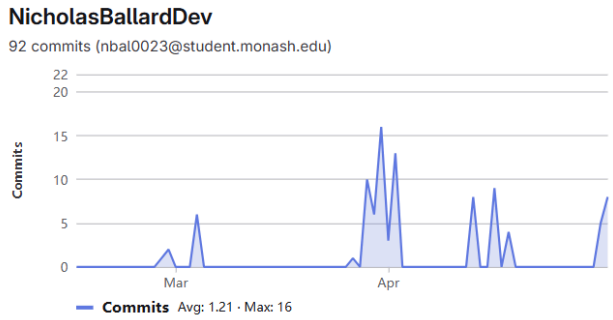
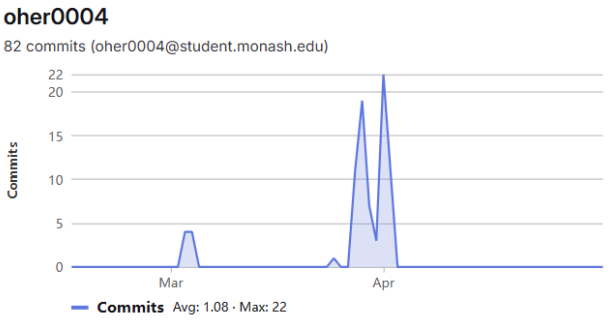
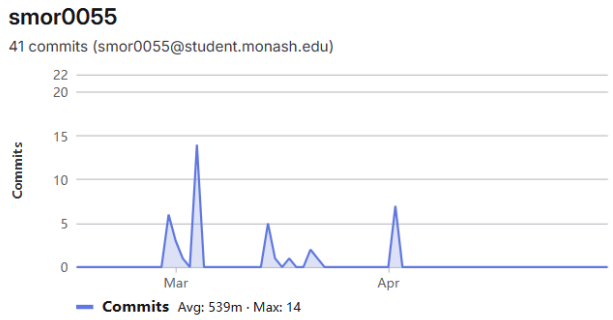
Nick Ballard Sprint 3 Submission
33861536

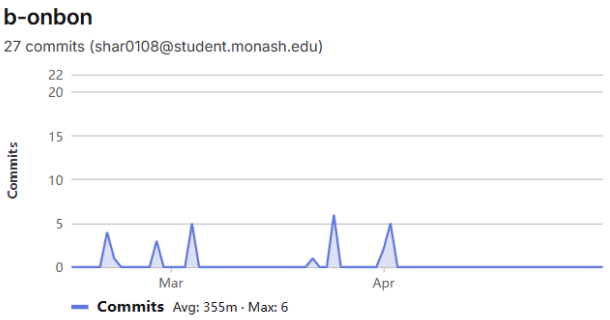
Table of Contents

Contributor Analytics









Extension Descriptions

God Card:

- Zeus: On your build, your worker may build a block under itself. Note: a worker building a 3rd level block underneath itself does not count as a win condition

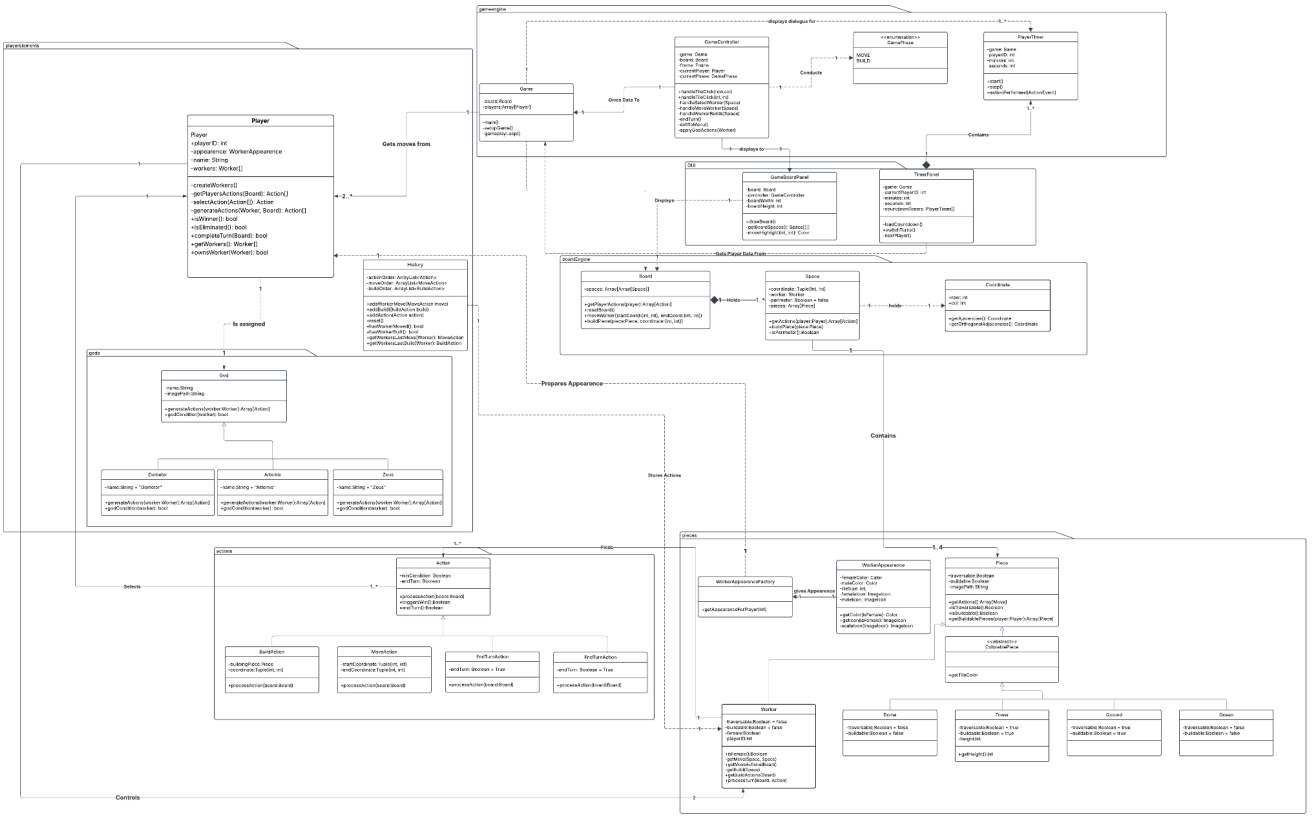
Extension From The List

- Timer: each player gets a fixed amount of time (e.g., 15 minutes) for “thinking” and executing their turns. Once a player has completed their turn, their clock pauses and the clock starts for the next player. If the clock of a player reaches zero, this player loses the game. The timers for each player need to be made visible in the user interface so that each player can see how much time they have left.

Self Defined Extension

- Human Value: Helpful
- At the start of the game each player gets a helpful token. On a player’s build they can choose to use this helpful token. When a helpful token is used, it allows the currently inactive worker (worker that did not move) to build for the active worker.

Object Oriented Design



Note: Look at Sprint 3 > docs > Sprint 3 Class Diagram.png for a clearer image

Design Changes and Rationale:

ColorablePiece:

ColorablePiece ensures that a piece's appearance/colour is stored within themselves. In the previous iteration of Santorini, the color of tiles was handled within the GameBoardPanel. However, the GameBoardPanel had too much responsibility, not only having to construct the board, but also having to handle the colours of each tile. That approach also did not allow for much extensibility, as if we needed to add a new type of piece, we would also need to edit the code within GameBoardPanel. With the addition of ColorablePiece I ensure that GameBoardPanel adheres to the single responsibility principle. This also meets the Liskov substitution principle as ColorablePiece can be used where piece has also been used instead.

GameController (Potential Rework):

GameController was a class I considered reworking, as it has many responsibilities. This includes not only selecting actions for the worker to complete, but also some game logic like detecting win conditions, and generating actions for the player. Not only does this violate the single responsibility principle, but also the open close principle, as if I want to add a new type

of action, I need to alter the existing methods of GameController for it to work. Whilst I could have distributed some of these functionalities to other classes such as the game or player classes, due to the scope of the project and time constraints, I decided it was more beneficial to focus on polishing the core mechanics of the game, especially since the current implementation of game controller did hinder the addition of the extensions I chose to implement.

CRC Cards

Zeus	
Generates worker actions	Action, Worker, Board
Knows the condition that should be met for activation	Worker, Space
Knows its name	
Knows the game phase it gets activated in	
Knows if it is optional	

This CRC card represents the class zeus. It is responsible for generating worker actions, making the condition that it can activate, knowing its name (so it can be displayed to the player), knowing the game phase it is activated in (so that it can communicate with GameController), and if the god ability is optional. Zeus mainly collaborates with worker, as it needs to generate the actions for the worker, however, it will also need to collaborate with Space and Board to select a spot for the worker to build.

PlayerTimer	
Start timer	

Stop Timer	
Performs countdown operation	ActionEvent

This CRC card represents the class player timer. It is responsible for being able to start and stop itself, as well as performing a countdown operation, which is used to display the time to the screen.

TimerPanel	
Loads the Countdown	PlayerTimer
Switch countdown timer between players	PlayerTimer

This CRC card represents the class TimerPanel. It is responsible to load the countdown, which is used to start the countdown for the current player. It is also responsible for switching the countdown timer between players, so that the countdown only goes when it is the current player's turn. It collaborates with PlayerTimer to achieve both of these responsibilities.

Design Rationale For Extensions:

Zeus:

Zeus was implemented using the template methods pattern that was already established. With this pattern, functionality such as displaying the god's name during popups were easy to implement and ensure that existing methods do not have to be altered when adding new gods in the future (open close principle).

PlayerTimer:

PlayerTimer was implemented to update the player time that gets displayed to the screen. PlayerTimer utilises the observer pattern as it needs to implement ActionListener. This allows it to observe timer events and count down whenever an event gets triggered. Because I want it to count down every second, it will be triggered every 1000 milliseconds. A potential design pattern that could have been implemented is the template method pattern. For this to work I could have made PlayerTimer an abstract class and construct methods that all timer classes could follow. This would allow for the addition of different types of timers, for

example a timer that goes at 1.5x speed. However, for simplicity I decided to not implement it that way, as player time countdowns should remain the same for all players and potential game modes.

TimerPanel:

The design pattern I utilised in the TimerPanel was the iterator pattern to iterate through each player's timer. Whilst I did not formally use an iterator, the nextPlayer function worked in a similar way to traverse the list of timers in order. The façade pattern was also used, as TimerPanel acts as a centralised place to handle some of the logic for the timers e.g. switching between players. Instead of holding the count down timers in a single data structure, an alternative approach would be to only allow for 2 timers to be activated, for player 1 and player 2. Whilst this approach would have been suitable, since I was not implementing extensions that alter the number of players, this also would prevent the possibility for the timer working for more players at once in the future.

Self-Directed Extension Rationale:

I chose to use helpful as my human value of choice. The value of being helpful promotes qualities such as collaboration, and a positive and supportive environment. From a societal context, being helpful can strengthen communities, and lead to a diverse range of people coming together. Without helpfulness, people would be very segmented, and we would progress slower as a society.

This idea is manifested within my extension of having 'helpful tokens'. This allows for inactive workers to build for the current active worker when the player consumes a helpful token. This conveys the theme of being helpful, as workers are now able to assist your team beyond their usual capabilities. For players to experience this, before the start of each build phase, players are given the opportunity to use a helpful token. If they select yes, building is now the responsibility of the currently inactive worker. This value of helpfulness is especially illustrated when you have chances to block your opponent from winning, without having to sacrifice the position of either of your workers.

The main logic that had to be changed within the game was just to check if each player has a helpful token at the start of each turn, as well as generating build actions for a player's currently inactive worker.

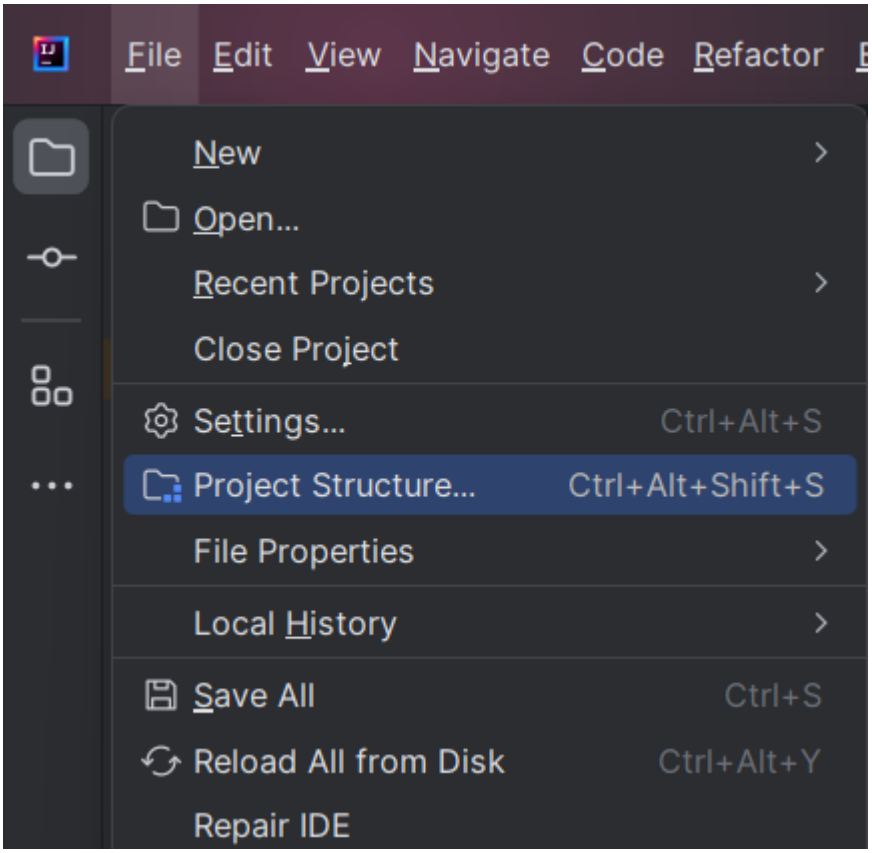
Executable Specifications

Description: The executable will be able to run the software prototype for my implementation of Santorini.

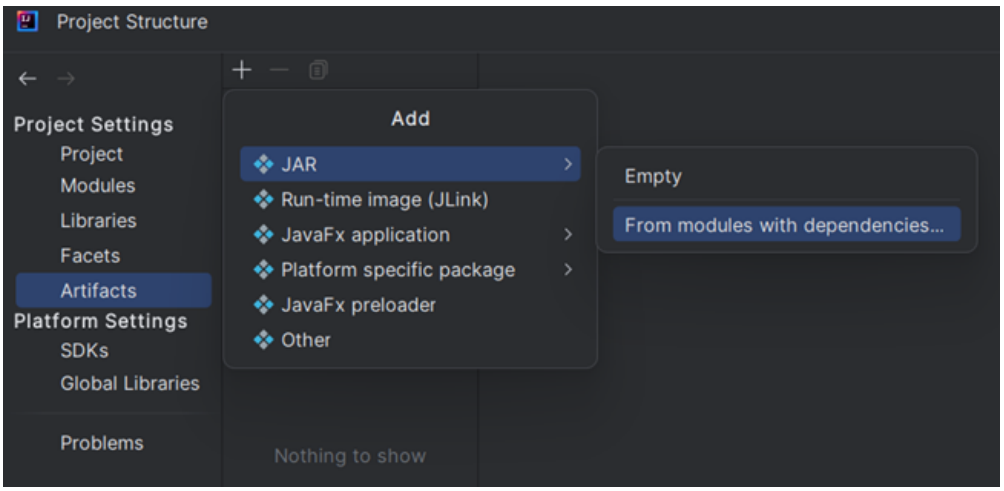
Platforms: Windows 11, Mac Sequoia

version: Oracle OpenJDK Version 24

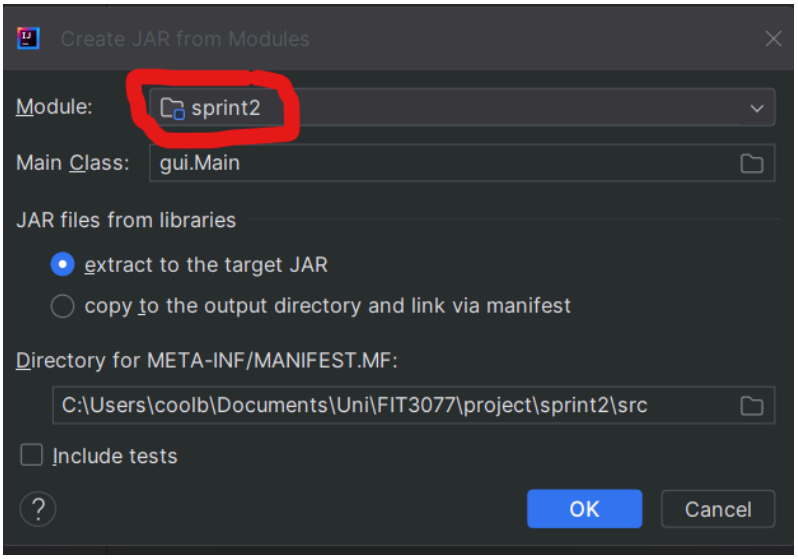
1. Open the folder ‘sprint2’ in IntelliJ.
2. Navigate to Project Settings by selecting File -> Project Structure.



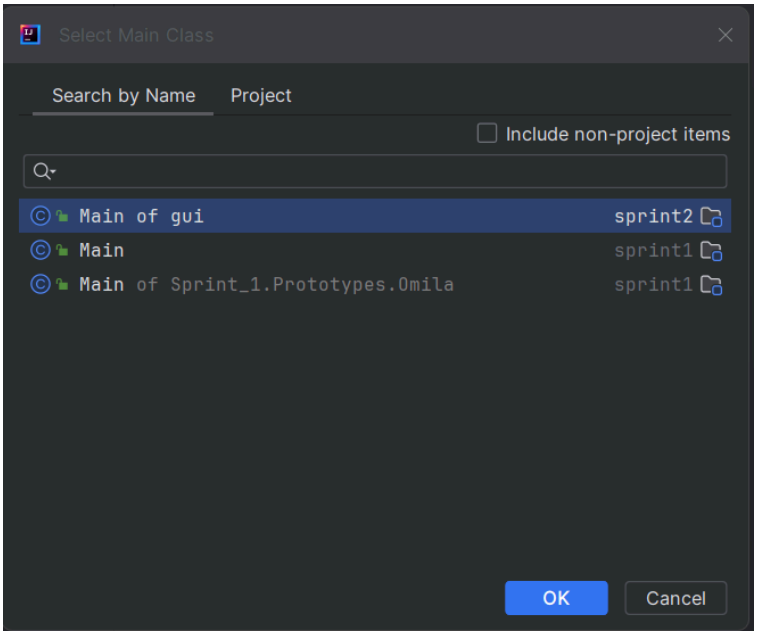
3. Click on 'Artifacts', then select the '+' icon to generate a JAR file.
4. Select 'from modules with dependencies'.



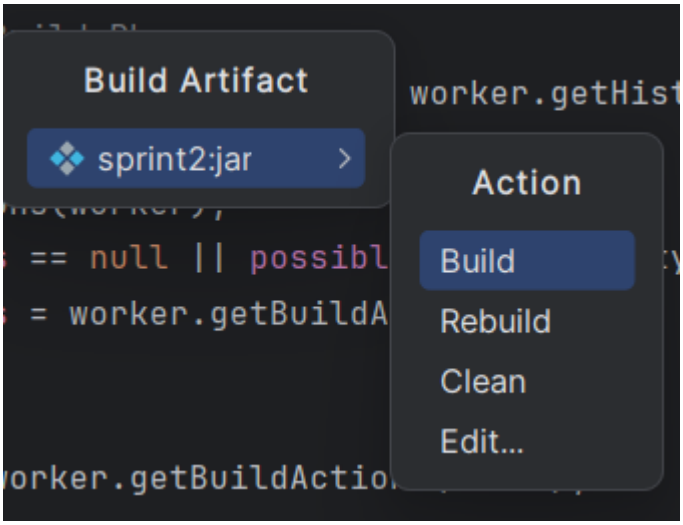
5. Select ‘sprint3’ as the module, then click on the 'Main Class' text box (The picture below says ‘sprint2’ but ensure it is set to ‘sprint3’ instead).



6. Select ‘Main of gui’ and press OK.



7. Press OK again to close the window, then select Build -> Build artifacts -> build.



8. Next, open your terminal and navigate to the jar file built. This file will be inside the 'out' folder (project\out\artifacts\sprint3_jar)

9. Next, type 'java -jar sprint3.jar' in the terminal.

10. If this doesn’t work, you can also try and double clicking on sprint3.jar

Sprint 3 Design Reflection

Zeus

Whilst implementing the Zeus god card looked relatively simple on the surface, there were some nuances that made it somewhat difficult to implement. For example, in the worker class from sprint 2, there was no simple way to get access to the space the worker was currently standing on. This made it difficult to check for the build height of the worker's current position (since we cannot build on a height of 3). To work around this, I made board as an input for worker and made a getSpace function for worker.

Another issue that I encountered was the code smell of having long methods. In the GameController class from sprint 2 there were several methods that were quite long. This made it difficult to navigate through the class, and make necessary changes, to allow the player to build on the space a worker was already standing on. Because the implementation of the function handleSelectWorker automatically deselected the worker when clicked again, I had problems allowing the player to choose to build underneath an already occupied space. Furthermore, because it was hard to navigate through several functions of GameController, it took more time than expected to get my implementation of Zeus to work.

Timer Extension

The timer function was quite straightforward to implement overall. The main difficulty that arose was that there was no way to tell the game to finish when a timer ended. This is because the victory screen functionality was embedded within the GameController class, which I was unable to give the PlayerTimer class access to. To fix this I needed to move that functionality into the game class, which the PlayerTimer could then communicate with. However, apart from this small issue, our code from Sprint 2 was easily extensible to fit the timer extension into our implementation of Santorini. This is because it provided a clear structure for how each screen was established, and how it fits into the context of the game and menus.

Helpful Token Extension

The Helpful Token Extension was simple to implement. Because we followed the open close principle for our Player and Worker class, by having well defined methods, as well as scope, I was able to add a simple capability to add a list of 'helpful actions' to the worker's already existing actions. To do this I simply created another function called generateHelpfulBuildActions, and then in the game controller class I was able to replace these actions with the worker's already existing actions. One thing I could change in the future is to remove some functionality out of the game controller class. This is because the GameController class can be considered as a "God Class" as it has many responsibilities. By being less dependent on this class to carry out game logic, I would be able to contain most of the worker and player related functionalities in the worker and player classes respectively.