Sprint 4 - Advanced Requirement (Undo & Upload)

Team Cabbage

Euan Lim, Zoe Tay Shiao Shuen and Lim Zheng Haur

Faculty of Information Technology, Monash University

FIT3077: Software Engineering Architecture & Design
Dr Jean-Guy Schneider

11 June 2023

Table of Contents

Demo Video Link

https://youtu.be/LJypLlupGvM

User stories

Revised User Stories for advanced requirement(s)

1. As a game player,
   I want to be able to undo a move that I made,
   so that I am able to return the state of the board to the previous state when I accidentally made a wrong move.

2. As a game board,
   I want to remember the past moves made,
   so that I can reset states when the players undo their move.

3. As a game player,
   I want to be able to upload a game state,
   so that I can replay the game from different scenarios.

4. As a game board,
   I want to be able to read game states,
   so that I can start games from different scenarios.

Revised User Stories

1. As a game player,
   I want to see my current number of pieces left on the board,
   so that I know if I am winning or losing.

2. As a game board,
   I want to know all available valid moves,
   so that I am able to end the game if a player has no valid moves left.

Unchanged user stories

1. As a game player,
   I want to see all the moves I made on the grid,
   so that I can plan for my next move.

2. As a game player,
   I want to see the opponent's moves,
   so that I can respond with my next move.

3. As a game player,
   I want to see the board
   so that I can place and move my pieces around.

4. As a game player,
   I want to place a piece on the board
   so that I can occupy a place on the board.

5. As a game board,
   I want to validate if a move is legal before execution,
   so that I can prevent illegal moves from occurring.

6. As a game board,
   I want to be able to detect when three in a row is formed,
   so that I can remove a piece from the opponent's.

7. As a game player,
   I want to have my piece removed off the board when my opponent gets 3 pieces in a row,
   so that I know how many pieces I have left.

8. As a game board,
   I want to alternate between both players making a move,
   so that they have equal turns to make a move.

9. As a game board,
   I want to be able to detect when a player has less than 3 pieces,
   so that I can end the game appropriately.

10. As a game board,
    I want to send an alert indicating the outcome of a match,
    so that players will be informed when the game has ended.

11. As a game token,
    I want to have all placed tokens displayed on the board,
    so that I know if an adjacent position is empty.

12. As a game token,
    I want to be able to move to an adjacent empty position,
    so that I would be able to change my position and try to form a three in row.

13. As a game token,
    I want to be able to jump over opponent tokens to an empty position,
    so that I would be able to try and form a three in row.

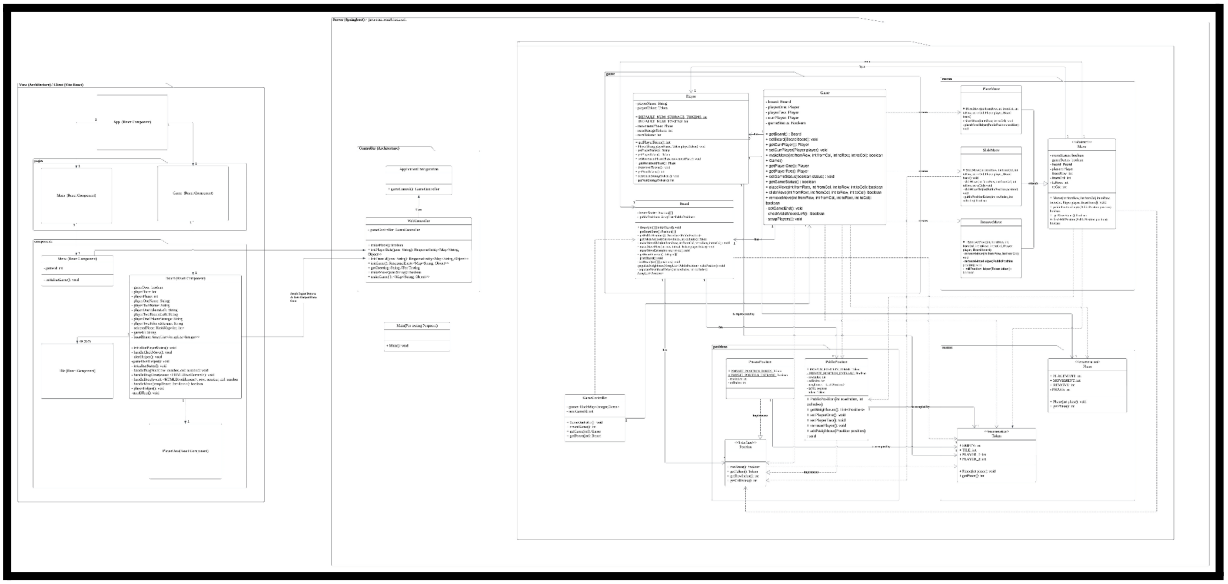
14. As a game board,
    I want to be able to detect when a player is unable to move any tokens,
    so that I can end the game.

15. As a game board,
    I want to be able to detect when a player has only 3 tokens remaining,
    so that I know if they should be allowed to jump over tokens.

UML Class Diagrams

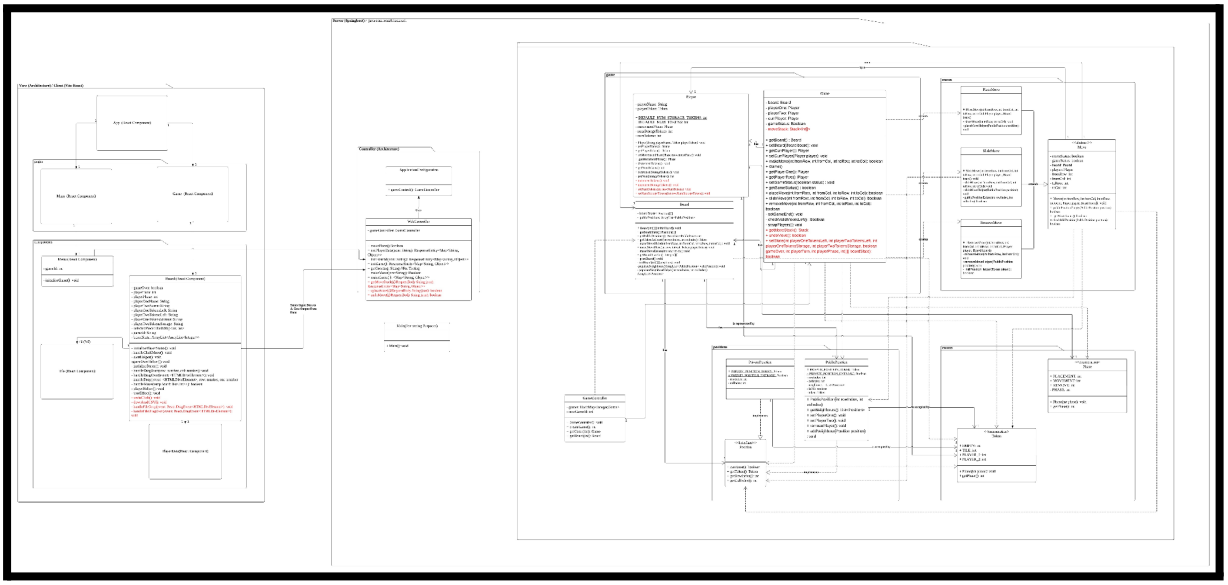
Past Class Diagram

Please refer to attached PDF documents **Sprint4_UMLClassDiagram_Previous.pdf in Git Repository for better clarity.*



Revised Class Diagram

Please refer to attached PDF documents **Sprint4_UMLClassDiagram_Latest.pdf in Git Repository for better clarity.*



Architecture and Design Rationales

We decided on Advanced Requirement B

Undo Feature Implementation

We have mainly used the stack concept to incorporate the undo functionality in the game. A stack is a form of abstract data type that practises the idea of last in first out. This means that the latest item stored, also known as pushed into the stack will be the first to be extracted, also known as popped out of it. For each of the player moves, the state of the board is pushed into the stack. Whenever an undo is requested, the latest previous state is popped out of the stack and reinstated into the board of the game, allowing players to return to the previous state of the board. Due to the nature of a stack, if an undo is requested continuously, the previous state of the board will be continuously popped from the stack and reinstated on the board.

The reason for the decision to use a stack to implement the undo moves functionality is that it ensures only the latest previous move is retrieved, and that there will be no moves to undo on an empty board. Within the stack, each move is stored in an array of integers containing the player who made the move, the movement phase, and the rows and columns of the origin and destination on the board. An alternative considered by the team to implement the undo functionality is storing game states as opposed to just the moves, which would entail the game board simply being restored to the previous state upon the clicking of the undo button. This alternative was decided against due to the fact that its implementation would be very space consuming.

Download & Upload Feature Implementation

This feature was implemented by extracting the existing data in the frontend and then parsing it into a CSV file format. The frontend makes a call to the backend requesting the move stack, however the backend sends a string of the serialisation of the stack instead of the actual stack resulting in the game data being able to download on the CSV file. When the game data is uploaded to the backend, the stack is deserialized and set to the games stack allowing for undo moves to still work on a game that is uploaded.

Architecture Changes

There were virtually zero architectural changes when implementing the new feature outside of adding a few more endpoints in the webcontroller and a few more getters and setters for the Player class, as well as a method within Game. One key aspect in retrospect is that we decided instead of having a game loop utilising a while loop. Due to the MVC based, we decided on a backend that listens for inputs and updates accordingly

As a result of not having a game loop sequence, there was very little refactoring required for implementing the ability to undo moves or upload a new game state, as this event could occur at any time due to the backend simply listening for changes or inputs on the frontend.

Implementation Reflection

Overall the implementation of the new advanced requirement took little redesign, instead only a few methods were implemented to make the function work, including the setter methods in the Player class. Had we decided on another way of implementing the game loop cycle as mentioned via a while loop that repeats until the game is over, the implementation would have been much more difficult. However upon consideration it is because we did not implement with a game loop, that we did not end up implementing the feature of an ai opponent for players to go against as a potential advanced requirement. So it is clear there are trade-offs in every design decision and we were fortunate that the undo and upload functions correlated well with an implementation that had the backend simply listening for inputs of a user.

Our team's choice of implementing the management of the game state has contributed greatly to the convenience of implementing the additional feature. In our implementation, the Board class constructor provides flexibility in terms of being able to create a board of any state. This is due to the initBoard method that initialises the game board according to a given matrix containing data of a board state. This has made the process of implementing the advance requirements rather convenient, since both acts of undoing moves and downloading/uploading the game state relies on modifying the state of the board. Hence, choosing to initialise a Board this way makes it easy to modify the board state in order to implement these new features. This is also present in the web controller endpoints, in that the frontend predominantly utilises 2 main initialization methods to update the View's board state and game data, of which the new feature was also able to use leading to fewer modifications being required.