University of
Southern Denmark

Shores of Tomorrow

University of Southern Denmark

Faculty of Engineering (TEK)

Software Engineering

Development of software program

First semester

Project Authors:

Santiago Matzkin (smatz25@student.sdu.dk)

Nikolajs Kozevnikovs (nikoz25@student.sdu.dk)

Emil Adzhygryei (emadz25@student.sdu.dk)

Marija Savkina (masav25@student.sdu.dk)

Project Supervisors:

Mubashrah Saddiqa (msad@mmmi.sdu.dk)

Alexandra Moraru (almor24@student.sdu.dk)

Project duration period: 01/09/2025 – 29/01/2026

Course Code: T630032401

Group number 16

Table of Contents

1. Abstract

1. Abstract

Our project is a text-based game “Shores of Tomorrow” inspired by the United Nations (UN) Sustainability Goal number 14 - protecting life below water, focusing on the problems of overfishing and ocean pollution. A player navigates by typing relevant commands - it is possible to go to the different rooms, interact with items and complete quests provided by NPCs. Based on the choices, player gets different endings, where they can see the consequences of their actions. The goal of this game is to raise awareness and educate people about overfishing and water pollution, because these growing problems now threaten both marine life and human livelihoods. The game was written in C# in Visual Studio code. All the changes were documented on GitHub. Main takeaways were about the importance of having good communication in the group, having organized work and making everything on time.

2. Introduction

Such problems as overfishing and ocean pollution appeared a long time ago and proceed to grow nowadays. It contributes to fish species going extinct and people losing their livelihoods. Fish populations are rapidly declining; animals unintentionally caught in nets die and communities that depend on fishing are losing their primary source of income.

Since the problem is unlikely to be solved in the nearest future, our goal is to spread awareness by showing real-life situations and consequences through the interactive game, where the player takes the role of a character who explores different locations and faces choices that can either help protect the ocean or worsen its problems.

At the start of the game, the player is welcomed and given the first quest. To complete it, they must go to the right room and meet a character who provides more information about the task. The player must then decide between completing the following quests quickly, which brings short-term success but long-term harm, or spending more time making a sustainable choice that benefits the environment. Based on the player’s choices several endings are possible, where the player will face the consequences of his actions.

The game targets a broad audience, but especially those who might be unaware of the current environmental situation. Our goal is to educate people, encourage them to speak up, and inspire them to act. In the end of the game, it will be possible to explore how even small choices can make a big difference. Recognizing the problem and spreading awareness is an essential step toward problem solving.

The rest of the report will provide a deeper insight into the environmental issues behind the game and a detailed description of the game's development, including requirements, design, implementation, and code. Finally, we present an evaluation of the project and our conclusions.

3. Problem Analysis

Nowadays, marine life is consistently damaged by human activity. Waste is poured into the oceans; fish gets caught faster than it reproduces. Rare species go extinct because living conditions become unbearable. However, humans lose their jobs, livelihoods, and money because of the same issue. According to recent data [1], over two-thirds of coastal states (107 out of 152) failed to meet any of the four 2020 UN Sustainable Development Goal 14 (Life Below Water) targets related to marine protection and sustainable fisheries. Only 8.4% of the global ocean is currently protected, far below the 30% target set for 2030 [2]. Overfishing continues to worsen, with 35.5% of fish stocks overexploited in 2021 — a number that keeps rising every year [2]. Additionally, ocean acidification reduced the average pH to 8.04 by 2023, and between 2023 and 2025, 84% of coral reefs were affected by a massive bleaching event. These statistics demonstrate an urgent need for greater awareness and action to protect marine ecosystems [3].

The consequences of overfishing and pollution are severe. They create an imbalance in marine life, destroy habitats, and contribute to species extinction. According to the report by The International Union for Conservation of Nature, more than one-third of sharks, rays, and chimaeras are now at risk of extinction [4].

Beyond environmental impacts, these issues also threaten human livelihoods — millions of people, especially in coastal communities, depending on fisheries for food and employment [5]. The decline in fish populations and degradation of marine environments lead to economic instability, food insecurity, and loss of biodiversity [5].

The demand for wild-caught fish has been increasing even faster than the world's population, putting pressure on the fishing industry to catch more than is sustainable. This growing demand creates strong financial incentives for industrial fishers to exceed legal limits and encourages illegal fishing practices. At the same time, water pollution continues to worsen due to industrial production, where fertilizers, chemical byproducts, and waste are often released into the ocean [6]. One of the main reasons this happens is poor waste management and the high cost of proper disposal, which leads many companies to take cheaper, harmful shortcuts. Together, these actions contribute to the ongoing damage to marine ecosystems and make it harder to restore ocean health.

All things considered, this led to us making following problem statement:

4. Problem statement

How can "Shores of Tomorrow" influence people's understanding of overfishing and water pollution while raising awareness about their possible consequences?

5. Limitations

The development of our project had several limitations and delimitations. Our team consists of four members that use three different operating systems: Windows 11, Arch Linux and MacOS, which may cause difficulties while trying to work together. Since the "World of Zuul" framework is a terminal-based game, it does not include a graphical user interface (GUI), which limits us to simple gameplay. Another limitation is that none of the group members have experience with C# and we all have different levels of programming knowledge, therefore someone needs to spend more time understanding the code. Additionally, being a newly formed team also led to some communication and coordination challenges.

We also had several delimitations: we aimed to use accurate data, however it is practically impossible to include everything in a text-based game. Some information must be simplified or adapted to make it engaging and clear for a player.

6. Methods

To address the problem and develop our educational game, we followed a structured process:

1. **Research:** Gathered information about overfishing, ocean pollution, and their global impacts.
2. **Code Analysis:** Studied the original "World of Zuul" source code to understand its logic and framework.
3. **Design:** Planned the game structure, including rooms, NPCs, quests, and interactive choices.
4. **Implementation:** Expanded the game world and added mechanics such as inventory, events, and commands.
5. **Education Focus:** Incorporated informative quests that teach players about ocean protection and responsible choices.
6. **Testing and Debugging:** Regularly tested the game for errors and playability issues.
7. **Collaboration:** Used GitHub for version control and effective teamwork.
8. **Documentation:** Recorded progress, challenges, and decisions for the final report.

7. Requirements

A. Functional

Interface:

- The game must have simple text-based interface; commands typed in lower-case.
- The system must include an introductory paragraph, explaining the purpose of the game, rules and gameplay mechanics.
- TUI – canvas draws background images and a minimap overlay using ANSI colors.

Navigation:

- It must be possible for a player to move between rooms by using relevant commands, such as north, south, east and west.
- It must be possible for a player to control the state of the player and the game – to read the description of the current location by typing "look", return to the previously visited room by typing "back", end the game by typing "quit", get a short command reference by typing "help".

Interaction:

- It must be possible to add items to the inventory, remove and list them.
- It must be possible to interact with an NPC, which triggers scripted conversation and optional quest offer.
- It must be possible to accept, track, and complete quests; outcomes modify game state.

Error Handling:

- Immediate feedback after each command; clear prompts when input is missing or invalid.

- Unrecognized commands must produce "I don't know that command".

B. Non-functional

- **performance** – startup ≤ 5 s; command response ≤ 100 ms.
- **reliability** – no crashes on supported terminals; graceful handling of missing asset files.
- **scalability** – parser designed for easy addition of multi-word commands and new game features.
- **constraints** – minimum terminal size 132 × 43 characters; ANSI-compatible console required.

System related:

- **hardware** – any PC capable of running .NET 6+ and displaying Ansi colors.
- **software** – .NET 8 runtime.
- **compatibility** – works on Windows, macOS, GNU/Linux terminal emulators.
- **integration** – no external; all assets reside in ./assets/graphics/.

Interaction Scenarios:

Scenario 1 – Core navigation

- **initiation** –the player launches the executable; the engine performs a terminal-size validation and blocks progress until a minimum of 132 × 43 cells is available.
- **command** – the user enters the directional keyword north.
- **Input** – the command interpreter confirms the input belongs to the approved verb set and forwards it to the movement subsystem.
- **relocation** – the navigation module consults the current location's adjacency map, updates the "previous location" reference, and swaps the active location pointer to the northern neighbour.

- **Renderer** – the rendering pipeline loads the new room's background asset, recomposes the minimap overlay, and pushes the composite frame to the terminal.
- **feedback** – a concise textual description of the new environment is emitted, confirming successful relocation.

Scenario 2 – Inventory manipulation

- **acquisition** – while situated in a room containing a key entity, the player issues the phrase take key.
- **parsing** – the interpreter extracts the action verb and the object identifier, verifies the object's presence in the current scene, and authorizes the acquisition.
- **update** – the item is appended to the player's personal collection, and the scene's object registry is decremented accordingly.
- **inspection** – the player invokes inventory to audit held items.
- **enumeration** – the system iterates over the stored collection, emitting each item's name and descriptive metadata.
- **disposition** – the player later commands drop key; the engine removes the entry from the personal collection and reinstates it into the ambient object list of the current location.

Scenario 3 – NPC dialogue & Quest lifecycle

- **encounter** – entering a designated chamber activates an npc.
- **dialogue** – the player types talk; the interaction manager initiates a scripted exchange, presenting a greeting and a binary choice regarding quest participation.

- **decision** – upon receiving affirmative input, the quest orchestration component registers a new quest instance, marking its status as active and populating the active-tile matrix.
- **map** – the rendering subsystem overlays the active quest coordinates onto the minimap using a distinctive hue, providing spatial guidance.
- **goal** – the player navigates to the highlighted tile and executes complete.
- **resolution** – the quest engine transitions the quest state to completed, unlocks any dependent outcome quests, and locks alternative branches, thereby updating the global quest graph.

Scenario 4 – Unexpected / Undesigned input

- The player types an unknown command such as fly or a very long random string.
- The input validator does not find the verb in the allowed list, so the system treats it as invalid.
- The program responds with a friendly notice: "i don't know that command." and prompts again for input.
- If the player tries to move in a direction that has no exit (e.g., west from a room with no western door), the navigation check fails and the game prints "you can't go west!" instead of crashing.
- The game continues to run, awaiting a valid command, preserving the current state unchanged.

Outcome: even when the user attempts actions the game was not designed for, graceful error handling keeps the session stable and informs the player of the mistake.

8. Analysis of the Requirements

A. Nouns

- **player** – the human-controlled avatar.
- **room** – a discrete location in the game world.
- **direction** – north / south / east / west.
- **description** – long textual narrative of a room.
- **command** – textual instruction entered by the player.
- **inventory** – collection of items owned by the player.
- **item** – any object that can be taken, dropped or used.
- **npc** – non-player character that can converse and give quests.
- **quest** – a goal-oriented task with a life-cycle (accept → track → complete).
- **canvas** – drawing surface for the TUI (background, minimap).
- **minimap** – schematic view of rooms and player's active position.
- **parser** – component that translates raw input into verb + noun tokens.
- **renderer** – subsystem that composes and outputs frames to the terminal.
- **asset** – static resource (image, text file) loaded from ./assets/.

B. Verbs

- **navigate** – move the player between rooms (north, south, …).
- **look** – display the long description of the current room.
- **back** – return to the previously visited room.
- **quit** – terminate the game cleanly.
- **help** – print a short command reference.

- **add / remove / list** – manipulate the inventory.
- **talk** – initiate scripted dialogue with an NPC.
- **accept / track / complete** – manage quest life-cycle.
- **draw** – render background images and minimap overlay on the canvas.
- **highlight** – mark quest-active rooms on the minimap.
- **validate** – check command syntax and recognize unknown input.
- **load** – read assets from disk into memory.
- **store** – persist player progress or configuration.
- **respond** – emit textual feedback to the player.

C. Noun - Verb Mapping

- **player** → move, look, back, quit, help, add, remove, list, talk, accept, track, complete.
- **room** → draw, highlight, load.
- **direction** → navigate.
- **description** → look.
- **command** → validate, parse.
- **inventory** → add, remove, list.
- **item** → add, remove, list, drop, take.
- **npc** → talk, give-quest.
- **quest** → accept, track, complete, highlight.
- **canvas** → draw, compose.
- **minimap** → highlight, update.
- **parser** → validate, extract, map.
- **renderer** → draw, compose, push.

- **engine** → initialize, loop, shutdown, store.
- **asset** → load, cache.

D. CRC Cards

Player:

- Responsibilities: hold current location, manage inventory, track previous room.
- Collaborators: Room, Inventory, QuestManager.

Room:

- Responsibilities: store description, exits, contained items, NPCs; provide adjacency map for navigation.
- Collaborators: Player, Item, NPC.

Item:

- Responsibilities: represent pick-up-able objects; expose name, description, usable flag.
- Collaborators: Inventory, Room.

Inventory:

- Responsibilities: add / remove / list items; serialize / deserialize contents.
- Collaborators: Player, Item.

NPC:

- Responsibilities: store dialogue script; offer quests on interaction.
- Collaborators: QuestManager, DialogEngine.

Quest:

- Responsibilities: maintain state (inactive, active, completed); define objectives and rewards.
- Collaborators: NPC, QuestManager, MapOverlay.

QuestManager:

- Responsibilities: register new quests, track progress, resolve completion; update minimap highlights.
- Collaborators: Quest, NPC, Renderer.

CommandParser:

- Responsibilities: validate raw input, extract verb + noun tokens, dispatch to appropriate subsystem.
- Collaborators: Engine, Player, Room.

Renderer:

- Responsibilities: draw background images, compose minimap with active-quest highlights, output ANSI frames.
- Collaborators: Canvas, Room, QuestManager.

Engine:

- Responsibilities: initialize subsystems (parser, renderer, assets), run main game loop, handle shutdown & persistence.
- Collaborators: Player, CommandParser, Renderer, QuestManager.

AssetLoader:

- Responsibilities: load graphics and text files from ./assets/, cache resources for reuse.
- Collaborators: Renderer, Engine.

9. Design

Iteration #1

System Architecture:

The game follows a modular architecture designed to separate core functionalities into distinct components for ease of development, testing, and maintenance. The main modules include:

- **Game:** Handles game loop, state updates, and renders logic in the terminal.
- **Command and Parser:** Handles player input, filters out invalid commands.
- **QuestManager and Quest:** Manages quests – their state, active tiles, branching storylines, and story progression.
- **Room:** Contains data and logic for locations, interactable objects, events, and environmental states.
- **Technical interface (TUI):** Formats and displays textual interfaces, menus, dialogue boxes, descriptions.
- **convertGraphics.py:** Utility script that converts image files into their RGB representations pixel by pixel and stores them in .csv formal, which are later used to render images in the terminal.
- **NPC:** NPCs act as interactable "objects" that the player can talk to to accept or finish a quest, or just for fun.

Storing Data:

Game data such as quests, dialogue, items, and world configurations should be stored in structured text files, allowing designers to modify content without altering code. This feature is not implemented in the first iteration since the plot and, consequently, quests have not been designed yet.

User Interface Design:

The interface is fully text-based, optimized for clarity and immersion within a terminal window.

Players interact via keyboard commands such as:

- Movement (north, south, west, east)
- Actions (e.g., look, talk, take, use)
- Menus (e.g., quests, inventory)

Text colors are used to differentiate environments, highlight interactions, and convey game mood.

Algorithms and Logic

Key algorithms include:

- **Quest Graph Traversal:** Determines available quests by evaluating completion status and branching conditions.
- **Event Triggering System:** Randomly or conditionally activates environmental events. (Not implemented in the first iteration of development)
- **Input Parsing Algorithm:** Interprets textual user commands and maps them to internal actions.
- **RGB Conversion Algorithm (Python):** Processes image files pixel-by-pixel to output RGB data for color rendering in the console.

Security and Error Handling:

The system includes basic error handling for invalid file paths, malformed data, and unexpected input. All player input is validated to prevent crashes.

Performance and Scalability:

Since the game runs in a terminal environment, resource usage is minimal. The architecture supports adding new content (quests, maps, events) without major code changes. The quest system's graph structure allows narrative complexity.

Version Control:

Version control (Git) is used to track all changes. Each new feature or fix is implemented in a dedicated branch and reviewed before merging. This ensures compliance with internal design standards and project maintainability.

Iteration #2

System Architecture:

The game follows a 3-layer architecture, consisting of Data, Logic and Presentation layers. This modular design allows for ease of development, testing and maintenance.

In the Data layer, game state is saved and loaded using GameStateSaver and GameStateLoader, which interact with JSON files to store and retrieve player progress.

The Logic layer features Registries and lists to store immutable data related to NPCs, items and quests. Quest progression is now implemented, tracking the state of quest completion, available and active quests and player decisions.

The Presentation layer includes an updated Terminal User Interface (TUI) that manually renders elements on screen, eliminating the dependency on the Spectre.Console library. A minimap has also been introduced.

Storing Data:

Game data, such as items, NPCs and quests, is stored in JSON files, Player progress is stored in player.json, while items and NPCs are stored in static JSON files.

User Interface Design:

The interface remains text-based, optimized for clarity and immersion within a terminal window. Players interact via keyboard commands. The minimap provides a visual representation of the game world.

Algorithms and Logic:

Key algorithms include quest progression, which tracks player decisions and quest completion and random events, which introduce unexpected gameplay elements. The input parsing algorithm continues to interpret textual user commands and map them to internal actions.

Performance and Scalability:

The game's architecture supports adding new content without major code changes and the 3-layer design allows for easy maintenance and updates. Resource usage remains minimal due to the terminal environment.

UML Diagram: see Appendix D

10. Implementation

Iteration #1

During the implementation phase we concentrated on turning software design into a working game in the Visual Studio Code with the C# extension using C# for object-oriented programming, that is based on the .NET framework. The main development environment was Visual Studio Code, which offered all the tools required for project management, coding, and debugging.

We also used some additional tools for better organization during the development process. For version control and code repository management we were using GitHub, which allowed several people to work on the code simultaneously without facing any problems. Miro.com was used to create UTM diagrams, flowcharts, and visual plans and Trello was used to organize tasks.

Coding followed standard C# conventions, for instance, PascalCase for class and method names and camelCase for variables. Code reviews were done to make improvements.

The first steps in the development process were brainstorming the game's plot, planning the backstory, the main player, and the NPCs. We also came up with the quests available to the player and how these would result in different outcomes.

UML diagrams were created to plan code architecture with the necessary classes, their inheritance relationships, and connections between them. This approach helped to ensure that the program structure was logical and matched the gameplay design. Once the planning phase was complete, the team proceeded to implement the code in C#, developing most of the features that had been planned earlier.

When the game starts running it checks if the terminal is the right size, then background and minimap are loaded, dialogue box are displayed. Every time a player moves, the background updates based on the current room and the player location changes on the minimap. All user inputs are being checked by the parser. If the command is unknown, the game prints: “I don't know that command” and you are offered to input again. If a command is valid, it goes through the switch statement and based on the case executes the action. Each NPC holds a list of quests and can be interacted with by using “talk” command. A scripted dialogue is then displayed, offering to complete a quest in the end. If a player chooses to complete it - more information is provided, such as where it should be completed and what do you need to do. Each quest has a status, showing if it is available, in progress or completed. Rooms are stored in an array, and each room holds lists of quests, NPCs, and items associated with that location. Items are stored in a list, while NPCs and quests are stored in dictionaries. Items, Quests, Rooms and NPCs are managed by dedicated manager classes.

Iteration #2

After the second iteration several features were added. After starting the game, it is possible to save it by entering the name of your “world”, which writes the current state into JSON files. Along with the dialogue box, minimap and background, introductory paragraph is displayed. Furthermore, random events were added. They can occur at any time except while in dialogue with NPCs. We also refactored all the code into three-layer architecture – game state controller is in the Data folder, everything about game logic is in the Logic folder and processing of input/output we included in the presentation folder. We have additional folders such as “assets”, that include all files with data, such as images, saves, etc.

After some features were finished, we tested them by writing unit tests. Classes that were tested in a separate file for each class: first, a test game state was created, then based on what is being tested, new rooms, NPCs or items were created and the corresponding methods were executed and verified. Furthermore, throughout the whole process of game development we were testing the code manually – ran the code to check if implemented features work as intended.

11. Evaluation

Iteration #1

After completing the first iteration of our project, we evaluated the game design, functionality, and user experience to understand how well it met our initial goals and what needed refinement. While not all requirements have been fulfilled yet, we aim to complete them by the end of November. All navigation requirements have been met except for save/load. For NPCs and quests logic has been implemented. Furthermore, error-handling has also been implemented. All the commands go through the parser and in case of an invalid command the game prints “I don't know that command”. Additionally, we have introduced a terminal user interface (TUI) that provides a clearer and more engaging representation of the game, helping players better understand its structure and purpose. However, for example, items in the rooms have not been implemented so far. At this stage, we have added NPCs to the game, allowing players to communicate with them and receive quests.

However, some requirements have not yet been met. For instance, our main goal - to educate players about overfishing and water pollution. This is because we have focused primarily on the technical aspects, and the game plot has not been fully implemented in the code. Addressing this will be a priority during the second iteration, along with some other features, such as relevant assets for the background, introductory paragraph, minimap and inventory.

During the game development process, we encountered several challenges related to understanding the game structure. This required us to spend a significant amount of time analyzing the plot and visualizing how to implement it effectively within the game. If we had not faced these obstacles, we would have been able to implement more features, and by now the game would have additional functionality.

Iteration #2

During the second iteration we realized that our code does not follow the correct structure and is inefficient. We refactored the code into three-layer architecture and rewrote the logic behind it, so that it is easier to meet all the requirements. After that we have added new features: implemented an introductory paragraph, uploaded all relevant background pictures, created a minimap. It is now possible to interact with NPCs, receive quests, complete them and save the progress of the game. In short, all the requirements have been met, and the game is now finished.

12. Conclusion

For the past few months, our team has been working on the text-based game “Shores of Tomorrow”, with which we intended to raise awareness about global problems connected with water pollution and overfishing. Providing the choice of either doing “bad” or “good” actions for the environment makes the player think about how their actions are going to impact the state of the island. Throughout the game, the player faces such dilemmas as whether they should help bad people by doing things that harm the environment and benefit from it or miss that opportunity. We believe that it makes the player contemplate the realistic image of people having an impact on the environment and how every small detail can make a difference.

After the completion of the game, we also had several takeaways. It is crucial to be able to work in teams, split roles between everyone, and communicate with each other in order to prevent misunderstandings. Having clearly defined requirements from the beginning is critical for work to be as efficient as possible. During the project, everyone also gained experience with version control and learned to use suitable tools to support it.

We would like to thank everyone who took part in the game development - our supervisor, teachers, and instructors. They made the process more comprehensible and helped us when we were stuck. For the future we have some recommendations. It would be better if we were provided with concrete steps on how to define the requirements and start the game development process from the very beginning.

13. Process Evaluation

Takeaways

- Effective use of version control is essential in team-based development, as it helps prevent code conflicts and supports coordinated collaboration.
- Clearly defined and well-documented requirements significantly reduce misunderstandings and minimize the need for refactoring the code. They also help to communicate with the team since everyone knows what exactly they are supposed to create.
- Meeting deadlines is critical, as unfinished tasks tend to accumulate over time and increase overall project complexity and pressure.
- It is important to understand the role of every team member. Throughout the game development, each person had their own main tasks. This is crucial for successful development, because it makes the work more efficient and allows everyone to focus on their strengths. We had a leader of the group, which also significantly helped us to maintain organised and do everything on time.
- Maintaining clear and consistent communication with the supervisor ensures that the overall process remains transparent and well-organized. It is always preferable to ask clarification through communication rather than risk making an error due to misunderstanding or lack of information.

Working process

- For this project, the work was divided into smaller tasks that followed the main stages: planning, implementation, testing, and review. Regular communication with the supervisor helped to clarify expectations and avoid misunderstandings. Progress was checked often, and feedback was used to make small changes to the plan when needed. These methods helped keep the work organized and focused on the project goals.

- Report writing was also split between all the members. Each was responsible for their own part. Those who had worked on the code more had done less for the report writing and vice versa.

Skills learned

- Team management – we learned how to split tasks between a team of developers with different interests and skill levels.
- Three-layer architecture and its benefits - this architecture reduces the complexity of individual code snippets, helping to understand the code for other teammates.

14. Generative AI Usage

Artificial Intelligence has been used for several purposes during the project. It has bene used to help with the creative process of creating the plot, creation of characters and a connected backstory that showcases our objective. Furthermore, AI has been used to improve programming of code, by debugging, finding mistakes, simplifying methods into shorter lines of code, and explaining certain parts of our code. Finally, AI has been used to improve text formulation in the report, rephrasing stuck sentences and contributing to clear certain parts of text.

15. References

Bibliography

[1] United Nations, “The Sustainable Development Goals Report 2025,” 2025. Available: https://unstats.un.org/sdgs/report/2025/The-Sustainable-Development-Goals-Report-2025.pdf

[2] United Nations, “SDG Indicators,” *UN.org*, 2025. https://unstats.un.org/sdgs/report/2025/Goal-14/

[3] T. R. Walker, “(Micro)plastics and the UN sustainable development goals,” *Current opinion in green and sustainable chemistry*, vol. 30, p. 100497, Apr. 2021, doi: https://doi.org/10.1016/j.cogsc.2021.100497.

[4] International Union for Conservation of Nature, “A third of sharks, rays, and chimaeras are threatened with extinction, as new report narrows in on solutions,” *IUCN*, Dec. 02, 2024. https://iucn.org/press-release/202412/third-sharks-rays-and-chimaeras-are-threatened-extinction-new-report-narrows

[5] United Nations Environment Programme, “Goal 14: Life Below Water,” *UNEP - UN Environment Programme*, 2019. https://www.unep.org/topics/sustainable-development-goals/why-do-sustainable-development-goals-matter/goal-14

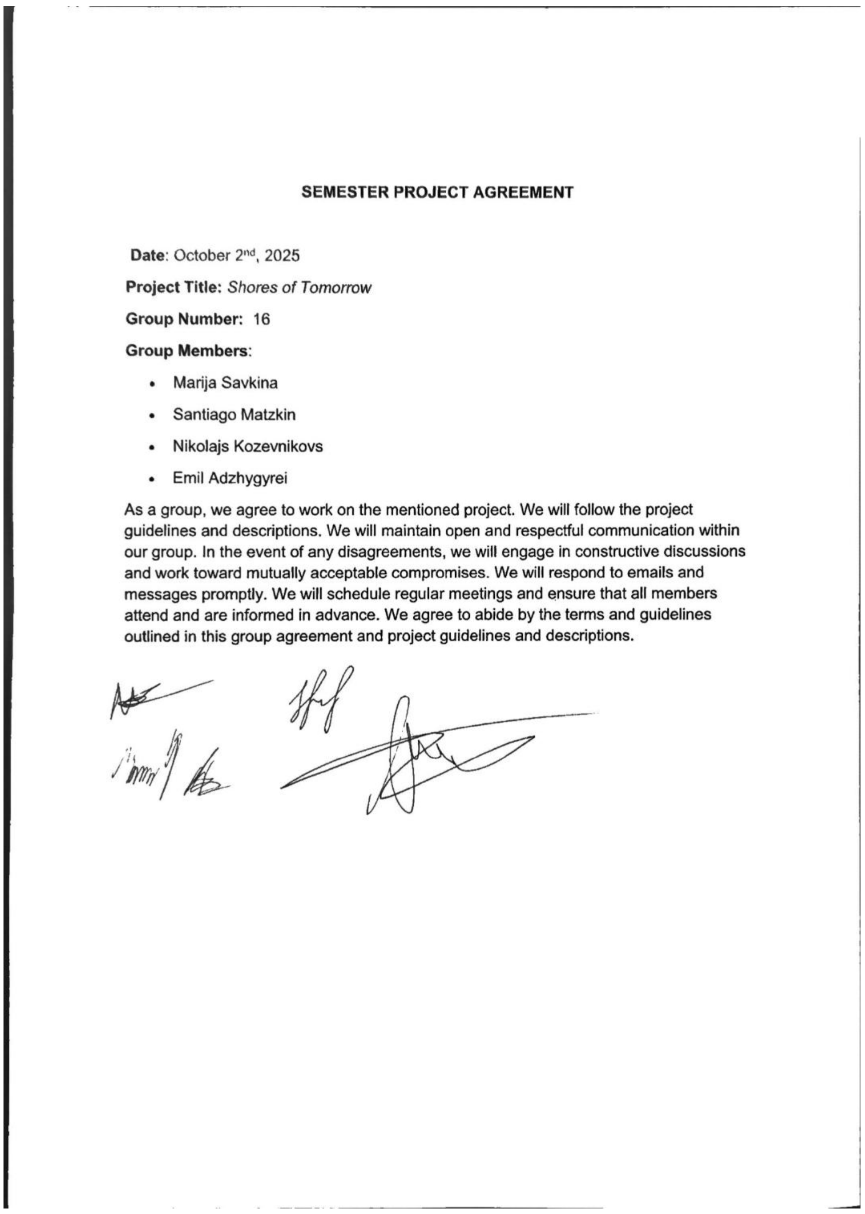
[6] Global Plastics Hub, “Global Plastics Hub,” *Globalplasticshub.org*, 2025. https://globalplasticshub.org/es/data/maps

[7] The Global Goals, “Goal 14: Life below water,” *The Global Goals*, 2024. https://globalgoals.org/goals/14-life-below-water/

[8] Clean Seas, “Did You Know,” *Clean Seas*, Apr. 15, 2021. https://www.cleanseas.org/did-you-know

16. Appendixes

Appendix A: Collaboration Agreement



SEMESTER PROJECT AGREEMENT

Date: October 2nd, 2025

Project Title: *Shores of Tomorrow*

Group Number: 16

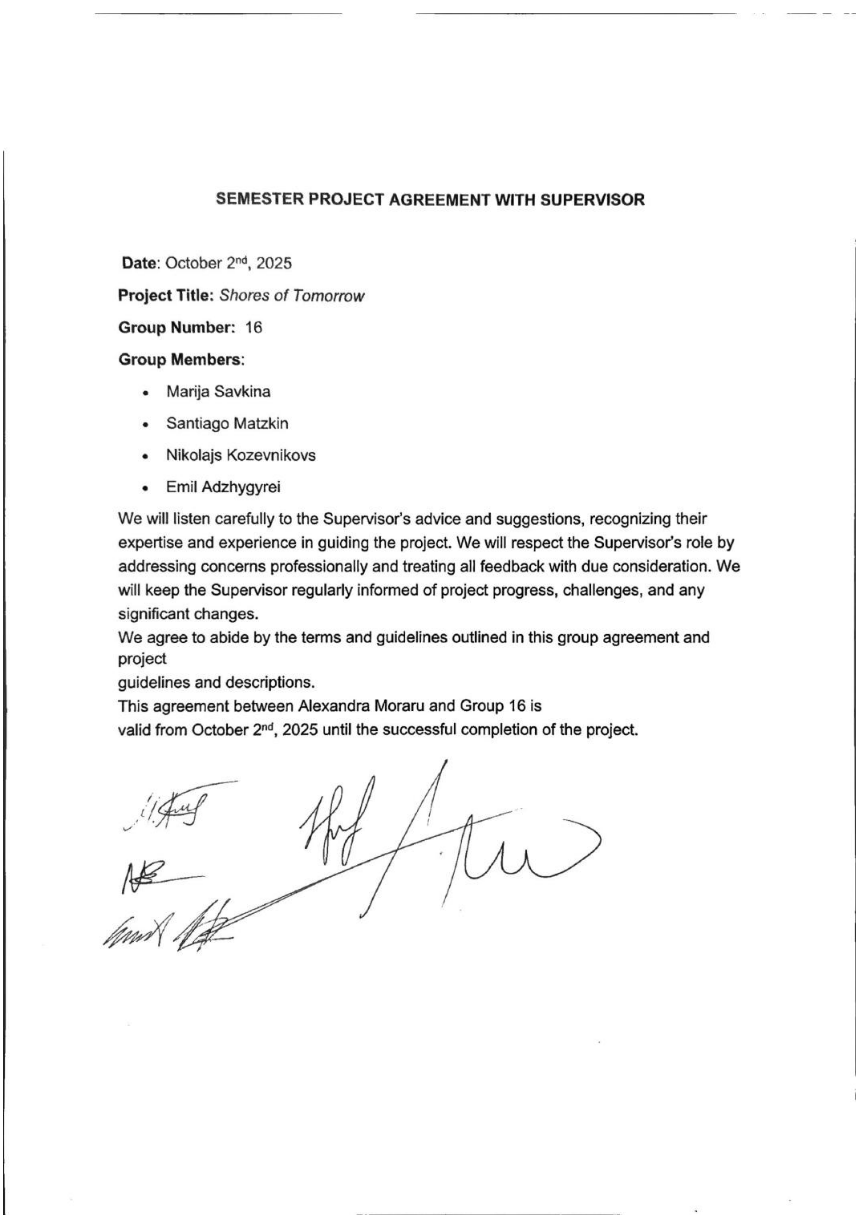
Group Members:

- Marija Savkina
- Santiago Matzkin
- Nikolajs Kozevnikovs
- Emil Adzhygyrei

As a group, we agree to work on the mentioned project. We will follow the project guidelines and descriptions. We will maintain open and respectful communication within our group. In the event of any disagreements, we will engage in constructive discussions and work toward mutually acceptable compromises. We will respond to emails and messages promptly. We will schedule regular meetings and ensure that all members attend and are informed in advance. We agree to abide by the terms and guidelines outlined in this group agreement and project guidelines and descriptions.

[illegible] [illegible] [illegible] [illegible]

Appendix B: Supervisor Agreement



SEMESTER PROJECT AGREEMENT WITH SUPERVISOR

Date: October 2nd, 2025

Project Title: *Shores of Tomorrow*

Group Number: 16

Group Members:

- Marija Savkina
- Santiago Matzkin
- Nikolajs Kozevnikovs
- Emil Adzhygyrei

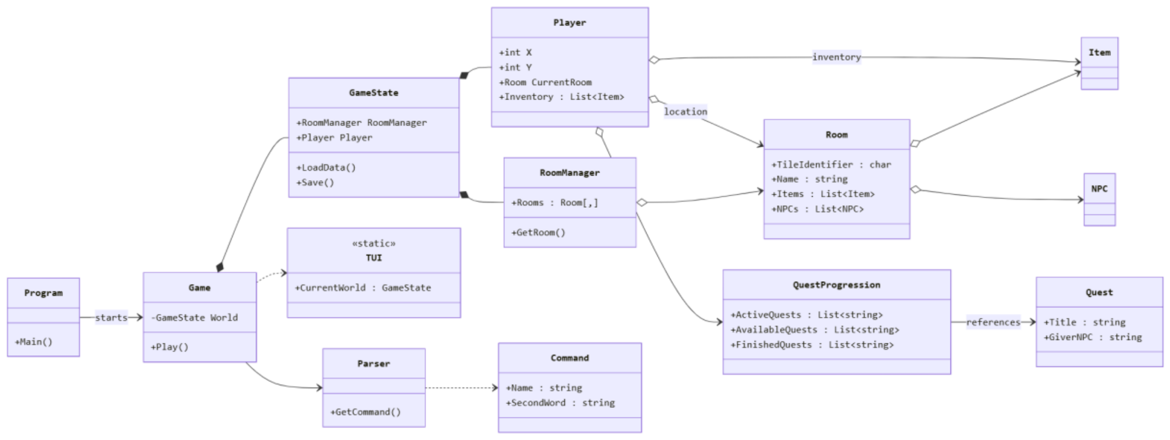
We will listen carefully to the Supervisor's advice and suggestions, recognizing their expertise and experience in guiding the project. We will respect the Supervisor's role by addressing concerns professionally and treating all feedback with due consideration. We will keep the Supervisor regularly informed of project progress, challenges, and any significant changes.
We agree to abide by the terms and guidelines outlined in this group agreement and project
guidelines and descriptions.
This agreement between Alexandra Moraru and Group 16 is
valid from October 2nd, 2025 until the successful completion of the project.

[illegible]

Appendix C: Project Log

Meeting logs

Appendix D: UML Diagram



Iteration #2 – UML Diagram (Simplified)

Appendix E: Portfolio Checklist

Chapter	Content	Fulfilled +/-
Cover Page	Project title, educational institution, faculty, department, program, course code, project period, supervisor, project group, and project participants (first name, last name, SDU email). May include illustrations.	+
Table of Contents	Comprehensive table of contents for the entire project portfolio.	+
Introduction	Project context and background for the project.	+
Problem analysis	Description of the delivered framework. Problem statement and limitations. Methods. Timeline.	+
Requirements	Total requirements (received and formulated by you). Analysis of the requirements, including: • Noun-Verb analysis	+

	• CRC cards	
Design	Description of the design, including one or more UML class diagrams. The description includes architecture and user interface design.	+
Implementation	Description of the implementation with key parts of the program code.	+
Evaluation	Evaluation of whether the solution meets the requirements, e.g., through user surveys or user testing. Description of what has been achieved and what has not been achieved in the project compared to the expected outcomes described in the introduction. Description of strengths and weaknesses in the results and whether better results could have been achieved. Description of known issues.	+
Iteration #1 and #2	Have results from both iteration #1 and iteration #2 been included in the above sections (Requirements, Design, Implementation, and Evaluation)?	+
Conclusion	Summary of the results and the evaluation. Answer to the problem statement.	+
Process Evaluation	Reflection on the process and the group's consideration of the process: The learning process, team roles, internal group and supervisor cooperation, project work methods, working methods, methodologies, writing process, time management of the project, project leadership, project work distribution, etc.	+
Collaboration Agreement	Insert the collaboration agreement as Appendix.	+
Supervisor Agreement	Insert the supervisor agreement Appendix.	+
Project Log	Address and a link to the project log Appendix.	+
Portfolio Checklist	Completed Portfolio Checklist Appendix.	+