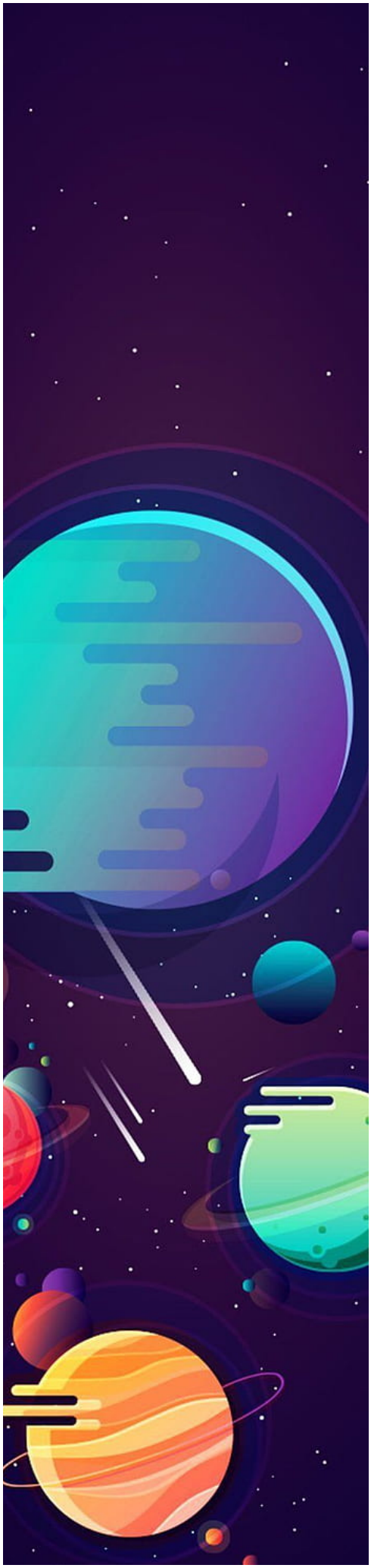


FIT3077
Sprint 1
Team Rocket

MONASH University

Team Information

Team Name and Team Photo

Team Name: Team Rocket

Team Members: Rohit Valanki, Neth Botheju, Karan Luthra

Team Membership

Team Rocket

Neth Botheju

Bachelor of Software Engineering

E: nbot0003@student.monash.edu

Faculty of Information Technology of:

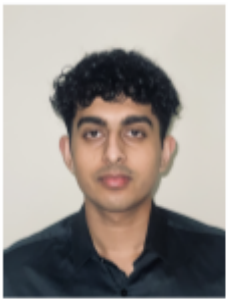
MONASH University

Fun Fact:

"My comfort drink is milk tea!"

Open Day Access

Team Rocket



Rohit Valanki

Bachelor of Software Engineering & Commerce

E: rval0008@student.monash.edu

Faculty of Information Technology of:

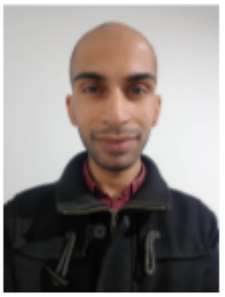
MONASH University

Fun Fact:

"I am scared of birds"

Open Day Access

Team Rocket



Karan Luthra

Bachelor of Software Engineering

E: klut6@student.monash.edu

Faculty of Information Technology of:

MONASH University

Fun Fact:

"My favourite musician is Floating Points."

Open Day Access

Technical and Professional Strengths:

Neth Botheju

- Proficient in multiple front-end and back-end languages such as Python, Java, JavaScript, HTML/CSS, React and SQL
- Currently in the third year of a Bachelor of Engineering degree with a specialisation in Software Engineering.
- Works at a technology retail company and is experienced in repairing mobile devices' hardware and software troubleshooting.
- Engaging in communication skills and working in multiple customer-facing roles to get better confidence and experience speaking to different people with different levels of technical expertise.

Rohit Valanki

- Proficient in multiple languages such as Python, Javascript, C, Java and SQL and frontend languages such as HTML/CSS
- Experience using frontend frameworks such as React JS as well as backend architecture services such as Flask and APIs.
- A team player with great communication and collaborative skills who enjoys working in fast-paced group environments

Karan Luthra

- Comfortable with Python, Java, SQL, TypeScript and HTML/CSS.
- Completed a full-year project building a mobile application for Monash Health using React Native.
- Has a Bachelor of Human Science granted as an exit pathway from the first three years of a medical degree; has a strong interest in applications of software, particularly AI, to healthcare.
- Most comfortable in a position that allows for a mixture of product management and coding responsibilities.
- Enjoys public speaking and working with a team, but prioritises allocating time for solo development and creative expression.

Team Schedule

Team Meetings

As per the first meeting, team members have agreed to allocate Sunday afternoons every week for a weekly stand-up meeting. If a member cannot be present for the agreed time or has prior commitments they are expected to notify the team before the meeting in order to reschedule. If a member is unable to attend the meeting due to unforeseen circumstances, the meeting will still go ahead and the absentee will be caught up to speed from a meeting minutes sheet written by the attendees.

The meeting minutes will have the things discussed, any conclusions or decisions made, and tasks allocated. This serves as a responsibility for all members to be aware of their expected workload and what they must complete by the next meeting.
Members will not have any expected times or hours they need to work on the project, however, the expectation has been set that they will dedicate a fair and reasonable amount of time to the development of the project each week. These developments will be logged on the contribution log each week so that the team can keep track of what tasks have been done and what haven't.

The additional use of project management tools will be utilized throughout the development, testing, and documentation phase of the project to ensure progress is measured and kept track of.

Task Allocation

Tasks are arranged by level of difficulty and time required rather than allocated to individual team members based on their skills. To avoid silos of members working on single deliverables or specific functionality, each deliverable is split into three roughly equal workloads, decided and approved by all team members.

Team members have the freedom to pick whichever tasks they would prefer to do however in the event that members don't have a preference, tasks are randomly allocated with no bias and members have the opportunity to reject or accept their given tasks. Task allocations will be finalised with the guarantee that all team members have similar difficulties and workloads to complete. This method promotes fairness and grants members the authority to take on tasks they are happy with.

Each user story should be divided into the Design, Implementation GUI, Implementation Back-End, Testing, and documentation aspects in its description so that it is easier to track the progress and put it into commit messages. The user stories will be broken down into their respective parts and expectations which clearly outline what it will be expected to look like on the GUI, what it will do in the back end section, and what the final outcome will look like in terms of overall functionality.

Technology Stack and Justification

Language Considerations:

Java

Java was one of our main suggestions for this project since every member in this group has used Java when learning about object-oriented programming. The main advantages and disadvantages of Java can be summarised as follows:

Advantages:

- Java is a fully object-oriented language. All variables and functions in Java must be defined with classes, effectively enforcing OOP rules
- Java is a compiled language, which results in better performance in comparison to interpreted languages such as Python
- Java has various libraries and frameworks that support frontend development
- Java is statically typed, meaning variables and classes must explicitly define data types. This enforces good data consistency practices that adhere well to OOP.

Disadvantages:

- Convoluted syntax relative to other languages such as Python and C#, which may result in decreased readability and maintainability of production code
- Java runs on top of Java Virtual Machine, making it consume more memory and relatively memory inefficient
- Java has automatic garbage collection, which can hinder performance as developers cannot directly allocate and deallocate memory.

Python

Python was a primary alternative to Java, as a highly readable language with strong support for object-oriented programming and a high level of comfort for all team members considering the use of Python in the context of other university units. However, concern existed over the potential for dynamic typing and no compile-time checking to lead to messy code without significant review.

Advantages:

- Clear, readable syntax with minimal boilerplate code, allowing developers to easily read and refactor the existing codebase.
- Support for key object-oriented concepts including polymorphism and interfaces through abstract base classes, as well as useful tools like generics.
- Comparable popularity to Java, giving rise to a mature ecosystem of libraries and ample documentation and community support.
- Support for wrappers and metaclasses to significantly reduce boilerplate code, such as properties and data classes.
- Interpreted, allowing developers to quickly write and test snippets of code without the need for compilation, permitting faster learning and turnaround on new ideas.

Disadvantages:

- While type hinting and static type checkers are available, dynamic typing does not reach the level of type safety achieved by statically typed languages like Java and C#, increasing the risk of bugs or rushed implementation.

- Weaker performance than typically compiled languages like Java and C# due to the need for garbage collection and less information with which to optimise code at run time.
- Limited support for app development on a wide variety of platforms without the use of cross-platform enabling libraries like Kivy, which perform worse and are far more limited than development using natively supported languages. This is particularly damning given that 9MM is likely to have greater appeal to mobile than the desktop-based market.

C#

C# offers another suitable alternative, as a statically typed language geared towards object-oriented programming with a focus on Windows applications. However, although C# was considered, it was a less comfortable choice for all team members; only one team member had limited experience with the language and the others had none.

Advantages:

- Similar syntax to Java due to common ancestry with C++, smoothing the learning curve for developers and creating an opportunity to try a new language.
- Support for static typing, allowing a comparable level of type safety to Java.
- Boilerplate code reduction features such as properties.
- Prominent programming language like other alternatives, with a strong community and support from Microsoft.
- Closer relationship to C than other alternatives, which was of interest to the team from an educational perspective given the influence of C on high-level languages today.
- Compiled language, allowing better performance and compile-time type checking.

Disadvantages:

- *Significantly* less comfort and pre-existing knowledge for developers, likely leading to a decreased pace of development in early sprints while picking up the language. This was a significant factor for the team.
- Strongly associated with the Windows ecosystem compared to the portability of Java, despite some cross-platform support through Xamarin and .NET Core.

Chosen backend language:

After weighing the pros and cons of the various approved backend languages, our group has decided that Java would be the most appropriate language for this project. Java being an object-oriented language that is compiled and has various libraries and frameworks that support frontend development, makes it the perfect language for this project. Furthermore, each member has experience using Java unlike C#, which only one member has used previously. Java also offers more type safety than Python which is great for building a robust program while learning new design patterns.

Front-End Programming Considerations:

Java FX

Out of the many frameworks available for Java GUI, JavaFX is one of the newer flagships of Oracle to be used. Despite its latest breakthrough into the GUI rankings, it is counted to be one of the best among the top few for its rich desktop interfaces with new libraries and cross-platform support. JavaFX is included in the Java Development Kit (JDK) and allows developers to create applications that are compatible among various operating systems like Windows, MacOS, Linux and more.

Swing

Among the GUI toolkits, Swing is one of the older frameworks for desktop application development. It is a part of the Java Foundation Classes and was introduced as a replacement for Abstract Window Toolkit. However, despite its maturity, Swing remains a widely used GUI kit and continues to be supported and maintained by Oracle.

Swing provides a rich set of components like buttons, text fields, labels, tables and trees with more advanced components such as tabbed panels, sliders, progress bars and more. It provides what is called a pluggable look and feels architecture which allows developers to customise the appearance of their application

Apache Pivot

Unlike the other two, Apache Pivot is an open-sourced platform for building rich internet applications through Java. Apache Pivot also provides a useful set of GUI components such as buttons, text fields, tables, trees, charts, and multimedia players, as well as a bit more than Swing with features such as data binding, drag-and-drop, animation, and skinning.

Apache Pivot is built on top of the Java Standard Edition platform, which allows it to support a variety of other programming languages such as Groovy and JavaScript.

GUI Framework	JavaFX	Swing	Apache Pivot
Components - The amount, quality and applicability of the components of the toolkit and how much can be achieved	JavaFX does not feature a lot of individual components. As an up-and-coming GUI, it does not possess the advantages that legacy APIs that other Toolkits possess	Swing has a lot of components and a wider library to access from. As one of the first few GUIs, Swing has a full-featured Legacy library and full access to APIs.	Apache Pivot has a lot more native components than Swing and JavaFX but however because its also a web development app, its library is limited in the way that can be utilised for a desktop app
User Interface	For a modern GUI, JavaFX has a lot more modern interface looks and feels. Despite the limited amount of components, JavaFX allows developers to create with advanced looks and feels to their UIs	Despite the wide range of components, Swing uses Standard UI designs for developers to achieve what they need without any modern-looking interfaces.	Pivot also has the ability to very quickly “theme” the UI with a whole different set of colours and icons, simply by changing the master configuration file. Since it is open source you can make your own with just as rich a set of functionality as any others.

Development	JavaFX is mainly used for desktop app development using Java. It comes with Java Development Kit (JDK) so additional installation is not needed. Its versatility makes JavaFX cross-compatible among most operating systems.	Being a mature and developed GUI, Swing is still fully supported and maintained by Oracle however installation is needed to get Swing up and running. This maturity allows Swing to have various IDE's which offer a tool for rapid development	One of the advantages of JavaFX is the ability to declare the UI using FXML, this advantage was first developed on Pivot and hence it has been enhanced and allows for easy customisation.
Functionality	The biggest advantage of JavaFX up and coming status is the ability to grow and introduce new functionality in the future. This opens the gateway for JavaFX to expand, develop and improve its look as more libraries and tools are introduced. This allows JavaFX to concur with the times.	Due to Swing's age, it is only currently being supported and maintained by Oracle. This means no new functionalities are to be released for it and no planned developments are available	Like JavaFX, Apache is a growing and developing GUI which allows any apps built on it to keep up with modern Looks and Feels and overall adapt to trending changes made with user interfaces in the near or far future.

Front-End Programming Chosen GUI:

After considering and weighing the pros and cons, the team decided to choose Swing to build the user interface for the Nine Men's Morris Game. The reasoning behind this decision relies on the resources and tech community support readily available when learning Swing. Despite Swing being limited in customization and advanced UI looks and feels, its maturity allows new users to pick it up quickly through well-versed youtube videos and free online courses.

However JavaFX was voted easiest language to learn for a beginner out of the three, the reason why it did not appeal to the team was that JavaFX is still a relatively new language so resources to thoroughly learn it having no Java front-end experience by any of the team members made it a less reliable choice. Another downside to JavaFX to add it its recent emergence is that due to this, debugging and community support are less frequently available whereas Swing has common problem debugging answered by multiple tech community websites like StackOverflow.

This problem was also prevalent in the use of Apache Pivot, hence it would be more difficult for the team to learn and debug in the short development time given for the project. Apache Pivot is also built for developing web applications and while its components and toolkit can conform to desktop apps, its full potential lies in web development. This makes JavaFX and Swing more suitable candidates for our desktop game application.

Overall, Swing was the chosen GUI for its broad legacy library of components, ease of learning and use and abundant support in the tech community.

Project Management Software Considerations:

Jira
Jira provides ready-to-use Scrum and Kanban boards that align with the quasi-agile development approach that will be used for this game. Allows for easy collaboration with teammates, where all members can view and update shared Kanbans. Roadmap features provide a clear progression of assigned tasks, currently in progress tasks and completed tasks of each team member, making it easy to track team progression.

User stories that are added to the backlog can be given priorities and difficulties which will help determine the order in which user stories should be completed during development. Supports integration to GitLab, which can allow us to directly manage code committed to user stories. Roadmap features provide visual progress of tasks completed.

Trello
Trello is another Atlassian software, that provides KanBan-style project management and progression software. Similar to Jira, Trello allows for easy collaboration between team members, with 3 separate sections for to-do, currently in progress and completed user stories in the backlog. Trello provides various templates such as Kanban, agile board template, project management etc, which are all accessible with the education package. However, the main drawback of Trello, is that user stories are not as customisable as Jira user stories, where we are unable to easily add priorities and difficulty to each user story. Furthermore, there are limited visual progression tools apart from just the basic Kanban.

LucidSpark
LucidSpark provides us with empty templates that allow us to model any form of document we want, ranging from project timelines to sprint dashboards. The customizability makes LucidSpark an attractive option as we have complete control over how we would like our team's dashboard to look like, vs other software which only allows you to add stories and not customise the board itself.

However, even with the education pack, we are limited to only 3 documents we can create on all Lucid-related platforms. Furthermore, project timeline, Gantt chart, sprint dashboard and other agile-related templates are premium features even with the education pack, which makes LucidSpark an extremely limiting software.

Project Management Chosen Software:

After comparing the pros and cons of Jira, Trello and LucidSpark, we have decided that Jira would be the best choice as our project management tool. Jira provides access to multiple tools with respect to our agile development process, whilst giving access to templates for free, unlike LucidSpark. Furthermore, Jira has additional features that Trello lacks, such as priority ranking of user stories and separate project progression charts. As a result, Jira would provide the best tools to manage this project.

User Stories

Basic Game Play

As a player, I want to place a token into an empty intersection, so that I can set up a mill.

As a player, I want to know when it is my turn to move so that I can place my next token.

As a gameboard, I want to detect when an intersection is free so that I can mark it as open

As a player, I want to know which intersections are open, so that I can know where to place a token

As a gameboard, I want to detect when an intersection is taken, so that I can mark it as closed

As a gameboard, I want to alert the player when a token is placed on a closed intersection so that the player knows an illegal move was made

As a gameboard, I want to alert the player when they select a token that cannot be moved so that the player knows a closed token was picked

As a player, I want to know where I can move placed tokens, so I can move the token to set up my next move

As a gameboard, I want to know when a player places all 9 pieces down so that I can allow the player to move a piece currently on the board

As an experienced user, I want to be able to move a token horizontally or vertically across the board so that I can attempt to find ways to create mills

As a novice user, I want to be able to have visual hints to see where I can move my piece so that I can make the right move and understand the game

As a novice user, I want audio feedback when I attempt to make an illegal move so that I know I can't move my token to that intersection

As a novice player, I want to receive audio feedback when the token I have selected cannot make a move so I know I cannot use that piece until there is an empty intersection nearby.

Token Free Movement

As a player with 3 tokens, I want to move my pieces freely to any intersection, so that I can recover my position.

As a player with 3 tokens, I want to move my pieces freely to any intersection, so that I can form a mill.

As a player with 3 tokens, I want to move my pieces freely to any intersection, so that I can block my opponent from forming a mill.

Mill Formation
As a gameboard, I want to detect three pieces placed horizontally, so I can mark them as a mill.

As a gameboard, I want to detect three pieces placed vertically, so I can mark them as a mill.

As a player, I want to place a new token on an intersection so that I can form a mill.

As a player, I want to place a new token on an intersection so that I can block my opponent from forming a mill.

As a player, I want to move an existing token to an intersection, so that I can form a mill.

As a player, I want to move an existing token to an intersection, so that I can block my opponent from forming a mill.

Mill Offence
As a gameboard, I want to detect tokens that are not in mills, so that I can make them targetable.

As a gameboard, I want to detect tokens that are in mills if no tokens are not in mills, so that I can make them targetable.

As a player, I want to remove an opponent's token from the game when I form a mill so that I can gain an advantage.

As a player, I want to see clearly when a token has been removed, so I can register a significant event in the game.

Mill Defence
As a gameboard, I want to detect tokens that are in mills, so that I can make them immune to opponent attacks.

As a player, I want my token to be immune when placed in a mill so that I can avoid losing tokens.

As a player, I want to know which of my opponent's tokens can be removed, so that I can determine which removal gives me the greatest advantage.

As a player, I want to know which of my opponent's tokens are defended by a mill, so that I can plan out my offence ahead of time.

Winning

As a gameboard, I want to detect when a player has two tokens left on the board so that I can finish the game

As a gameboard, I want to detect when the current player has no legal moves so that I can finish the game

As a player, I want to see when I have won, so that I can rub it into the other player's face

Game GUI

As a general player I want to have a main screen so I know what game I am playing

As a novice player, I want to be able to have the option to view the rules so that I can understand how to play the game before beginning

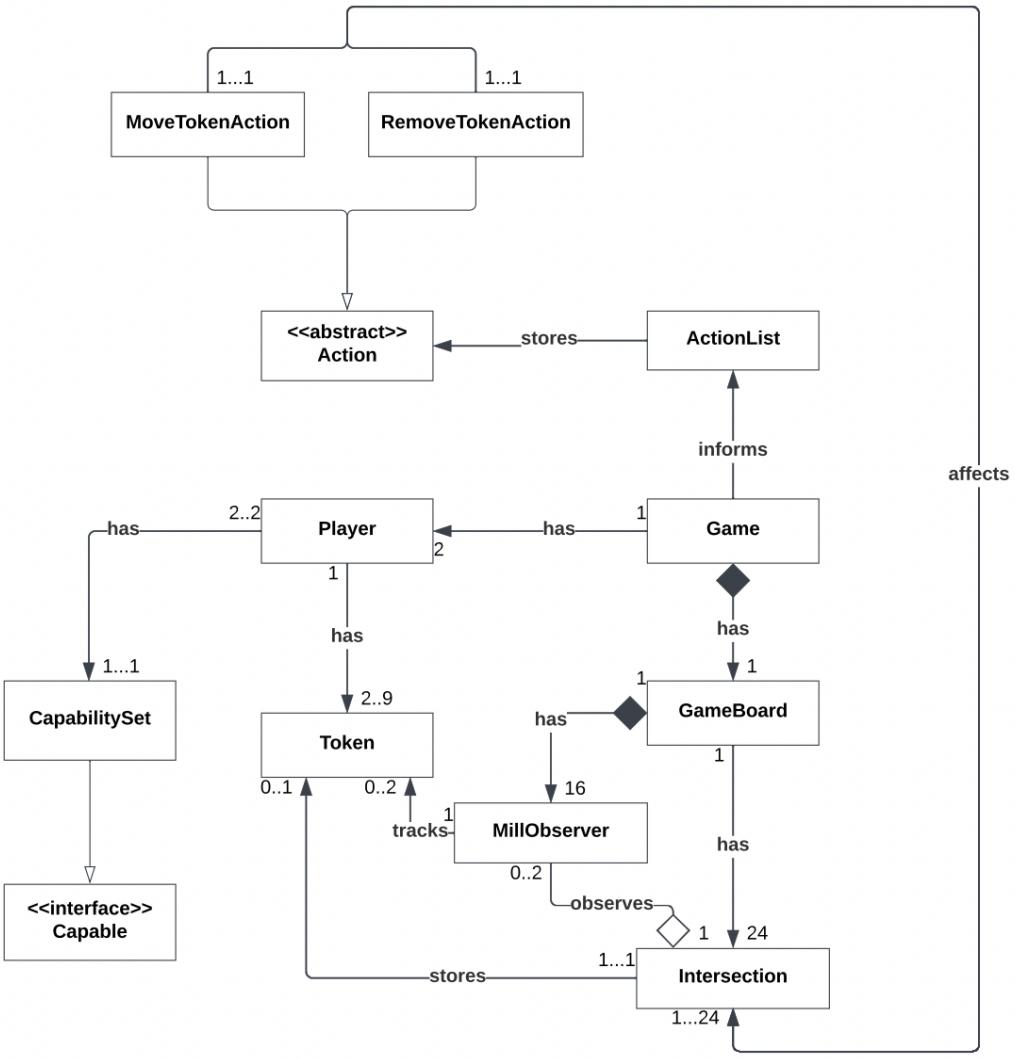
As a general player, I want to be able to have a play game option so that I am prepared and know that the game will begin

As an experienced player, I want to see where the 18 tokens are on the board so I know which tokens I can move to begin playing

As a general player, I want to see how many tokens I have left out of 9 so that I can plan my next move

Basic Architecture

Domain Model



Design Rationale

Game
The Game class encapsulates the core game logic and comprises the highest-level class in the program. It is responsible for running the tick function that runs each turn in the game, as well as tracking which player is currently playing their turn and awarding victory. As such, the Game has a dependency on the Player, from whom it retrieves the player's capabilities for each given turn. Given these capabilities, the Game can generate an action list to affect the game. The Game class is also composed of a GameBoard, from which it can glean information about which Intersections are accessible to a player to interact with each turn or whether mills have been formed, allowing the Game to enqueue the appropriate Actions.

GameBoard
The GameBoard is responsible for storing references to the Intersections in the game, from which it can pass information to the Game class to help determine the permitted actions in a given turn. The GameBoard can use its references to check which Intersections are full. Additionally, the GameBoard is composed of MillObservers, from which it can learn whether or not a mill has been formed; this allows it to inform the Game class of exactly which Intersections are not accessible for RemoveTokenActions and whether a RemoveTokenAction should be enqueued.

MillObserver
The MillObserver follows the Observer design pattern, watching with each tick to check whether the Intersections to which it has been attached are carrying a Token. MillObservers can check whether or not the Intersections to which they are attached are storing a mill by comparing the colours of all of the Tokens associated with those Intersections; if the colours are identical, a mill is present.

Alternative designs considered include providing the Intersection with an algorithm to traverse neighbouring Intersections and find a mill, as well as storing Intersections in a coordinate system in the GameBoard and having the GameBoard check for similarly coloured tokens in sets of coordinate combinations. However, in comparison to these alternatives, the MillObserver offers a focused solution with no need to bloat the Intersection or GameBoard class, a clear responsibility for the class, and clearly defined design terminology with greater readability for future developers.

Token
Tokens represent the 'men' in the 9MM game. Tokens are not aware of their Players to avoid a circular dependency and are instead aware only of a colour attribute with which it is initialised, which matches its Player's colour attribute. Importantly, the Player who owns the Token also retains a reference to the Token, allowing the Player to have a constant awareness of how many Tokens they have remaining so that they can receive the appropriate Capability. When a Token is removed from an Intersection, its instance is deleted, nullifying the reference being held by a Player. The Token is therefore a simple but powerful representation of the data of a player's ownership over Intersections.

Player
The player class represents the two players of the game, player 1 who uses white tokens and player 2 who uses black tokens. The player class has reference to tokens, allowing the player to know at any time how many tokens out of 9 it has remaining. The player is given a special capability set, which describes what capabilities the player has at the current state of the game (the player can make a jump move, and the player can remove a token). Based on the current capability of the player, the game will grant the player an appropriate action for the move, such as removeTokenAction which is granted when the player makes a mill or MoveTokenAction which is granted when the player has already placed all 9 tokens and now must move one of its placed tokens. Thus the player is an important entity in the game, as the player keeps track of their own token as well as receives actions from the game based on the current state of the game.

Capable interface + CapabilitySet
Capable is a special interface used to identify different “capabilities” that players can have. A capable instance effectively describes the current legal moves that a player can perform. The player will check its current token bank; if the token bank is not empty the player will have a capable instance such as “placeable” which tells the game the player can place a token. If the token bank is empty the player could have a capable instance “moveable”, which tells the game class that the player can only use the move action to move a piece of the board. The capabilitySet will store all the capable instances the player has at the current time and the game will check the player's capabilitySet to determine which action to grant the player. Having a capable interface supports the extensibility of the game, which supports the open-closed principle, as new capable instances can be added through the interface without having to refactor the game layout. Thus having a Capable interface and CapabilitySet is important so that the game can check what actions the player can be granted for each iteration of the game.

Intersection
The intersection class is the heart of the game and is used to represent one of the 24 intersections that appear in the nine men's morris game. The gameboard will have reference to 24 instances of the Intersection which will make up the game board, and each intersection will have a relevant connection to other intersections to make up the overall board. Each intersection contains a container to hold a token, and when a player places a token, a reference of the token will be stored in the intersection. This will then allow the mill observer to check whether a token is present at the intersection and whether a white or black token is placed. The player's possible actions, MoveTokenAction and RemoveTokenAction, will result in a token being placed or removed from the intersection instance. This represents the basic gameplay that the player will do. The intersections adhere to the single responsibility principle as it is effectively only given one responsibility, which is to handle token placement and removal from itself, whilst other classes, such as MillObserver, can utilise the intersection to complete other tasks.

Action
The abstract action class is set to provide a base for all the possible actions that the tokens can take when the player chooses to make a move. This was created to abide by the object-oriented principle of do not repeat yourself hence allowing the classes it extends to take on methods from the abstract action class and further add additional methods. Having this class also allows the domain to abide by the Open/Closed principle as it makes its

extending actions implement new features by new code and is loosely coupled with the ActionList class which makes it easy to maintain and test.

ActionList
The concrete class ActionList relays information to the Game class by letting the Game know during a playing turn, which possible moves a player can make. All actions a player depend on the state of the game and the player's current status, for example, if the player is currently playing a turn to move the token and makes a mill, ActionList keeps track of the actions, to move a token and then to remove a token so that the game allows the player to remove a token off the board in the same turn before moving to the next player. This was implemented in such a way because there needed to be a way for the game to know a player can still remove a mill, even though they have already played their moving token action.

MoveTokenAction
The concrete class MoveTokenAction extends from the abstract class Action, allowing to reduce redundancy of the Action Class's code and avoid violating the DRY principle. This class is responsible for letting the player move their chosen token from one intersection to another. Hence the association with the Intersection Class as the MoveTokenAction takes the information from that class to determine which token is attempting to be moved and to which receiving intersection. The MoveTokenAction class determines how many intersections from the currently selected intersection the token can move to, which depends on the Player's current capabilities. For example, if the player has the capability "JUMP" then the MoveTokenAction allows the game to let the player move to any open intersection on the board, whereas if it only is playing a regular move, the player can only move on an intersection adjacent. This was done in such a way that it reduces unnecessary code repetition and removes the need for an extra class like JumpToken which does exactly the same thing but with a wider range of intersections.

RemoveTokenAction
The concrete class RemoceTokenAction also extends from the abstract class Action, however, unlike MoveTokenAction, this action takes the association with the Intersection class to let it know the token on it needs to be removed entirely from the game. This action only occurs when a player has made a mill and would like to remove an opponent's non-mill tokens. The class extends from the abstract class Action to avoid violating the DRY principle but to also separate the difference in the action from the MoveTokenAction class. Despite the action initially doing the same thing, the RemoverTokenAction is deployed by the player only when a mill is formed and only to take a token off the board and not to move it anywhere, making it very different from the MoveTokenAction Class's methods.

Basic UI Design

Provided in Git and Separate File.
Figma Link for more clarity:
https://www.figma.com/file/zykWeEoB9Swq8gFw1tNX4C/FIT3077-Low-Fi-Prototype?node-id=243%3A1&t=sSGORiA3cDsURhby-1