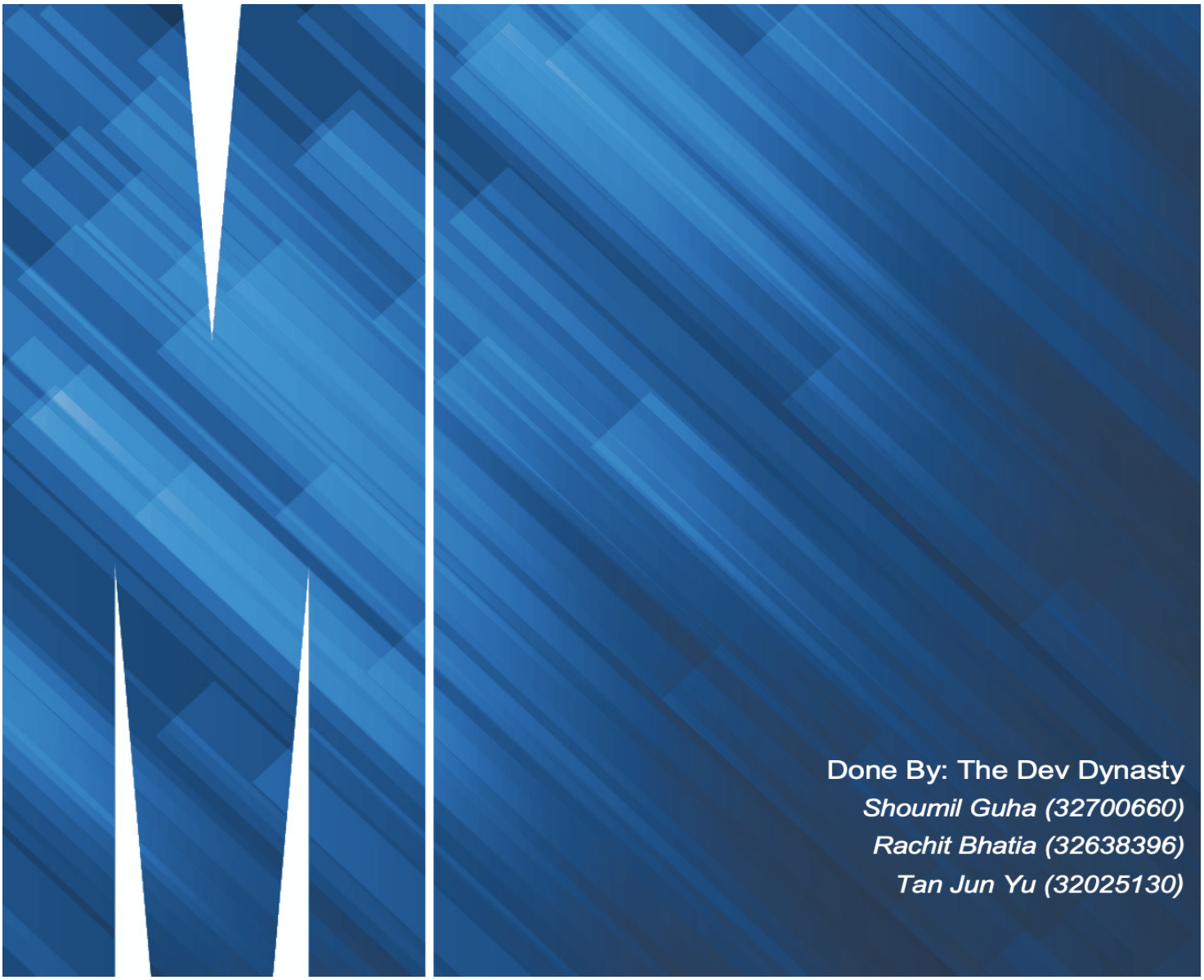
MONASH University

FIT3077
SPRINT 4: PRODUCT DELIVERY



Done By: The Dev Dynasty

Shoumil Guha (32700660)

Rachit Bhatia (32638396)

Tan Jun Yu (32025130)

Choice of Advance Feature

Playing against Computer Feature:
“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”

9MM: User Stories

The user stories have been modified from Sprint 1 and listed below. The user stories for the *advanced feature of playing against the computer* have been provided with task descriptions to give more details.

1. As a game client, I want each player to be allocated with 9 tokens initially, so that they can use these tokens to make their moves throughout the game.
2. As a game board, I want to have 24 intersection points, so that the players can place their respective tokens on them and track the progress of the game.
3. As a token, I want to be differently coloured from the opponent player’s tokens, so that my player can distinctly track their moves.
4. As a player, I want to take alternate turns with my opponent, so that the game is fair and each player gets an equal number of moves.
5. As a game client, I want a mill to be formed when 3 tokens are placed together along a line on the board, so that the player with the mill gets an advantage.
6. As a token, I want to be able to slide along the board line to an adjacent empty intersection, so that my player can position me strategically.
7. As a game client, I want a player to be declared as the winner when the opponent has less than 3 tokens left on the board, so that the game can come to a conclusion.
8. As a player, I want to remove one of my opponent's tokens from the board after forming a mill, so that I can reduce my opponent’s winning chances.
9. As a token, I want to be able to fly to any empty intersection when there are only 3 of my instances left on the board, so that it becomes more challenging for the opposing tokens to block my mill.
10. As a player, I want to be able to play the game with an opponent on the same device and application, so that I can enjoy a streamlined multiplayer game experience.

Advanced Feature: Play against Computer

11. **As a player, I want to be able to start a game against a computer bot, so that I can practice and enjoy playing individually**

    Tasks Involved:

    a. Create a new separate game board view for playing with CPU
    b. Create a button which loads this view and add it to home page

12. **As a computer bot, I want to be able to make every possible valid move in the game and take completely random decisions to ensure a fair and simple gaming experience**

    Tasks Involved:

    a. Create methods for generating random moves of each valid move type in the game (placing, sliding, flying, removing)
    b. Add mill formation functionality to Computer Player's tokens

13. **As a computer bot, I want to take a fixed response time before every move, so that the player gets a more realistic experience while being able to observe the state of the game clearly**

    Tasks Involved:

    a. Ensure that the game loop is running in a separate background thread to allow UI changes to happen smoothly without overloading the main thread
    b. Add a time delay before each move Computer Player makes a move by forcing the background thread to sleep for a fixed amount of seconds (2 seconds)
    c. Add an additional delay between formation of mill by Computer Player and removal of random token (5 seconds)

UML CLASS DIAGRAM

Note: New additions have been shown in green, while deletions/replacements have been shown in red. All the other colours used in the arrow lines are merely for clear distinction between overlapping lines and do not signify anything specific

ARCHITECTURE & DESIGN RATIONALE

1. What has changed and Why has it changed?

Type of Change	Changes	Justification
New Method	Added addImage() function as a new method in MainPagePanel class	This method is added to remove code duplication in the class. The MainPagePanel is responsible for the homepage of the application, and it contains 2 different images (one for background and one for title). Since the addition of an image has the exact same code with a few different parameters like size and positioning, the code block was converted into a method to prevent code repetition, thus following the DRY principle.
New Attribute	Added millLabel as a new attribute in MainWindow class	This is needed to display a message to inform the user that a mill is formed. The message regarding mill formation is displayed to avoid confusion and inform the players about the state of the game. It is made a static variable so that its value can be modified via MillChecker class where it is actually needed to be displayed. It is an attribute of MainWindow because all the display elements are inside this class. So to ensure consistency, millLabel was also added in MainWindow.
New Method	Added voidInstance() as a new method in the following classes: MainWindow, GameBoard, Game	This function is added as a result of adding the new feature of returning back to homepage and then starting a new game. This function nullifies the states of the current instances of MainWindow, GameBoard, and Game Since all 3 of these classes use singleton instances, setting their instance to null is a way to reset their state so that whenever a new game is started, getInstance() will be called and that will create a new instance. This state reset helps to enable the feature of returning to the homepage and starting a new game.
New Method	Added generateButton() as a new method in MainWindow class	This method is used to create a new button. Although this method is only used once to create the ‘Return’ button, the code block was still put inside a method to enhance modularity. Since

		this is a newly added button, the code was put inside the addAllItems() function initially, however, that function already handles addition of several components. Thus, to avoid a god method, the code for creating a button with the reset function was modularised into a separate method.
New Method	Added getMillLabel() as a new method in MainWindow class	Getter method for newly created JLabel millLabel
New Attribute	Added protected attribute isComputer in Player class	This attribute is added to recognise if the player is a computer player or not. A True value is used to identify a computer player. Its visibility is set to protected to ensure it is shared with the child class i.e. ComputerPlayer. The need for the attribute arose when specific actions had to be modified when the current player was a computer player type.
New Method	Added typeIsComputer() method in Player class	The getter method for isComputer
New Method	Added generateRandomNumber(int) as a new method in ComputerPlayer class	This method is required to generate a random number within a given range in order to perform a random move including SLIDING,FLYING and PLACING. The random number is used to select a random intersection point among the valid intersection points. This is needed as a method as its implementation is used in two different places which are the generateRandomSlidingMove() and generateRandomFlyingPlacingMove(). Hence, repeated code can be avoided and DRY principle will not be violated.
New Method	Added new method produceDelay(int) in ComputerPlayer class	This method is added to make the background thread sleep for a fixed amount of seconds to introduce fixed time delays between computer player's moves. The time delays were introduced to make the player's experience with computer more engaging and easier to follow
New Signature of Method	Added a new parameter to an existing method checkMill(intersectionPoint) to checkMill(intersectionPoint,boolean)	The boolean parameter is needed to identify if the mill is formed by a normal Player or ComputerPlayer. This differentiates the display message to be shown when a mill is formed as the display message is different when a ComputerPlayer forms a mill.

Class Removed	HumanPlayer class removed from the architecture	It was realised that this class was redundant as it did not have any of its own unique implementations and completely used inherited components from the Player class. Hence, this class was removed from the architecture and the usage of HumanPlayer was replaced with Player by converting the Player class from abstract to concrete and shifting the entire implementation of setPlayerTurn() inside the Player class itself.
Dependency Relationships removed	The dependencies of RemoveMove to MillChecker, IntersectionPoint, and GameBoard were removed	These dependencies were removed by introducing a new interface TokenRemoval and shifting the token removal code block inside this interface.
New Interface Created	TokenRemoval interface created with new default method performRemoval(Token)	New interface created to contain the method responsible for removal of a token. The interface contains a default method performRemoval which is responsible for token removal from the game. This method was shifted from RemoveMove to a new interface because the exact same code block was used by the ComputerPlayer class for token removal. ComputerPlayer couldn't have had a dependency to RemoveMove because RemoveMove is created for Token type only, and creating dependencies using dummy instances is a bad practice as it leads to unnecessary coupling. Furthermore, according to the Dependency-Inversion Principle, high-level modules should depend on abstractions rather than low level modules, hence creating a new interface is a better design choice since a lot of dependencies between RemoveMove and other concrete classes are now removed. This could have been done using an abstract class because ComputerClass already inherits from Player class, and since there can be only 1 inheritance but multiple interface implementation, the interface design was chosen over an abstract class.
New	ComputerPlayer and	The interface is implemented by both of

implement interface relationships	RemoveMove implement the TokenRemoval interface	these classes because both use the functionality in the performRemoval function for token removal.
New Dependency Relationships	The interface TokenRemoval has dependencies to Player, Token, Game, GameBoard, IntersectionPoint, MillChecker	These dependencies are there for the token removal part since communication is needed between multiple classes to remove a token. However, these dependencies are from an abstraction, hence, the coupling is not very strong.