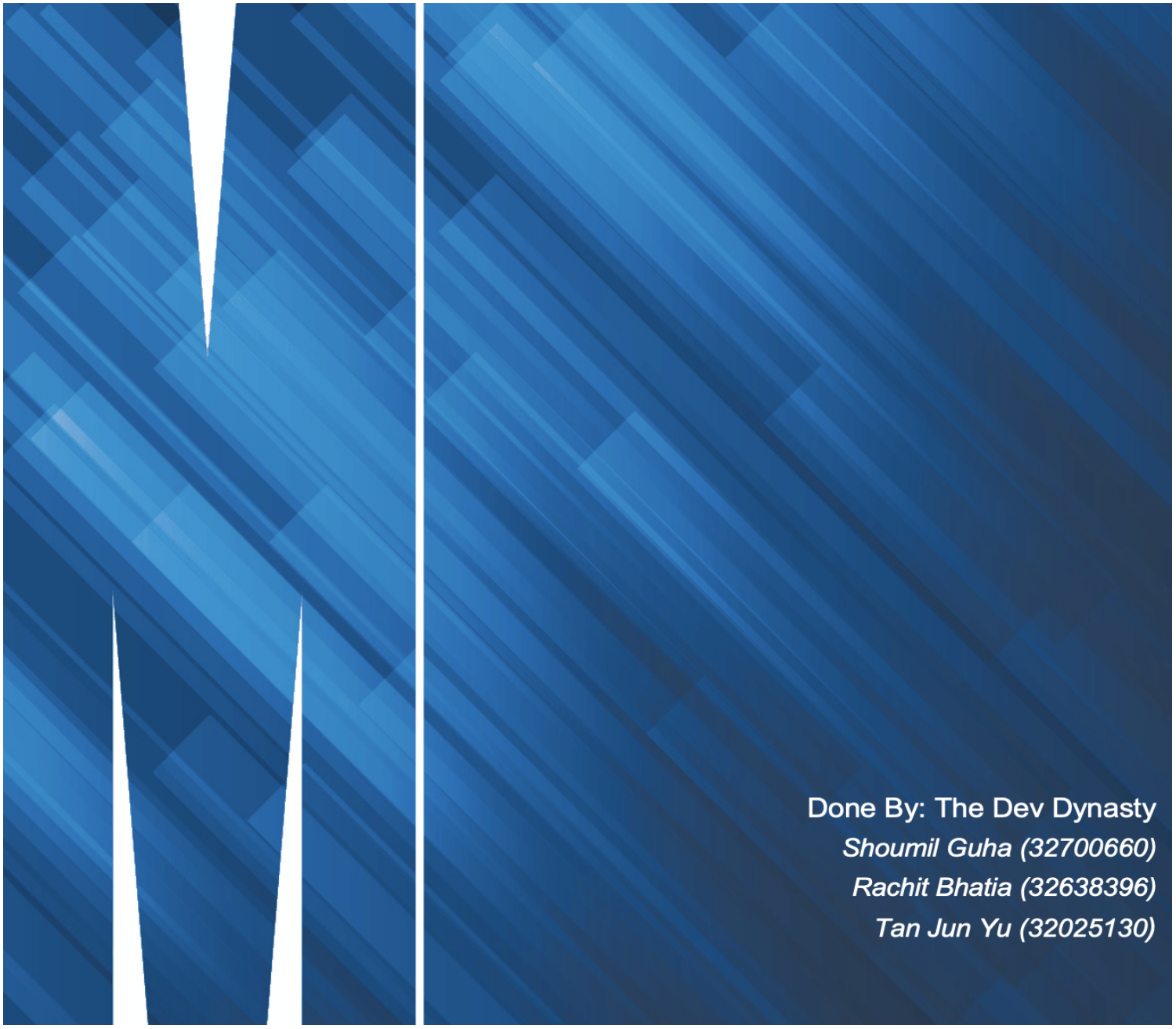
MONASH University

FIT3077
SPRINT 1: PROJECT INCEPTION



Done By: The Dev Dynasty

Shoumil Guha (32700660)

Rachit Bhatia (32638396)

Tan Jun Yu (32025130)

Team Name : The Dev Dynasty

Team Photo :

Team Members Contact Details

1. Shoumil Guha
    - **Phone**: 011-5640-9250
    - **Discord**: shoumil#3182
    - **Email**: sguh0003@student.monash.edu

---

2. Rachit Bhatia
    - **Phone**: 019-405-7032
    - **Discord**: racrage#2757
    - **Email**: rbha0031@student.monash.edu

---

3. Tan Jun Yu
    - **Phone**: 016-278-3792
    - **Discord**: junyu#7430
    - **Email**: jtan0245@student.monash.edu

Team Members Info

1. Shoumil Guha
    - **Technical and Professional Strengths :**
        - ➢ Well versed in object oriented programming and its design using Python and Java.
        - ➢ Experience in database management systems, web development, image processing.
        - ➢ Fast learner and starts work early which leads to no panicking close to deadlines.
        - ➢ Proficient in the agile working pattern due to past experience in Scrum projects
    - **Fun Fact :** I eat breakfast an hour before lunch just because I can.

---

2. Tan Jun Yu

   - **Technical and Professional Strengths :**
     - ➢ A well-developed understanding on Object Oriented Programming principles which is vital for this assignment .
     - ➢ Good collaboration skills by being able to work effectively with team members and actively participate in any group discussions for the best project outcome
     - ➢ Familiar with the programming language we have chosen for this assignment that is Java being applied with Object Oriented Programming
     - ➢ Experienced in the field of agile projects
   - **Fun Fact :** I dislike playing 9 Men's Morris and will not ever want to play or see this game again after this assignment

---

3. Rachit Bhatia

   - **Technical and Professional Strengths :**
     - ➢ Very familiar with Object-Oriented Programming principles due to experience from prior projects.
     - ➢ Equipped with some knowledge in UI development for iOS mobile applications
     - ➢ Well-experienced with Java principles and semantics
     - ➢ Proficient in team management
     - ➢ A good team player with the ability to provide valuable input to every discussion while respecting everyone's opinions
     - ➢ Well adapted to an agile team format
   - **Fun Fact :** I love to listen to music in all moods except when feeling overwhelmed or stressed

Team Schedule

- ❖ Meeting Schedule
  - ➢ At the beginning of each sprint, the team will hold a sprint planning meeting to discuss and agree on the sprint goal, scope, and tasks to be completed. Task division for the sprint will be done in this meeting.
  - ➢ A weekly standup meeting will be held to check on the team's progress to ensure that everything is on the right pace to lessen the chance of missing deadlines. Each team member will update one another on the progress of one's assigned tasks and make sure no one is left behind. This meeting will usually be on every Friday at 4pm. The timing may change subject to the team's convenience but the meeting cannot be cancelled.
  - ➢ The frequency of these weekly standup meetings will change from once a week to 4 times a week when closer to the end of the sprint. When this change takes place will depend on the remaining amount of work for that particular sprint (frequency of meetings could increase from 7 to 10 days before the end of the sprint).
  - ➢ After the end of each sprint (preferably the same or next day after a sprint ends), an additional sprint retrospective meeting will be held to discuss the shortcomings of the completed sprint and possible improvements for future sprints.

- Work Schedule
  - Every member is expected to spend at least 1-2 hours per day on the allocated work. This may increase based on the workload for the sprint.
  - The tasks given to each team member will be assigned an internal deadline considering the availability of the member and the difficulty of the task.
  - In every standup meeting conducted, the work done by each member will be discussed and reviewed. At the same time the team will look at any issues faced by a member and try to resolve them to smoothen the workflow.
  - It will be aimed to allocate roughly the same amount of workload to each sprint to ensure consistency in the workflow.

- Workload Division and Management
  - Every type of activity (writing user stories, coding, prototyping) is distributed equally
  - Tasks allocated to each member will cover all aspects of the project to prevent siloing.
  - Tasks will be created at the beginning of the sprint and will be managed through a Trello board. Each task's priority will also be recorded. All of this will be done during the sprint planning meeting.
  - Each member will update the status of task completion on the Trello board.
  - Team members should voluntarily choose tasks from different aspects to make sure everyone is comfortable with their workload. The final decision should be agreed upon by the other members.
  - Any task that might be harder to implement can be divided into smaller tasks and assigned to more than one team member.
  - Any severe difficulty encountered by a team member on the given tasks can be further discussed in a standup meeting and a decision whether to reassign the particular task to another member can be made.

Technology Stack and Justification

Definition and Purpose:

Chosen Programming Language: Java

Pros:

1. Strong structured language
2. Industry standard for Object-Oriented Programming
3. Widespread support online as it has been around for decades
4. Platform independent

Cons:

1. Slow
2. Poor GUI tools especially for complex GUIs
3. Verbose. Takes much more code to implement the same tasks in other languages

Justification for Selection:

Java is selected as our programming language due to its strict rules of syntax that precisely conforms to the Object-Oriented Programming principles. With Java being an object oriented language, methods and variables are declared or implemented in each of their respective classes. Hence, it allows a better readability of the code. Besides that, Java provides easier employment of encapsulation by just using access modifiers for the variables declared as compared to other languages.

Most importantly, all of our team members have experience with using Java as an object-oriented language from prior projects and thus will ease the process of code development rather than trying to familiarise ourselves with other languages being applied with Object-Oriented Programming.

Discarded Alternative Programming Language: Python

Pros:

1. More unstructured with less rules to follow when programming
2. Overwhelming support online
3. Extensive support libraries
4. Has many third party packages
5. Simplicity

Cons:

1. Requires additional libraries to implement and use classes
2. Slower than Java as code is interpreted at runtime
3. Unconventional for developing object oriented programs

Justification for Discardment:

Due to the flexibility of Python which supports a variety of programming styles, Object-Oriented may not be rigorously enforced. Scripts can still be executed without any classes being declared and this strongly violates the rules of Object-Oriented Programming.

Moreover, none of our team members have experience in using Python as an object oriented language that may end up being time consuming to adapt to the new kind of formats in which Python supports.

<u>Chosen GUI Tool: JavaSwing (Might need support from the tutor)</u>

Pros:

1. Vast amount of component types
2. Components support additional features well
3. Code base is entirely implemented in Java
4. Standard GUI Library
5. Platform independent
6. Components easy to customize

Cons:

1. Hard to move components around in the Swing Designer
2. Slower Performance than AWT if the programming is not well designed

Justification for Selection:

JavaSwing and JavaFX are the two main GUI Frameworks we have been considering. After trying out both the GUI Frameworks, JavaSwing is found to be the easier one to apply with Object-Oriented Programming. It is found that the components created in JavaSwing can be easily delegated into their respective classes while we encountered some difficulties in JavaFX.

Discarded Alternative GUI Tool: JavaFX

Pros:

1. Designing UI can be easy with the help of external applications like SceneBuilder.
2. Independent on operating system
3. Easy to create complex UIs with lesser code

Cons:

1. Smaller community support compared to Swing

Justification for Discardment:

Although JavaFX was initially more preferable than JavaSwing (chosen GUI Framework) , but we have discovered a hurdle while testing out JavaFX that we have yet to find a solution for it. While creating components and adding them to the GUI frame, it is found that the code implementation for the all the different components will have to be added to the same one and only one Controller Class .This could be a major issue that might result us in not being able to conform to Object Oriented Programming principles as we might not be able to separate different components into different classes together with their own methods and variables.

Discarded Alternative GUI Tool: AWT (Abstract Window Toolkit)

Pros:

1. Low crashing rate
2. Basic user interface design, easy to use especially for beginners

Cons:

1. Limited types of component in the toolkit
2. Components may not have enough support for special features like images
3. Simple layout management makes it difficult to build complex GUIs
4. Limited customization for components
5. No extensibility.

Justification for Discardment:

The Abstract Window Toolkit(AWT) has a considerably lesser amount of components and functionalities as compared to both JavaSwing and JavaFx. With that being so, code implementation may be more abundant when AWT is used due to the lesser amount of in-built features and components. In addition,the appearance of the components in AWT is mainly unconfigurable which might hinder the UI designing process of the 9 Men’s Morris game.

Discarded Alternative GUI Tool: SWT (Standard Widget Toolkit)

Pros:

1. Rich component types
2. Components support additional features well

Cons:

1. Not in the standard library of the JRE (Java Runtime Environment)
2. Steep learning curve if unfamiliar with underlying native GUI libraries
3. Can be complex to use due to its sophisticated features and customizability
4. Harder to find support

Justification for Discardment:

Standard Widget Toolkit(SWT) is known to have unstable performance when being operated on any other operating systems other than Windows. In other words, SWT is vulnerable to crashes in other operating systems like MacOS in which two of our members are currently using. This may be an aggravating hindrance that could impact the development process that our team should certainly avoid.

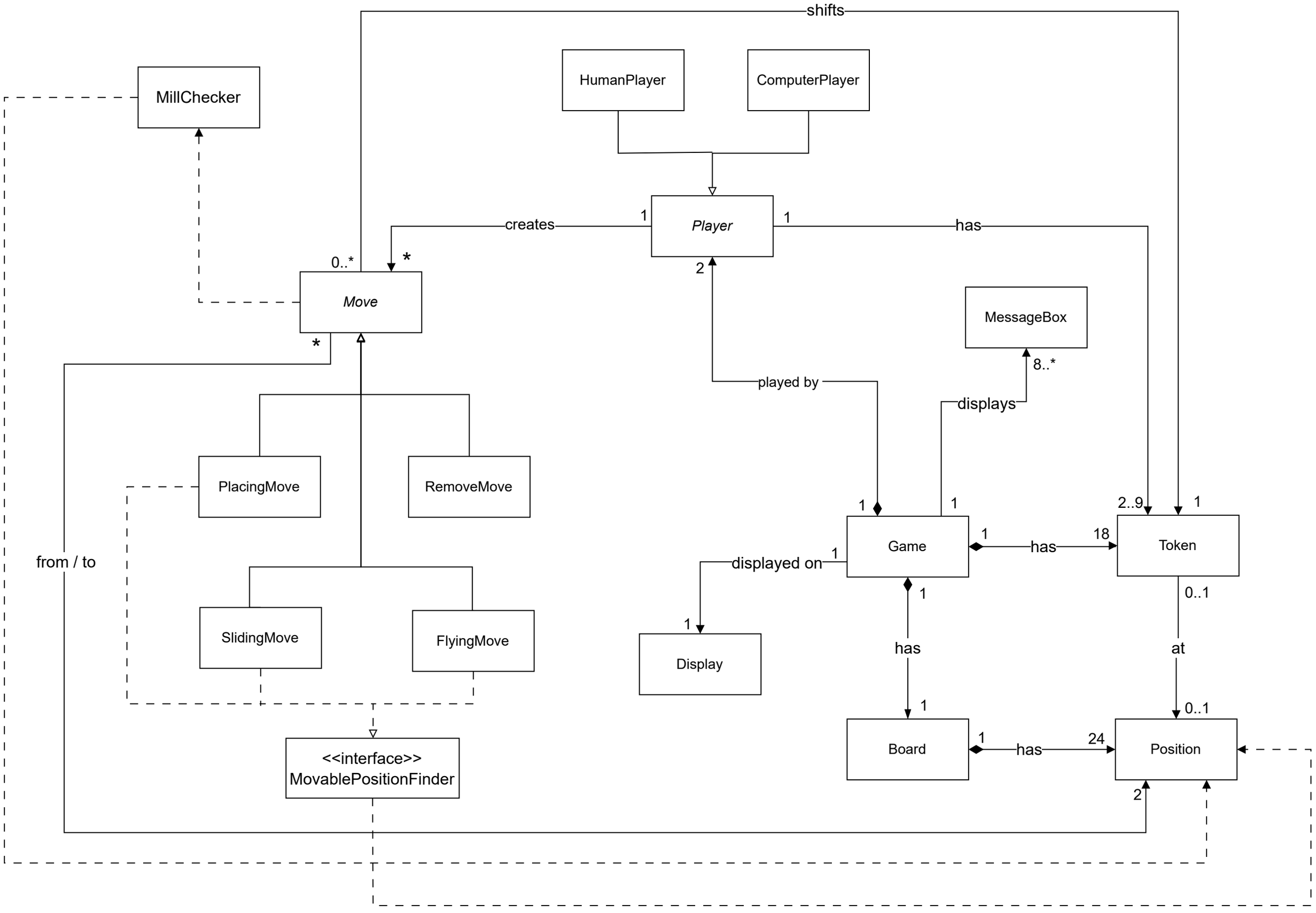
9MM: User Stories

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 player, I want to be able to place my tokens on the intersection points as per my preference, so that I can strategize my gameplay.
4. As a token, I want to be differently coloured from the opponent player's tokens, so that my player can distinctly track their moves.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. As a player, I want the game to display which player's turn it is, so that I can be aware when I need to make a move.

[Advanced Feature]

13. As a player, I want an option to play against the computer, so that I can practice and enjoy playing individually.
14. As a computer, I want to be able to play random moves among all valid moves, so that the player has a complete fair gaming experience.

DOMAIN MODEL DIAGRAM


shifts
MillChecker
HumanPlayer
ComputerPlayer
Player
creates
1
1
has
0..*
*
Move
2
*
MessageBox
8..*
displays
played by
PlacingMove
RemoveMove
from / to
SlidingMove
FlyingMove
<<interface>>
MovablePositionFinder
Display
displayed on
1
1
1
1
Game
1
has
18
Token
2..9
1
1
has
0..1
at
1
0..1
Board
1
has
24
Position
2

DOMAIN MODEL RATIONALE

Player

Reason for choosing this domain

Since both HumanPlayer and ComputerPlayer will later have the same attributes (such as an arrayList of tokens) and methods (such as the basic moves of moving the token), Player is introduced as a parent class for both HumanPlayer and ComputerPlayer. It defines the standard for all the actions any type of player can take during the game, hence, Player will be defined as an abstract class so that there is no repetition of code in the HumanPlayer and ComputerPlayer classes. Having an abstract super class for Player, ensures that each type of player can have its own implementation details within the same domain.

Discarded Alternative: The HumanPlayer and ComputerPlayer classes could repeat the same skeleton code for a Player's base actions but that would violate the DRY principle. Hence, an abstract class was a better option as most of the elements can be reused and some method implementations can be made mandatory (by using abstract methods). This solution also applied the Open-Closed Principle since extension of basic Player behaviours is easy and such changes do not need to be copied to multiple classes.

Reason for the relationships

- Player – Move : Player can use Move to move their tokens at each turn to any valid position.
- Player – Token : Each Player has a total of 9 tokens at the start of the game. It might be reduced slowly one by one to a minimum of 2 tokens in which the game will end.

HumanPlayer

Reason for choosing this domain

The 9 Men's Morris game requires at least one HumanPlayer for it to be runnable. The HumanPlayer domain defines the implementation of any behaviours that any user of the game does

Reason for the relationships

- HumanPlayer – Player : HumanPlayer is chosen to be a child class to the Player in order to prevent repetition of items in multiple player classes. It extends from the abstract Player class.
- HumanPlayer – Game : A Game will have at least one HumanPlayer (ComputerPlayer Mode) and a maximum of 2 (HumanPlayer Vs HumanPlayer) depending on the game mode chosen.

ComputerPlayer (Advanced Feature)

Reason for choosing this domain

The user can select to play against a bot whose actions and behavioural implementation is defined by the ComputerPlayer domain.

Reason for the relationships

- ComputerPlayer – Player : ComputerPlayer is chosen to be a child class to the Player in order to prevent repetition of items in multiple player classes. It extends from the abstract Player class.
- ComputerPlayer – Game : A Game might have a ComputerPlayer is a user chooses to play with a bot.

Game

Reason for choosing this domain

The Game domain defines the state of the whole game at all times. It initialises the main required elements (like board, tokens, players, etc.) during the startup of the game. The game runs for as long as there is no result (win or draw). Since the Game domain defines the initialisations of certain main elements, this implies that some of these are a part of the game itself and cannot exist on their own without the presence of a running Game. Hence, a composition identifier is used for the relationships between Game and these elements because none of these elements can exist without a running Game. Even the Player domains will not be able to act unless there's a running Game, hence the Game is composed of the Player as well (especially ComputerPlayer).

Reason for the relationships

- Game – Player: Every game has two players.
- Game – MessageBox: During the lifetime of a game, multiple messages may be displayed to inform the players the state of the game. Composition is not used here because MessageBox is not a part that makes up the Game, it is merely contained by the Game.
- Game – Board: Every game is comprised of a game board
- Game – Token: Every game is comprised of 18 tokens
- Game – Display: The state of the game at every point is presented on the display.

MessageBox

Reason for choosing this domain

A message box is mainly required for 2 functions. Message box is displayed when a mill is formed by either of the players as an indication for the player to remove a token from the opponent. Other than that, a message box is displayed when the game ends to display the result of the game.

Reason for the relationships

- Game – MessageBox : The Game will display a minimum of 8 messages (7 mills minimum to win a game formed by the same player + 1 winning message).

Token

Reason for choosing this domain

Token is the main element in the game for the players to move. Actions can be made on these tokens by the respective player. The combinations of these tokens form the basis of the game's actions and result.

Reason for the relationships

- Token – Position: Each Token can either have 0 Position (Removed or yet to be placed) or 1 Position (Placed on the board)

MovablePositionFinder

Reason for choosing this domain

MovablePositionFinder is used to locate the empty positions that a selected token can be moved to. This is an interface which defines the behaviours that are used to look for available empty positions on the board. The implementation details of these methods depend on the type of Move selected. Only the types of moves which involve moving a token to a new position will implement this interface. For example, the FlyingMove will implement this interface and set up the behaviours to look for available positions along the entire board, while the SlidingMove will set up the behaviours of this interface to look for available positions along the current row.

Discarded Alternatives: The required behaviours for this interface cannot be included as abstract methods in the Move parent class because not all Move subclasses require these behaviours (eg: RemoveMove). The behaviours are defined in an interface as abstract methods to enable reusability and extensibility (DRY and Open-Closed Principle). Methods cannot be directly implemented in the respective Move subclasses due DRY violation, and because each Move subclass has its own implementation of these methods.

Reason for the relationships

- MovablePositionFinder – Move subclasses: The respective Move subclasses implement this interface because all of these types of Moves (placing tokens, sliding tokens, flying tokens) require looking for empty positions on the board.
- MovablePositionFinder – Position: The behaviours of the interface iterate through the selective locations based on the type of move to return available empty positions.

Position

Reason for choosing this domain

Position is required in the game to form the basic design structure of 9 Men's Morris's board so that tokens can be placed on them.

Reason for the relationships

- No outgoing relationship

Board

Reason for choosing this domain

The Board consists of positions along which all the tokens can be moved by the players. This domain defines the base location where all the actions of the game take place.

Reason for the relationships

- Board – Position : The Board has 24 Position where a token could be placed. Board and Position relationship is chosen to be represented as Composition since Position is part of the Board. Without the Board, the Position will not even exist as they are supposed to be placed on the Board at specific locations according to the 9 Men's Morris's board design.

Display

Reason for choosing this domain

The Display is used to show the current state of the game at every point.

Reason for the relationships

- No outgoing relationship

Move

Reason for choosing this domain

The Move domain represents an abstract class which would define the basic action for all the various types moves a player can make. All the types of moves involve changing the position of a particular token in the game. The position could also be out of the board, like in the case of RemoveMove. Since, all the move classes have the same basic action of shifting a token from one state to another, all of them extend from the Move super class. The Move class contains the necessary behaviours required to shift the token's state, hence it can be commonly accessed by all subclasses. This prevents violation of the DRY principle and allows for new types of moves to be added in the future without significant changes to the design.

Discarded Alternative: The Move class may not consist of too many methods and attributes since it only defines the basic shifting position action, while the actual action is still individually defined by each Move subclass. Hence, an alternative to this design could be having these limited attributes and methods in every individual Move class. This alternative was discarded because it was deemed unnecessary to contain repeated code elements when it could just be reused. Repetition would make each move class more complicated and the design would be harder to extend in the future. Additionally, it is always a possibility that the Move class could have added functionality common to all types of moves if the requirements change in the future, hence, having a separate Move abstract class was a better option.

Reason for the relationships

- Move – MillChecker : After every move of sliding, placing, and flying, the MillChecker checks if a mill has been formed. In the case of removing, it is used to ensure that

the token removed by the player is not part of a mill. Since all the subclasses (PlacingMove, SlidingMove, RemoveMove, FlyingMove) of the Move will eventually have to use the same implementation of MillChecker, the MillChecker has a direct dependency with the abstract class Move instead of all the individual subclasses.

- Move – Position : Move updates the Position of a token when it is repositioned.
- Move – Token : Move changes the positions of Token

SlidingMove

Reason for choosing this domain

This move provides a way for tokens to move to adjacent positions, which is the basis for the game itself. SlidingMove is only available to the players once they have placed all of their tokens on the board.

Reason for the relationships

SlidingMove – Move : SlidingMove inherits the basic position shifting behaviour from the Move parent class

FlyingMove

Reason for choosing this domain

This move provides a way for tokens to move to any unoccupied position on the board (flying). FlyingMove is enabled when players have three tokens remaining.

Reason for the relationships

FlyingMove – Move : FlyingMove inherits the basic position shifting behaviour from the Move parent class

PlacingMove

Reason for choosing this domain

This move is used by the players to place each of their 9 tokens on the board during the start of the game.

Reason for the relationships

PlacingMove – Move : PlacingMove inherits the basic position shifting behaviour from the Move parent class

RemoveMove

Reason for choosing this domain

This move is used to remove one of the opponent’s tokens from the board. A player that has formed a mill will get to remove one of the opponent’s tokens.

Reason for the relationships

RemoveMove – Move : RemoveMove inherits the basic position shifting behaviour from the Move parent class

MillChecker

Reason for choosing this domain

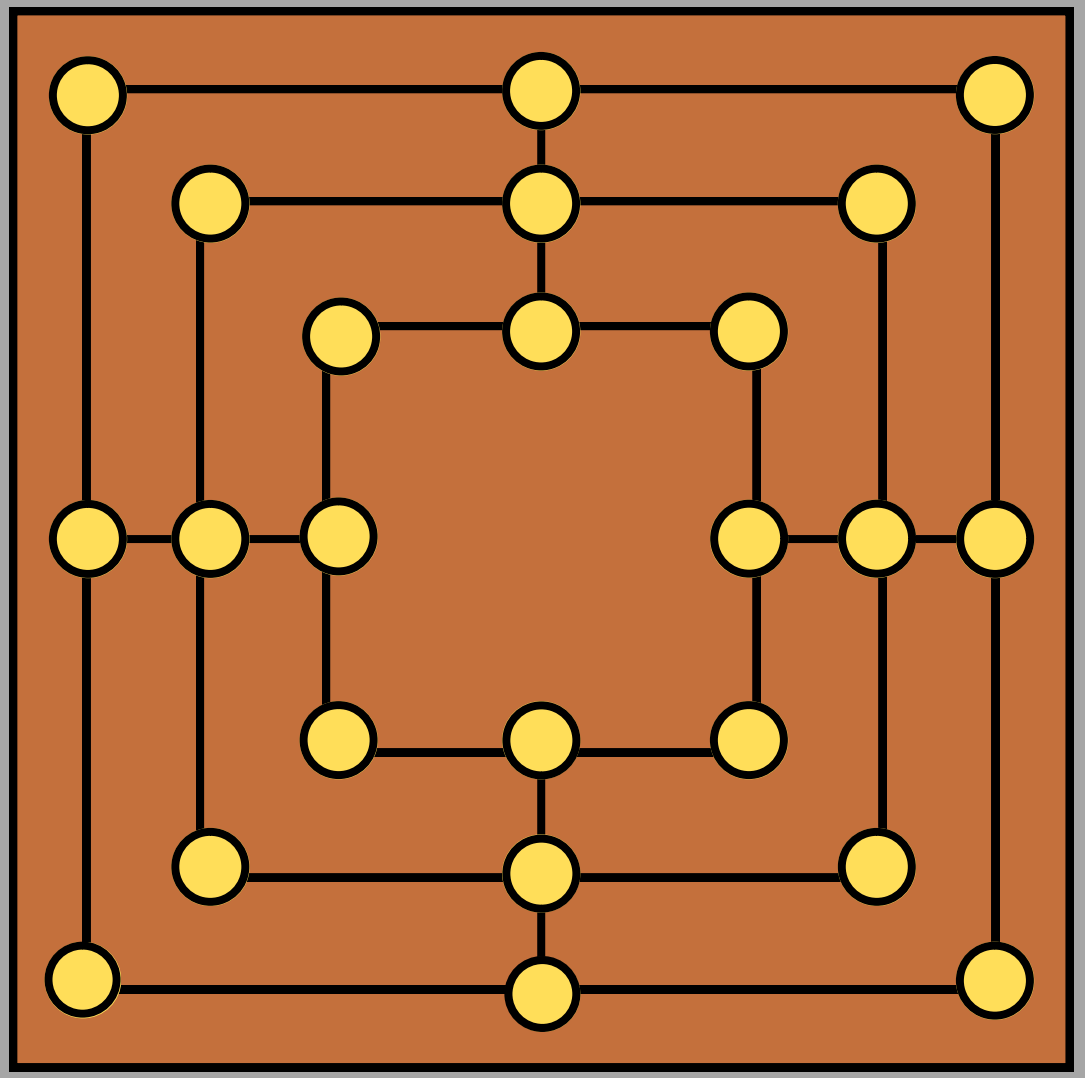
The MillChecker domain defines behaviours to loop through all positions on the board and find if a mill exists. After every move, these behaviours are run to check for a mill on the board. If a mill is found, a message is displayed. Depending on the type of move done, additional functionality may be added to support the MillChecker functionality. For example, for RemoveMove, the mill checking behaviours will have to be run ***before the remove action*** takes place and additional functionality will have to be added to ensure that the tokens in the mill are not removed (since tokens in a mill cannot be removed). But for other Move actions, the mill checking is needed to be done only ***after the move action*** takes place so that the updated state of the game can be displayed to the player.

Reason for the relationships

- MillChecker – Position : The MillChecker goes through all 24 positions to find a straight row of 3 occupied positions (mill) aligned along the same board line.

LO-FI PROTOTYPE

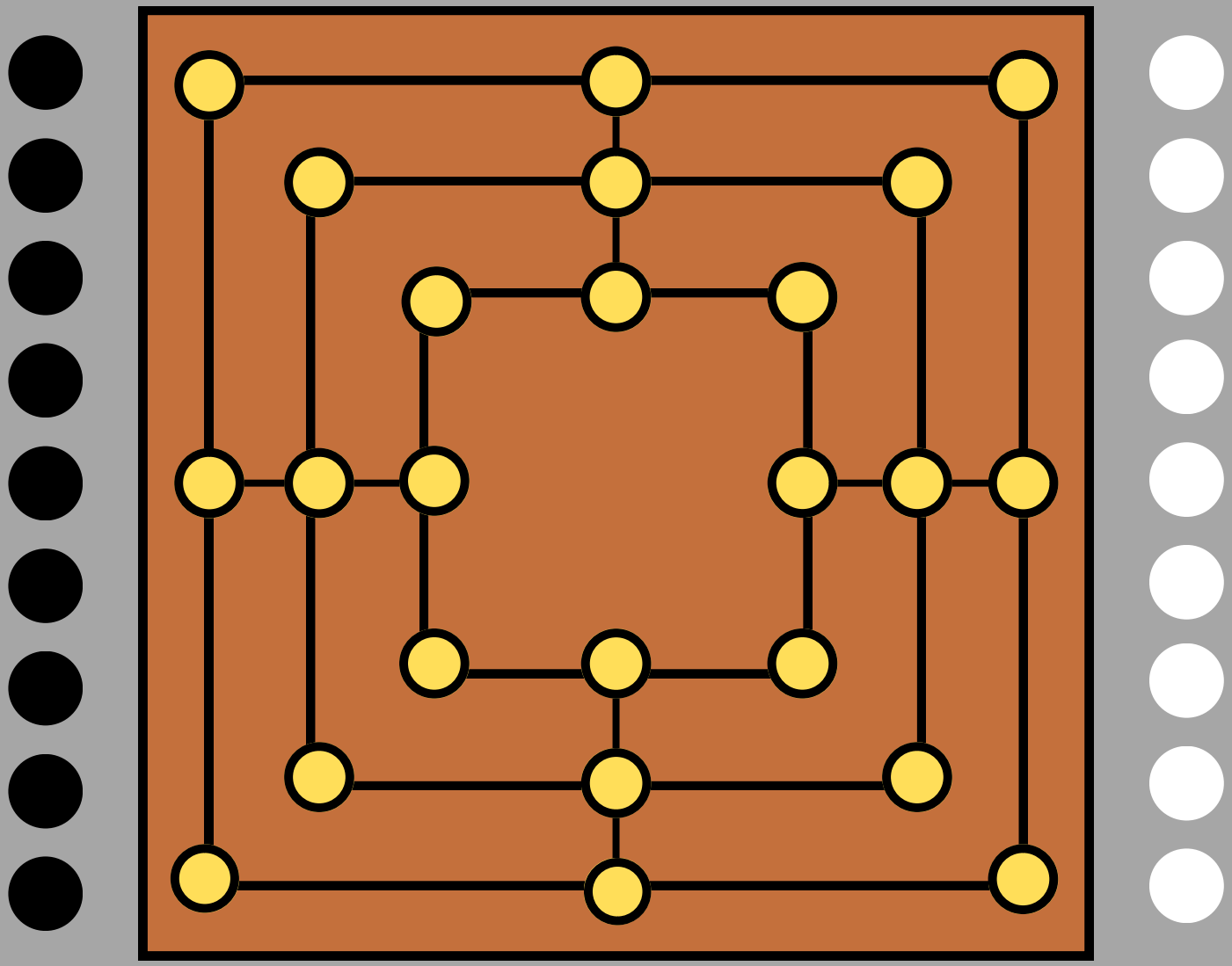
Selecting opponent before game



Player vs Player

Player vs CPU

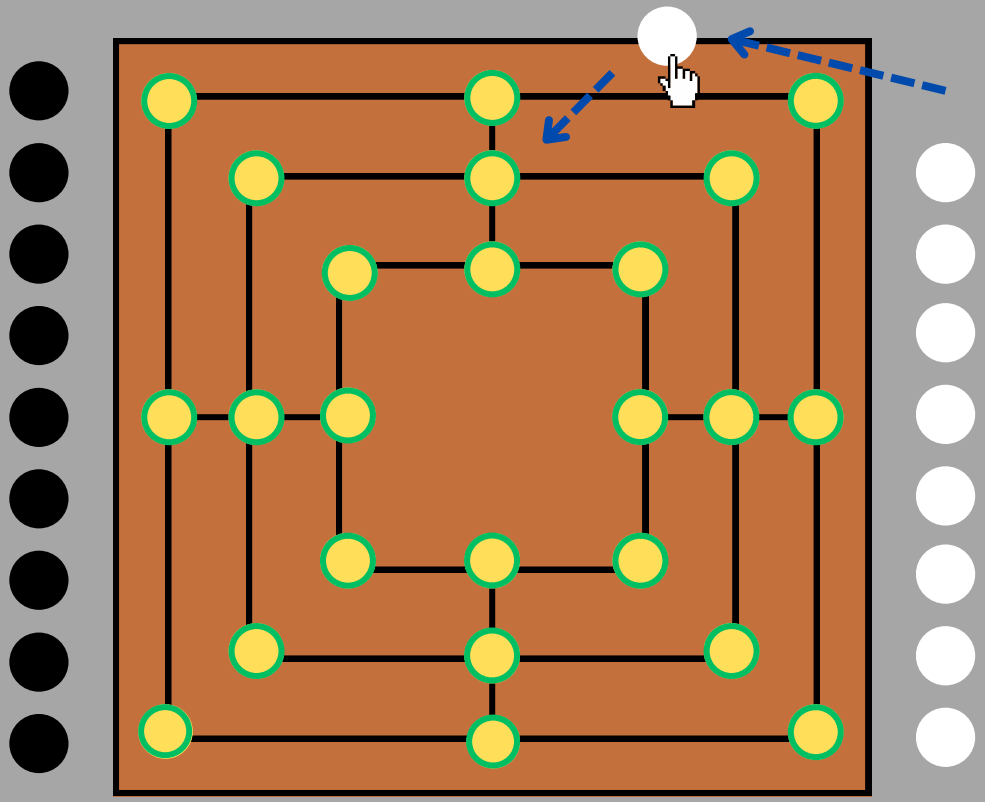
Initial Setup



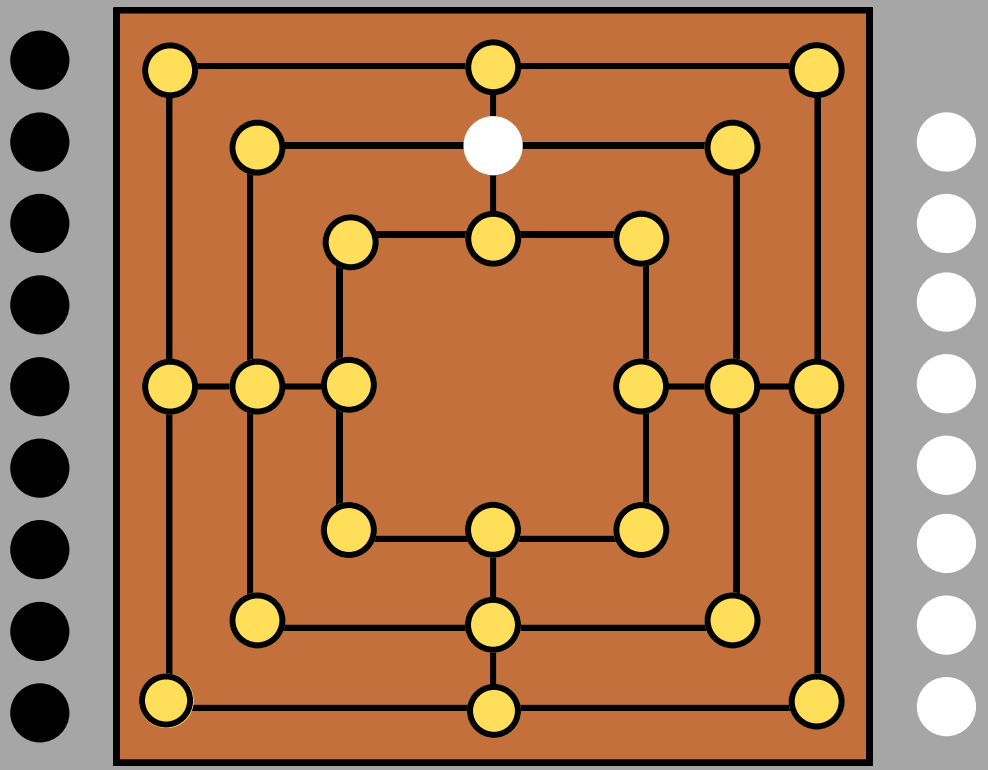
Player 1 Tokens placed on left side of board during initial setup

Player 2 Tokens placed on right side of board during initial setup

Placing of Token

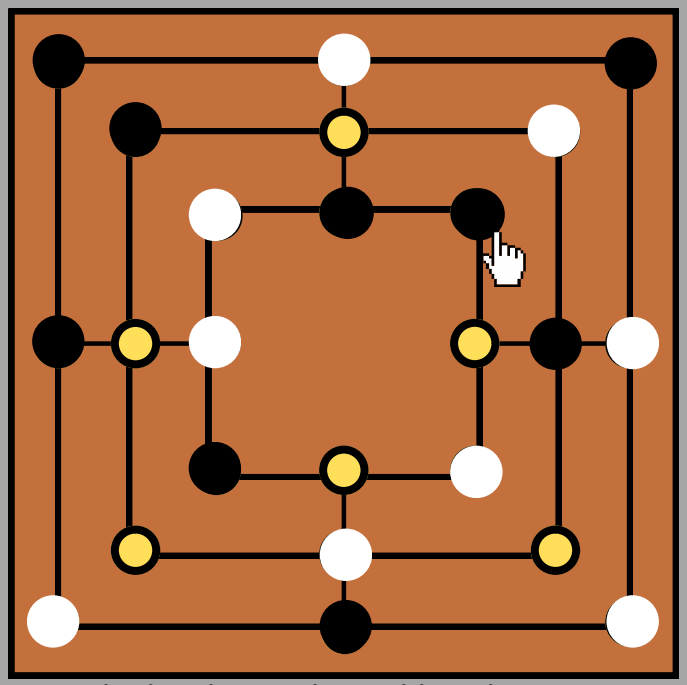


Dragging the token

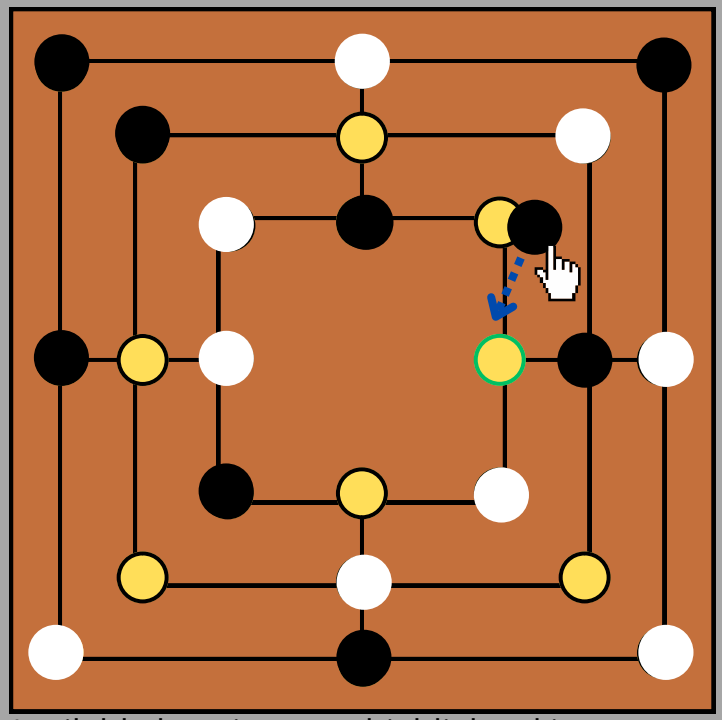


Token successfully placed

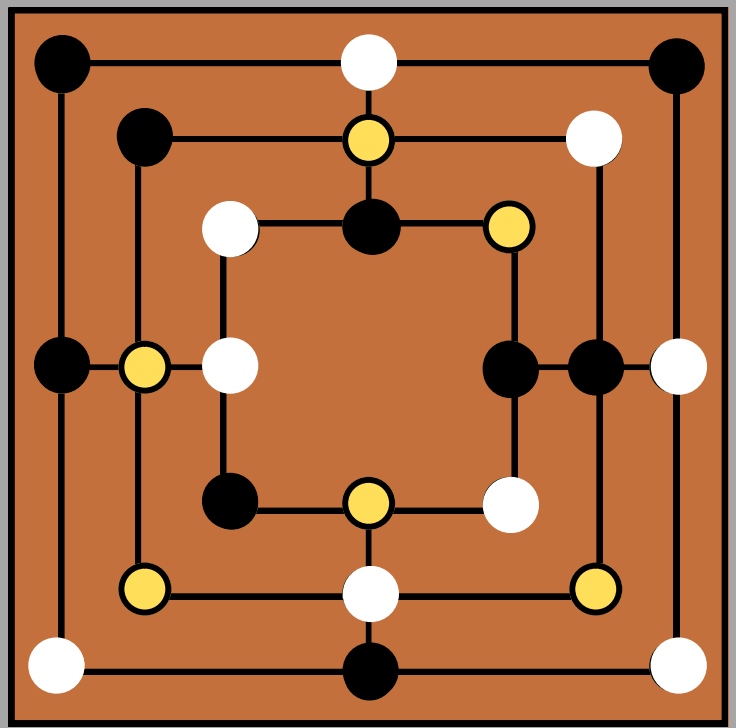
Selecting and Sliding Tokens



Black token selected by player to slide it to an available position

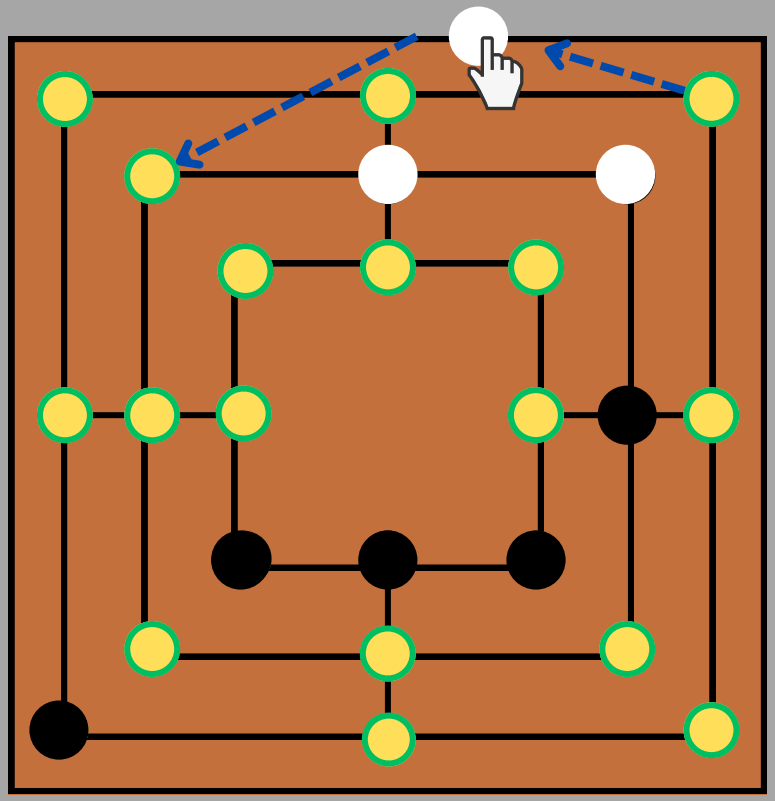


Available location gets highlighted in green, so player drags it to an available spot

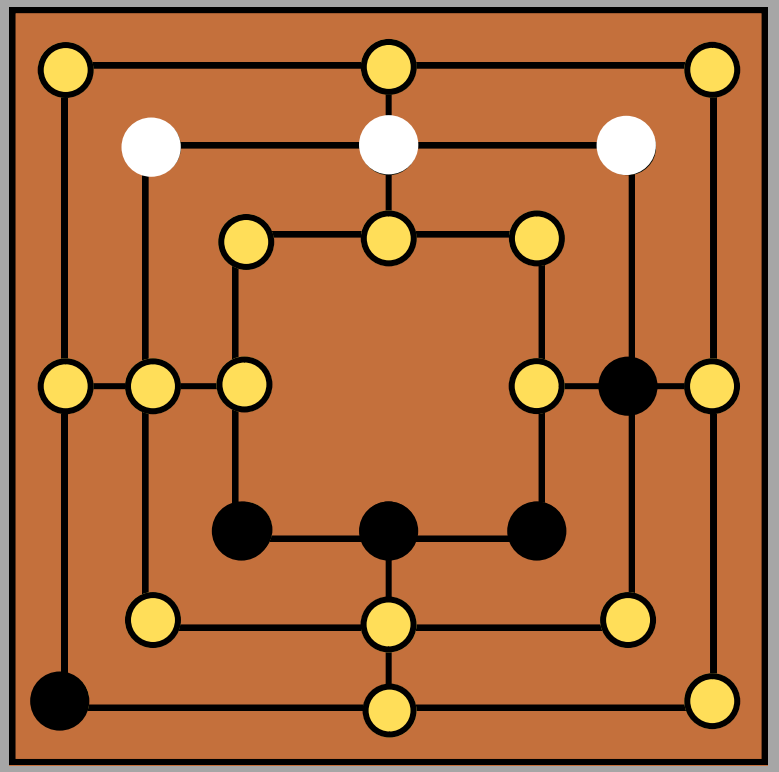


Black token moved to adjacent location

Flying of token

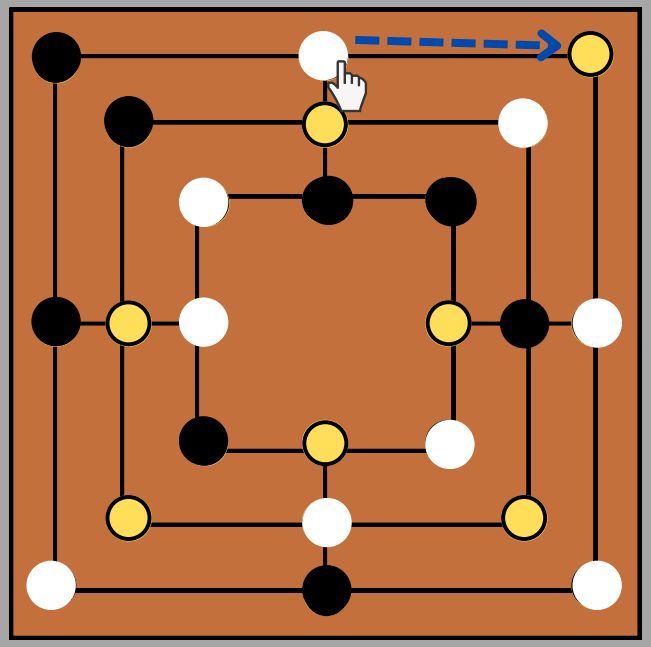


White has 3 tokens left . "Flying" is enabled and movable intersection points are highlighted in GREEN.

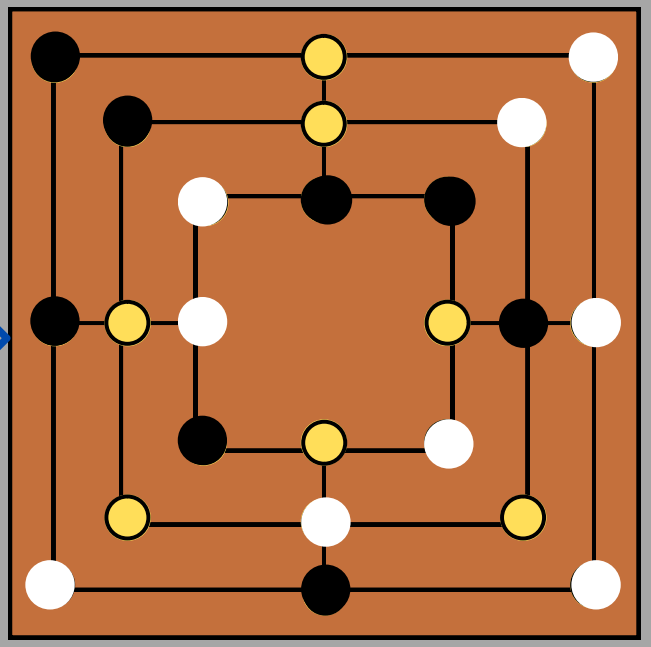


Token is successfully placed after "flying"

Forming a "mill" and removal of token

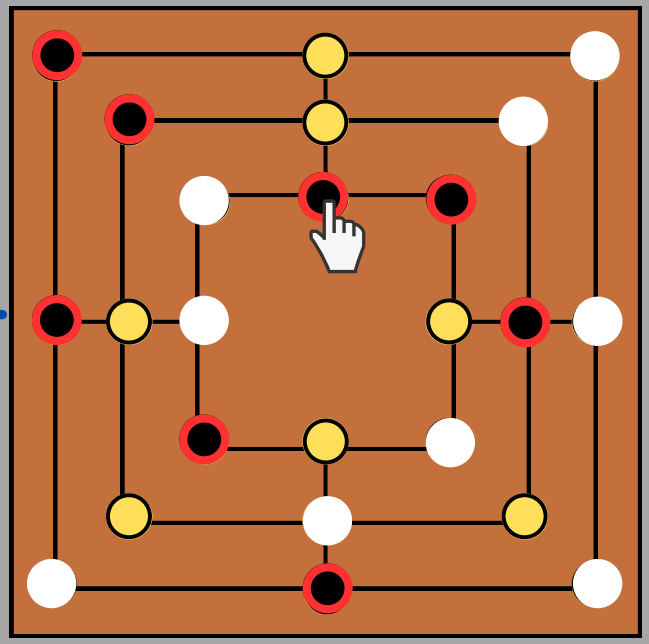


Before forming a "mil"

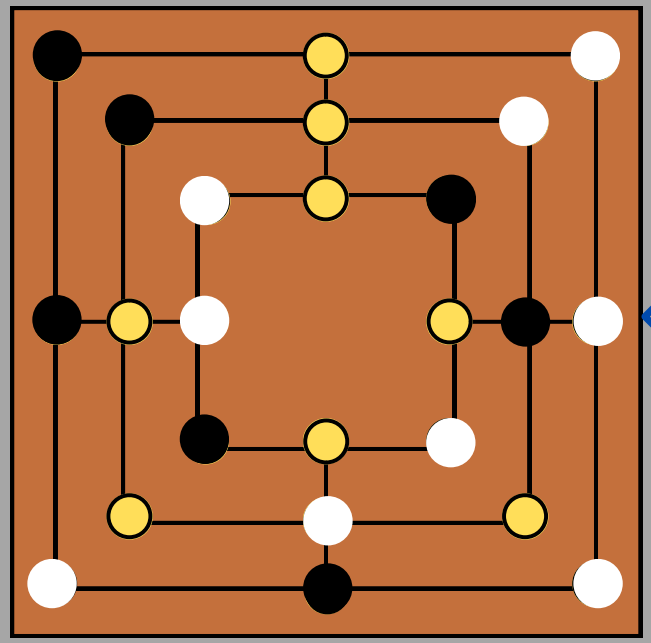


You've formed a mill, remove a piece

After forming a "mil" a message will be displayed

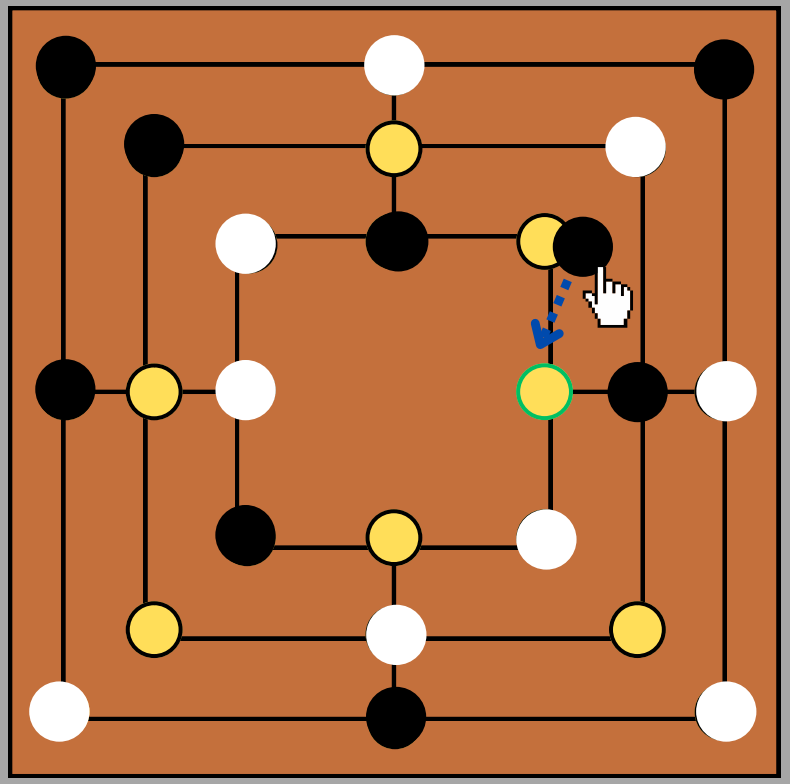


Removable pieces are highlighted in RED

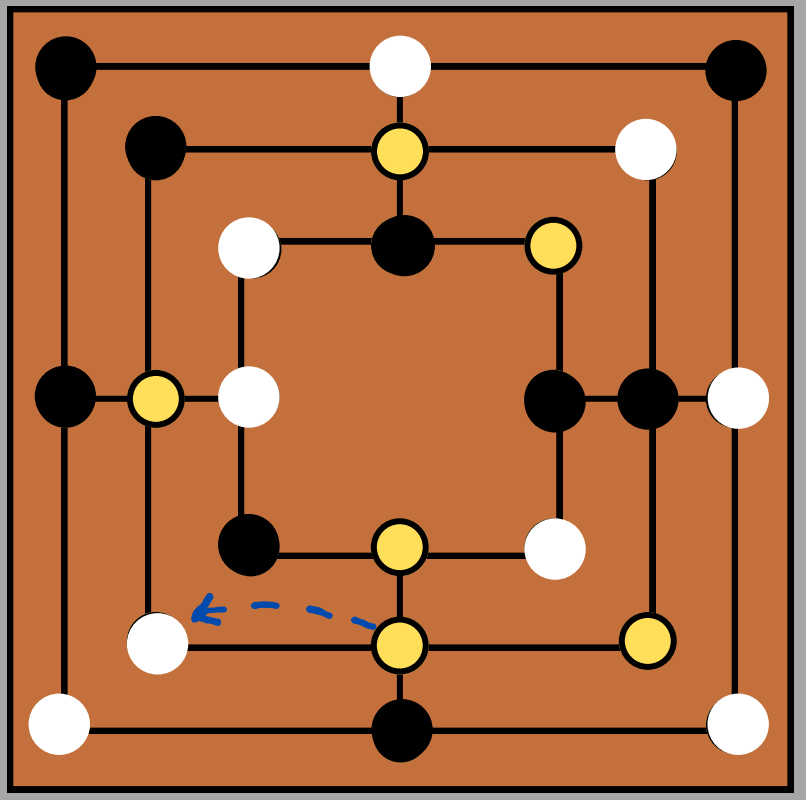


Selected piece is removed after click

Playing with Computer

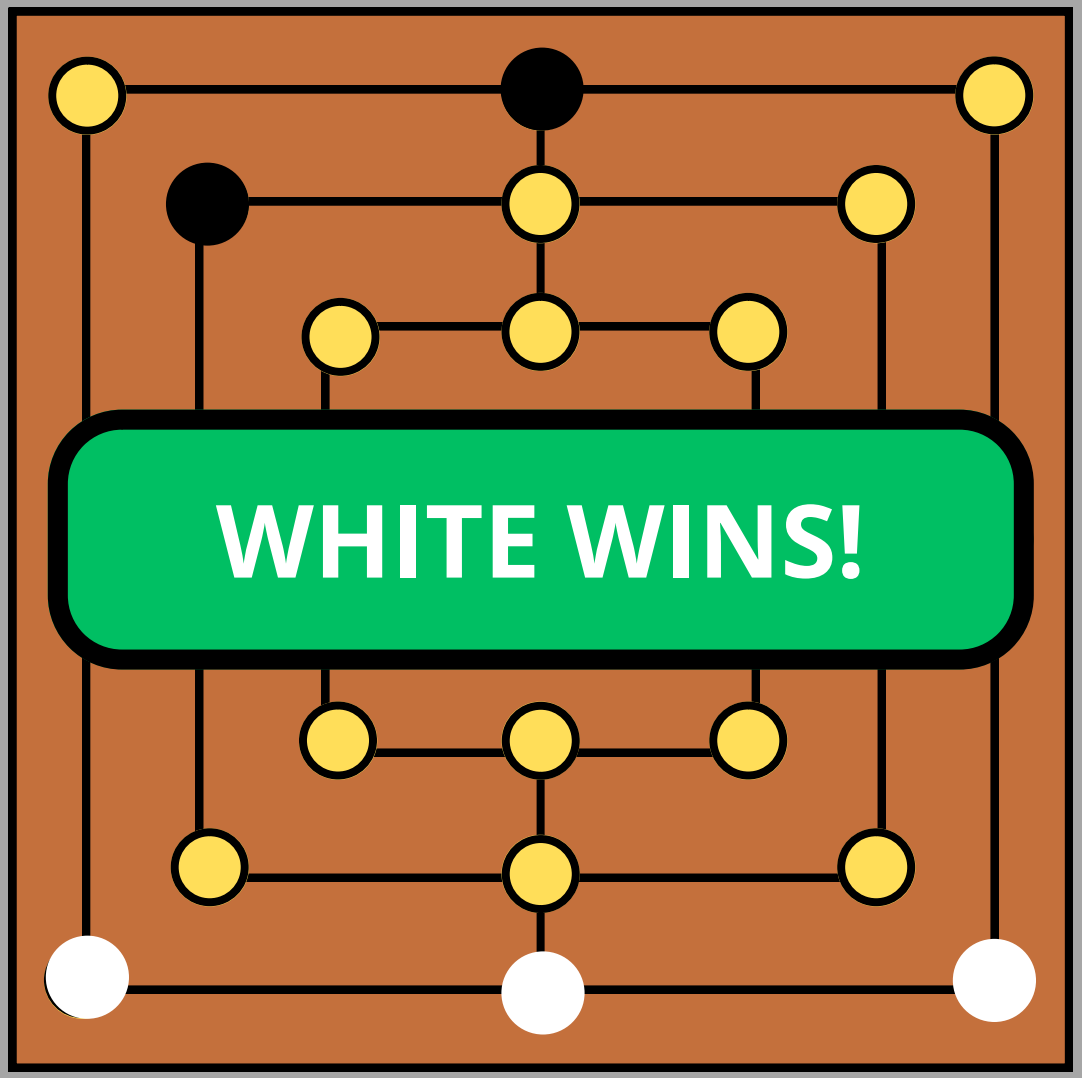


Player makes their move by dragging black token to available adjacent spot



After player makes their move, it is the computer's turn. Computer makes a random move by choosing a token and moving it to an available slot. The movement of the computer's token is shown by the dotted arrow on the board. When the computer moves its tokens, no green highlighted empty spots are seen because the computer quickly makes a random move without actually selecting a token first

Win Message



Win message displayed for Player 2 once
Player 1 only has 2 black tokens remaining