FIT3077 Software Engineering: Architecture and Design

Assignment 4: Sprint Four

MA_Friday2pm_Team5

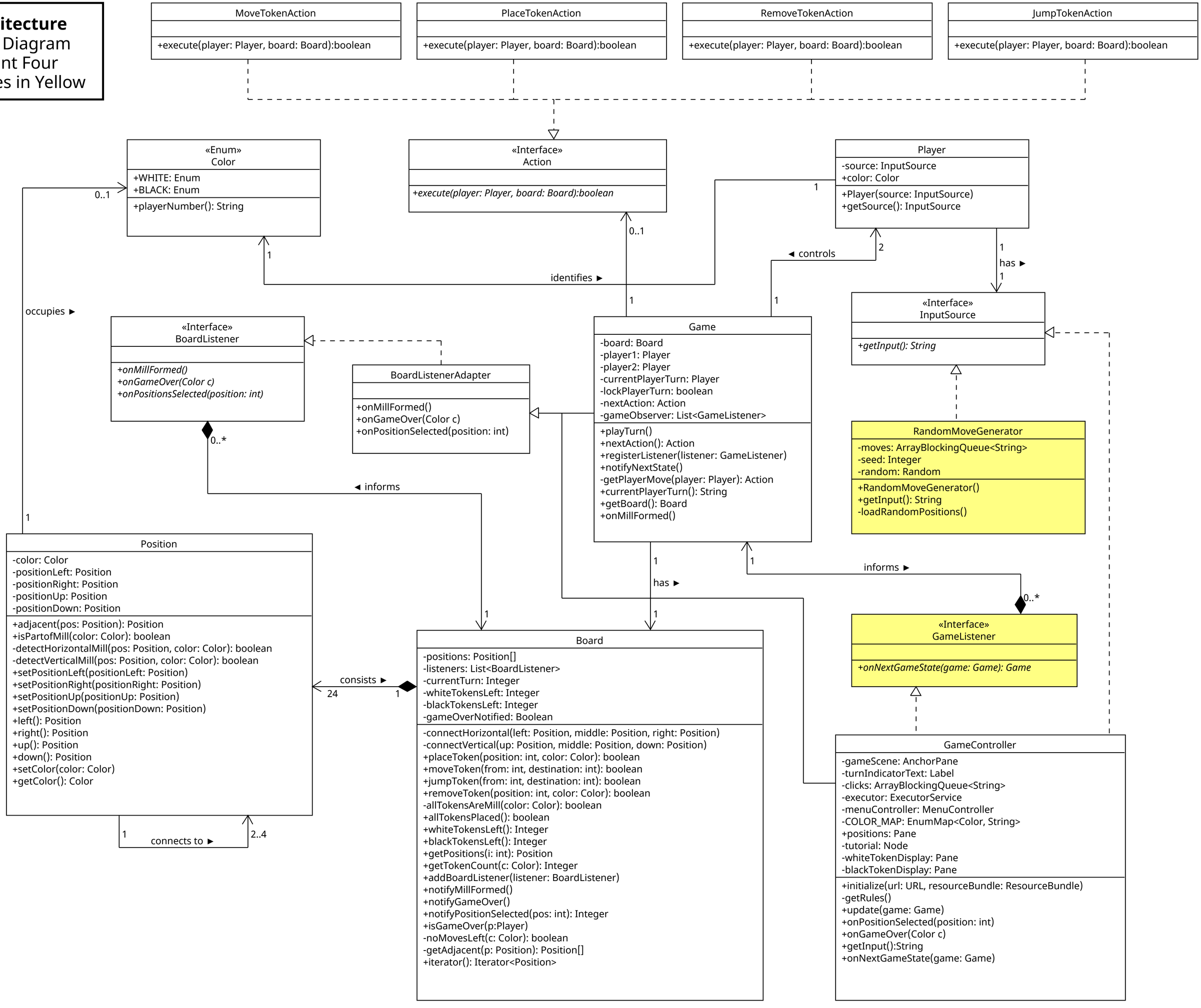
User Stories

For the advanced requirements, our team chose Advanced Requirements C:
“ A single player may play against the computer, where the computer will randomly play a move among all of the currently valid moves for the computer, or any other set of heuristics of your choice”

Advanced requirements:

1. As a human player, I want to be able to play against a computer player with a chosen colour so that I can play 9MM when another human is not available.
2. ~~As a computer player I want to randomly play moves that are legal so that the game is unpredictable for the human player.~~ (Removed)
3. As a computer player, I want to randomly place tokens on board in the first phase so that I can capture tokens if I form a mill.
4. As a computer player, I want to randomly move tokens to a connected position in the midgame so that I can capture the opponents’ tokens if I form a mill.
5. As a computer player, I want to randomly move tokens to any position when I have less than 3 tokens so that I can capture the opponents’ tokens if I form a mill.

Architecture
Class Diagram
Sprint Four
Changes in Yellow



Design Rationale

Explain why you have designed the architecture for the advanced requirement the way you have.

For the advanced feature, the team chose advanced requirement c:
A single player may play against the computer, where the computer will randomly play a move among all of the currently valid moves for the computer or any other set of heuristics of your choice.

This feature requires the team to implement a computer player that can play random moves against the player. Since the game is already built on top of the Model-View Controller architecture, it is set up in such a way that the game model is already agnostic to the type of the input source as long as it is implemented through a unified interface. As such, the scope of change is mainly limited to the Controller to find a way to generate a stream of random inputs. However, the team still has to decide on how a random move is generated, and the two reasonable approaches to implement the feature are outlined in the table below.

<table>
<tr>
<td>Approach A: Get all the legal moves from the current game state and randomly select a valid move to play. The game model is guaranteed to play the proposed move and proceed to the next game state.

Advantages
● More efficient compared to the alternative approach as the generated input is guaranteed to be valid.

Disadvantages
● The game model does not support checking if a proposed move is valid without altering its state. It takes a huge effort to rewrite the model to support such a feature.</td>
<td>Approach B: Generate sequence of random inputs in the input stream and let the game model reject inputs until it finds a valid move. There are only a limited number of moves that can be made, and the random input generator will inevitably find a valid move.

Advantages
● Supported by existing architecture. The model can retrieve the random inputs from the controller until a valid move is found.
● Ease of implementation. The scope of change is limited to the controller.

Disadvantages
● Inefficient approach. The game model has to constantly check the random inputs until a move is valid.</td>
</tr>
<tr>
<td colspan="2">Decision
The team decided to proceed with approach B. While the approach may have been more inefficient, the alternative requires rewriting parts of the game model is infeasible with the tight Sprint deadlines that we have. The team expects the inefficiency to not be an issue to our software as the valid input can still be generated instantaneously and adheres to the MVC architecture.</td>
</tr>
</table>

From the player's perspective, both approaches will still yield the same result as the computer will immediately play a move after the player's turn.

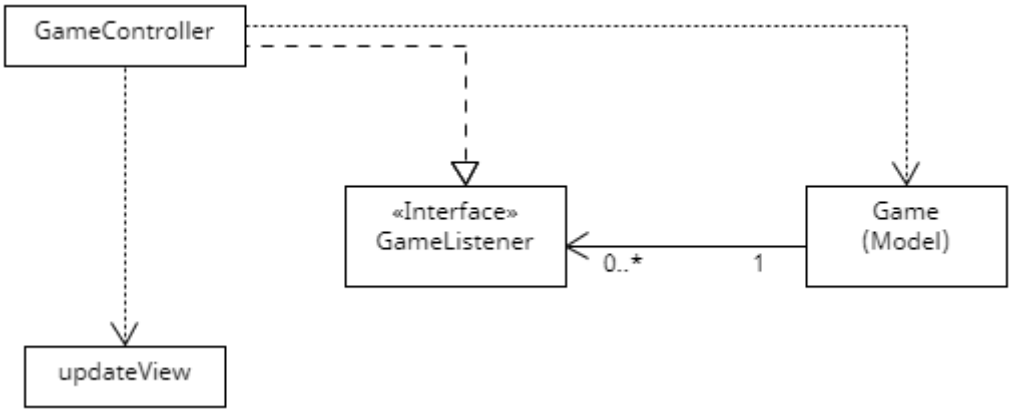
Explain why you have revised the architecture, if you have revised it.
When implementing the computer moves the team came across a problem where the game model needs to look for the computer player's input source. Originally, the game is built on a passive MVC since the game only reaches a new state after a human input. As the passive MVC is not independent from the game controller, there is no trigger for the game model when it is the computer's turn to move. The team can overcome this by changing the architecture into an active MVC, or alternatively notify the computer when the player has made a move.

Approach A: Change the passive MVC architecture to active MVC. Introduce a new GameListener interface to use the observer pattern for the random input source. Advantages ● Model is decoupled from the controller. ● Avoid procedural style approach compared to the alternative Disadvantages ● Introduces more levels of indirection with a new Listener class. The complexity of the architecture increases.	**Approach B:** Notify the computer player to make a move after the player's input source detects input. Advantages ● Straightforward implementation. Requires no change in the architecture. Disadvantages ● Human player's input source is coupled with computer player's as it needs to notify them. ● Not taking full advantage of Object Oriented concepts with many checks (Don't look for computer moves if it is multiplayer mode).
Decision The team decided to go with approach A, where the architecture of the software is revised to be an active MVC from a passive MVC. As the advanced feature of the game requires the game model to report a change of state after a valid move, active MVC brings its own benefit of being independent from the controller as it runs in parallel with it. That way, the model will automatically search for the computer player's move upon its turn. The setback of the approach introduces new indirection and classes as we utilise the observer pattern to notify the controller, but it is a compromisable approach for the team to avoid coupling between the human player with computer input and adhere to good object oriented practices.	

Explain when your advanced feature is finalised, how easy/difficult it is to implement? Was it due to good design practice/patterns you have applied in earlier sprints? Or was it difficult for the advanced feature?

The approach to implementing the advanced feature is very clear to the team after it is selected. The Model-View Controller architecture implemented within the software clearly segregates the responsibilities of each component, and for the team to implement computer moves we have a clear idea early in the Sprint that the scope of change lies mainly in the Controller. The approach taken by the team is to generate a sequence of random inputs in the Controller. However, as shown in the above section the existing architecture was

originally written in passive MVC which the team has to convert it to an active one to utilise good object-oriented practices. Aside from the change of the game model to introduce the observer pattern, the game model remains largely intact as it is independent from the type of player making the moves and the view component still renders the same game state. The figure below illustrates the MVC architecture that allows the advanced feature to be fairly easily implemented.



As seen in the figure above, the game model is independent from the game controller, allowing the Model to be the one to "pull" inputs out from the Controller regardless of human or computer player.