FIT3077 / S1 2025
Sprint 3 Submission

By: Suhani Gadgil

Contributor Analytics

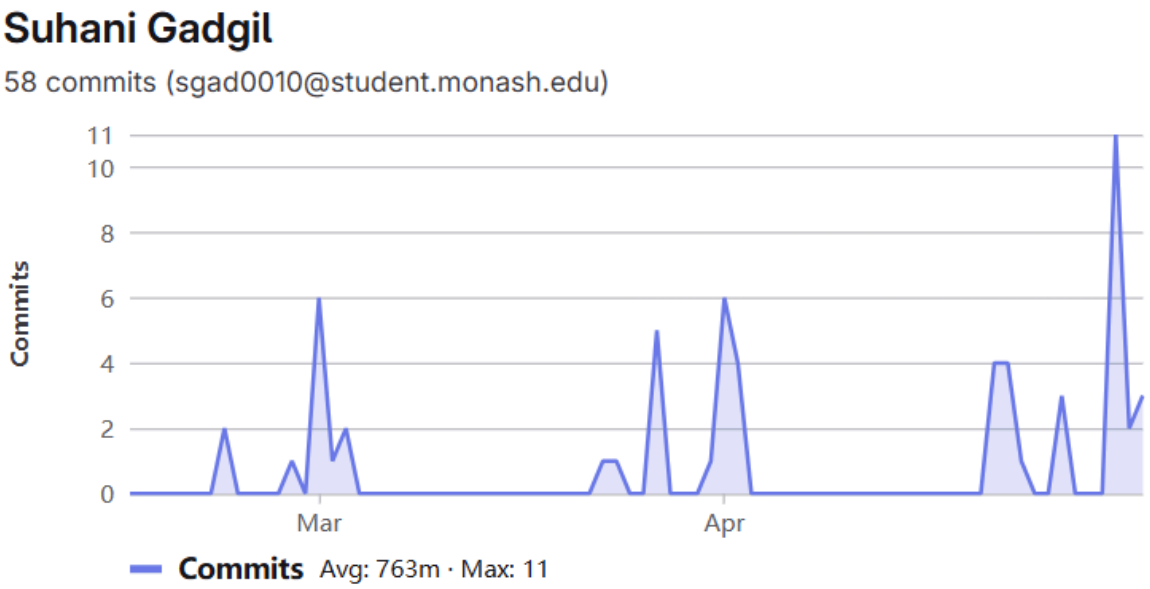

Figure 1: Contributor Analytics

Object-Oriented Design Deliverables

Class Diagram

The updated class diagram for the Santorini Game reflects key extensions added in Sprint 3: a new *GodCard(Zeus)*, a timer mechanism for player turns, and a forgiveness mechanic that allows undoing moves under controlled conditions. These additions require the introduction of new classes and relationships, as well as targeted modifications to existing classes to maintain a cohesive, extensible design.

Updated Class Diagram

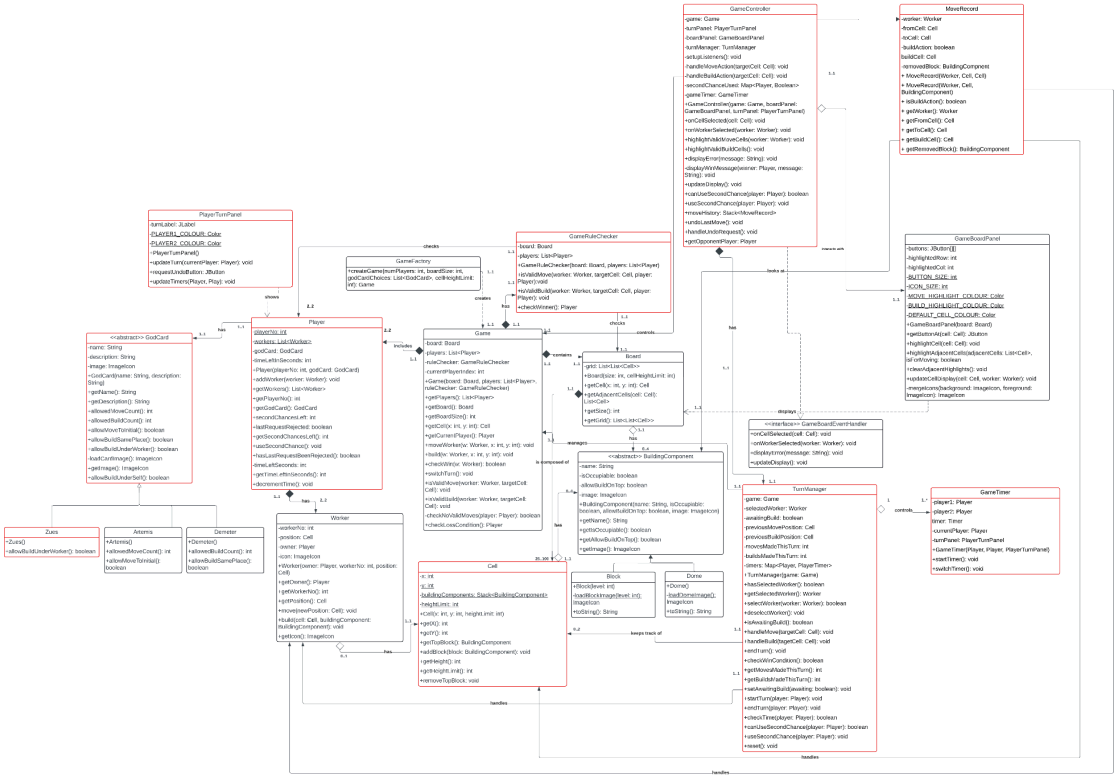

Figure 2: Whole updated UML Class Diagram

The class diagram maintains the core structure from Sprint 2 but introduces several significant enhancements:

- *GodCard* hierarchy extended with the new *Zeus* card.
- *TurnManager* now manages move history to support undo operations.
- *GameController* is augmented to handle player timer updates and forgive requests.
- *Player* is extended to track second chances and forgiveness state.
- *PlayerTimer* is introduced to manage individual player timers, ensuring fairness through visible countdowns.

New and Updated Classes

Zeus (New Class)

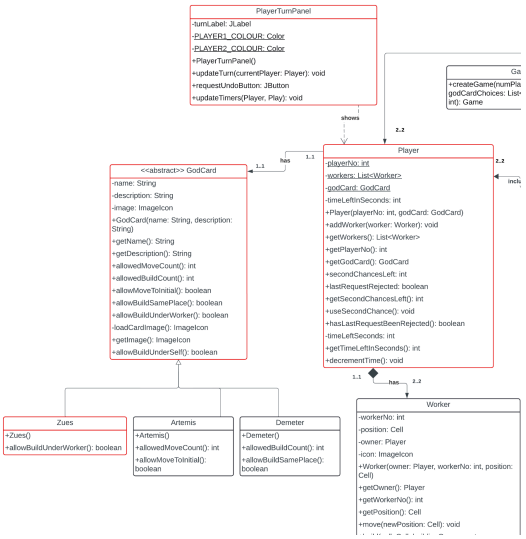

Figure 3: *Zeus* (new class)

A subclass of *GodCard* that implements special building behaviour, allowing a player to build under their worker. *Zeus* overrides *allowBuildUnderSelf()* returning *true*. This extension demonstrates the Open/Closed Principle (OCP) by enabling new functionality without modifying existing code.

Player (Updated)

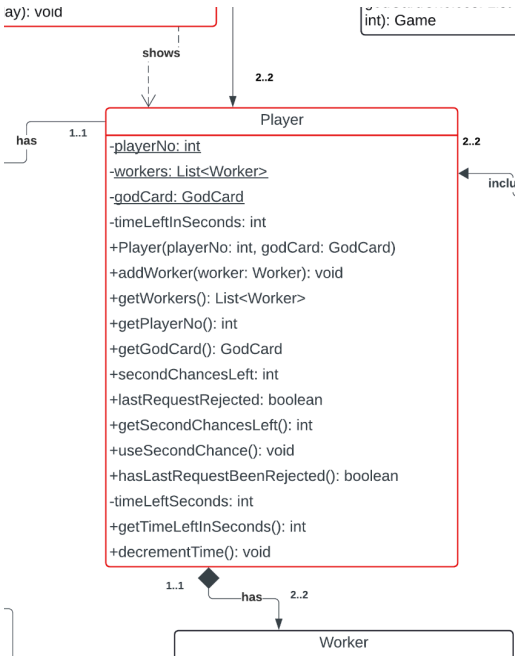

Figure 4: *Player* updated class

This class now tracks:

- *secondChancesLeft*: an integer representing the two available second chances per player.

- *lastRequestRejected*: a boolean flag enforcing acceptance on second forgiveness requests.
- Getters and setters to manage these properties, supporting the new forgiveness mechanic.

This enhancement integrates Human Values such as forgiveness and fairness, ensuring that players have limited but equitable opportunities to rectify mistakes.

GameTimer (New Class)

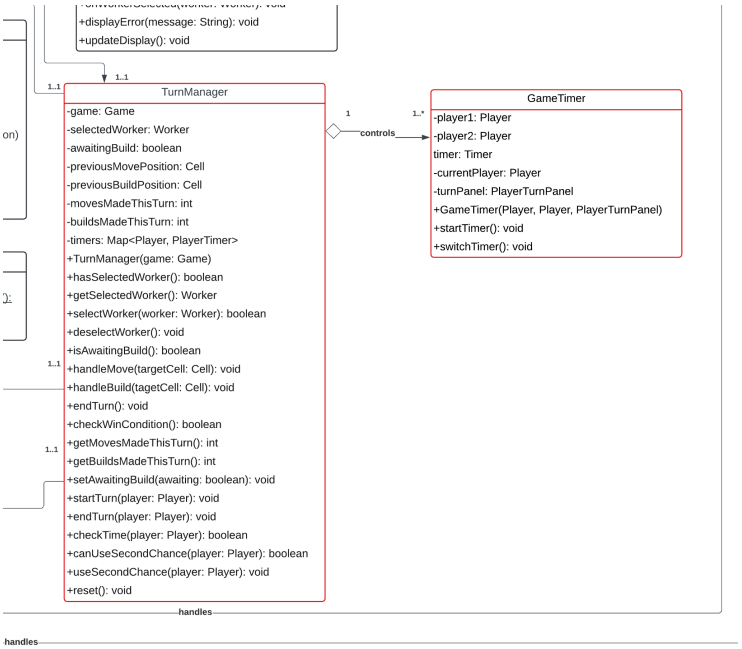

Figure 5: *GameTimer*

This class manages the timing system between two players:

- Attributes:
    - *player1*, *player2*: the two players being timed.
    - *Timer*: internal timer to handle countdown events.
    - *currentPlayer*: tracks whose turn it is.
    - *turnPanel*: reference to the PlayerTurnPanel to update the visible countdown.
- Methods:
    - *GameTimer*(Player player1, Player player2, PlayerTurnPanel turnPanel): constructor that sets up the players and UI panel.
    - *startTimer*(): begins the countdown timer for the current player.
    - *switchTimer*(): switches the timer to the other player at the end of a turn.

By centralising timer functionality inside GameTimer, the design achieves Separation of Concerns (SoC) and Single Responsibility Principle (SRP). Timing logic is decoupled from game logic, improving maintainability, reusability, and making it easier to test timing functionality independently from the rest of the game.

TurnManager (Updated)

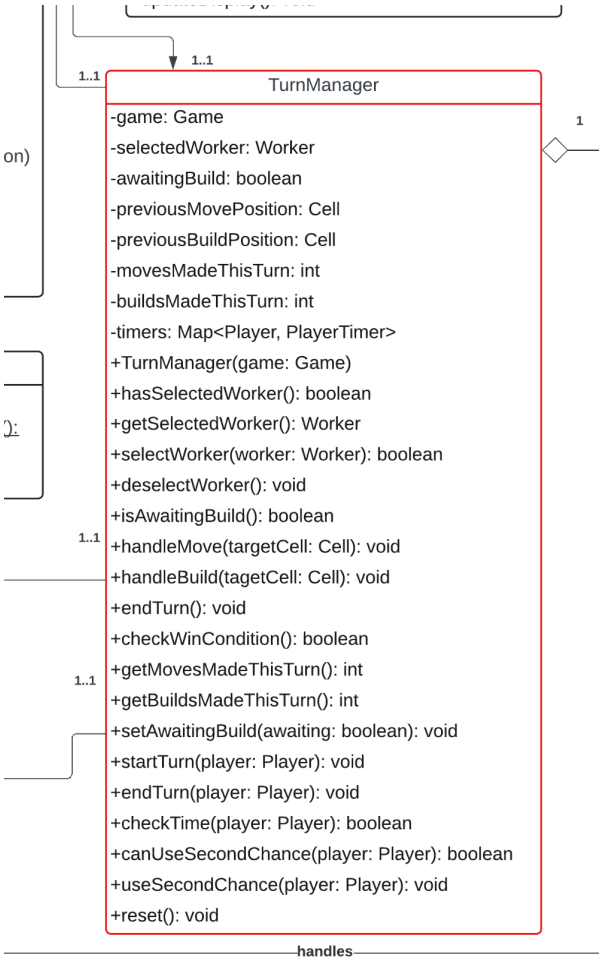

Figure 6: Updated *TurnManager*

This class manages:

- A *Stack<MoveRecords>* move history for supporting undo functionality
- Methods *recordMove()* and *undoLastMove()* to push and pop from the stack.

The Command Pattern principle is lightly reflected here via move recording and undoing, although not formalised as full Command objects due to sprint scope.

MoveRecord (New Class)

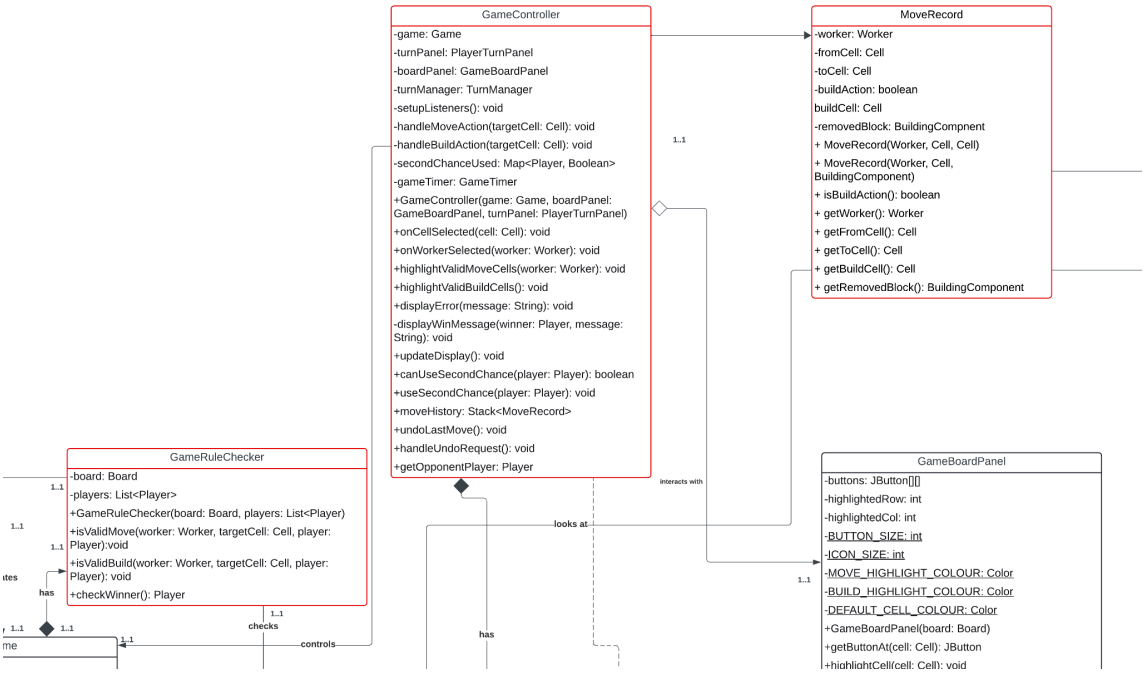

Figure 7: new class *MoveRecord*

This class stores data needed to undo a move:

- *Worker worker*
- *Cell previousPosition*
- *buildingComponent lastBuiltBlock*

This ensures minimal and focused responsibility (SRP) for managing undo information.

GameController (Updated)

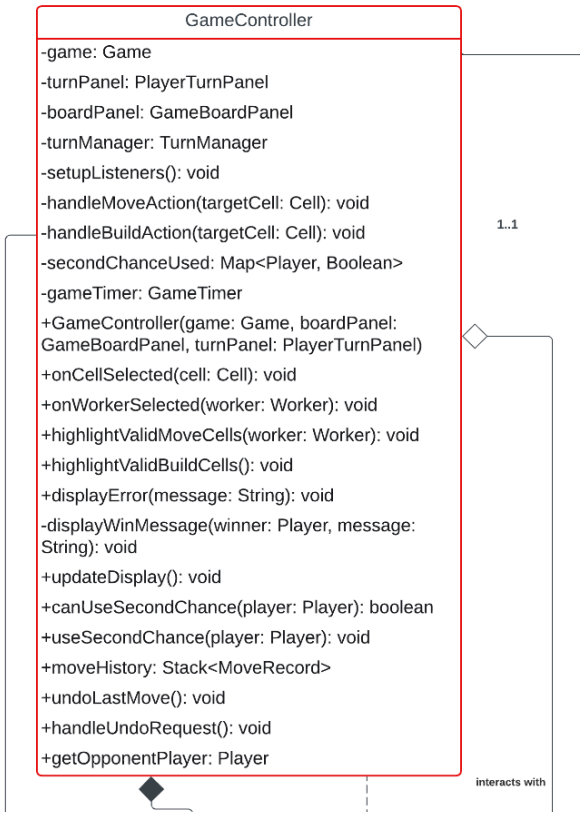

Figure 8: Updated *GameController*

This class is augmented to:

- Handle undo requests with confirmation dialogues
- Manage timer pausing and resuming between turns.
- Validate second chance rules and enforce forgiveness acceptance after rejection.

UI-related operations (popups, timers) remain separated from domain logic, ensuring low coupling.

CRC Cards

CRC (Class-Responsibility-Collaborator) cards were created to support the design and implementation of the new extensions introduced in Sprint 3. They document the key classes added to the system as part of the Timer, Undo (Forgiveness), and Zeus God Card features, helping clarify the allocation of responsibilities and collaborations during development.

CRC Card: *MoveRecord*

MoveRecord	***Subclass of the other class***
Store details of a single move of the *Worker* that moved	TurnManager
Store details of a single move of the *Cell* that moved to	TurnManager
Store details of a single move of the *Cell* built on	TurnManager
Store details of a single move of the *BuildingComponent* that was last built	TurnManager
Allow undo functionality by restoring the worker's previous position	TurnManager
Allow undo functionality by restoring the board state (remove last built block)	TurnManager

Figure 9: *MoveRecord* CRC Card

CRC Card: *GameController* (Modified)

GameController (modified)	***Subclass of the other class***
Control overall game flow	TurnManager
Detect when a plater loses	GameRulerChecker
Manage player turn transitions (move/build phases)	TurnManager, Board
Manage undo requests and apply forgiveness logic (Human Value: Forgiveness)	Player, MoveRecord
Allow players to request second chances and validate via handleUndoRequest()	Player, MoveRecord
Interact with MoveRecord stack to undo moves	MoveRecord
Update board state through Board and update visual display via PlayerTurnPanel	Board, PlayerTurnPanel
Update player timers and switch control between players after each turn	GameTimer, PlayerTurnPanel

Figure 10: *GameController* CRC Card

CRC Card: *GameTimer*

GameTimer	***Subclass of the other class***
Start and stop timers for each player	Player
Track remaining time per player	Player
Decrement time left each second and listen for time-out event	(Internal TimerTask)
Pause current player's timer and resume next player's timer when the turn switches	TurnManager
Declare a player loss if their time reaches zero	GameController
Reset timers and players' time left at the start of a new game	GameController

Figure 11: *GameTimer* CRC Card

CRC Card Justification

The CRC cards highlight the key classes that were newly introduced or modified to enable the successful implementation of Sprint 3 extensions. *MoveRecord* tracks move history to support undo functionality, *GameTimer* manages player timers and loss conditions based on timeouts, and *GameController* was extended to coordinate new forgiveness (undo) requests and timer updates. These CRC cards help ensure a clean separation of responsibilities and low coupling between new and existing classes.

Design Rationales

This design rationale explains the updates made to the Sprint 2 design to implement the Sprint 3 extensions, specifically the player timer, the undo (forgiveness) system, and the Zeus God Card power.

New Classes and Interfaces

MoveRecord

This class was added to track individual moves and builds. It stores the worker moved, the cell moved to, and the last building action, enabling the undo functionality. *MoveRecord* was created instead of overloading *TurnManager* to ensure separation of concerns. *TurnManager* focuses on tracking the current turn state, whereas *MoveRecord* archives historical actions.

GameTimer

A new class dedicated to managing player timers. It tracks, decrements, and switches timers, as well as declares losses when time runs out. Timer functionality was extracted from *Game* and *TurnManager* to avoid bloating their responsibilities and to ensure a clean, single-responsibility design.

GameController (Modified)

GameController was extended to coordinate second chance (undo) requests, interact with *MoveRecord* to revert moves, and manage timer control between players. Adding these responsibilities to *GameController* was appropriate because it already acted as the orchestrator between model and view, and the new forgiveness logic is closely tied to turn management.

Reassigned Responsibilities

In Sprint 2, the Game class handled turn transitions and could have been expanded to include timing and undo management. However, this would have violated the Single Responsibility Principle (SRP) and increased coupling.

Therefore:

- Timing logic was reassigned from Game to GameTimer.
- Undo management (including second chance control) was reassigned from *TurnManager* to *MoveRecord* and handled via *GameController*.

This ensures that each class remains cohesive and easier to maintain or extend.

Alternative Designs Considered

Embedding Timer Logic into Game:

Initially considered combining timers with *Game* or *TurnManager*. However, this would have mixed turn management and time management, increasing complexity and decreasing modularity.

Merging Undo into TurnManager:

Combining undo functionality directly into *TurnManager* was considered but rejected. Keeping history management and current turn management separate reduces complexity and prevents *TurnManager* from becoming a "God Class."

By maintaining modularity and adhering to SRP, the design ensures future extensions (e.g., adding new God Cards or different win conditions) can be integrated easily.

Design Patterns

There was no explicit implementation of a formal design pattern for these extensions. However, the following patterns were rejected:

- Command Pattern:
    - Often used for undo operations by encapsulating each action as a command. We rejected this due to the relatively simple undo mechanism (only need to track the last move and build). The Command Pattern would have introduced unnecessary complexity for a two-step undo history.
- State Pattern:
    - Could have modelled different game phases (move, build, request undo). However, this was unnecessary since the game has only two simple phases without complex transitions.
- Memento Pattern:
    - Provides the ability to capture and restore object states. We rejected it because full board snapshots would have been memory-inefficient. MoveRecord provides a more lightweight and efficient solution.

Human Value: Forgiveness

The human value chosen was Forgiveness — one of Schwartz's 58 basic human values. In this game, the power to grant a second chance lies with the opponent. A player may request to undo a move as an act of asking for forgiveness. Each player may be granted a maximum of two second chances over the course of the game.

The opponent can choose to accept or reject a second chance request. However, if they reject the first request, they are required to accept the next one. This ensures that forgiveness, while not guaranteed in every moment, cannot be denied indefinitely.
This mechanic reinforces mutual respect, empathy, and fairness, acknowledging human error while also giving players meaning.

Manifestation in the Game:

- Players can request a second chance after making a mistake.
- The opponent can deny the first request but must accept the second, ensuring forgiveness is ultimately granted.
- Each player is limited to two second chances, balancing fairness with compassion.

Changes Implemented:

- Added a second chance counter and last request rejection tracker to the Player class.
- Added logic in GameController to handle forgiveness requests, validating whether forgiveness must be granted based on the opponent's previous response.
- Introduced MoveRecord to support reverting moves and builds.
- Updated UI to show clear feedback about forgiveness requests and decisions.

This feature was designed not only to enhance gameplay fairness but also to encourage empathy and sportsmanship between players. By limiting denials, it ensures that players cannot withhold forgiveness indefinitely, reinforcing the value through both rules and player experience.

Reflection on Sprint 2 Design

In Sprint 3, the original game was extended by adding three new functionalities: Zeus God Card power, player timers and the human value of forgiveness for the final extension. The level of difficulty in incorporating these extensions varied based on the design choices made during Sprint 2.

Zeus God Card Power Extension

Implementing the Zeus God Card was relatively easy. Sprint 2 already has a polymorphic structure for *GodCards* with a clean abstract *GodCard* class and overrides like *Artemis* and *Demeter*. Adding a new subclass, *Zeus* and overriding the appropriate methods to allow building under the worker fir naturally into the existing design.

The prior use of inheritance and polymorphism greatly facilitated the extension. Zeus-specific behaviour was easily isolated in the subclass without requiring major changes to other parts of the system.

Since the Sprint 2 design was already modular and extensible for adding new God powers, there would be no need to make any significant changes to be ready for this extension.

Player Timer Extension

Adding the timer was moderately challenging. While Sprint 2's design had a well-defined *Game* and *TurnManager*, time management logic had not been anticipated. Initially, turn transitions were handled purely through logical state switches without regard for time, meaning there was no modular place to attach time tracking.

The absence of a dedicated turn management lifecycle and tight coupling between turn logic and the *GameController* made it slightly difficult to introduce a clean timer system. To overcome this, I introduced a separate *GameTimer* class to handle player-specific timers, ensuring that the *Game* and *GameController* classes were not overloaded with new responsibilities.

If I could go back to Sprint 2, I would have incorporated a basic TimeManger interface early in Sprint 2's design to anticipate timed turn transitions. A basic hook into the turn-switching logic would have made the timer easier to integrate.

Human Value: Forgiveness Extension

Adding a human value was the most difficult extension. Initially, I was stumped on any idea of how to go about it, but then the word "Forgiveness" stood out to me, and I knew that I wanted the extension to revolve around that idea. Slowly, I came up with the idea that each player should be allowed a maximum of two second changes each, as it is human to make mistakes and thus should be allowed to try again and learn from those mistakes. However, I did not want to "double-dip" into the undo function given as an option in the brief, so the difference for my human value extension is that permission must be granted by the opponent for a second chance to occur, making the opponent learn the value of forgiveness.

When implementing this extension, Sprint 2's design only tracked the current game state but did not maintain a history of moves. Without a move history or command pattern structure, there was no out-of-the-box way to revert a move or build action.

Moreover, the *TurnManager* class was heavily focused on enforcing move/build rules but did not store any past states. This made us realise that Sprint 2 lacked historical state management.

To solve this, we introduced a *MoveRecord* class, which captures every move and build step in a lightweight way. The *GameController* was also updated to interact with this move history for undo operations.

If I could redesign Sprint 2, I would have introduced a Command Pattern or a lightweight move history mechanism at the start. This would have significantly reduced the effort required to implement undo functionality in Sprint 3 and made state reversal more reliable and scalable.

Creating an Executable

Create a JAR file from the IntelliJ project

1. In IntelliJ IDEA, go to ***File*** > ***Project Structure***.
2. Select ***Artifacts*** on the left panel under project settings, and click the + button.
3. Choose ***JAR*** > ***From modules with dependencies***.
4. Select the main class that contains the entry point to the game.
5. Make sure that the ***extract to the target JAR*** is checked for the dependencies.
6. Set the output directory to a location of your choice.
7. Click ***OK*** to save the configuration.
8. Build the JAR by going to ***Build*** > ***Build Artifacts*** > ***FIT3077_A2:jar*** > ***Build***.

For Windows: Converting JAR to EXE [for our submitted executable]

Once you have created your **JAR** file:

1. Search on your internet browser and search for ***Launch4j*** to help convert the JAR file
2. Click on ***Download***, which will redirect you to a page called ***Launch4j Executable Wrapper Files***
3. Download the file that suits your operating system.
4. Once downloaded, open that ***.exe file***, and a tab will open
5. There are 2 required fields that must be filled:
    - Output file:
        i. Select the path where you want to generate an .exe file
            - NOTE: Please put .exe at the end of the path, otherwise it will show an error message.
    - Jar:
        i. Select the path where your .jar file is saved.
6. Click the ***gear icon*** at the top of the tab.
7. It will ask for an XML file to save to.
8. Your .exe file is now created.

Conclusion

Sprint 3 introduced significant extensions to the Santorini game, including player timers, an undo functionality designed around the human value of forgiveness, and the addition of the Zeus God Card. These enhancements refined the system’s modularity, incorporated historical state management for improved flexibility, and deepened the game's engagement through human-centred design. While some limitations from the Sprint 2 design, such as the absence of move history tracking, created challenges, careful updates allowed the system to adapt effectively. The process emphasised the importance of extensibility and modular design, ensuring that the architecture remains robust and adaptable for future iterations. Overall, the Sprint 3 extensions not only improved the technical foundation but also enriched the player experience by aligning gameplay with meaningful human values.