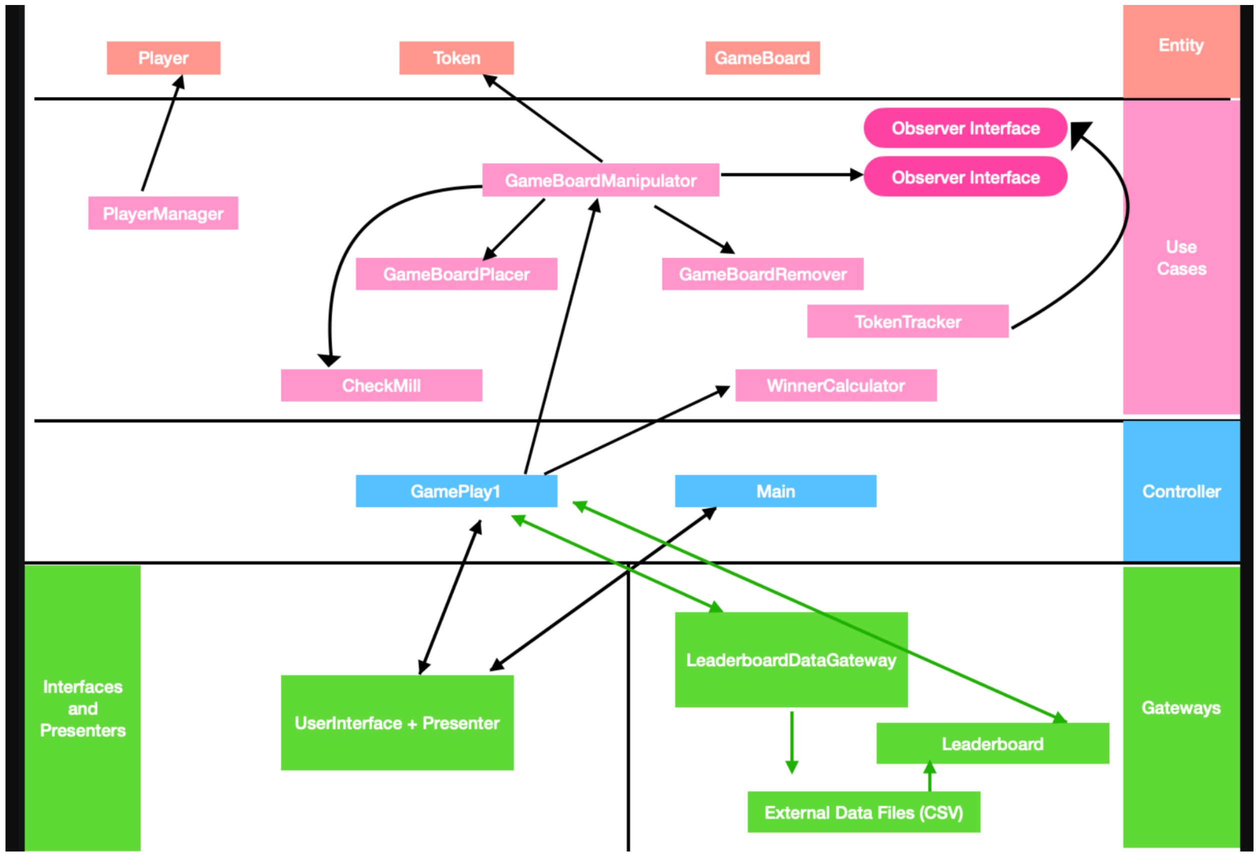
Updated Specification

Our main features that were added between Phase 1 and Phase 2 were the leaderboard, which kept track of players' names and how many times they had won the game after which their position on the leaderboard was determined. We also added a loading and saving component to the project where players were now able to start a game, to save and quit that game and restart the game without losing their previous progress. In addition, we also added a tutorial video to explain the rules of the game for players who may not be aware of how to play the game and it is in a step by step presentation format.

UML Diagram



Design Decisions and Patterns

<u>Observer:</u>

- Implemented for GameBoardManipulator and TokenTracker classes.
- GameBoardManipulator is Subject and TokenTracker is Observer.

- Whenever GameBoardManipulator adds or removes a token from the game board (the “action”), an observer (TokenTracker) is notified of such a change, and the appropriate update
- Method in TokenTracker is called to reflect new placement of tokens
- Ensures dynamic updating of token positions as game proceeds

Facade

- Implemented for GameBoardManipulator, to avoid violation of single responsibility
- Remove and add token functionalities are independent and subject to change
- So, introduce separate classes to carry out these individual functionalities
- Have GameBoardManipulator store instances of these classes as part of facade, to carry out these actions

Iterator

- Overlooked, could have been implemented for GameBoard
- To iterate over all game board positions, GameBoard holds an array of all positions
- Get an array then loop over it
- Alternatively: implement iterator, so we can loop directly over a game board object instead of having to get this array then loop over it

Testing

- Most major use cases are tested, since it is easy to test game logic
- However, controller and GUI classes especially hard to test
- Controller requires user input sent from higher layers to work
- GUI classes can only be tested by “trial and error” as we see the GUI
- Most testing focused on making sure game logic works correctly

Clean Architecture and SOLID Design Principles

1. Single Responsibility:

   “Every class should have a single responsibility”.

   For example:

   - GameBoardManipulator: adds/removes tokens from gameboard

   - implemented facade design pattern so adding/removing done by separate single responsibility classes

   - WinnerCalculator: determines who won at end of game - PlayerManager: Instantiates and stores Players in the game

2. Open Closed Principle:

   Example:

   - GameBoardManager: To add new functionality, simply add a new class to the facade. The existing code doesn't really affect new features.

3. Liskov Substitution Principle:

   Not really demonstrated in our project, since we didn't write abstract classes.

4. Interface Segregation:

   In our code, we have 3 interfaces, namely:

   - DataAdaptor.java:

   - Observer.java: Updates the position when a player adds/removes a token

   - Subject.java: Updates observer about the addition/removal of the token

   Only one class implements each interface, but interface methods are just enough to accomplish desired tasks (ex: updating gameboard status).

5. Dependency inversion

   Might not apply to our code since we did not introduce an abstraction layer between low-level and high-level classes.

Packaging Strategies

The packages were made according to Controller, Entity, Exceptions, Gateways, Interfaces, UseCases and UserInterfaces. The controller package contains main and gameplay1. The entity package contains GameBoard, player and token entity classes. The exceptions package contains all the exceptions that may be raised throughout the game for invalid moves. The Gateways package contains LeaderBoardDataGateway and the data folder that contains all the data used for the leaderboard. The UseCases package contains classes such as Winner Calculator, GameBoardManipulator, GameBoardPlacer etc. Finally, the UserInterface package contains all the GUI classes related to the login panel, token button, leaderboard panel etc. Generally, the UI gets the data and passes the information to the Controller package which then passes the information through the UseCases. The UseCases then takes this information and uses it to control the Entity classes after which an output is delivered.

Summary of what each group member has been working on:

Jason - Worked on implementing a new class, TokenTracker, for storing a GameBoard and keeping track of where the player Tokens are on the GameBoard. Also implementing the Observer pattern between GameBoardManipulator and TokenTracker, so TockenTracker has its map of player tokens updates whenever Manipulator makes a move on the Gameboard. Pull request: Significant because manually merged two conflicting but important branches, so we could get the code for my Observer pattern and Felipe's GUI for the game into our final project.
https://github.com/CSC207-UofT/course-project-group-053/pull/38

Mary - Worked on packaging and made an abstract parent class for gameplay1 and gameplay2 in order to make the sliding phase. Also implemented the GUI into the main branch. Deleted unused methods and renamed variables to be well recognized. Significance of pull request: it was the first step in linking the game algorithm to the GUI
https://github.com/CSC207-UofT/course-project-group-053

Aliza - Worked with felipe on implementing LeaderBoard and its csv file. Also worked on creating the accessibility report. Pull request: significant because it loaded all the data in order for the leaderboard to be displayed on the GUI and keep track of the players wins. Pull request: Shows the leaderboard implementation that loads and saves the results which is an integral part of the leaderboard.
https://github.com/CSC207-UofT/course-project-group-053/pull/35

Aalina - Worked on the design document for Phase 2. Pull request: Significant because it was setting the basis for GameBoardMap to be used to implement an observer.
https://github.com/CSC207-UofT/course-project-group-053/actions/runs/1463531923/workflow

Felipe - Worked on making a tutorial video and adding into the GUI as well as refining the visuals of the GUI (allow player to resize the game window, nicer looking buttons etc.). Moreover, fixed the issue where the game did not recognize the special occasion where a user had to remove an opponent's token while all the removable tokens were part of a mill. Worked with Aliza to implement the leaderboard to keep track of every player's number of wins. Updated the GUI to interact with LeaderboardDataGetaway and display the top 10 players. Finally, made the PlayerManager use case class so that the controller GamePlay1 could have access to all the necessary information without using the entity classes. Pull request: Significant because it consisted of most of the above mentioned changes thereby having a considerable impact on the overall code and game. https://github.com/CSC207-UofT/course-project-group-053/pull/34

Saksham - Worked on loading and saving the game so that players could exit the game while all the information of the game was still retained in order for them to restart the game if they chose to exit. Also worked on implementing this into the GUI by creating a button for saving the game and then reloading it again. Pull request: significant because it made all of the above possible.
https://github.com/CSC207-UofT/course-project-group-053/actions/runs/1547516073

Significance: the GUI for saving the game.