 4.37: The use of Audio Manager.

By implementing the Audio Manager as a singleton, it ensures that only one instance exists throughout the game. This allows the Audio Manager to be accessed and used from any part

of the code. Figure 4.37 shows an example of how the Audio Manager is used to play the sound effect for drawing a card.

4.2.2.10 Achievement System

The game features a simple achievement system that encourages player to explore the game further. It uses Unity's `PlayerPrefs` to store achievement data locally and Firebase Realtime Database to store and sync data online.



```
Unity Script (1 asset reference) | 3 references
public class FirebaseDatabaseManager : PersistentSingleton<FirebaseDatabaseManager>
{
    public DatabaseReference databaseReference;

    Unity Message | 0 references
    private void Start()
    {

        databaseReference = FirebaseDatabase.DefaultInstance.RootReference;
    }

    1 reference
    public async Task<AchievementDTO> LoadAchievementFromDatabase(string achievementId)
    {
        string userId = FirebaseAuthManager.Instance.GetUserId();
        if (string.IsNullOrEmpty(userId))
        {
            Debug.Log("User ID is null or empty. Cannot load achievement.");
            return null;
        }

        try
        {
            DataSnapshot dataSnapshot = await databaseReference
                .Child("users")
                .Child(userId)
                .Child("achievements")
                .Child(achievementId)
                .GetValueAsync();

            if (dataSnapshot.Exists)
            {
                string json = dataSnapshot.GetRawJsonValue();
                AchievementDTO loadedAchievementDTO = JsonUtility.FromJson<AchievementDTO>(json);
                Debug.Log($"Achievement {achievementId} loaded successfully: {loadedAchievementDTO.Title}");
                return loadedAchievementDTO;
            }
            else
            {
                Debug.Log($"Achievement {achievementId} does not exist in database.");
                return null;
            }
        }
        catch (Exception ex)
        {
            Debug.Log($"Error loading achievement {achievementId}: {ex.Message}");
            return null;
        }
    }
```

Figure 4.38: Implementation of Firebase Realtime Database (Part 1).

```
1 reference
public async Task SyncAchievementToDatabase(AchievementDTO achievementDTO)
{
    string _userId = FirebaseAuthManager.Instance.GetUserId();
    if (string.IsNullOrEmpty(_userId))
    {
        Debug.Log("User ID is null or empty. Cannot save achievement.");
        return;
    }

    try
    {
        string achievementId = achievementDTO.Id;
        string json = JsonUtility.ToJson(achievementDTO);
        await databaseReference
            .Child("users")
            .Child(_userId)
            .Child("achievements")
            .Child(achievementId)
            .SetRawJsonValueAsync(json);
        Debug.Log($"Achievement {achievementId} saved successfully.");
    }
    catch (Exception ex)
    {
        Debug.Log($"Error syncing achievement {achievementDTO.Id} to database: {ex.Message}");
    }
}
}
```

Figure 4.39: Implementation of Firebase Realtime Database (Part 2).

Figures 4.38 and 4.39 show the implementation of functions that utilize Firebase Realtime Database services to load and store achievement data. The data is handled in JSON format and organized in the database using a structured path based on the user ID and achievement ID.

Figure 4.40: Structure of the Data.

Figure 4.40 above shows the structure of the data stored in Firebase Realtime Database. The hierarchy begins with the "users" node, followed by the specific user ID, then the

"achievements" node, and finally the achievement ID. Each achievement entry contains the relevant variables required to track achievement progress.

4.2.2.11 Leaderboard System

The game features a leaderboard system to make it more interactive and competitive for players. The leaderboard uses Firebase Realtime Database to store and retrieve data for display. Figure 4.41 below shows the user interface of the leaderboard.

Figure 4.41: Leaderboard Scene.

The leaderboard displays the scores of all players and highlights the current player's position. Additionally, players can view separate leaderboards for each game level.

```
2 references
public void ListenToLeaderboard(GameLevel gameLevel)
{
    if (currentQuery != null)
    {
        currentQuery.ValueChanged -= OnLeaderboardChanged;
    }

    Query query = databaseReference.Child(gameLevel.ToString()).OrderByChild("score");
    currentQuery = query;
    currentQuery.ValueChanged += OnLeaderboardChanged;
    Debug.Log($"Now listening to {gameLevel} leaderboard in real-time!");
}
```

Figure 4.42: Implementation of Leaderboard (Part 1).

Figure 4.42 shows the function used to check for changes in the database. When a change occurs, it updates the data and executes the `OnLeaderboardChanged` function.

```
2 references
private void OnLeaderboardChanged(object sender, ValueChangedEventArgs args)
{
    if (args.DatabaseError != null)
    {
        Debug.LogError($"Leaderboard load failed: {args.DatabaseError.Message}");
        return;
    }

    DataSnapshot snapshot = args.Snapshot;

    List<LeaderboardEntry> newEntries = new();
    foreach (DataSnapshot childSnapshot in snapshot.Children)
    {
        LeaderboardEntry entry = new LeaderboardEntry
        {
            UserId = childSnapshot.Child("userId").Value.ToString(),
            Username = childSnapshot.Child("username").Value.ToString(),
            Score = int.Parse(childSnapshot.Child("score").Value.ToString()),
        };

        newEntries.Add(entry);
    }

    newEntries.Reverse();
    allEntries.Clear();
    allEntries.AddRange(newEntries);

    leaderboardMenu.UpdateLeaderboardUI(newEntries);
    Debug.Log($"Leaderboard updated! Total entries: {allEntries.Count}");
}
```

Figure 4.43: Implementation of Leaderboard (Part 2).

Figure 4.43 shows the `OnLeaderboardChanged` function. It retrieves the data from the database as a list and passes it to the `UpdateLeaderboardUI` function, which binds and displays the data on the leaderboard.

4.2.2.12 User Interface Design

This section describes the design and functionality of the user interface (UI) elements across the three main scenes of the game, which are Main Menu, Level Selection, and Gameplay. The UI is optimized for 1920x1080 landscape mode on mobile. All UI elements and components are scaled dynamically using Canvas Scaler and anchored appropriately.

Figure 4.44: Main Menu Scene.

The Main Menu scene is the first screen the player interacts with upon launching the game. It provides navigation to other parts of the game. The interface features a simple background with a solid color and faint object design. The game title and logo positioned at the center of the screen.

The Play button directs the player to the level selection scene to begin a new game. The Tutorial button opens the tutorial panel, where players can learn how to play. The Settings button allows players to adjust volume settings, while the Achievement button displays unlocked achievements and player progress milestones. The Profile button offers login and signup functionality, as well as displays player information such as email. Lastly, the Exit button allows the player to close the game application, and the Leaderboard button displays the scores of players who have played and completed the game.

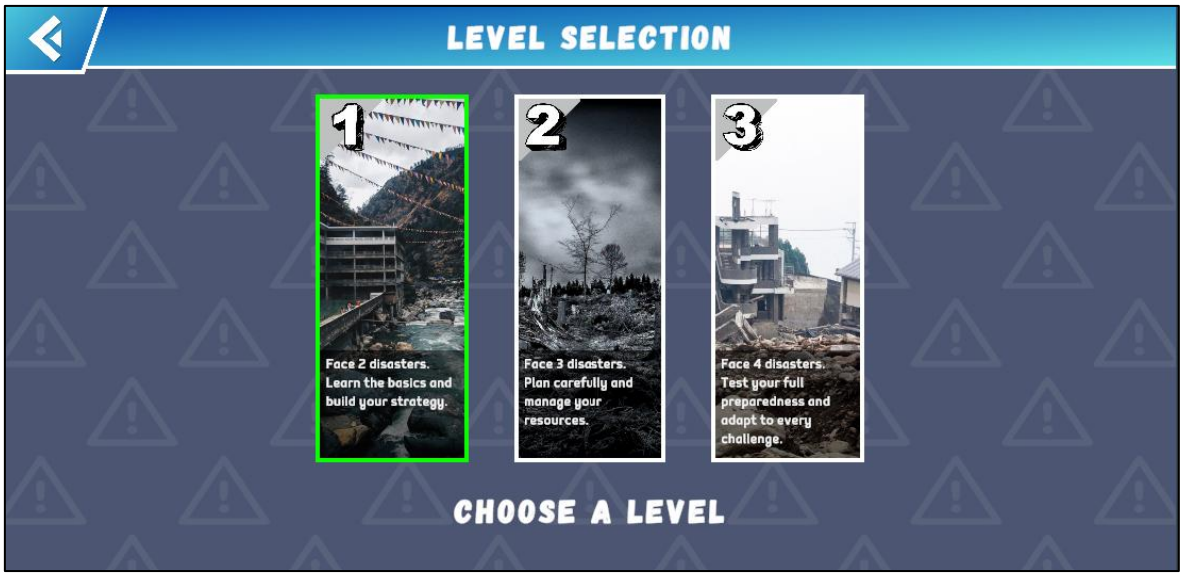


Figure 4.45: Level Selection Scene.

The Level Selection scene allows player to choose the level and difficulty they want to play. This scene provides an intuitive and visually engaging way to select scenarios by interacting with image-based options.

At the top-left corner of the screen, there is a Back button that returns the player to the Main Menu. The main level selection is performed by clicking on level images displayed on the screen. When a level image is selected, its border turns green, visually indicating the player's current selection.

Additionally, the text "CHOOSE A LEVEL" transforms into a Start Game button once a level is selected. Clicking this button will begin the game with the selected scenario and difficulty setting.

Figure 4.46: Gameplay Scene.

The Gameplay scene is the core interface where player interact with the game and carry out all turn-based actions. The background image dynamically changes based on the current enemy card, which represents a disaster such as a flood, wildfire, earthquake, or landslide.

At the top of the screen, three functional buttons are available, Settings button allows players to adjust the volume, Collection button opens a panel displaying all available cards of the game, and Tutorial button provides access to the game's instructions.

On the left side of the screen, there is a discard area that allows the player to discard up to two cards per turn. Below this area, the player stats panel displays two key indicators, preparedness value, which measures the player’s resistance against the current disaster, and health, which shows the player’s remaining life.

In the middle of the screen, the play area is where players drag and drop cards to activate their effects. This area also displays the enemy disaster card, with the current turn number shown above it. Below the play area is the hand area, where drawn cards are held and can be interacted with.

On the right side of the screen, the End Turn button is used to complete the player's turn. Below this button, the cost indicator displays the remaining cost points available for playing cards during the turn. A visual card deck is also shown in the lower-right corner to represent the remaining cards in the deck.

CHAPTER 5

TESTING

5.1 Functional Testing

Functional testing was conducted throughout the development process to ensure that all game features operated correctly and as intended. This type of testing focused on verifying the core functionality and user interactions within the game. The table below outlines the functional testing conducted during the development phase. Each test case was designed to verify that a specific feature within the game functions as intended.

Table 5.1: Feature Function Testing.

Function Tested	Description	Expected Result	Status
Navigation	Test the navigation of the buttons and transitions between scenes.	Buttons navigate to their correct destinations. Scenes load correctly between scenes.	Pass
Player Profile	Test login, signup, and display of user data.	Player can sign up, log in, log out, and view email and profile information.	Pass
Achievement	Verify that achievements unlock based on player actions.	Achievements unlock and appear on the achievement page when criteria are met.	Pass

Table 5.1 continued

Leaderboard	Verify that the leaderboard displays scores of players who completed the game.	The leaderboard correctly displays all player scores and highlights the current player's position.	Pass
Volume Settings	Check audio adjustment functionality.	Volume increases, decreases, or mutes as adjusted. Settings are saved and loaded when the game restarts.	Pass
Level Selection	Test the responsive level selection and level data accuracy.	Selected level is highlighted, and the start button is enabled. The correct level data loads when game starts.	Pass
Card Collection	Test viewing of all available cards in a scrollable panel.	All cards are displayed correctly, and the panel is scrollable.	Pass
Drag and Drop System	Test that the card can be dragged and dropped in the scene.	Card can be dragged and dropped at the intended positions.	Pass
Play Area	Test the area that accepts card drops and triggers card effects.	Card can be dropped in the play area and successfully triggers the card’s stats or effects.	Pass

Table 5.1 continued

Discard Card	Test that players can discard up to two cards per turn.	Selected cards are removed from the hand. After discarding two cards, no more can be discarded in the same turn.	Pass
Draw Card	Test card drawing at the start of each turn and after discarding cards.	Cards are drawn with animation and appear correctly in the hand area.	Pass
Defense and Health System	Test the update of defense and health based on disaster and played cards.	Stats change accurately according to the game logic and update visually in the scene.	Pass
Turn System	Test that turns proceed correctly and end after certain rounds.	Turns progress based on correct logic and stop after the defined round limit.	Pass
Cost System	Test deduction and display of cost when playing cards.	Cost reduces correctly when cards are played and prevents overuse beyond available points.	Pass
Game End	Test the end conditions and display of the correct UI.	Game ends after final turn or health reaches zero. End screen displays results correctly.	Pass

5.2 Responsive User Interface Testing

Responsive UI testing was conducted to ensure that all user interface elements scale properly and remain visually consistent across different mobile devices and screen resolutions. Since the game is intended to run on mobile devices in 1920x1080 landscape mode, the layout and positioning of UI components were tested on both real devices and emulators. Testing focused on verifying that elements such as buttons, cards, text, and indicators do not overlap, fall off-screen, or distort on various screen sizes.

Table 5.2: Device Responsive Testing

Device/Simulator	Screen Resolution	Aspect Ratio	Result
POCO F5	1080 x 2400	20:9	UI scaled correctly. All elements displayed without overlap.
Xiaomi Redmi Note 7	1080 x 2340	19.5:9	UI scaled correctly. All elements displayed without overlap.
Razer Phone	1440 × 2560	16:9	UI scaled correctly. All elements displayed without overlap.
Samsung Galaxy Note 10	1080 × 2280	19:9	UI scaled correctly. All elements displayed without overlap.

Table 5.2 continued

BlueStacks Emulator	1920 × 1080	16:9	UI scaled correctly. All elements displayed without overlap.

5.3 User Testing

User testing was conducted with 10 testers who played the prototype version of the game "Disaster Wise". After playing the prototype, the testers were asked to complete a Google Form survey to provide feedback. The purpose of the survey was to evaluate player understanding, game functionality, user experience, and overall satisfaction. The survey consisted of several sections, including:

- Demographic information
- Game Educational Value
- System Usability Scale (SUS)
- Game User Interface Design
- Game Experience

Game Education Value

This section evaluates the educational content presented in the game, particularly its effectiveness in raising awareness and understanding of disaster preparedness.

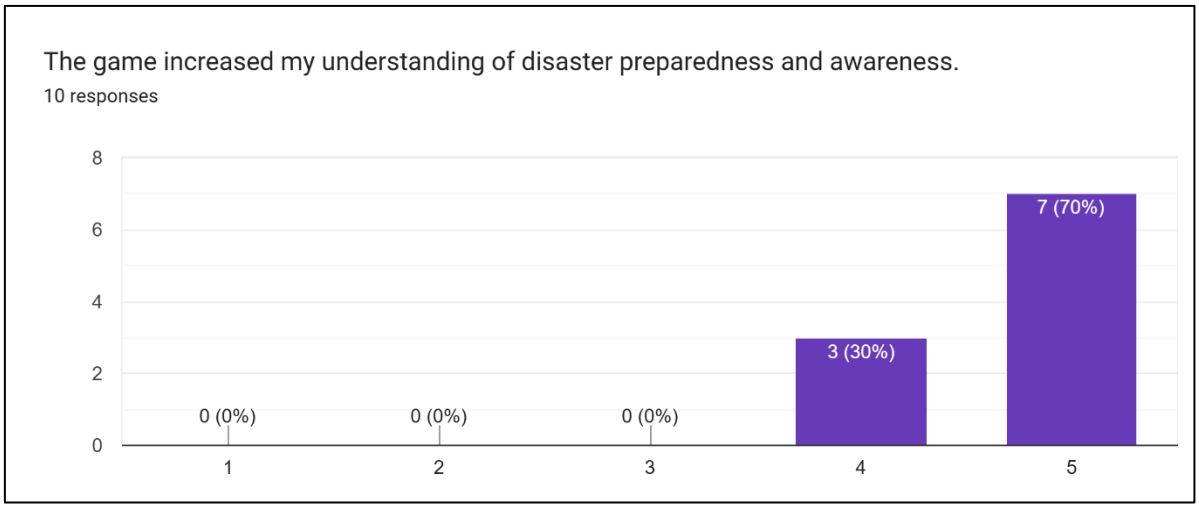


Figure 5.1: Responses on Understanding of Disaster Preparedness and Awareness.

The figure above indicates that the game help increase players' understanding of disaster preparedness and awareness, with 4 testers agreeing and 6 strongly agreeing.

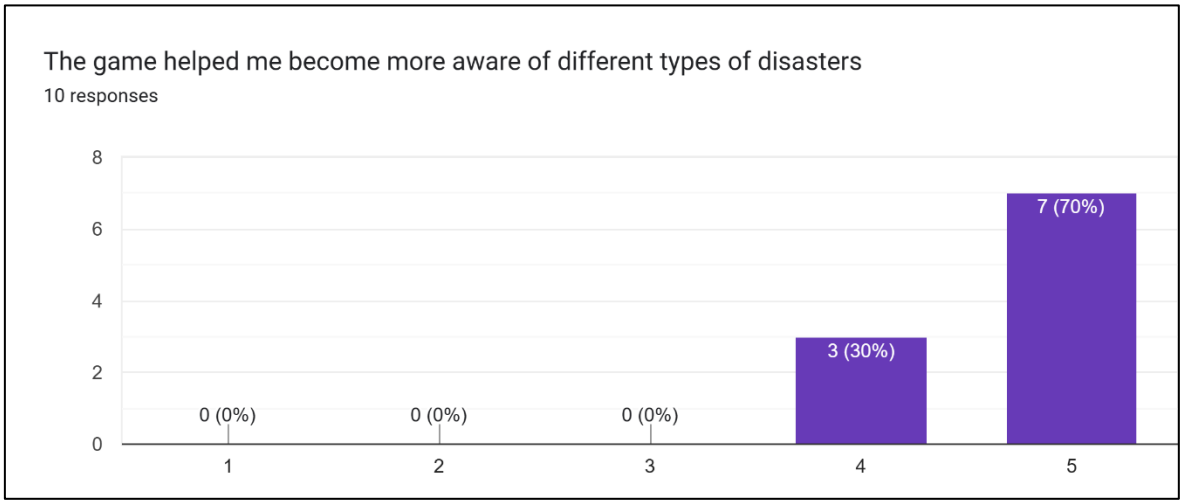


Figure 5.2: Responses on Disaster Type Awareness.

The figure shows that the game helped increase players' awareness of different types of disasters, with 3 testers agreeing and 7 strongly agreeing.

System Usability Scale (SUS)

This section includes 10 standard questions designed to evaluate the usability of the developed game using a Likert scale. The questions are as follows:

- I think that I would like to play this game frequently.
- I found this game unnecessarily complex.
- I thought this game was easy to use.
- I think that I would need help from someone to play this game.
- I found the various functions in this game were well integrated.
- I thought there was too much inconsistency in this game.
- I would imagine that most people would learn to play this game very quickly.
- I found this game very difficult to play.
- I felt very confident playing this game.
- I needed to learn a lot before I could play this game.

The SUS score was estimated based on aggregated responses. Since individual-level responses were not available, average values per question were used to calculate the final score. The estimated SUS score was 80.25, which suggests the game has above-average usability.

Game User Interface Design

This section evaluates the design, layout, and responsiveness of the game's user interface. Testers were asked to rate their agreement with statements related to visual clarity, button placement, ease of navigation, and overall interface consistency. The feedback collected helps determine whether the interface supports an intuitive and smooth user experience across different screens and devices.

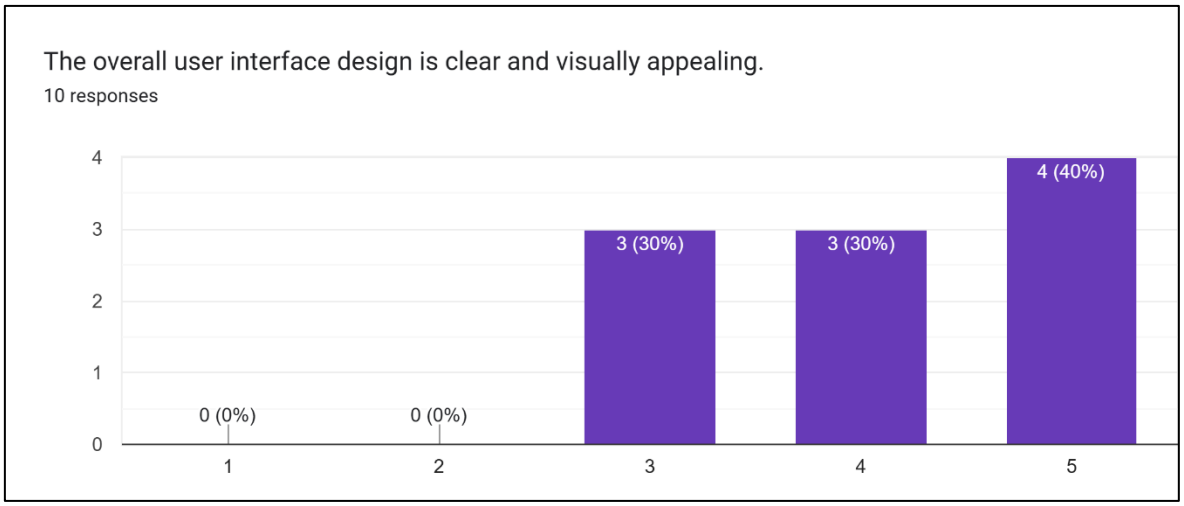


Figure 5.3: Responses on Overall User Interface Design.

The overall user interface design was found to be clear and visually appealing, with 3 testers giving a rating of 3, another 3 testers giving a rating of 4, and 4 testers giving the highest rating of 5.

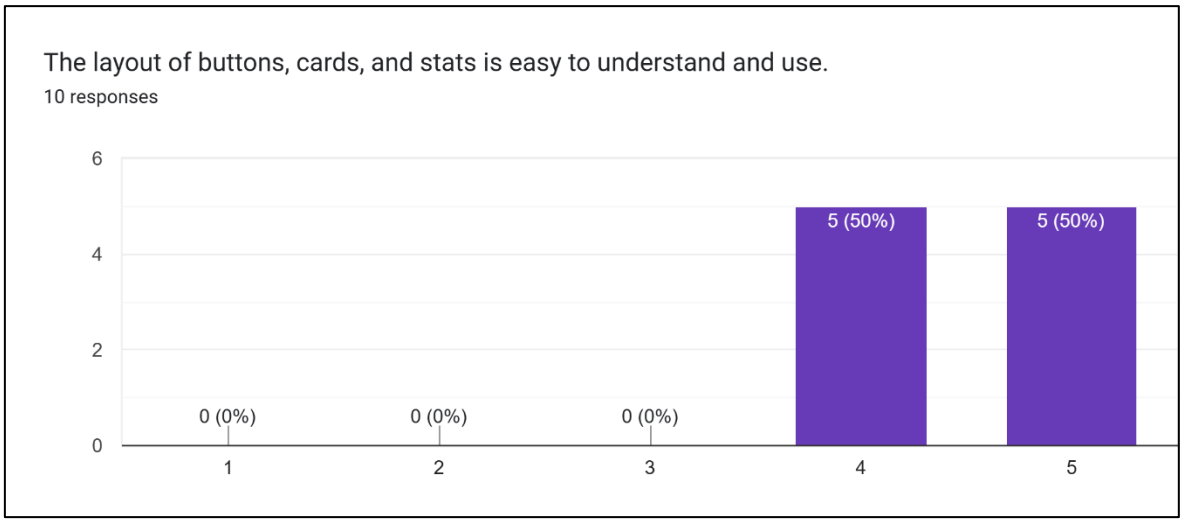


Figure 5.4: Responses on the Layout of the Game.

The layout of buttons, cards, and stats was considered easy to understand and use, with 5 testers giving a rating of 4 and the remaining 5 testers giving the highest rating of 5.

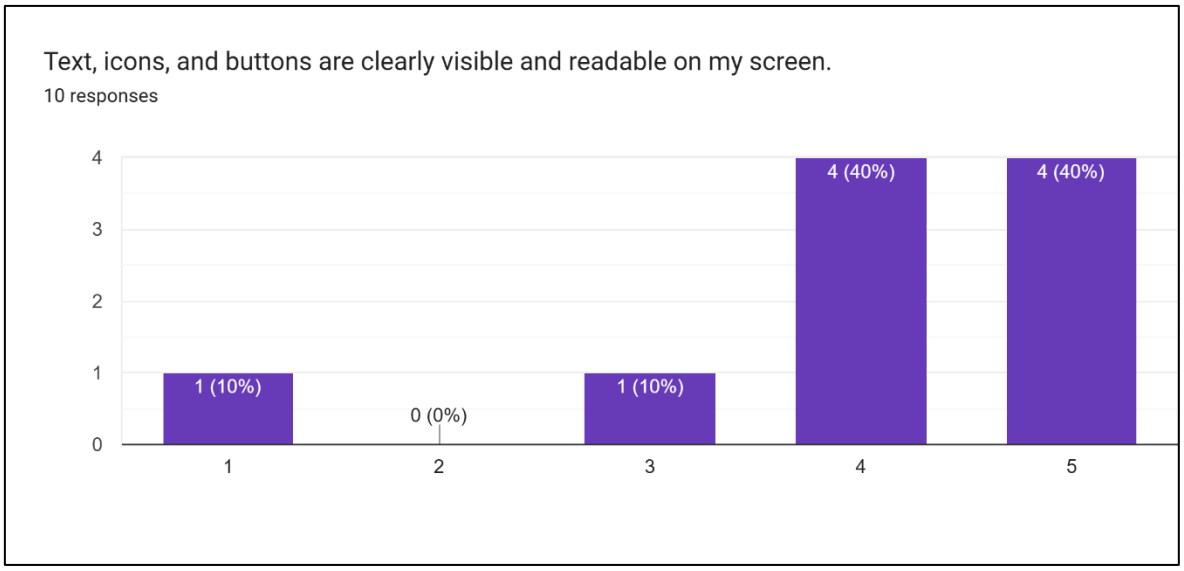


Figure 5.5: Responses on Visibility of User Interface Elements.

The text, icons, and buttons were generally reported to be clearly visible and readable on screen. However, one tester gave a rating of 1, noting that the text on the cards was difficult to read due to its small size. Despite this, the overall feedback was still positive, with 4 testers rating it 4 and another 4 giving the highest rating of 5.

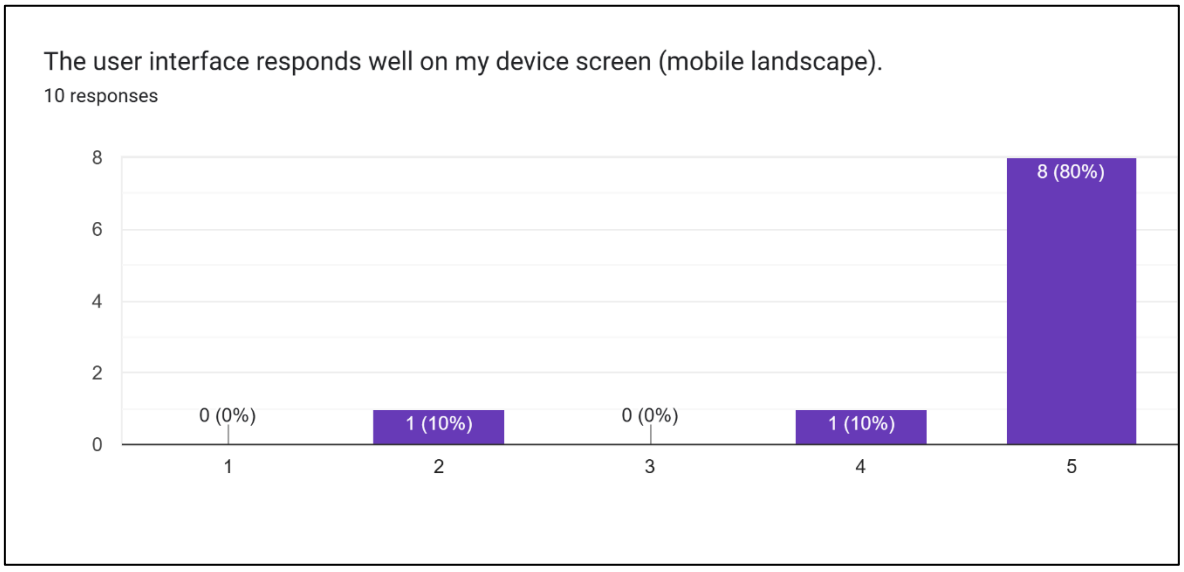


Figure 5.6: Responses on Responsive User Interface.

The user interface responded well on most testers' device screens. There are 8 testers who gave the highest rating of 5, 1 tester who gave a rating of 4, and 1 tester who gave a rating

of 2. The lower rating may be due to differences in screen size or resolution on the tester's device. The game was not designed for a specific screen size or resolution.

Game Experience

This section gathers testers' feedback on the overall gameplay experience, including the game's mechanics, rules, difficulty, feedback, and level of enjoyment. Testers were asked to share their thoughts on both the likes and dislikes of the game, as well as provide suggestions for improvement. These insights can help guide future improvement to the game.

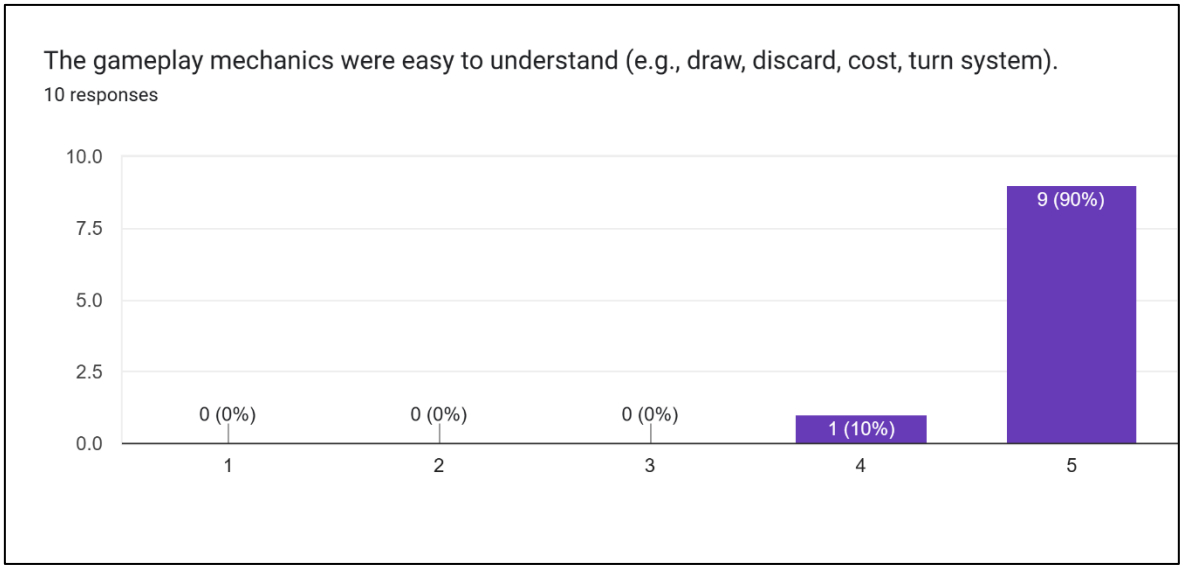


Figure 5.7: Responses on the Gameplay Mechanics.

The gameplay mechanics were easy to understand, with 9 testers giving the highest rating of 5 and 1 tester giving a rating of 4.

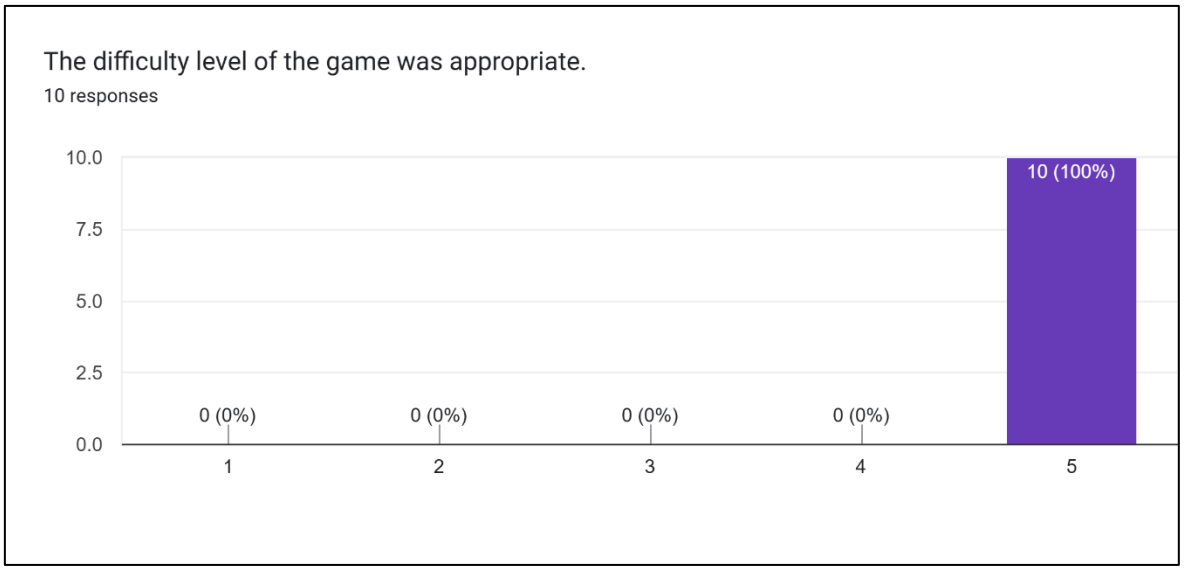


Figure 5.8: Responses on the Difficulty Level.

The difficulty level of the game was considered appropriate, with all 10 testers giving the highest rating of 5. This indicates that the game's level design is well-balanced and suitable across all stages.

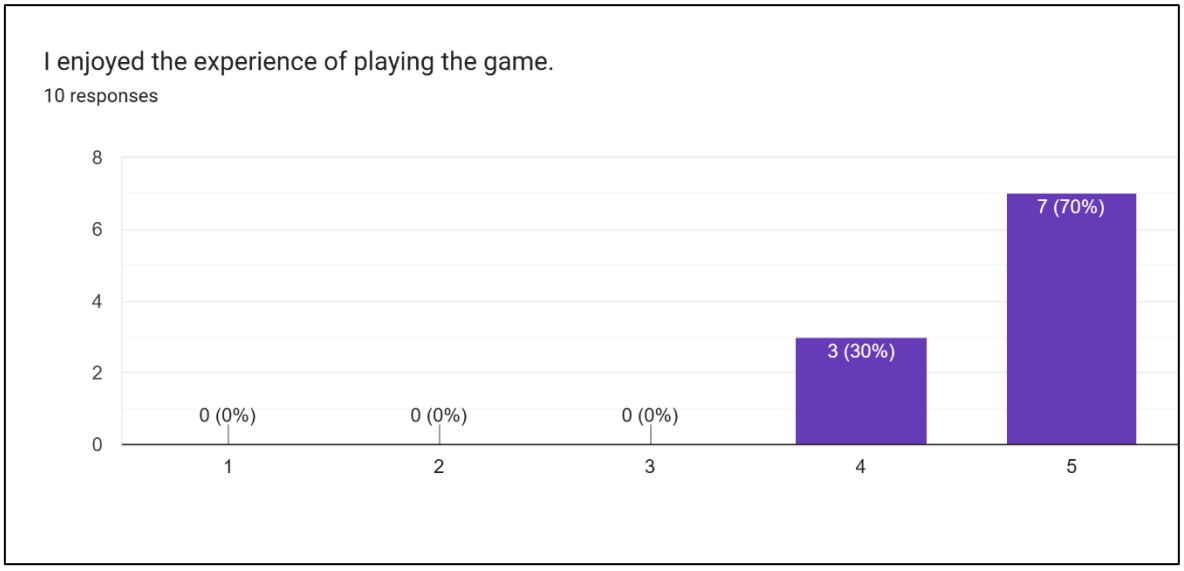


Figure 5.9: Responses on Game Experience.

Most of the testers enjoyed the experience of playing the game, with 3 testers giving a rating of 4 and 7 testers giving the highest rating of 5.

This section also included open-ended questions for testers to share their likes and dislikes. These responses are valuable for improving the game. Some feedback highlighted issues such as the text on the cards being too small, while others provided helpful suggestions, such as adding more realistic sound effects for each disaster type.

CHAPTER 6

CONCLUSION

6.1 Introduction

This chapter outlines the reflection on the overall development and evaluation of the game prototype titled Disaster Wise. It discusses the extent to which the project objectives have been achieved, the limitations of the game, and potential improvements for future versions of the game.

6.2 Objective Achievement

This section outlines the objectives of the project and describes how the game successfully achieved each of the objectives.

Table 6.1: Objectives and Corresponding Achievements.

Objective	Achievement
To design a serious game for disaster preparedness and awareness initiative.	The game was designed and developed with the theme of disaster preparedness and awareness. Educational content is subtly integrated into the gameplay through disaster and preparedness cards. The game helps players understand key aspects of disaster preparedness and raises awareness of different types of disasters.

Table 6.1 continued

To develop a functionality game using game development engine, incorporating components including game mechanics, player interactions, user interface and multimedia elements.	The game was designed and developed using Unity, incorporating various mechanics, including a turn-based system, card system, and action system. The user interface design includes the main menu scene, level selection scene, and gameplay scene. The animation, background music, and sound effects were integrated into the game.
To evaluate the functionality of the game by testing all features to ensure they operate as intended and are free of technical issues.	The game's functionality was tested through functional testing, responsive user interface testing, and user testing. All features operated as intended and were responsive on specific mobile screen sizes and resolutions.

6.3 Limitation and Future Work

Although the game prototype successfully met its objectives, several limitations were identified during the development and testing phases. One of the main limitations is the lack of adaptability across a wide range of device screen sizes and resolutions. While the game was tested on selected mobile devices and emulators, other devices may experience misaligned UI elements or display issues. This limitation affects the accessibility and the

reach of the game. Future versions should improve responsive design to support more device types.

Another limitation of the game is currently the game provides a limited variety of cards, disaster types, and gameplay scenarios. This may affect long term engagement for players. Future work can be focus on expanding the content by introducing new cards, additional levels, and different game modes to increase replayability.

6.4 Summary

This chapter reviewed the overall development and evaluation of the Disaster Wise game prototype. It discussed how the project objectives were successfully achieved through the design, development, and testing phases. Key components such as game mechanics, user interface, and educational content were effectively implemented using the Unity engine. Testing to evaluate the game's functionality and responsiveness on selected mobile devices. Despite its success, the project had certain limitations in aspect of UI responsiveness across diverse screen sizes and limited gameplay content. Suggestions for future improvements include enhancing responsive design, expanding game content, and introducing new features to increase engagement and learning effectiveness.

REFERENCES

Alshowair, A., Bail, J., AlSuwailem, F., Mostafa, A., & Abdel-Azeem, A. (2024). Use of virtual reality exercises in disaster preparedness training: A scoping review. SAGE Open Medicine, 12. https://doi.org/10.1177/20503121241241936

Centre for Research on the Epidemiology of Disasters. (2024). 2023 Disasters in Numbers: A Significant Year of Disaster Impact. In Emergency Events Database (EM-DAT). Retrieved October 20, 2024, from https://files.emdat.be/reports/2023_EMDAT_report.pdf

Darmawan, N. A., & Kamaluddin, R. (2021, September 29). Literature Review: The Effectiveness of Tabletop Disaster Exercise on Disaster Preparedness. Darmawan | Jurnal Studi Keperawatan. https://ejournal.poltekkes-smg.ac.id/ojs/index.php/J-SiKep/article/view/7740/3171

Department of Education and Training, Youth and Culture of the Canton of Vaud (DFJC). (n.d.). CoronaQuest. Retrieved from https://coronaquest.game/

Federal Emergency Management Agency (FEMA). (2024, May 9). Exercises. Ready.gov. Retrieved December 7, 2024, from https://www.ready.gov/business/training/testing-exercise/exercises

Federal Emergency Management Agency (FEMA). (2024, June 17). Disaster Mind. Retrieved from https://www.fema.gov/about/organization/region-8/disaster-mind-game

Federal Emergency Management Agency (FEMA). (2024, August 29). National Preparedness Month: Take Steps to Prepare with Your Family [Image]. U.S. Department of Homeland Security. Retrieved November 30, 2024, from https://www.ready.gov/collection/npm_2024

Federal Emergency Management Agency (FEMA). (2024, September 24). National Preparedness Month. U.S. Department of Homeland Security. Retrieved November 30, 2024, from https://www.ready.gov/september

FEMA. (2024, May 30). How to build a disaster supply kit [Video]. YouTube. https://www.youtube.com/watch?v=bIWIX9dCPHs

Frasca, G. (2001). Video games. Ludology. Retrieved from https://ludology.typepad.com/weblog/articles/thesis/FrascaThesisVideogames.pdf

Hannah Ritchie and Pablo Rosado (2022) - “Natural Disasters” Published online at OurWorldinData.org. Retrieved from: 'https://ourworldindata.org/natural-disasters' [Online Resource]

INFORM. (2024). INFORM Risk Index 2025: Country Profile for Malaysia. INFORM, Inter-Agency Standing Committee Reference Group on Risk, Early Warning and Preparedness, and the European Commission. Retrieved from https://drmkc.jrc.ec.europa.eu/inform-index/INFORM-Risk/Country-Risk-Profile

JPP. (2022, December 18). Community workshops – SEDAR Malaysia-Japan. https://jppsedar.net.my/2022/12/community-workshops/

Lamb, R. (2024, October 23). Serious Games. Oxford Research Encyclopedia of Communication. Retrieved December 8, 2024, from https://oxfordre.com/communication/display/10.1093/acrefore/9780190228613.001.0001/acrefore-9780190228613-e-1482

Nguyen, P. H. (2024, December 5). 10+ types of video game genres – Everything you need to know. AntGames. Retrieved from https://ant.games/blog/lists/video-game-genres/

Panggabean, Francisco. (2023). Enhancing Flood Disaster Preparedness Through Virtual Reality: A VR-based Flood Simulator Game. Engineering, MAthematics and Computer Science (EMACS) Journal. 5. 69-72. 10.21512/emacsjournal.v5i2.9988.

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of Game-Based Learning. Educational Psychologist, 50(4), 258–283. https://doi.org/10.1080/00461520.2015.1122533

Purnomo, E., Hamid, A. Y. S., Gayatri, D., & Setiawan, A. (2024). Challenges and needs in disaster preparedness: A qualitative study. Salud Ciencia Y Tecnología, 5, 1225. https://doi.org/10.56294/saludcyt20251225

Schumacher, L., Senhaji, S., Gartner, B. A., Carrez, L., Dupuis, A., Bonnabry, P., & Widmer, N. (2022). Full-scale simulations to improve disaster preparedness in hospital pharmacies. BMC Health Services Research, 22(1). https://doi.org/10.1186/s12913-022-08230-9

SeDAR Malaysia-Japan. (n.d.). JPP SeDAR Resource Materials. Retrieved from https://jppsedar.net.my/jpp-sedar-resource-materials/

Sharrieff, M. (2023, May 30). The impact of natural disasters. Sciencing. https://www.sciencing.com/impact-natural-disasters-5502440/

Stikova, Elisaveta. (2016). Disaster preparedness. South Eastern European Journal of Public Health. Special volume 2016. 10.4119/UNIBI/SEEJPH-2015-106.

Taheri, Meisam & Javeid, Deniz. (2021). A STUDY OF HOW VIDEO GAMES ASSIST WITH LEARNING. 10.21125/edulearn.2021.2080.

United Nations Office for Disaster Risk Reduction (UNDRR). (n.d.). Stop Disasters!. Retrieved from https://www.stopdisastersgame.org/