MONASH University

FIT3077 Sprint 1
Team Information and Technology Prototypes

Tutorial: MA_Thursday08am

Team 013: SoftWheres

Prepared by

Teng Kong Cheng
Byran Hii Neng Yuan
Foo Jia Wei Bryan
Benjamin Tan En Zhe

Table Content

1. Team Information

1.1 Team Name and Photo

Team Name: SoftWheres

1.2 Team Membership

Name	Contact	Email	Roles
Teng Kong Cheng	012-437 6760	kten0008@student.monash.edu	Scrum Master, Technical Lead
Bryan Hii Neng Yuan	013-815 0318	bhii0003@student.monash.edu	Product Owner, Quality Assurance Tester
Foo Jia Wei Bryan	018-202 9792	jfoo0028@student.monash.edu	Tech Dev, UX Specialist
Benjamin Tan En Zhe	017-525 9818	btan0068@student.monash.edu	Tech Dev, Technical Writer

1.3 Roles and Responsibilities

Product Owner (Team Lead) Responsibilities

- Collaborate with team members to gather requirements and convert them into user stories that define specific product features.
- Prioritize user stories based on their relevance to the assignment and expected impact.
- Maintain and refine the product backlog, making adjustments as needed based on team discussion, feedback and evolving priorities.
- Define the overall vision and goals of the project, ensuring development efforts align with the assignment objectives and guidelines.

Scrum Master Responsibilities

- Facilitate team meetings, including discussions on progress, planning, reviews, and retrospectives.
- Guide the team in understanding and applying Agile principles effectively.
- Encourage team members to self-organize and make collaborative decisions.
- Assist in keeping the backlog well-structured and prioritized.
- Ensure transparency by keeping project information accessible and clear to the team.

Technical Lead Responsibilities

- Provide technical guidance on design, architecture, coding standards, and technology choices to help the team make informed decisions.
  Enforce best coding practices, conduct code reviews, and ensure code consistency and maintainability.
- Work with the Scrum Master to distribute tasks based on team members' strengths.
- Foster collaboration within the team and support knowledge sharing.
- Oversee the quality of the codebase by managing technical debt and ensuring dependencies are up to date.

UX Specialist Responsibilities

- Design the structure and layout of the product to ensure clear information distribution.
- Create wireframes and prototypes to visualize the functionality and user flow.
- Enhance user interactions for a smooth and intuitive experience.
- Ensure the final product meets usability expectations based on the target audience.
- Share design decisions with the team and incorporate feedback for improvements.

Technical Writer Responsibilities
- Develop clear, well-structured documentation to assist users and developers.
- Keep documentation updated with each project iteration.
- Write developer guides and explain features in a way that is easy to understand.
- Ensure documentation is suitable for the intended audience, including non-technical users.
- Simplify technical concepts into user-friendly explanations.

Quality Assurance Tester Responsibilities
- Review requirements and create a test plan outlining the testing approach.
- Develop test cases based on user stories and requirements.
- Execute tests, identify bugs, and report issues.
- Verify that software features function correctly and meet the assignment's criteria.
- Document test results, including issues found, their impact, and suggested fixes.
- Maintain test-related documentation for reference.
- Collaborate with team members by sharing test results and improvement suggestions.

Team Members Responsibilities
- Commit to completing assigned tasks and contributing to the project’s success.
- Take ownership of tasks from the sprint backlog and ensure they meet the expected completion criteria.
- Participate in discussions to improve team processes and efficiency.
- Adapt to changes in requirements and be open to feedback.
- Maintain the quality of work and meet the project’s standards.
- When clarification is needed, seek answers via the forum or schedule consultations with the lecturer.

1.4 Technical Proficiency and Fun Facts

Name	Technical Strength	Professional Strength	Fun Fact
Teng Kong Cheng	Skilled in full-stack development (Python, Java)	Problem-solving and system architecture	Likes to brew coffee
Bryan Hii Neng Yuan	System integration, requirement analysis, technical documentation	Attention to detail, ensuring good user experience.	High passion in gaming
Foo Jia Wei Bryan	Skilled in scrum process and create useful user stories that navigate the project.	Team collaboration and adaptability.	Doing sports is my main source of adrenaline
Benjamin Tan En Zhe	Database management and API integration.	Analytical thinker, efficient debugging	loves to collect vinyls

1.5 Team Schedule.

Regular Meetings: Every Monday at 2pm
Regular Work Schedule: Monday-Friday, 9 AM - 5 PM

2. Technology Prototypes

2.1 Comparison of Programming Languages (in term of OOP) and Their Respective Frameworks

Feature	Python	Java
Ease of Learning	Simple syntax, easy to read	Verbose syntax, requires more setup
Performance	Slower due to dynamic typing and interpreted execution, which increases overhead.	Faster execution due to static typing, Just-In-Time (JIT) compilation, and better optimization techniques.
Memory Management	Automatic garbage collection, but with less direct control over memory allocation.	More control over memory management, allowing for optimizations in high-performance applications
Type Safety	Dynamic typed, may lead to runtime errors and harder debugging in large OOP systems.	Strongly and statically typed, reducing errors, especially in complex OOP systems with better compile-time checks.

2.2 Framework Comparison: Tkinter vs. Java Swing

Feature	Tkinter	Java Swing
Ease of Use	Simple, lightweight	More powerful but complex
Performance	Suitable for lightweight apps	Better for complex applications
Look & Feel	Basic UI, less customization	Richer UI, more customization
Cross-Platform	Works on Windows, macOS, Linux	Works across all OS via JVM
Integration	Easily integrates with Python apps	Fits naturally with Java apps
Development Speed	Faster prototyping, easy to modify	More structured but slower

2.3 Selected Programming Language and Justification

After evaluating both **Python (Tkinter)** and **Java (Swing)** alongside with expertise of each team member, we have decided to proceed with **Java and Java Swing** for the following reasons:

1. **Team Expertise & Adaptability**

   Since all team members have experience in both **Python and Java**, we have the flexibility to choose the most suitable language for our project. Additionally, our team is proficient in **both Tkinter and Java Swing**, meaning we can effectively develop and troubleshoot applications using either framework.

   However, given the nature of our project—where long-term maintainability, structured OOP design, and performance are critical—**Java's static typing and strong OOP principles make it the more reliable choice**. Java enforces a disciplined programming approach, reducing the likelihood of runtime errors that can arise from Python's dynamic typing, especially in large-scale applications.

   Furthermore, Java provides **better documentation, industry-standard design patterns, and enterprise-level frameworks**, making it easier to collaborate, scale, and integrate with other technologies in the future.

2. **Performance & Scalability**

   Since the Santorini board game must be implemented in OOP to represent different components, Java just comes to mind naturally as it is strictly statically typed, ensuring a stronger type safety if working in a team. Even if a teammate does not fully understand a specific part of the code, Java's strict type system helps prevent accidental errors, ensuring that objects interact as expected. This reduces debugging time, as many potential issues are caught at compile-time rather than runtime, making the coding process smoother and more predictable.

   Additionally, Java's OOP enforcement allows for clearer separation of concerns, ensuring that each component—such as the Game, Board, Player, Worker, and God classes—has well-defined behaviors and interactions. The compiler enforces method signatures and expected data types, preventing unintentional misuse of objects, which is especially useful when multiple developers are working on different parts of the project.

   Beyond type safety, Java's Just-In-Time (JIT) compilation optimizes execution speed, making real-time move validation and game state updates significantly faster compared to an interpreted language like Python. Automatic memory management ensures that objects are efficiently handled, preventing memory leaks even as the game state updates dynamically.

Furthermore, Java's scalability and modularity mean that additional features—such as AI opponents, enhanced animations, or networked multiplayer—can be integrated seamlessly without refactoring the entire codebase. This makes Java not just a suitable choice for implementing the core Santorini gameplay, but also a future-proof option for expanding and refining the game during Sprint 3.

3. **User Interface & Customization**

   For a graphical user interface (GUI)-based application, the framework must provide flexibility, a modern look and feel, and seamless user interaction. Since Santorini is a board game, you need a UI framework that allows you to

   - Render a grid-based board dynamically
   - Display game pieces (workers) and buildings with different moves.
   - Update the UI in real-time as players make moves.

   Santorini's game involved multiple components, for example the board, player, pisces, game state information and UI buttons. Swing provided a very good advantage as it works well with Model-View-Controller (MVC) architecture, allowing us to separate the game logic from UI rendering, making our code more organised.

   Next, Java with Swing framework provided advanced UI components such as panels, layered panes and custom drawing using Graphics2D, which allow you to create and customise the game board efficeintly. Beside that, you can use custom painting techniques to render the board, workers and god power effects, making the game visually appealing. Last but not least, Unlinke Tkinter which has limited widget flexibility, Swing allows for better animation handling and layered rendering, essential for visualising the Santorini game mechanics.

2.4 Prototyping & Execution Testing

To validate our selection, we have conducted basic prototyping to ensure:

- Each team member can practically apply their knowledge.
- An executable can be created and run on another computer without our specific setup.
- Early identification of potential technical roadblocks.

Each team member has developed a prototype stored in our Git repository under the **'prototypes'** folder:

3 User Stories

3.1. Basic Gameplay User Stories (Sprint 2)

1. **As a player, I want to start a new game so that I can play Santorini from the beginning.**
   Independent: This story does not depend on other stories.

   Negotiable: The implementation (e.g., button placement, confirmation dialog) can be discussed.

   Valuable: It provides value by allowing players to start a new game.

   Estimable: The effort to implement a "New Game" button is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Clicking 'New Game' resets the board and workers."

2. **As a player, I want to select my preferred God Card from a given set so that I can use its special ability.**
   Independent: This story can be implemented without relying on other stories.

   Negotiable: The number of God Cards and their abilities can be discussed.

   Valuable: It adds strategic depth to the game.

   Estimable: The effort to implement a God Card selection screen is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Player can select a God Card from a list and see its ability."

3. **As a player, I want to place my two workers on the board at the beginning of the game so that I can start playing.**
   Independent: This story does not depend on other stories.

   Negotiable: The placement rules (e.g., starting positions) can be discussed.

   Valuable: It is essential for starting the game.

   Estimable: The effort to implement worker placement is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Players can place two workers on valid starting spaces."

4. **As a player, I want to take turns with another player so that the game follows the standard Santorini rules.**
   Independent: This story can be implemented independently.

   Negotiable: The turn order and rules can be discussed.

   Valuable: It ensures fair gameplay.

   Estimable: The effort to implement turn-based logic is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Players alternate turns, and the game enforces turn order."

5. **As a player, I want to move one of my workers to an adjacent space so that I can position myself strategically.**
   Independent: This story does not depend on other stories.

   Negotiable: The movement rules (e.g., adjacency, height restrictions) can be discussed.

   Valuable: It is core to gameplay.

   Estimable: The effort to implement worker movement is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Workers can move to adjacent spaces following movement rules."

6. **As a player, I want to build a level on an adjacent space after moving so that I can advance the game.**
   Independent: This story does not depend on other stories.

   Negotiable: The build rules (e.g., height restrictions) can be discussed.

   Valuable: It is core to gameplay.

   Estimable: The effort to implement building logic is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Players can build on adjacent spaces after moving."

7. **As a player, I want to be able to build up to three additional levels on a building so that I can construct a complete tower.**

   Independent: This story does not depend on other stories.

   Negotiable: The build rules (e.g., max height) can be discussed.

   Valuable: It adds depth to gameplay.

   Estimable: The effort to implement multi-level building is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Players can build up to level 3 on a building."

8. **As a player, I want to move onto a level 1, 2, or 3 building so that I can gain height advantage.**

   Independent: This story does not depend on other stories.

   Negotiable: The movement rules (e.g., height restrictions) can be discussed.

   Valuable: It adds strategic depth.

   Estimable: The effort to implement height-based movement is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Workers can move onto buildings of level 1, 2, or 3."

9. **As a player, I want to win the game immediately when I move onto the third level so that I can claim victory.**

   Independent: This story does not depend on other stories.

   Negotiable: The win condition rules can be discussed.

   Valuable: It defines the game's objective.

   Estimable: The effort to implement a win condition is clear.

   Small: It can be completed in a single sprint.

   Testable: Acceptance criteria: "Game ends when a worker moves onto level 3."

10. **As a player, I want to be unable to move onto domed buildings so that the game follows official Santorini rules.**

    Independent: This story does not depend on other stories.

    Negotiable: The dome rules can be discussed.

    Valuable: It ensures rule compliance.

    Estimable: The effort to implement dome restrictions is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Workers cannot move onto domed buildings."

11. **As a player, I want to undo my last move (before confirming) so that I can correct mistakes before submitting my turn.**

    Independent: This story does not depend on other stories.

    Negotiable: The undo rules (e.g., how many moves can be undone) can be discussed.

    Valuable: It improves player experience.

    Estimable: The effort to implement an undo feature is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Players can undo their last move before confirming their turn."

12. **As a player, I want an indicator to highlight valid movement and build options so that I don’t make illegal moves.**

    Independent: This feature can be developed without relying on other stories.

    Negotiable: The design of the indicator (e.g., color, animation) can be discussed.

    Valuable: It improves the player experience by reducing errors.

    Estimable: The effort to implement visual cues is straightforward.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Valid moves and builds are visually highlighted, and invalid ones are not."

13. **As a player, I want the game to enforce movement and build rules so that I do not accidentally break the rules.**

    Independent: This story can be implemented independently.

    Negotiable: The specific rules (e.g., height restrictions, adjacency) can be discussed.

    Valuable: It ensures fair and rule-compliant gameplay.

    Estimable: The effort to implement rule enforcement is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "The game prevents illegal moves and builds."

14. **As a player, I want to see whose turn it is so that I know when it is my turn to play.**

    Independent: This feature does not depend on other stories.

    Negotiable: The display format (e.g., text, icon) can be discussed.

    Valuable: It improves clarity and reduces confusion.

    Estimable: The effort to implement a turn indicator is straightforward.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "The current player’s turn is clearly displayed on the screen."

15. **As a player, I want to receive a notification when I win or lose so that I know the game outcome.**

    Independent: This story can be implemented independently.

    Negotiable: The notification style (e.g., pop-up, sound) can be discussed.

    Valuable: It provides closure and enhances the player experience.

    Estimable: The effort to implement win/lose notifications is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "A notification appears when a player wins or loses."

16. **As a player, I want to restart the game after finishing a match so that I can play again.**

    Independent: This feature does not depend on other stories.

    Negotiable: The restart mechanism (e.g., button placement) can be discussed.

    Valuable: It improves replayability and convenience.

    Estimable: The effort to implement a restart button is straightforward.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Clicking 'Restart' resets the game to its initial state."

3.2. Extension User Stories (Sprint 3)

17. **As a player, I want to play on a customizable board with different sizes and shapes so that I can experience different gameplay variations.**

    Independent: This feature can be developed independently.

    Negotiable: The customization options (e.g., board dimensions, shapes) can be discussed.

    Valuable: It adds variety and replayability.

    Estimable: The effort to implement board customization is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Players can select and play on boards of different sizes and shapes."

18. **As a player, I want to play with 3 or 4 players so that I can enjoy Santorini with more friends.**

    Independent: This story does not depend on other stories.

    Negotiable: The rules for additional players can be discussed.

    Valuable: It increases social play options.

    Estimable: The effort to implement multiplayer support is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "The game supports 3-4 players with correct turn order and rules."

19. **As a player, I want to have additional God Cards with unique abilities so that I can experience more strategic options.**

    Independent: This feature can be developed independently.

    Negotiable: The number and abilities of God Cards can be discussed.

    Valuable: It adds depth and variety to gameplay.

    Estimable: The effort to implement additional God Cards is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Players can select and use additional God Cards with unique abilities."

20. **As a player, I want to save my game progress so that I can continue later.**

    Independent: This story does not depend on other stories.

    Negotiable: The save mechanism (e.g., local vs. cloud storage) can be discussed.

    Valuable: It improves convenience and flexibility.

    Estimable: The effort to implement save functionality is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Players can save their game progress and resume later."

21. **As a player, I want to load a previously saved game so that I can resume where I left off.**

    Independent: This feature can be developed independently.

    Negotiable: The load mechanism can be discussed.

    Valuable: It improves convenience and flexibility.

    Estimable: The effort to implement load functionality is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Players can load and resume a previously saved game."

22. **As a player, I want to have restricted spaces on the board (e.g., holes or unbuildable tiles) so that gameplay is more challenging.**

    Independent: This story does not depend on other stories.

    Negotiable: The placement and rules for restricted spaces can be discussed.

    Valuable: It adds complexity and challenge.

    Estimable: The effort to implement restricted spaces is clear.

    Small: It can be completed in a single sprint.

    Testable: Acceptance criteria: "Restricted spaces prevent movement and building."

4. Domain Model

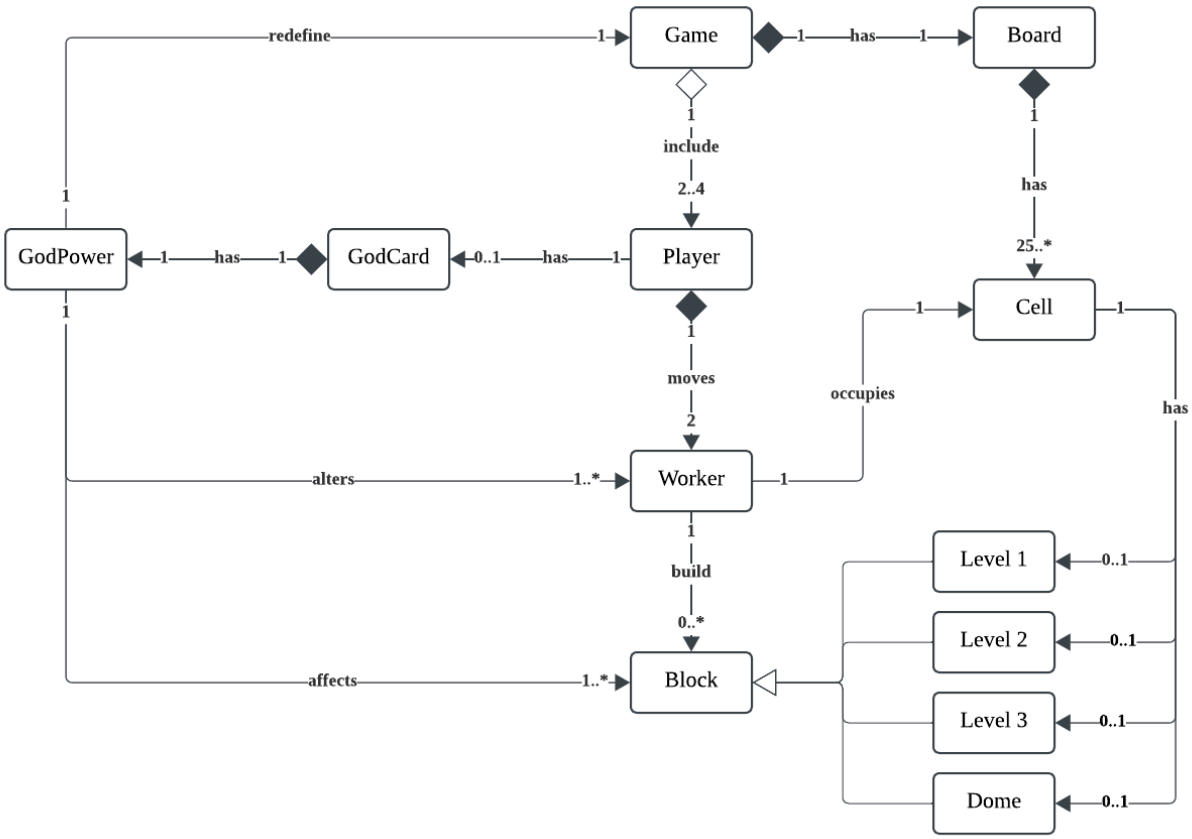


Figure 3.1 Domain Model Diagram of Santorini Board Game

4.1 Justification for Domain Entity

Domain Entity	Description
Game	Represents the overall structure of a Santorini match, managing the flow of play, players, and board state.
Board	The playing area consists of a grid of cells where players move their workers and construct buildings.
Cell	A single unit on the board that can hold workers and structures, forming the foundation for gameplay interactions.
Player	A participant in the game who controls two workers and competes to achieve victory through movement and building.
Worker	A piece controlled by a player that moves across the board and interacts with buildings to progress in the game.
Block	Represents different building levels that players construct during the game, affecting movement and win conditions.
Level1/Level2/ Level3/ Dome	Represents the different stages of construction in the game. Buildings start at **Level 1** and can be built up to **Level 3**, with a **Dome** placed on top to complete the structure and block movement.
GodCard	A special ability assigned to a player that modifies the standard rules and introduces unique strategic elements.
GodPower	The effect granted by a GodCard, defining how a player's abilities differ from the standard rules of the game.

4.2 Justification on Relationships between Domain

Core Game Structure

Relationship	Type and Justification	Description
Game (1) → (1) Board	**Composition** is used as the Board is created and managed by the Game. IT does not exist outside the game session.	A single game session is played on one board, which serves as the fixed playing area for all players and interactions.
Game (1) → (2..4) Player	**Aggregation**, as the Player can conceptually exist outside of a specific game session or even participate in multiple games. Game is responsible for managing players during the session, but if the game ends, the players as an entity is not inherently destroyed.	The game must have at least two players (for standard gameplay).
Player (1) → (2) Worker	**Composition**, as each worker is uniquely tied to a Player. A worker cannot exist independently without a player controlling it.	Maintain the basic game rule where each player controls exactly two workers, which they use to move, build structures, and attempt to win the game.
Board (1) → (25..*) Cell	**Composition** is best to represent the relationship as a Cell is no longer relevant if a board does not exist.	The board consists of a 5×5 grid, meaning it contains 25 cells, with each cell serving as a position for workers and buildings.
Worker (1) -> (1) Cell	**Association**, worker and cell can exist independently, a worker can move between different cells throughout the game.	A worker always occupies exactly one cell at any given time. This captures the core actions which is Move.
Worker (1) → (0..*) Block (builds)	**Association**, worker and block can exist independently, a worker can build any number of blocks, even none through the game.	Workers construct buildings by adding blocks to cells. This captures the core action of a Worker which is built.

Buildings and Levels

Relationship	Type and Justification	Description
Cell (1) → (0..1) Level 1/ Level 2/ Level 3/ Dome	**Association**, as cells and all blocks can exist independently.	Each Cell can contain at most one block at any level. Representing the stacking mechanism in Santorini, where buildings grow progressively.
Level 1/ Level 2/ Level 3/ Dome → Block	**Generalization**, as all Levels (1, 2, 3) and Dome is a type of Block	All different level blocks are a type of block and share a general structure.

God Powers and Cards

Relationship	Type and Justification	Description
Player (1) → (0..1) GodCard	Association, as a Player and GodCard are two independent components in the Game	Each Player may have a GodCard (or not if in a basic gameplay), which grants them special ability throughout the game session.
GodCard (0..1) → (1) GodPower	**Composition**, as a **GodPower** is an intrinsic part of a **GodCard**, meaning each card comes with a unique ability, and it cannot exist without its card.	Each **GodCard** grants a **specific ability** (GodPower) that affects gameplay.
GodPower (1) → (1..*) Worker	**Association,** since GodPowers can influence multiple workers, blocks or even the Game itself, but all the components (Block, Worker, Game) exist independently.	Some GodPowers (e.g., "Artemis") allow special movement actions for Workers. This models how GodPowers affect Worker actions but do not own them.
GodPower (1) → (1..*) Block		Certain GodPowers (e.g., "Demeter") allow special building rules, impacting how Blocks are placed.
GodPower (1) → (1) Game		GodPowers influence the setup of a game or even the winning condition of the game.

4.3 Special Choices and Rationale and Assumption

1. **Blocks is modelled as a Separate Entity instead of just Levels in a Cell**. A fixed integers for levels in Cell restrict the game in further expansion. With this implementation, it allows future flexibility for tracking individual blocks and adding special types of blocks if expansions introduce new mechanics. For example, one new God might have the ability to upgrade a certain block and grant some ability. This will be hard to achieve if block is just a number in a class.

2. Introduce a **separate class GodPower instead of just implementing the GodPower inside the GodCard**. The reason for this design choice is that some gods possess powers that can change or evolve depending on the circumstances or the game's progression.A separate `GodPower` class allows for flexibility, as it can handle different types of god powers that may vary across turns or be conditionally activated. This separation ensures that each god's ability can be dynamically adjusted and easily managed without overloading the `GodCard` class.

3. The domain model stated that a **board can have 25 or more cells**, this implementation supports the further implementation of board extension. However, I made an assumption that a minimum of 25 cells like a basic board are needed to act as a safeguard for core gameplay, while open-ended multiplicity decoupled spatial logix enable future creativity.

4. **Board has been modeled as part of the Game instance instead of making it a separate singleton entity.** This implementation allows multiple games to run independently, each with its own board state. If provide future flexibility to support replay ro saving games, each Game instance can have a self-contained board.

5 Lo-Fi Prototype

5.1 Introduction

The low-fidelity prototype for the Santorini board game project is designed to provide a clear user flow and interactive elements necessary for gameplay. The prototype includes essential components such as the ocean background, tiles, building blocks, builders, and God cards. The design ensures an intuitive and visually appealing interface while maintaining core gameplay mechanics.

5.2 Overview of Designed Pages

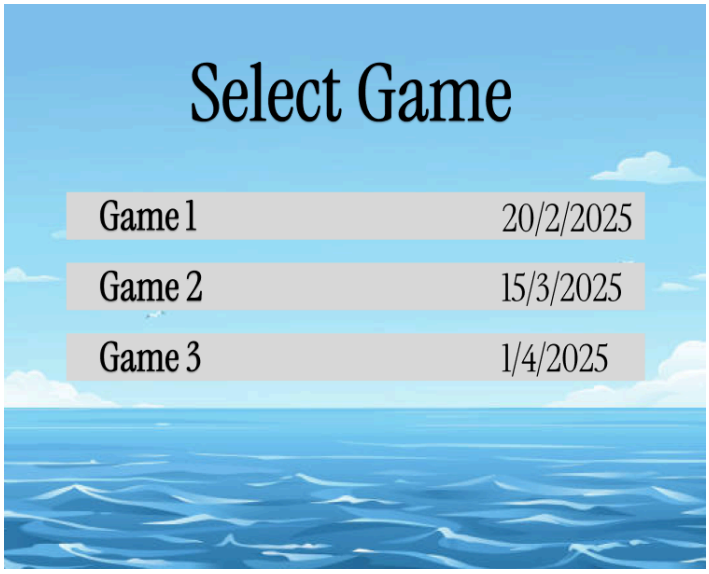
Main Menu Page

The main menu serves as the entry point for the game, offering three key options:

- **New Game** – Starts a new game session.
- **Load Game** – Allows players to resume a previously saved game.
- **Quit** – Exits the game.

This page ensures that players can easily navigate between different game states before starting a session.

Load Game Page



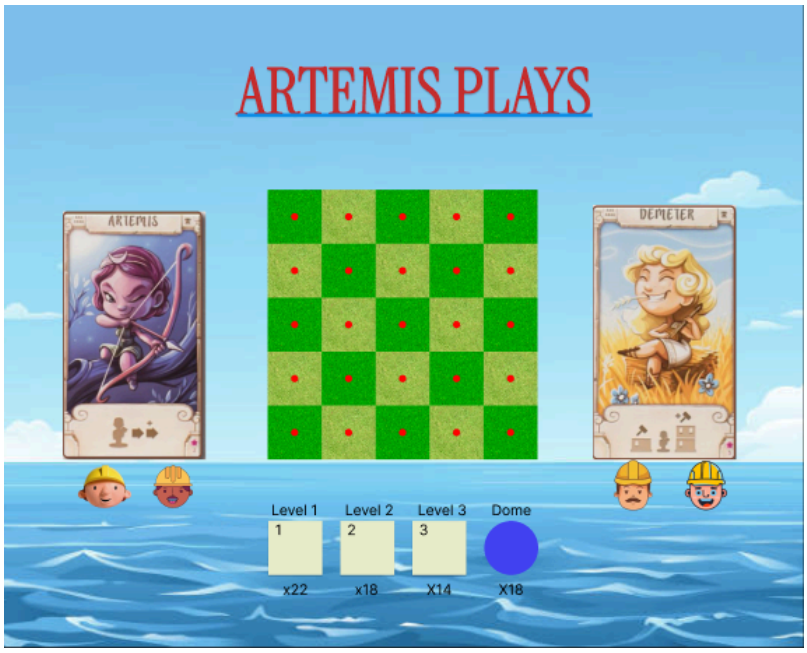
This page allows players to resume a previously saved game. It displays a list of saved games (e.g., Game 1, Game 2, Game 3) along with the date they were last played.

- Players can select a saved game to continue playing from where they left off.
- This feature enhances user experience by allowing flexibility and continuity in gameplay.

God Card Selection Page

This page enables players to choose a God card, which grants them unique abilities during gameplay. The interface provides a smooth selection process, ensuring that players understand their chosen God card's effects before entering the game.

Gameplay Setup Page (Placing Builders)



This page presents the game board where players place their builders. Key design elements include:

- Red dots on the board indicating valid positions for builder placement.
- A turn display at the top, ensuring players know whose turn it is.
- A building block pool at the bottom, showing available blocks and their quantities.

This step ensures that players properly position their builders before the game begins.

In-Game Action Selection Page

Once the game starts, players can interact with the board and perform actions. Key features:

- When a player selects a builder, red dots appear around it, indicating valid target tiles for movement or construction.

- Players execute their desired action by clicking on the corresponding red dot.

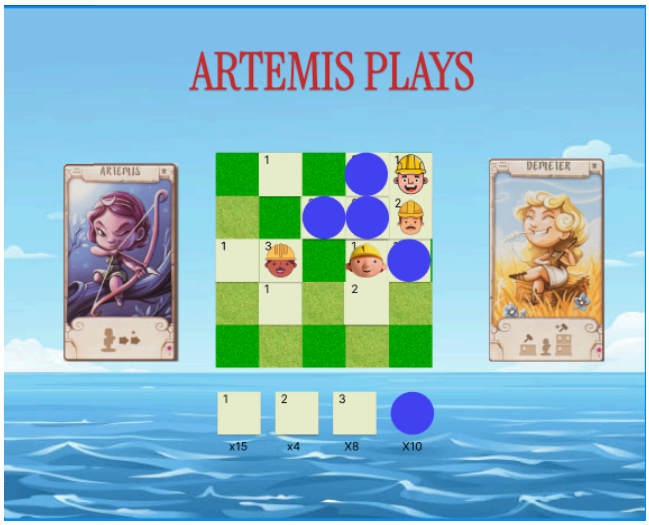
Mid-Game Progress Page

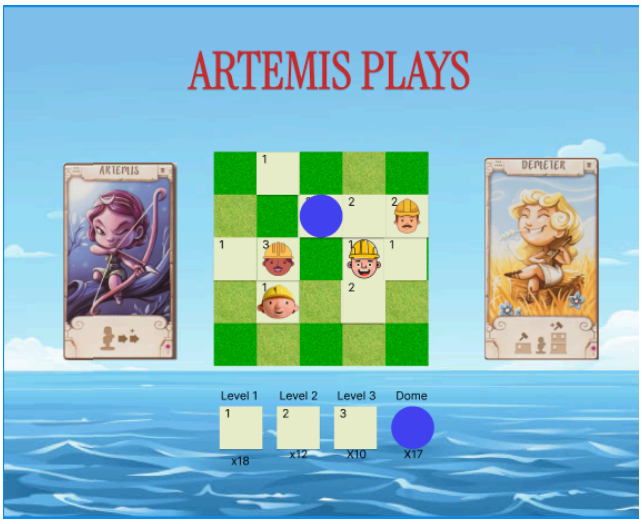
This page represents the game after several turns, showcasing the evolving board state with:

- Buildings, domes, and builders in different positions.
- Builders standing on different levels of buildings, highlighting the vertical movement mechanic.
- A real-time display of turn order and available blocks, ensuring that players can strategize effectively.

This visualizes how the board develops as the game progresses and helps players track their progress.

Winning and Losing Conditions



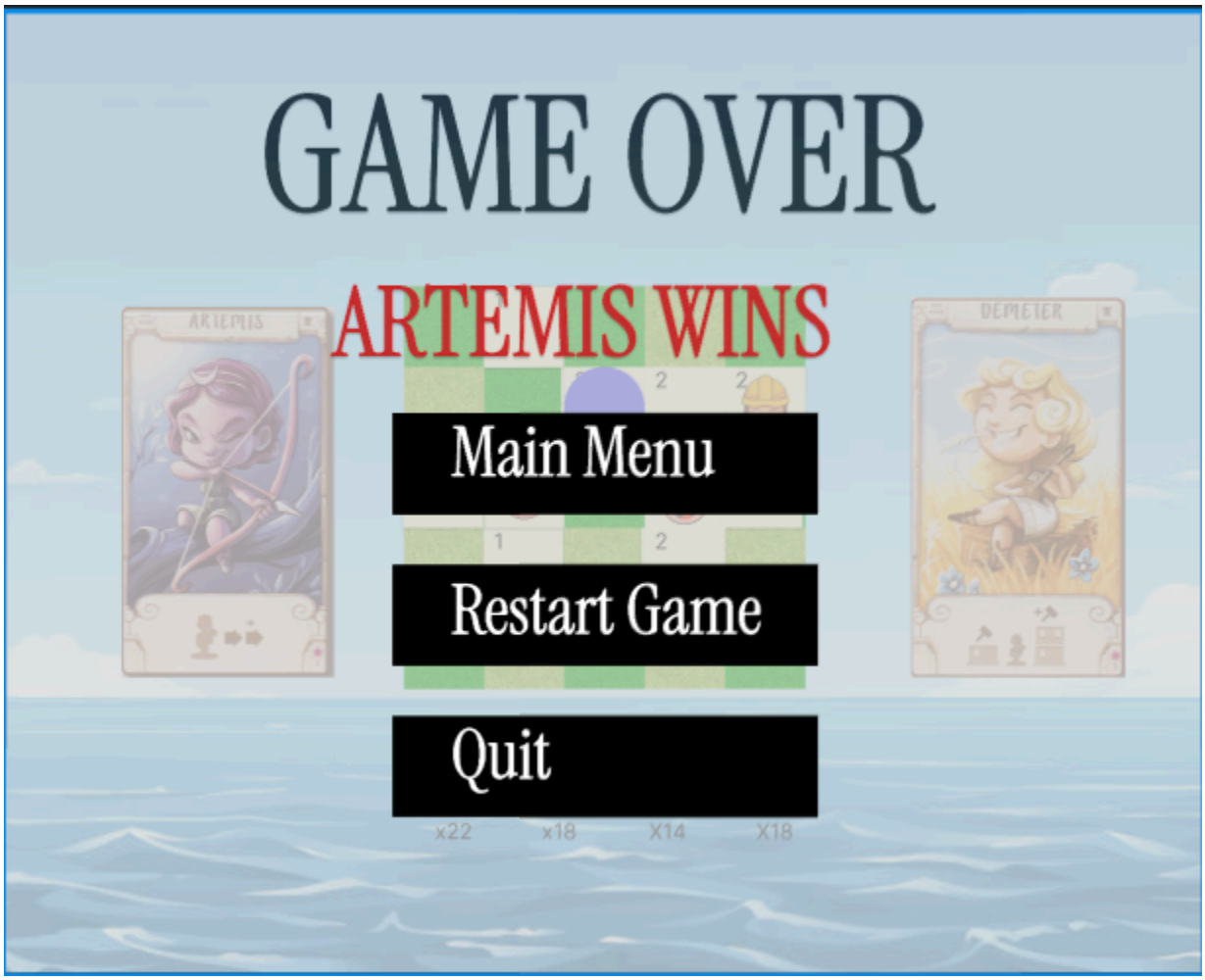


This page visually represents both the winning and losing conditions:

- Winning Condition (Left Side): A builder standing on level 3, signifying victory.
- Losing Condition (Right Side): Both builders are surrounded by domes, preventing any legal moves or constructions, resulting in a loss.

This clear distinction helps players quickly understand the endgame rules and how outcomes are determined.

Game Over Page



When a winning condition is met, this page appears with "GAME OVER" and the name of the winning God displayed at the top.

Players are given three choices:

- Main Menu – Returns to the main menu.
- Restart Game – Begins a new session immediately.
- Quit – Exits the game.

This screen provides a clear and simple conclusion to the game, ensuring players can seamlessly start a new match or exit.

Dynamic Board Layout Page

This page shows a bigger board layout compared to the normal sized board to satisfy the extensions to be implemented in the future.