MONASH University

FIT3077

SPRINT ONE

Team Nine Bytes

SUBMITTED BY

Shyam Kamalesh Borkar (32801459)

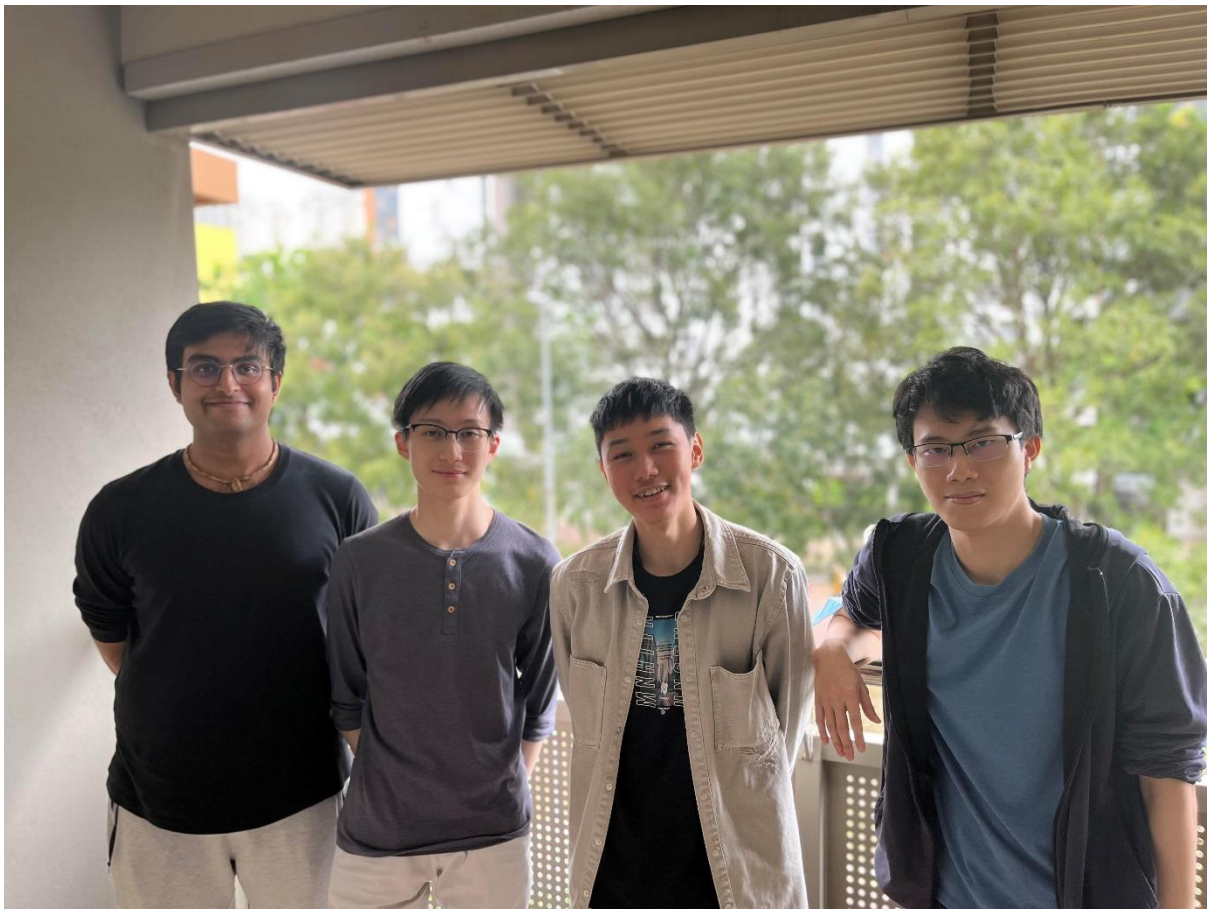
Lai Carson (32238436)

Richardo Husni (32767269)

Victor Lua (31362834)

Team Information

Team Name and Team Photo



Hello we are,

Nine Bytes

Team Membership

Basic Information

Name and Student ID	Contact Number	Student Email (@student.monash.edu)	Program	Programming Languages Known (in order of proficiency)
Lai Carson (32238436)	60 162125882	lcar0029	Software Engineering	1. Python 2. Java 3. JavaScript 4. MATLAB 5. HTML, CSS
Richardo Husni (32767269)	60 132258399	rich0006	Data Science	1. Python 2. Java 3. R 4. HTML, CSS
Shyam Kamalesh Borkar (32801459)	60 1123249218	sbor0018	Software Engineering	1. Python 2. Java 3. HTML, CSS 4. MATLAB

Victor Lua (31362834)	601123371225	vlua0001	Computer Science	1. JavaScript 2. Python 3. R 4. HTML, CSS 5. TypeScript 6. Haskell 7. MATLAB

Member strengths and fun facts

Lai Carson has had experience in previous university projects which utilised both procedural programming and object-oriented principles. He has also been involved in projects that used test driven development, which involved writing unit tests for source code. Over the summer break, he created a web application that aggregates news articles from different sources on the internet. This introduced him to the basics of web development, and helped him discover that this is where his passion lies!

Fun fact about Carson: Carson likes playing MMORPGs (massively multiplayer online role-playing games).

Victor Lua has had experience in project engineering and has developed a few computer projects. He has experience in coding a project in TypeScript, making a small game that runs synchronously. He has experience working with others in a group project. Other parts are that he also has experience in Algorithm and Data Structure and image processing.

Fun Fact about Victor: Victor wanted to be a badminton player when he was younger.

Shyam Kamalesh Borkar is a second-year software engineering student. He is an enthusiastic learner; he is proficient in programming languages like Java and Python. He is familiar with the use of git for development purposes as a result of using it effectively in prior software projects. He has used Python in many projects like creating lightweight applications using the Python tkinter library and more. Through a previous project that involved creating a Java-based terminal game, he developed an interest and expertise in object-oriented design and implementation. He is also skilled at creating clean UML diagrams to represent system designs.

Fun Fact about Shyam: When Shyam was younger it was his dream to be a professional footballer.

Richardo Husni has had a few experiences in programming and data science. He had contributed in the development of the Pokémon Game previously in the unit FIT2099 which utilized his knowledge of OOP concepts and Java programming. Moreover, he has completed an internship before where he worked collaboratively in a team in the data science field, gaining valuable experience and skills mainly in data visualization. He has also done a few personal projects in the data science field to further expand his expertise in it especially R and Tableau.

Fun Fact about Richardo: Richardo likes to sleep for more than 8 hours and sometimes less than 5 hours.

Team Schedule and Work Distribution

Team Meeting Schedule

The team will meet on the following days and times from the table each week to discuss matters related to the sprint and project. They might be subject to change depending on various variables affecting team members. Any member that is unable to make it to the meeting must notify the team through the communication channel in advance upon which the team might decide to reschedule that particular meeting or continue without them if needed.

WEEKLY MEETINGS			
	DAY	**START TIME**	**END TIME**
Meeting 1	Wednesday	9 PM	11 PM
Meeting 2	Sunday	8 PM	10 PM

Note: All meetings will take place on zoom.

Any additional required meetings will be scheduled on the go especially when nearing the sprint deadline dates. Stand-up meetings can also be scheduled if any member feels the need for it.

Regular Work Schedule

All members will regularly attend the weekly forums tutorials and engage in learning the content. During tutorials all members will actively participate in the tutorial activities and also make use of the face-to-face setting to clear any doubts or discuss sprint related matter that is best discussed face-to-face. If all members are interested, they might even stay back a couple of hours after the tutorial class to conduct team discussion or even work on the project.

Members will contribute and complete their assigned work related to the sprints that requires no collaboration in their own free time during the weekdays and the weekends. They will also make time if needed to understand and learn new technologies and the application of design patterns for use in the project. Members will attend all the scheduled meetings and actively take part in discussion and conversation to make advancements in the project sprints.

How Workload is Distributed and Managed

Distributing workload within a team can be tricky sometimes, but it is important for the team to achieve its goals and for team members to learn and grow from the experience. The first step in distributing the workload is to understand the deliverables and the tasks that are required to be complete. So, during meetings and discussions the team will highlight all the work that is to be completed and which of them are critical for the team to complete sooner. Another aspect of distributing workload and managing it is effective communication between team members, specifically about delegated tasks, expected quality, deadlines and more. This will make things clear for each member on how they are contributing to the team and what is expected from them.

Regular stand-up meetings are also crucial in managing workload to monitor and understand the progress of each member, and also provides the opportunity for help to be given to members facing problems and also offer feedback on their work. In addition, upon discussion during these meetings it is also possible for workload to be adjusted among team members based on unexpected difficulty of tasks at hand.

Working in silos means that there is no collaboration between team members and each member works on an entire portion of a deliverable. This can be detrimental to the success of the project and can harm the learning and growth of each member as part of the team. To avoid working in silos the team members will divide similar work portions among themselves and at the same time work in a collaborative teamwork environment. This will ensure that nobody is over worked and everybody in the team is contributing equally in all parts of the project. Using collaborative tools like Lucid Chart, Figma and others will help promote collaboration and allow all members to work on the same deliverables at the same time avoiding siloing.

With careful planning, consideration and communication the team will distribute workload equally among all members.

Technology Stack

Candidate Programming Languages

The team has decided to choose between **Java, C++ and Python**. Everyone in the team has had experience in projects which primarily use Java and Python, but C++ is a language that is new to everyone in the team.

Comparing Java with C++

The team included C++ as a candidate programming language based solely on the fact that it is a common language used in the game development industry. Although C++ and Java are syntactically similar, there are many differences between the two. Java clears unused objects (garbage collection) periodically, and this process cannot be controlled by the developer. In C++, memory has to be freed up manually, although it can still be automated if desired. This is one of the reasons why C++ is such a popular choice for games. The ability to precisely determine when memory should be freed allows for quicker applications. However, this is a double-edged sword for beginners like us, since poor memory management will undoubtedly lead to slower applications. It is also possible for C++ to exhibit unusual behaviours, where indexing goes beyond array bounds, or when uninitialised variables are referenced. Java prevents this by throwing run-time errors to signal when something is wrong with the code. C++ does not force object-oriented programming (OOP) principles, since a global scope exists, variables and methods are not required to be strictly kept within classes. Additionally, C++ allows multiple inheritance – a design choice that cannot be implemented in Java. To summarise, C++ is a language that requires relatively more micromanagement since it allows more control over resource management when coding, with the goal of optimising performance.

Comparing Java with Python

As with C++, Python does not enforce OOP like Java does. Encapsulation boundaries can be crossed if one does not pay much attention. It offers some flexibility at the cost of introducing code smells. Besides that, Python syntax is different from Java and C++ (e.g. how a code block is defined in each language). This, however, would not be an issue for the team.

Candidate Frameworks and APIs

1. Pygame (Python)

Pygame is an application programming interface (API) specifically for game development in Python. Classic games such as Chess, Tic Tac Toe and Tetris have been previously implemented using Pygame. It allows developers to create open-source, and even commercial games, completely free of charge.

Pygame has all necessary support to develop a basic 2D game. This includes basic features such as vector art rendering, input listeners, event listeners and queues. It also introduces more advanced game development functionalities like coordinate systems, time control, audio, and a lot more since Pygame can be used together with other external python libraries. All these modules can be used independently.

Pygame is easy to learn since an existing GUI is not needed to test a feature. Developers can experiment by typing code snippets in the console. There are also plenty of resources online for beginners, such as a comprehensive documentation and tutorials made by the community. Pygame is highly portable as the developed application runs on every system. It is frequently updated (every few weeks), and the last stable patch was released on 14th of March, 2023.

2. JavaFX (Java)

JavaFX is an API for general application development built into Java. JavaFX is capable of creating UI components like calendars, forms and charts. Its last stable release was 13th September 2022. Like Pygame, it supports image rendering, animations and user input handling. UI elements are styled using CSS syntax, which is helpful since everyone in our team is familiar with basic CSS. UI can also be abstracted and reused, although that may be less useful in the context of developing of a simple board game. JavaFX is integrated with Swing (another UI framework for Java), so using JavaFX also provides tools from Swing. Additionally, applications developed using JavaFX can be used on mobile, not just on desktop.

JavaFX has a well-organised documentation that explains each function and module in great detail, complete with examples – one that is more comprehensive than Pygame's. However, JavaFX does have a steeper learning curve than Pygame due to the variety and

amount of functionalities it offers, not all of which will be helpful for our game. JavaFX has a testing library called TestFX, that uses robots to simulate user interaction.

Although JavaFX might be designed for more "general-purpose" software and UI, it offers a special game engine library called FXGL that is syntactically similar to JavaFX. This library complements JavaFX by including common game development functionalities that are missing in the original API. FXGL has entity animations, networking, customisable UI looks and game update loops, which may help to simulate a turn in our game of 9 Men's Morris. An interesting aspect that FXGL provides is application states – the ability for blocks of conditional logic to be swapped around or turned on and off based on the state of the game. FXGL's last patch was 27th March 2023.

3. jMonkeyEngine (Java)

jMonkeyEngine (also known as JME), is a game engine API written in Java, for Java applications. JME is free to use and games developed using JME can be published. JME offers extensive support for 3D games such as entity physics, shaders and 3D rendering. However, it does have libraries for the implementation of 2D games (Lemur and IGUI). JME has interfaces for update loops and application states, similar to FXGL. It also supports input detection and processing – a feature common across all APIs that have been discussed thus far. Unique to JME is its Nifty GUI library. Nifty GUI is implemented using layers and panels similar to HTML wrappers – a concept that everyone in the team is familiar with. It also has mouse picking, which is the ability for cursor to select entities and interact with them and manipulate their properties in multiple ways. Heads-up display (HUD) can also be implemented in JME, which may be useful for the tutorial mode that the team will eventually implement.

There are advanced modules such as networking, custom keyboard configuration, particle effects and game physics which may add aesthetical value to the game, but are not necessarily required for this project. Regardless, JME is modular – the team can choose to only use certain modules offered by JME. To add on, each module has multiple libraries that developers can choose from to suit their needs, and they can be easily found using the search function on JME's website.

JME provides many niche features which may be an overkill for a simple 2D board game. As JME was designed to develop 3D games, the game engine (at first glance) is hard to

learn, although it is modular. JME is, however, well documented just like JavaFX. There is also a community forum where developers engage in frequent discussions.

4. **Qt (C++)**

Qt is UI framework, similar to JavaFX, but for C++ and Python. It has capabilities almost identical to JavaFX. Graphics can be designed using in-game tools or be imported as images. It has user input detection, UI state and animations, as well as precise or relative element positioning. One interesting aspect about Qt is its syntax – it is declarative and uses a json-style syntax for designing UI, which makes code more readable and hence easier to understand.

One major downside of Qt is that it was not designed for game development. While JavaFX has FXGL to compensate for its lack of game development functionalities, there does not seem to be any additional module that can be used together with Qt that helps to create game UI. Furthermore, the team would have to code a working game engine from scratch as Qt is merely a UI framework.

5. **LITIENGINE (Java)**

LITIengine, also known as LITI, is an open-source game engine produced specifically for Java game developers. It is a lightweight game engine that provides a 2D game framework with features like game loops, input handling, collision detection, and more. In addition, it has its own built-in physics engine based on Box2D. LITI also comes with a project management and mapping tool called the utiLITI editor that makes development with the engine simpler. Not only does LITI provide easy integration with other libraries and tools but it also comes with cross-platform support allowing developed games to be run on Windows, Linux and macOS. The LITI website provides comprehensive documentation with detailed tutorials with a great lookup library in their API reference. The LITIengine API is designed to be easy to learn, and beginners in game development can do great things with it. Its 2D render engine was built entirely using plain Java AWT graphics, which is an impressive achievement.

Overall, LITI is a great API/engine, and being a lightweight game engine, it makes a great candidate for building simple board games like 9 men's morris along with its beginner-friendly Java construct and its amazing 2D game development features.

6. **libGDX (Java)**

   libGDX, like LITIengine, is an open-source game engine that can also be used with other programming languages than Java, like Kotlin and Scala. In terms of features and capabilities both libGDX and LITIengine are similar. They both support game features that include game loops, graphics rendering, input handling, game loops, and asset management, and they both also support cross platform use. Not to mention both of their physics engines are based on Box2D.

   The key factor that differentiates libGDX from LITIengine is that it also offers support for 3D graphics, which would be beneficial for the team if they were to make a 3D version of the 9 Men's Morris game. To add to the differences, libGDX has a larger community and more resources available online.

   At the end, the choice comes down to the needs and preferences of our team and their preferred programming language to work with.

Chosen Technology Stack

The team has decided to go with Java as the programming language for the development of the Nine Men's Morris game. As previously mentioned, Java is excellent for object-oriented development because it is entirely an object-oriented language, in contrast to Python which is weak at enforcing certain object-oriented concepts. Apart from Python and Java, the team is not familiar with any other languages that support object-oriented programming, which made the decision to use Java very evident. In addition, every member of our team has worked with Java before on projects that also prioritise good object-oriented design. Java is also widely used to develop desktop and mobile applications, which fits the criteria for the developed game to be a standalone application.

As for the game engine/APIs, the team has decided to choose JavaFX for the GUI and FXGL a game development framework built on top of JavaFX for making the game development process simpler. Because JavaFX has more online resources for its use, including YouTube, it was chosen over other available APIs like LITIengine, libGDX, and others. Although the other APIs have excellent setup guides and documentation on their websites, the team's learning resources are constrained by a lack of YouTube videos and online tutorials. FXGL includes 3D graphics support, which the team thought was good for the chosen game development framework to have in case, in the future, the

team were to decide to make the 3D version of the game. This requirement assisted in removing some frameworks as an option, such as LITIengine, which only supports 2D development.

JavaFX has some interesting features that it provides like the use of FXML files. FXML is a markup language that can be used to define the user interface for JavaFX applications without having to write Java code. While the logic for the application will all be in Java code. JavaFX supports the use of CSS, which is used to alter the appearance of graphical user interface elements. JavaFX also comes with JavaFX scene builder, a standalone application, which is a tool to visually layout UI components that aid in designing the UI for JavaFX applications. It exports files as an FXML file and it also supports the use of CSS to enhance the look of the UI.

The above mentioned justifications and arguments made it clearer why JavaFX and FXGL were chosen for the game's development. The team believes that there will be some learning curve at the beginning of development for JavaFX and FXGL but with the help of online resources and the IT faculty, it will be overcome. The team has also decided to use IntelliJ IDEA as the IDE for the development of the game application as it provides excellent Java development support.

User Stories

Basic Requirements

Player

1. As a player, I want to see a game board, so that I know where to place and move my tokens.
2. As a player, I want to place my tokens on the board so that I can have up to 9 token pieces on the board.
3. As a player, I want to move placed tokens on the board so that I can try to form a mill, meaning 3 of my tokens on a straight line.
4. As a player, I want to be able to choose and remove one of my opponents tokens from the board, once I make a mill.
5. As a player, I want to be able to move a token out of an existing mill, then move it back, so that I can form the same mill.
6. As a player, I want to be able to fly my pieces (move to any unoccupied spot on the board) once I am down to three pieces, so I am not limited to moving my remaining tokens to adjacent points.
7. As a player, I want the game to notify me once either player has 2 tokens left, so I know there is a winner.
8. As a player, I would like to end a game prematurely and return to a main menu so that I do not have to complete a game.

Game Board

9. As a game board, I want to have 24 positions that each can have a token, so that both players have enough space to place and move their tokens around.
10. As a game board, I want to make sure that a tile can only occupy one, and only one token so that player's tokens do not overlap.
11. As a game board, I want to know if a player has moved their token to an adjacent available position on the board, so that I know a player is making a valid move.
12. As a game board, I want to make sure that tokens from a formed mill cannot be removed unless no other tokens are available, so that game rules are followed.
13. As a game board, I want to make sure that captured tokens cannot be played again, so that the game rules are obeyed.
14. As a game board, I want to indicate which player's turn it is, so that the correct player can make their turn.

Token

15. As a game token, I want to be placed on the board, so that a game can be initiated.
16. As a game token, I want to move to legal positions on the board based on game rules, so that the game can progress.
17. As a game token, I want to fly from one position on the board to another position of the board if my player has 3 tokens on the board so that I can have an advantage over the opponent.

Game State

18. As a game state, I want to keep track of whose turn it is, so that no player can play two consecutive turns.
19. As a game state, I want to keep track of the number of tokens that both players have respectively so that the game is not over and players on each side have tokens of 3 or more respectively.
20. As a game state, I want to keep track of which phase the game is in now so that I can identify what actions the tokens can make.
21. As a game state, I want to make sure that both sides have 3 or more tokens so that each side has a chance to form a mill.
22. As a game state, I want to keep track of a counter and check if the number of total turns completed has reached 200, so that the game can be declared a draw.
23. As a game state, I want to keep track of each player's tokens to determine which phase each player of the game is currently in.

Advanced requirement (a)

24. As a novice player, I want the game to show prompts and guides in tutorial mode, so that I can learn the game as a beginner and improve my skills.
25. As a game player, I want to be able to see a list of game rules, so that I can refer to the rules if I am new or if I forget some of the rules.
26. As a player who needs assistance in game, I want to be able to toggle hints, so that I can visualize all possible moves for a token and make better game decisions.

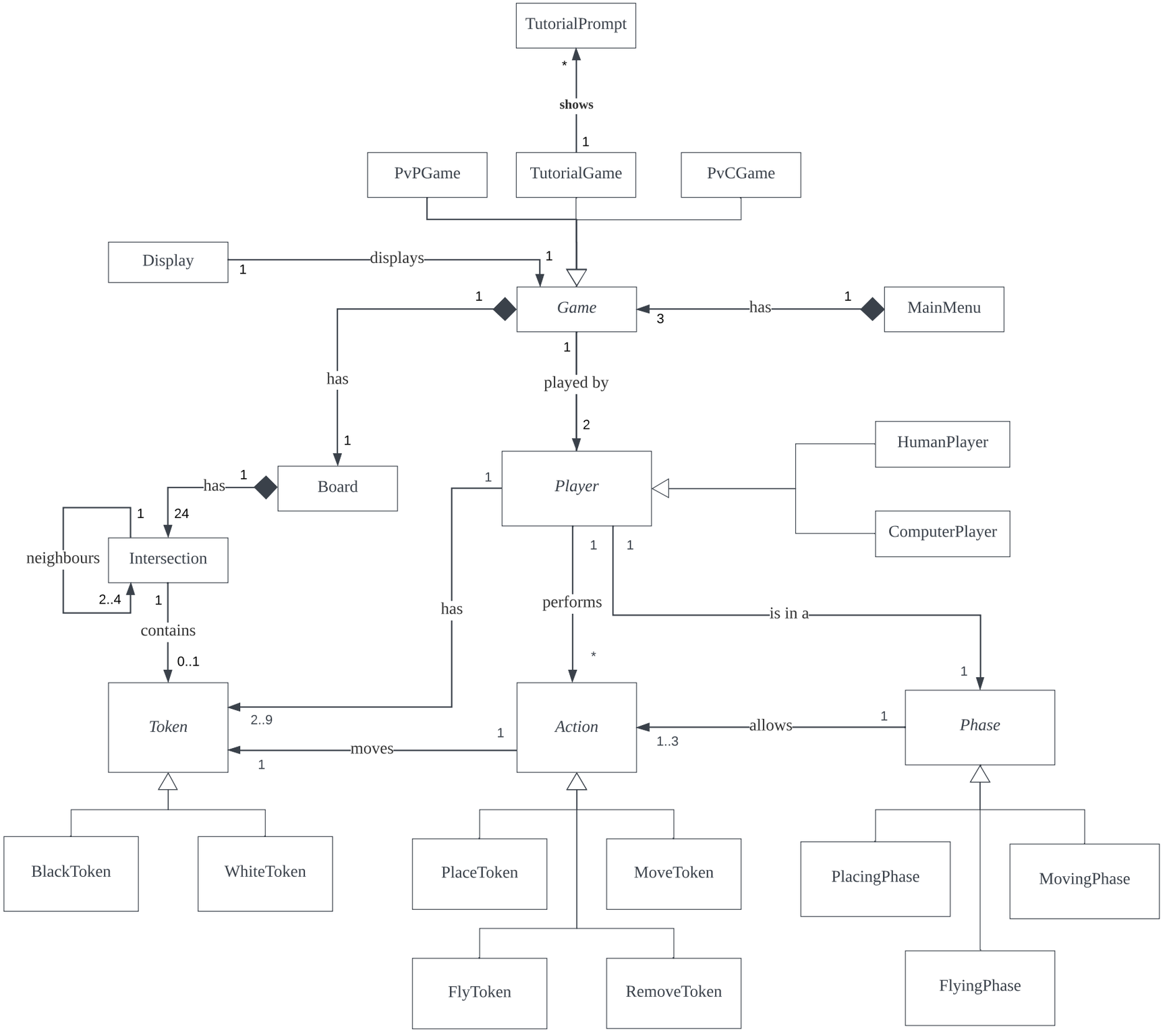
Advanced requirement (c)

27. As a game player, I want to have an option to play against the computer, so that I can have someone to play with.

28. As a game computer player (bot), I want to be able to do the basic game interactions, so that I can play the game properly against a real player.

29. As a game computer player (bot), I want to know all the possible moves for my tokens, so that I can randomly make a move for my turn.

9 Men's Morris Domain Model



Design Rationale for Domain Model

Advanced requirements Chosen: (a) and (c)

Domain Model and Design Decisions

Player

1. There are currently two types of players - a human and a computer.
2. Each player has 9 tokens at the start of each game. Each player must have at least 3 tokens for the game to continue, otherwise the game ends. Note that a player can have 2 tokens, and this event triggers a game to end.
3. A player can perform one specific action (see Action), depending on which phase (see Phase) of the game they are in.
4. Since a computer player is essentially simulating the behaviour of another human player, a computer player can perform any action that a human player can.
5. However, a computer player can be different from a human player in multiple aspects. For example, a computer player may also need incorporate additional algorithms to compute their moves.

Game

1. A game must be played by exactly 2 players.
2. A game must be shown on one display to convey latest information to the players whenever the board changes.
3. A game must be played on a board, and a board only represents one game.
4. A main menu has 3 types of games to choose from
5. There can be multiple game types (player versus player, player versus computer and tutorial mode).

6. Each game type is played according to the same rules shown above (1-4).
7. Each game type might have different combinations of player types.

Abstraction of Game and Player

1. This design choice is inspired the Liskov Substitution Principle.
2. Collectively, the abstraction of these two classes introduces flexibility into the game by making different combinations of game setups possible.
3. For example, a player (HumanPlayer) may prefer to play a game with a computer (PvCGame). However, they (HumanPlayer) may prefer to play another game (PvPGame), with another human (HumanPlayer) after the first game.
4. A game with different player types and game modes does not change how a game of 9 Men's Morris is played. In other words, these two components can be interchanged, without changing the game rules and logic.

TutorialGame

1. Everything related to tutorial is hosted in the TutorialGame domain entity that inherits from the Game entity to have all functionality of a normal game.
2. TutorialGame includes prompts for the players to follow through. The domain responsible for prompts is TutorialPrompt and TutorialGame has many prompts.

Token

1. Tokens are manipulated by players in order to progress a game. A player with fewer tokens indicates that they are losing (arguably).
2. Tokens currently come in black and white to represent each player in a game.
3. All tokens behave similarly, but will look different from each other.

4. There may be more token colours introduced in the future.

Action

1. Actions represent the moves that a player can perform on a token.
2. A player can perform one or more actions per turn (they can move, form a mill, then remove an opponent's token in the same turn). Throughout the game, they would have performed many actions.
3. One action manipulates only one token.
4. A token can move or fly across the board. It can also be placed or removed by a player. All these actions share a common goal of manipulating the position or state of a token, but they do so differently.
5. For example, both moving and flying moves a token to another empty intersection. A token can only be moved to an adjacent intersection, but a token can fly to any intersection on the board.

Phase

1. A phase defines a set of allowable actions a player can perform.
2. A player can by in any one of three phases based on how they can manipulate their tokens. They can either place, move or fly tokens.
3. For example, a player with all 9 tokens currently in play is allowed to move and remove opponent tokens (MovingPhase), but they are not allowed to place or fly their tokens until they have 3 tokens (FlyingPhase).
4. Players can remove opponent tokens at any phase.

Abstraction of Action and Phase

1. Abstracting both action and phase makes new game rules easier to maintain and extend.

2. If a new game rule specifies that players can fly their tokens during the placing phase, it only needs to be added in the PlacingPhase class, without affecting other phases.

3. A new phase with a combination of new and existing actions can also be added just by creating new classes that implement Phase or Action.

Board

1. A board must exist in the game so players can interact with tokens.
2. A board is a visual indicator of how the game is progressing for both players.
3. The board is essentially a collection of 24 intersections.

Intersection

1. An intersection is an interactable part of a board.
2. Players play the game by moving tokens around different intersections. They can also place or remove tokens in an intersection.
3. One intersection can contain a maximum of one token, but it may also be empty.
4. One intersection neighbours at least two, but not more than four other intersections on the board.

Design choice for Board and Intersection

1. It is possible to store the position of each token in a single Board object. However, it would be hard to maintain whenever the board layout changed.

2. A board would also need to store the exact positions of every single intersection and keep track of every token which would make the code within this class very convoluted.

3. It was decided that certain responsibilities would be delegated to a new class called Intersection.

4. Each intersection is responsible for its own position on the board.

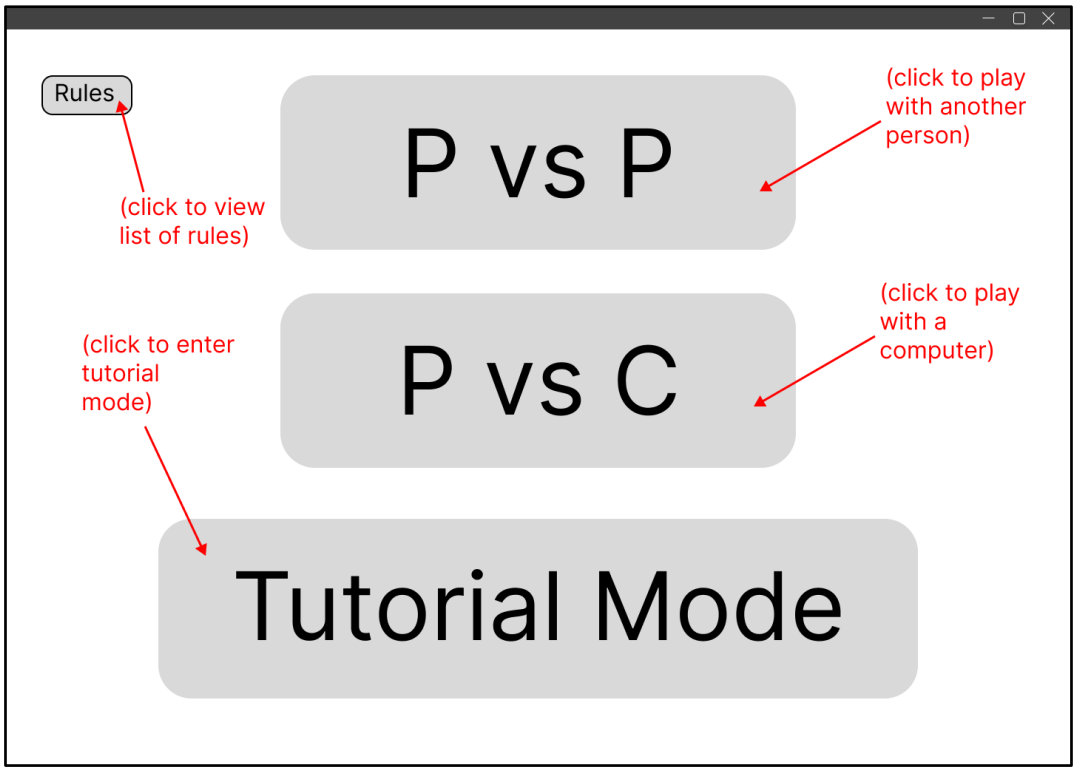
5. Each intersection records if it contains a token or not.
6. In this design, a board only needs to keep track of the relative positions of the intersections, and query each intersection to determine where all the tokens on a board are.

Assumptions

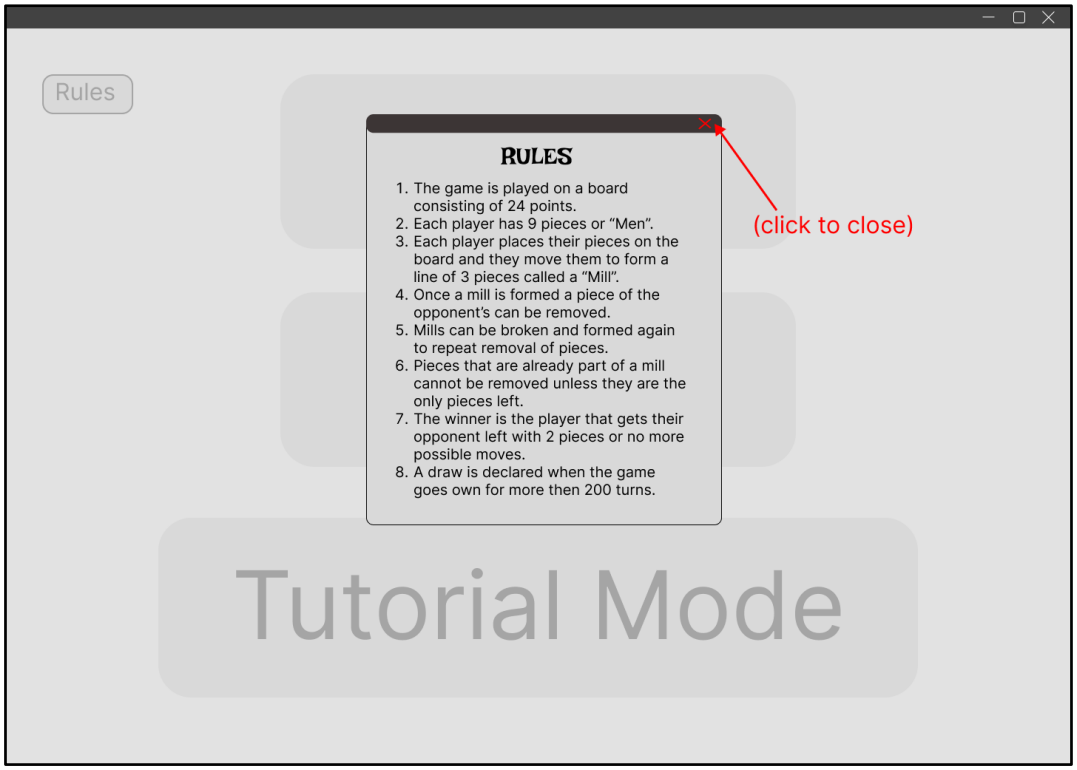
1. For the start turn the white tokens moves first.
2. For the main menu, we decided to allow the user to choose between new game, tutorial game and computers. We find this easier to implement the logic when coding.

Lo-Fi UI Prototype Design

Main Menu

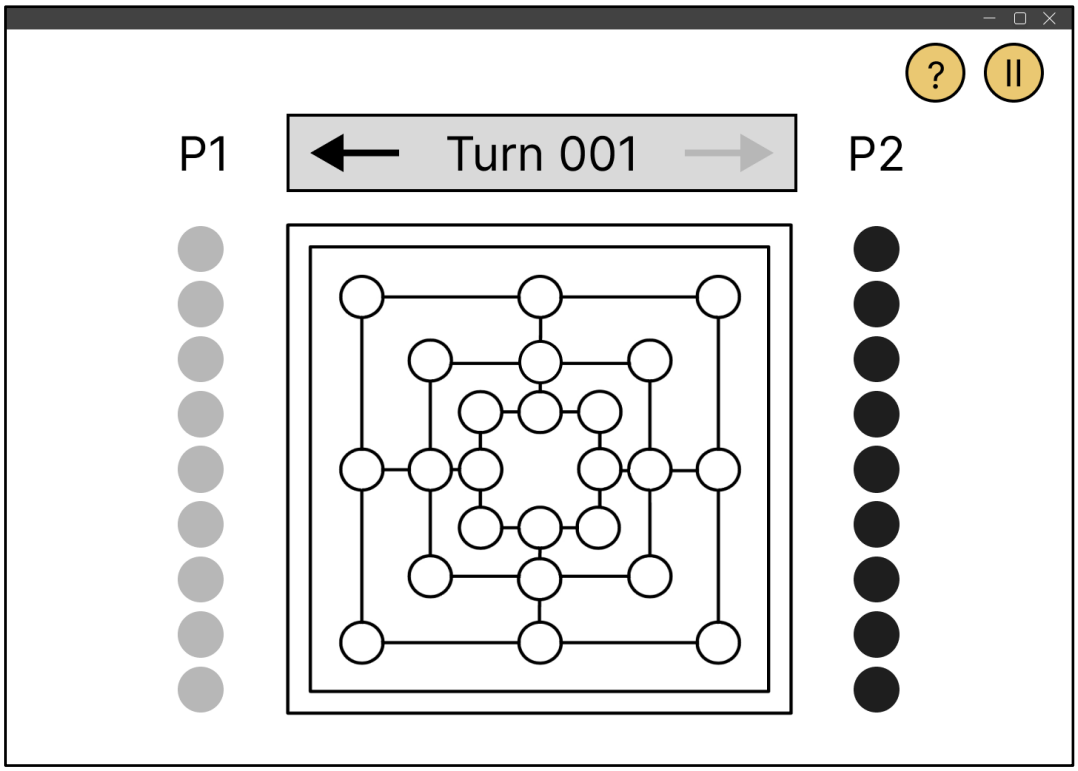


The Main Menu of the game that is shown when the game client is launched.



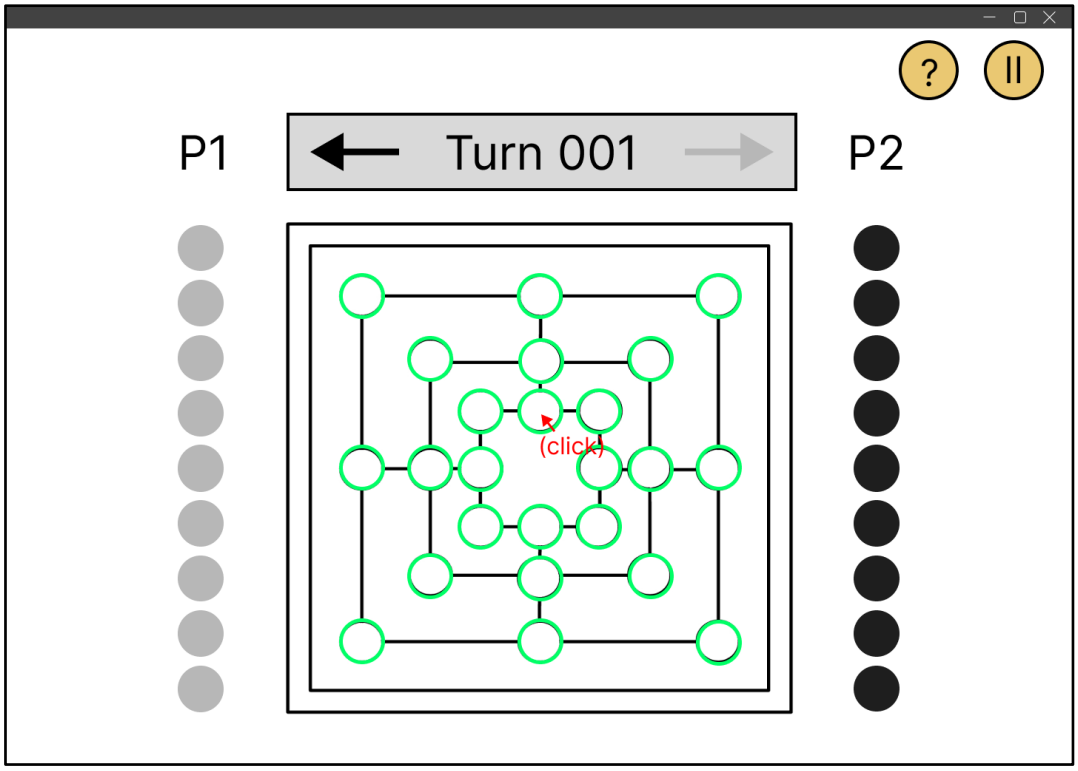
A list of Nine Men's Morris game rules that can be viewed in at the main menu.

Main Game UI

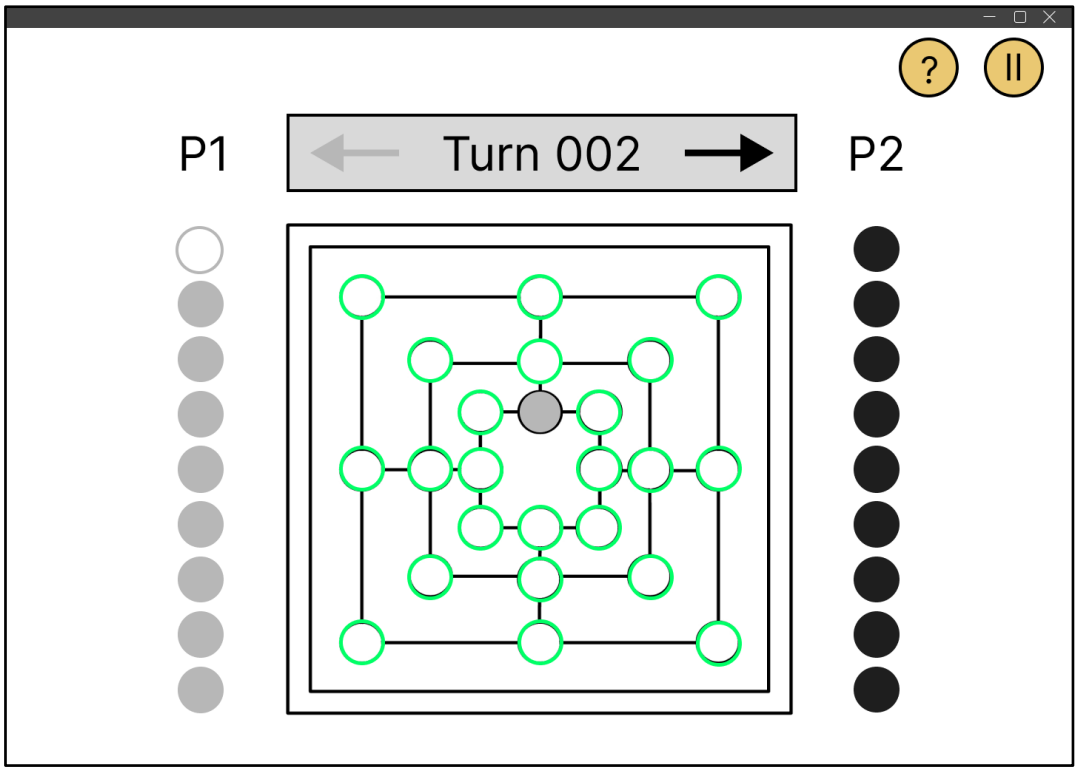


The UI of the game that is shown when the game starts.

Placing Phase

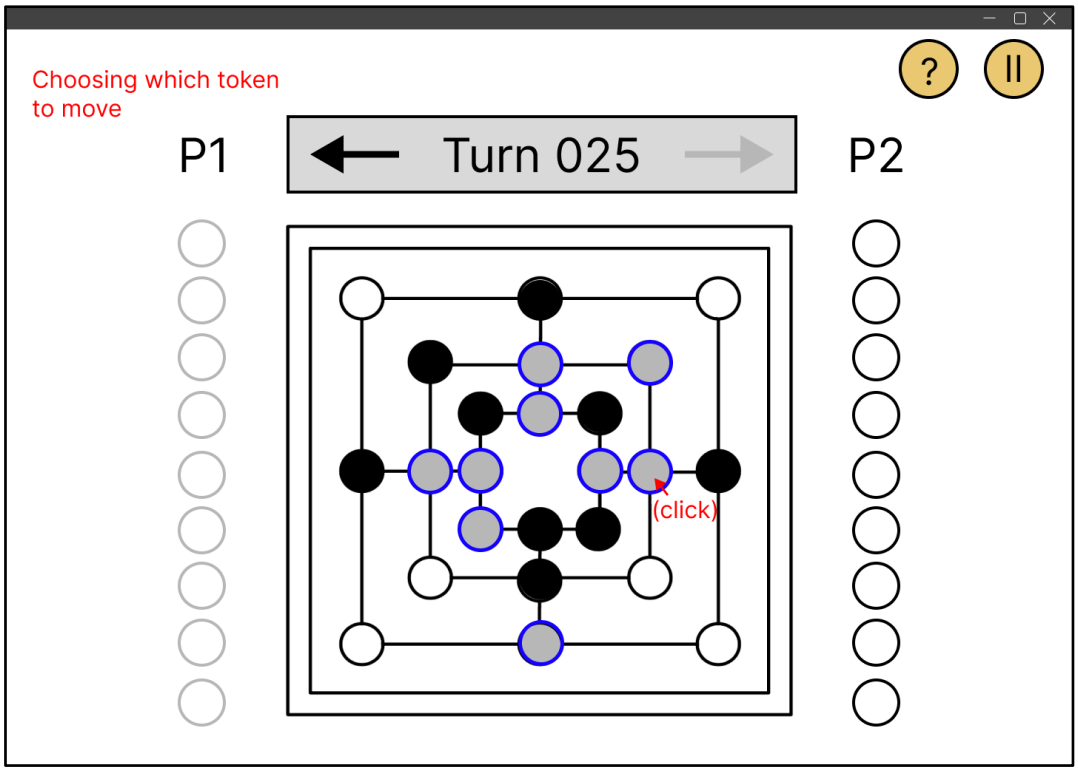


Frame 1: Green highlights indicate available position at the placing phase. Then a player can click on any of the position available.

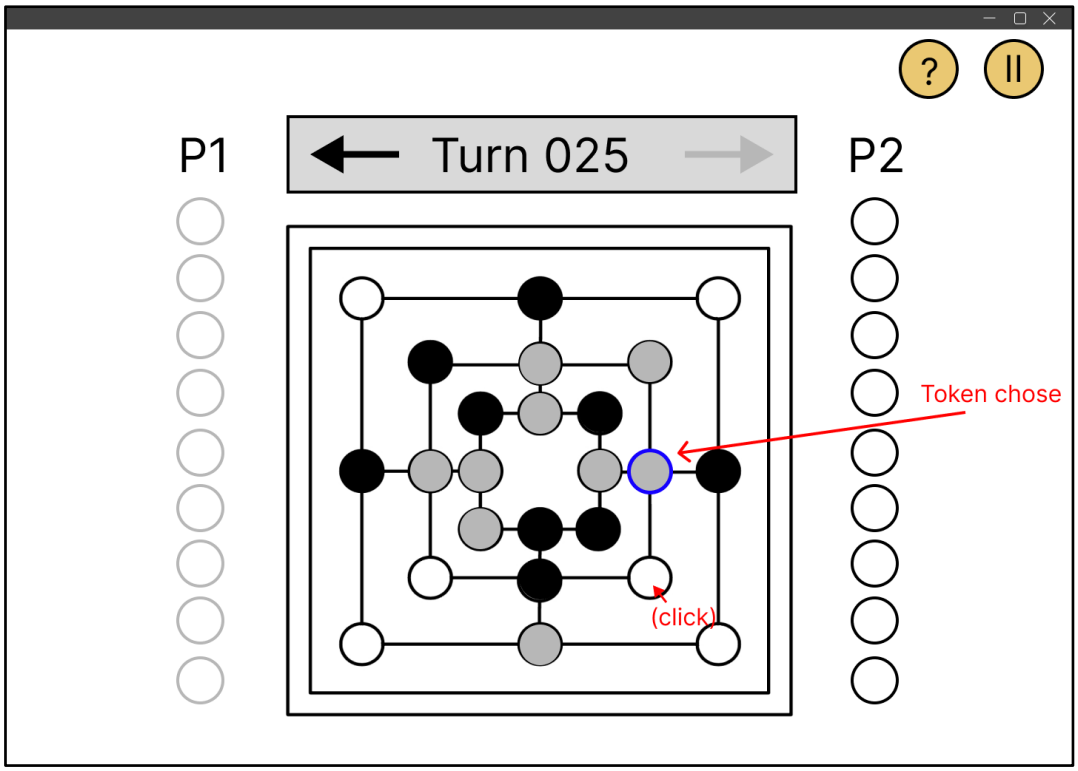


Frame 2: When a position is clicked, the player's piece is automatically placed.

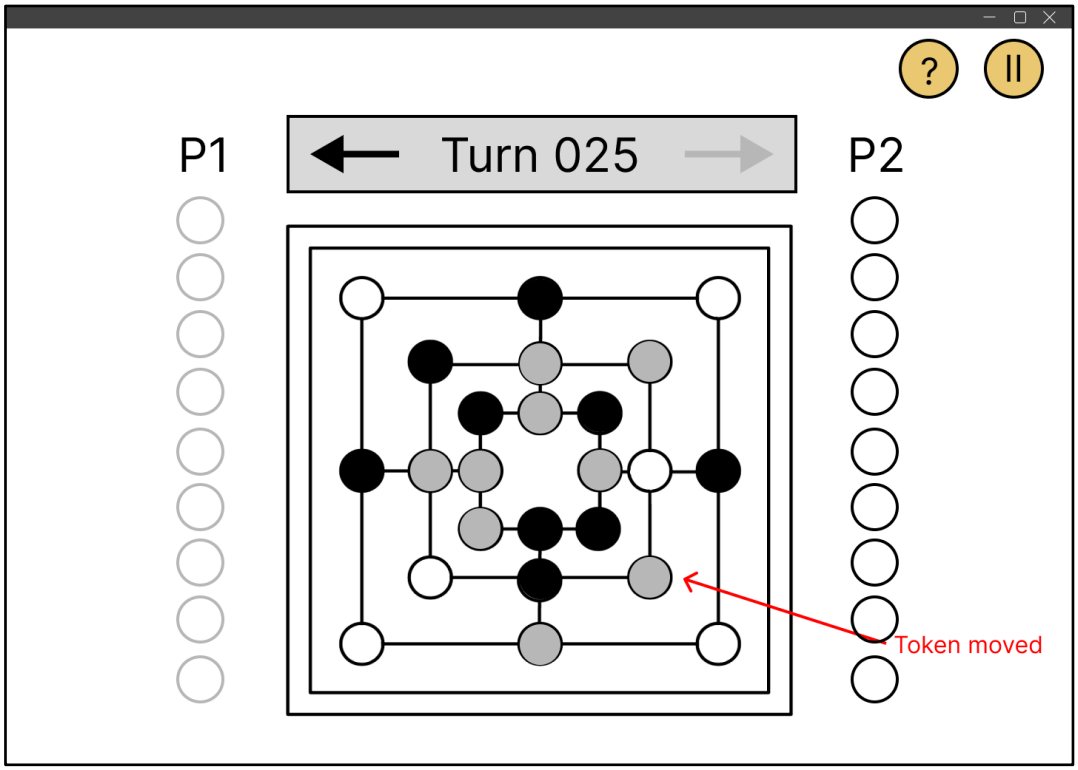
Moving Phase



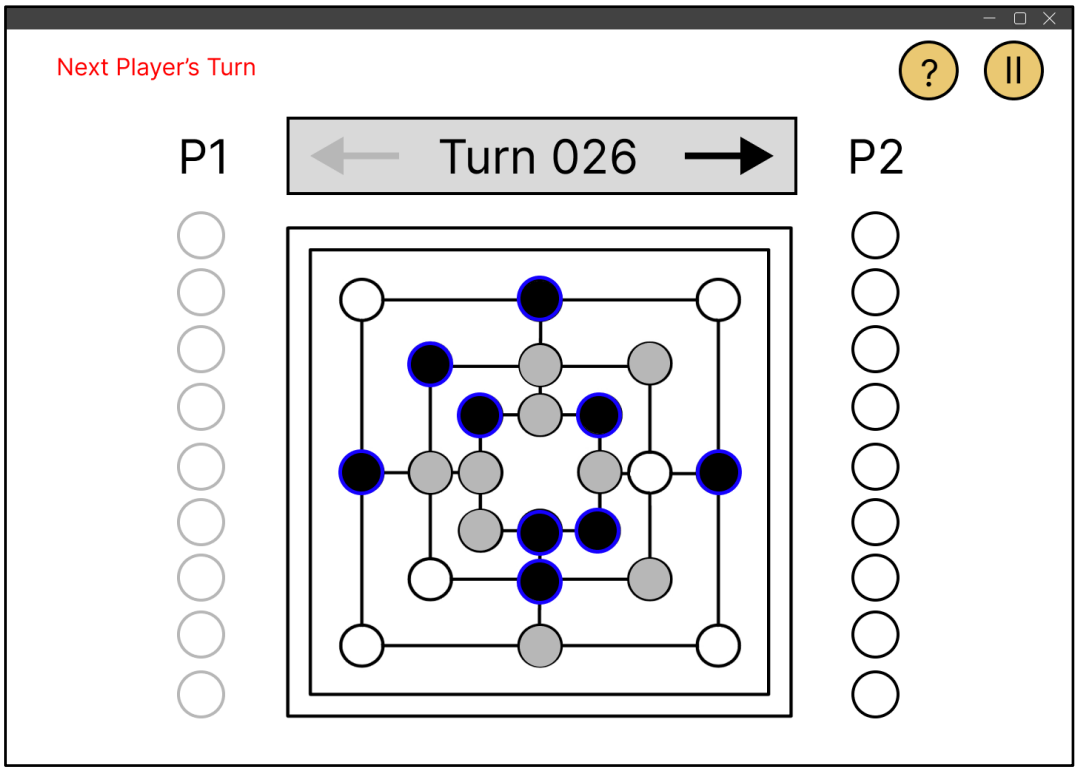
Frame 1: Blue highlights indicate available tokens to be chose. Then a player can click on the chosen token.



Frame 2: After choosing a token, the blue highlight on that token stays to indicate which token has been chosen. Then the player can click to choose a position for the token to be moved to.

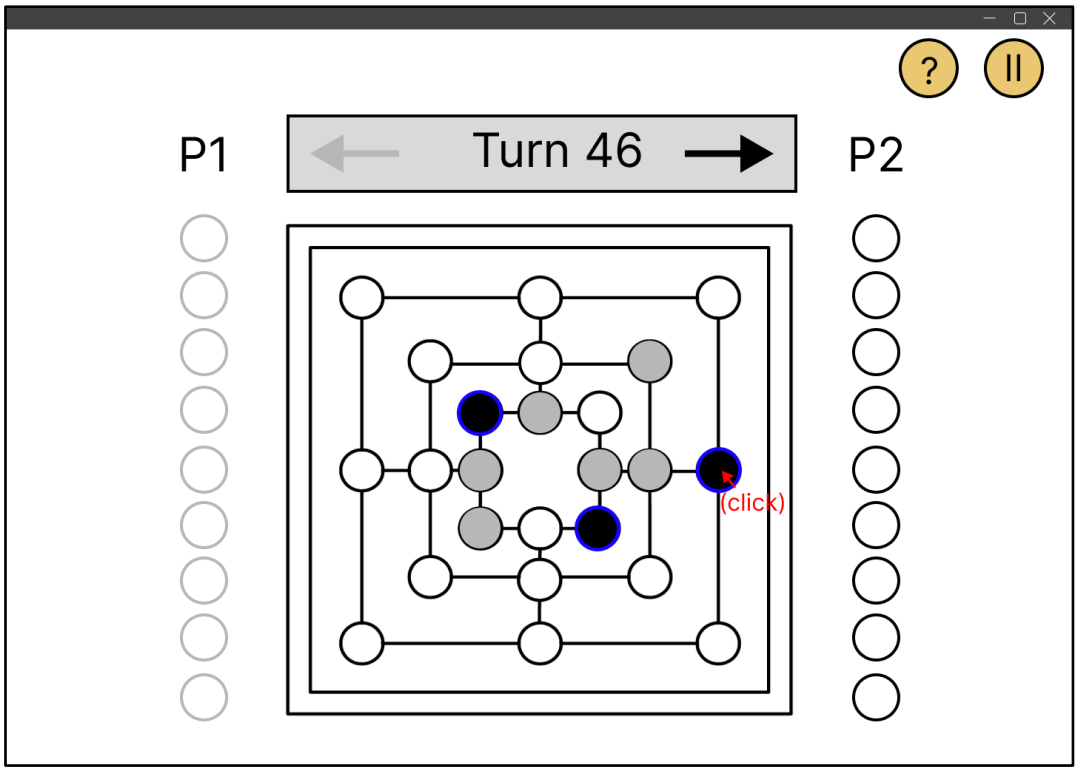


Frame 3: When a player has clicked on the position, the token will be automatically moved.

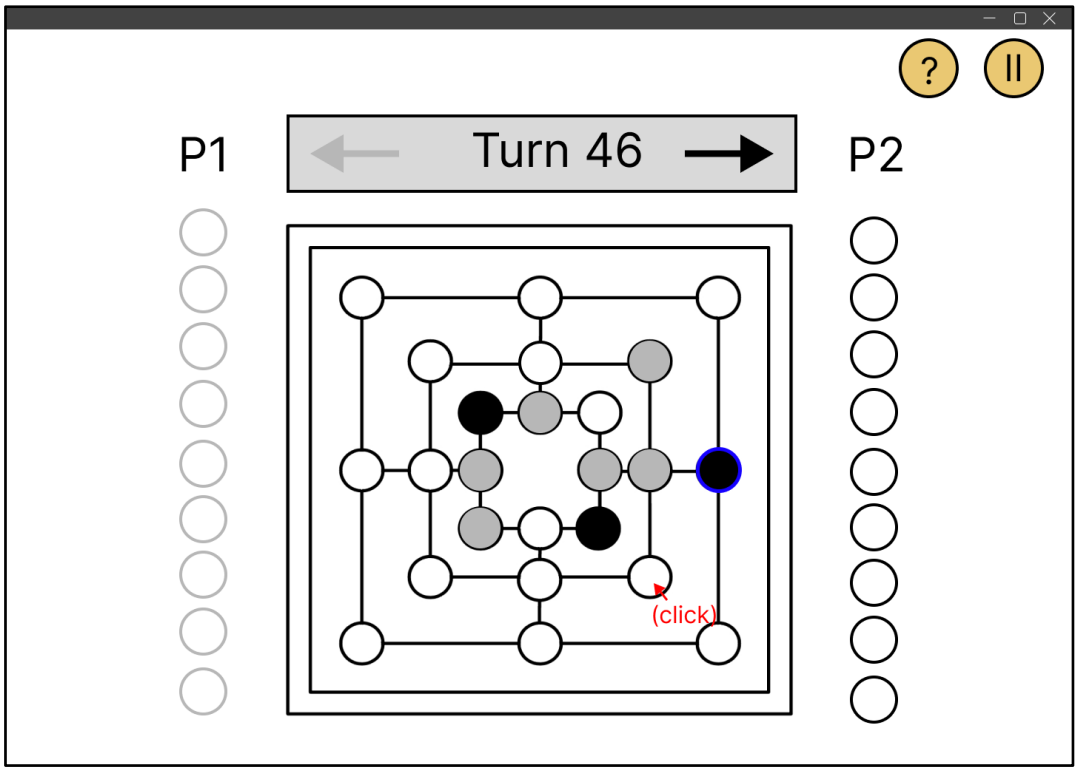


Frame 4: Then the turn changes and the same process repeats itself on the next player's turn.

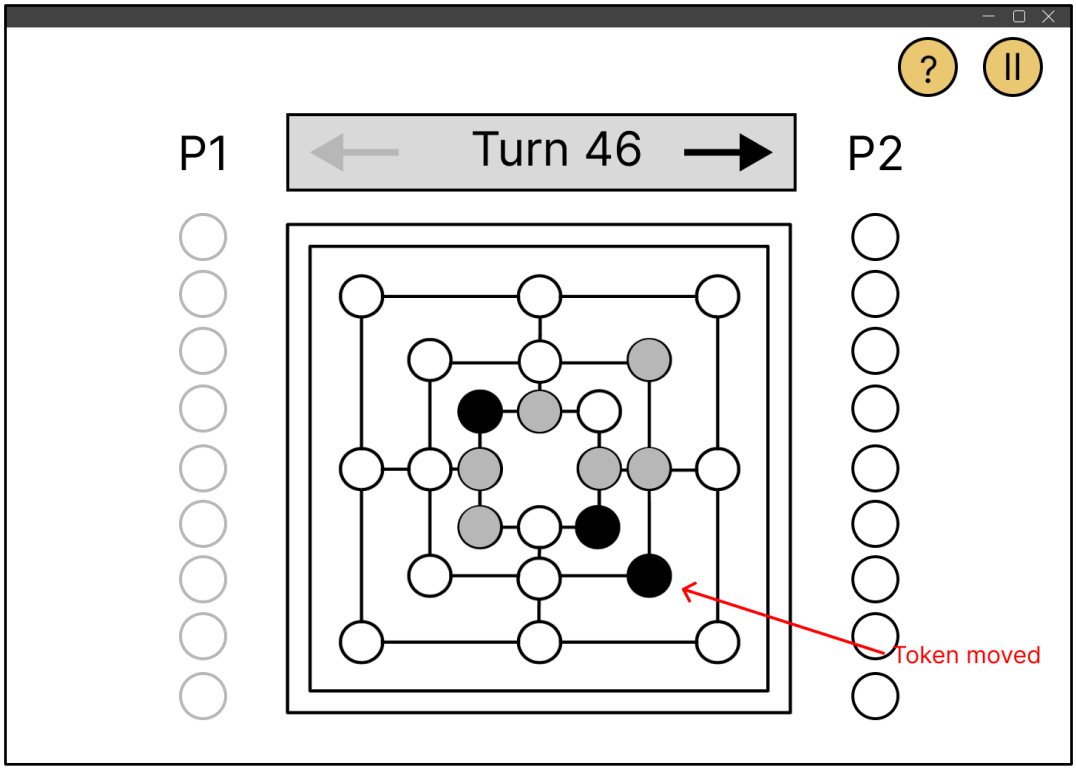
Flying Phase



Frame 1: Blue highlights indicate available tokens to be chose. Then a player can click on the chosen token.

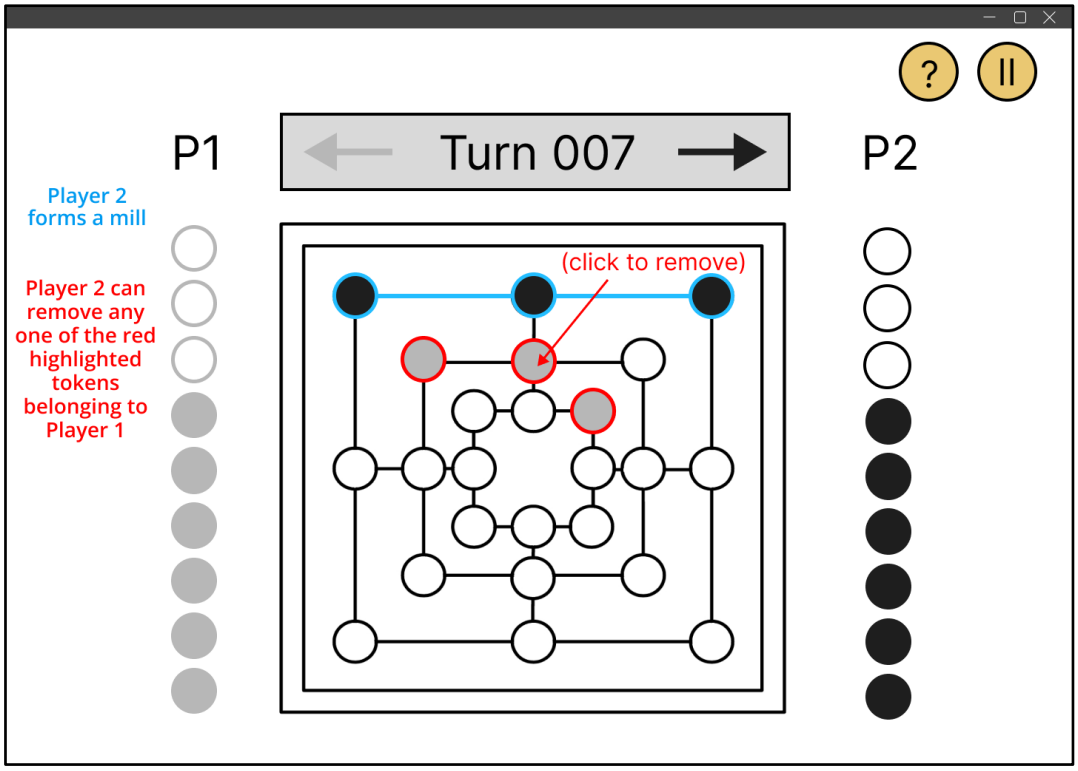


Frame 2: After choosing a token, the blue highlight on that token stays to indicate which token has been chosen. Then the player can click to choose a position for the token to be moved to.

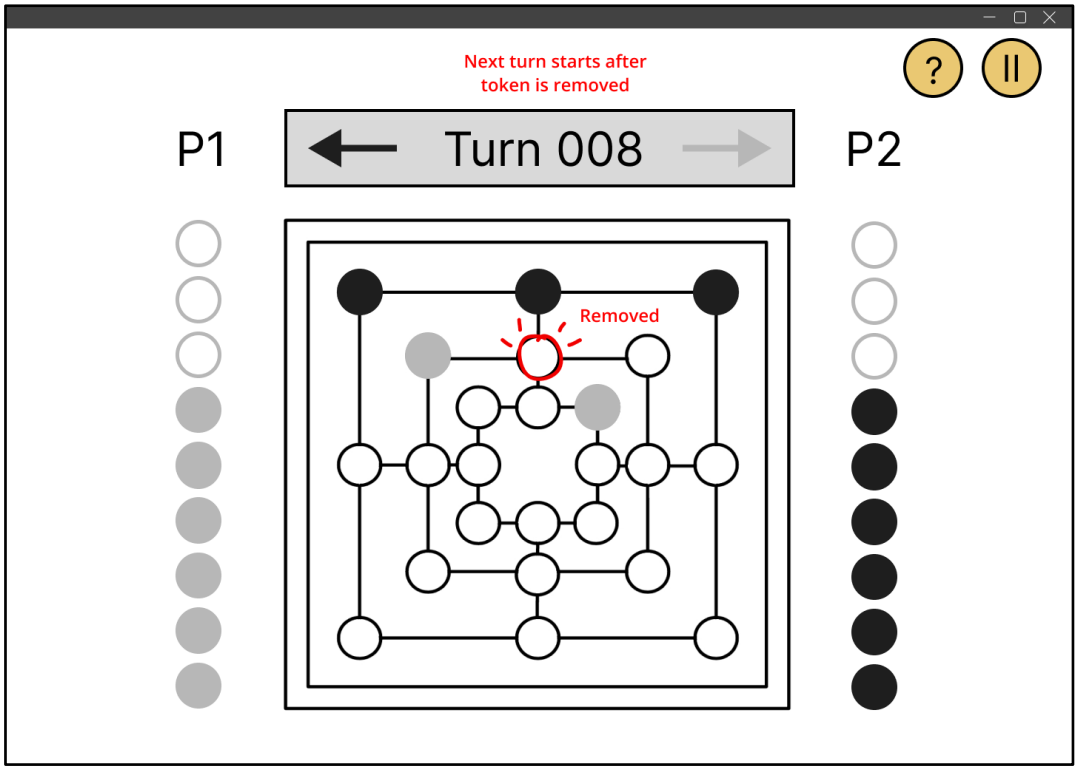


Frame 3: When a player has clicked on the position, the token will be automatically moved.

Forming a Mill and taking opponent's token

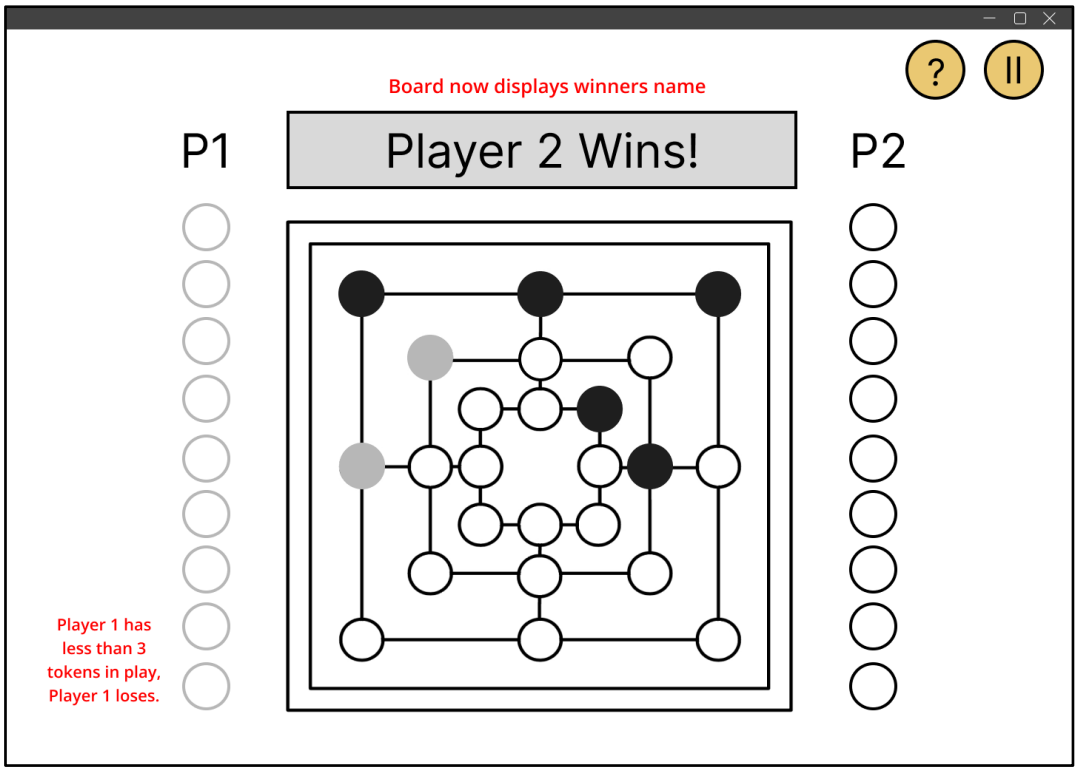


Frame 1: Player 2 forms a mill (shown in blue) since he has 3 tokens in a row.

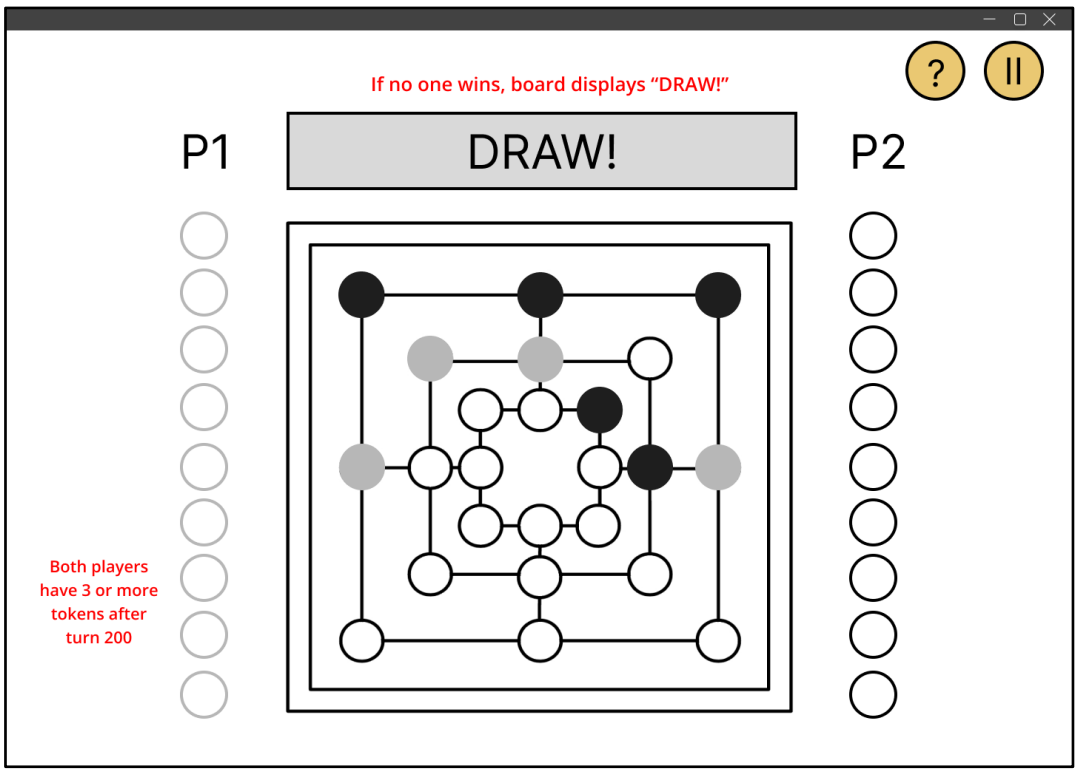


Frame 2: Player 2 removes one of Player 1's tokens (shown in red) from the board by clicking on it. Removing the token ends Player 2's turn.

Win/Loss and Game Draw

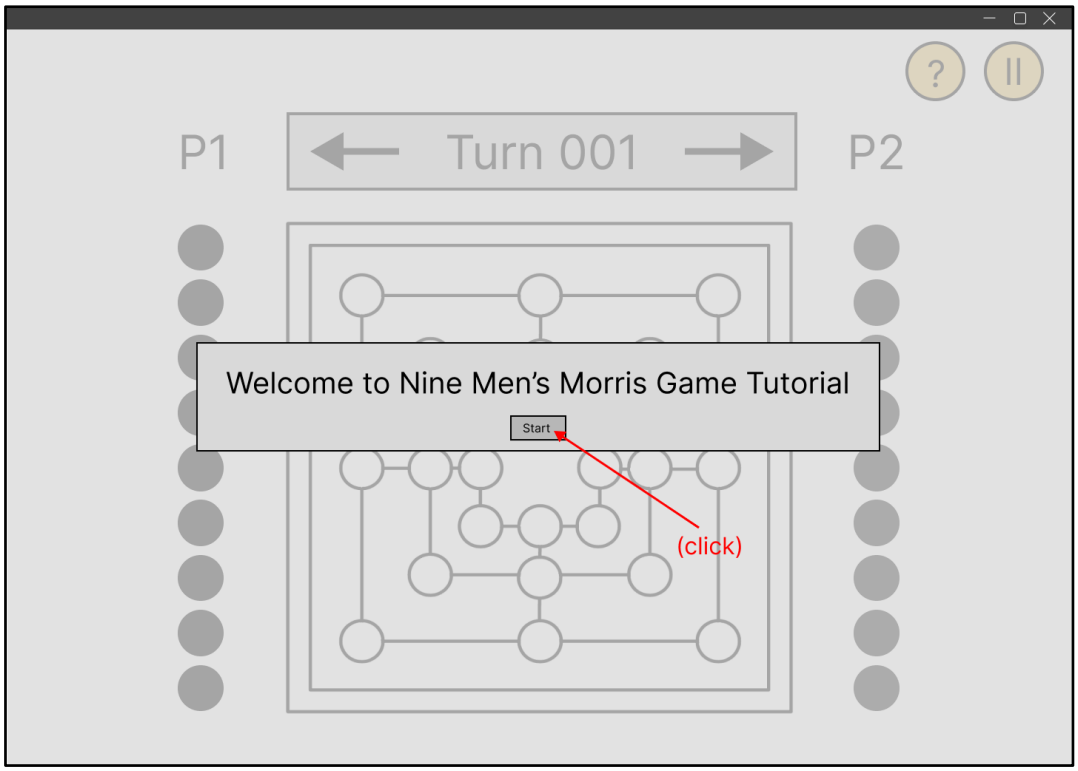


Frame 1: When any player has two tokens in play, it triggers the game to end. The turn indicator displays the name of the victor to indicate that the game has concluded.

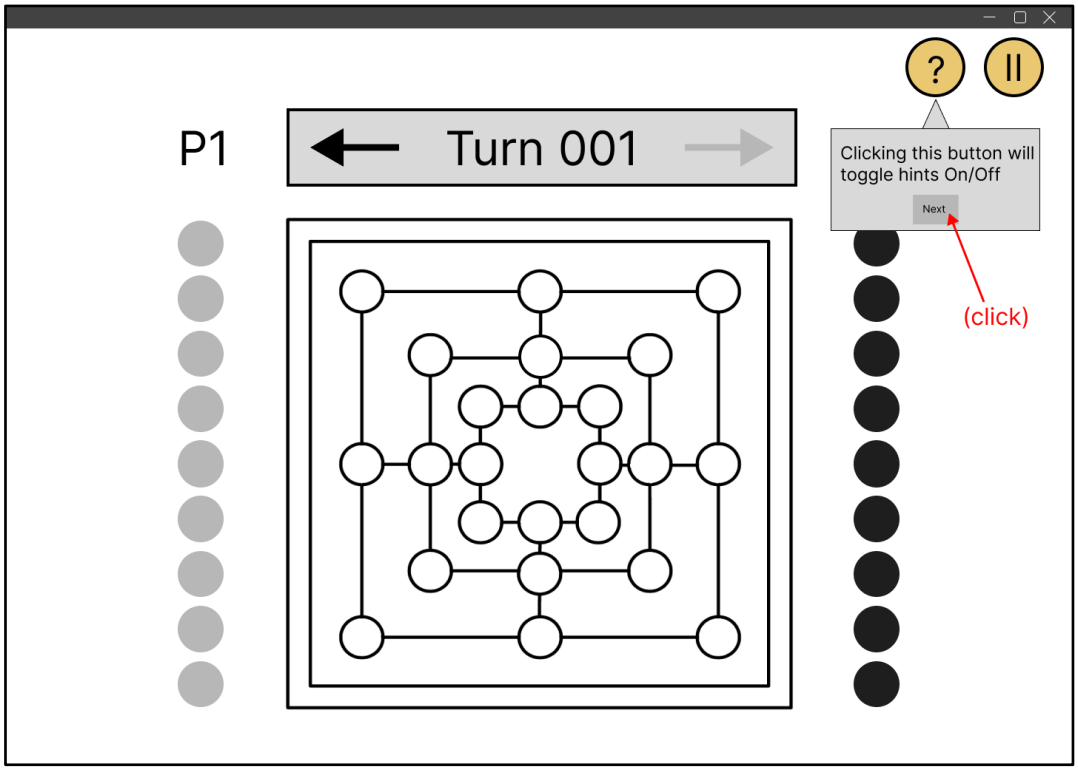


Frame 2: If both players have 3 or more tokens after turn 200, the game results in a draw. The turn indicator displays “DRAW!” to notify the players that the game has ended.

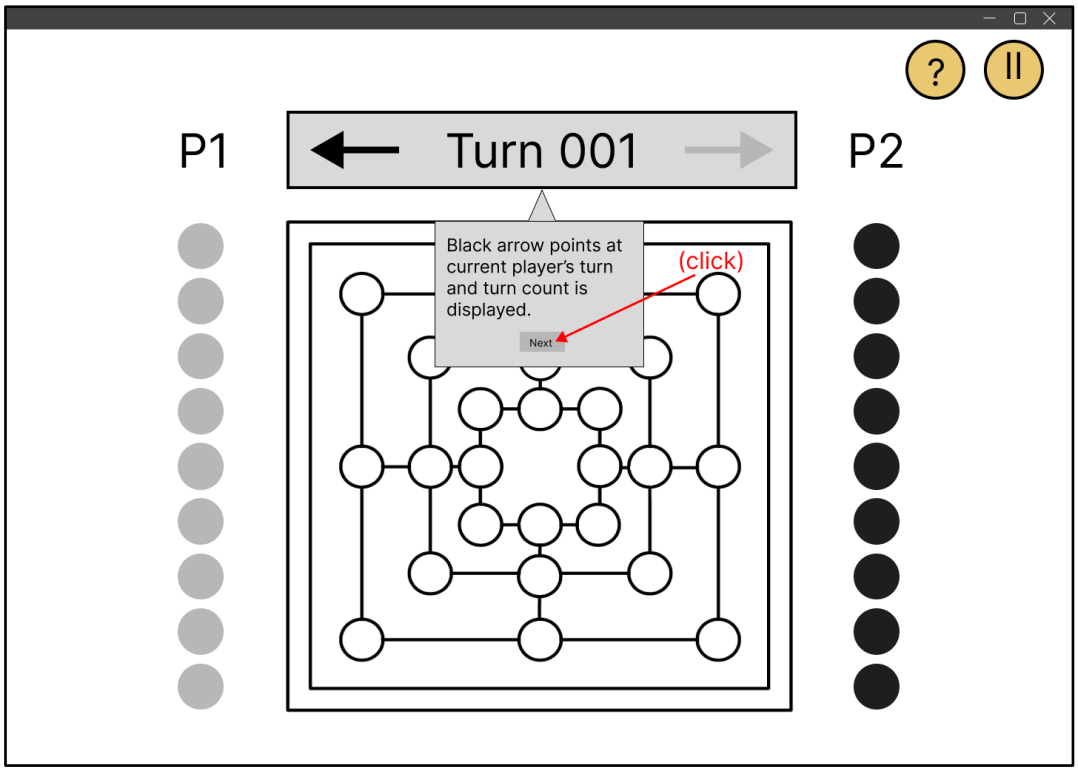
Tutorial Mode (Advanced Requirement(a))



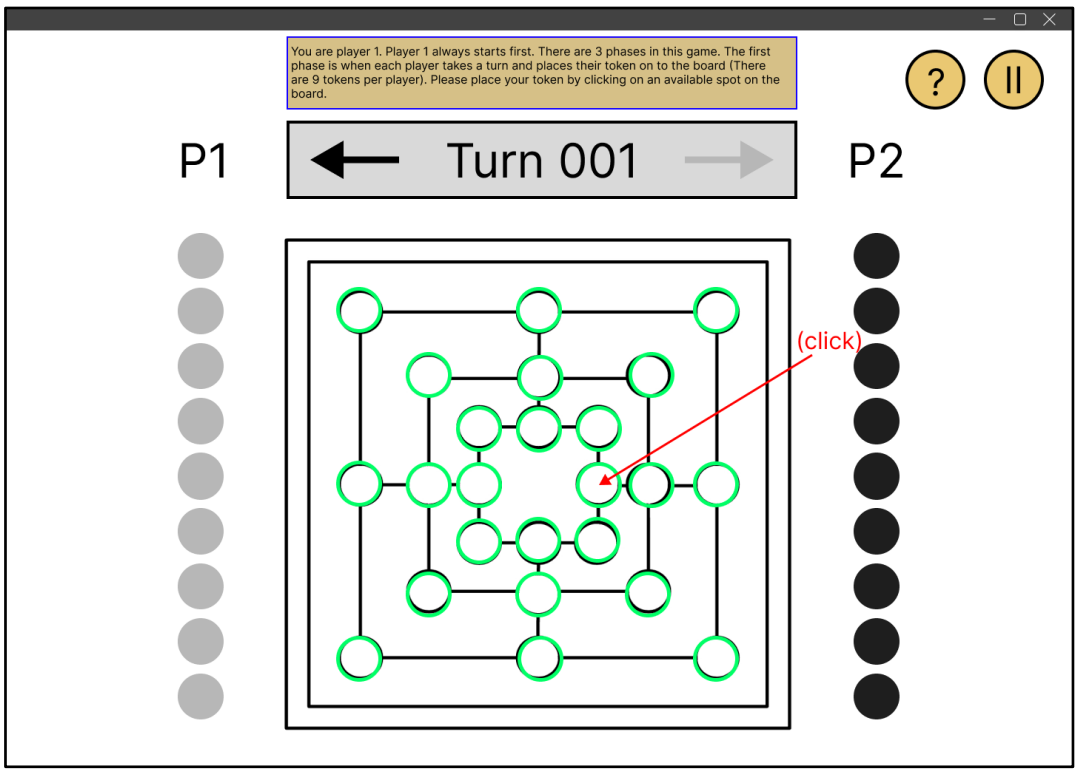
Frame 1: When a player clicks on the tutorial mode in the main menu, they are greeted with this screen and message to start the tutorial. The player clicks start.



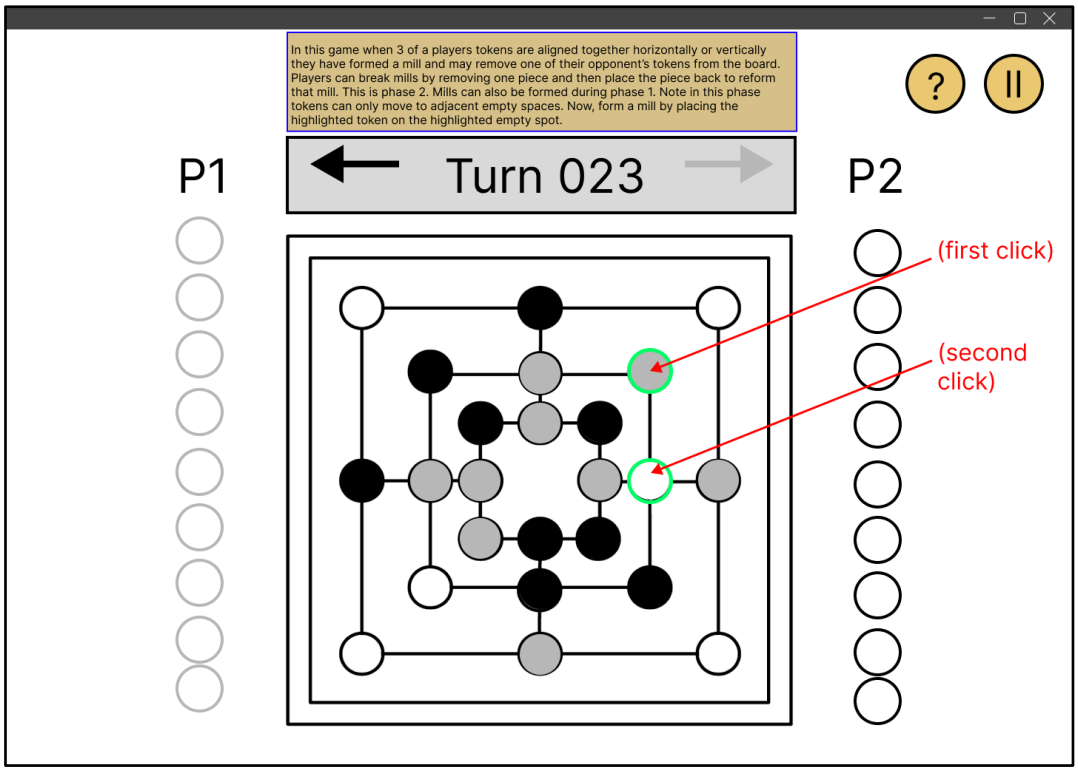
Frame 2: At the beginning of the tutorial a prompt box appears showing the player how to toggle hints on/off. The player clicks next.



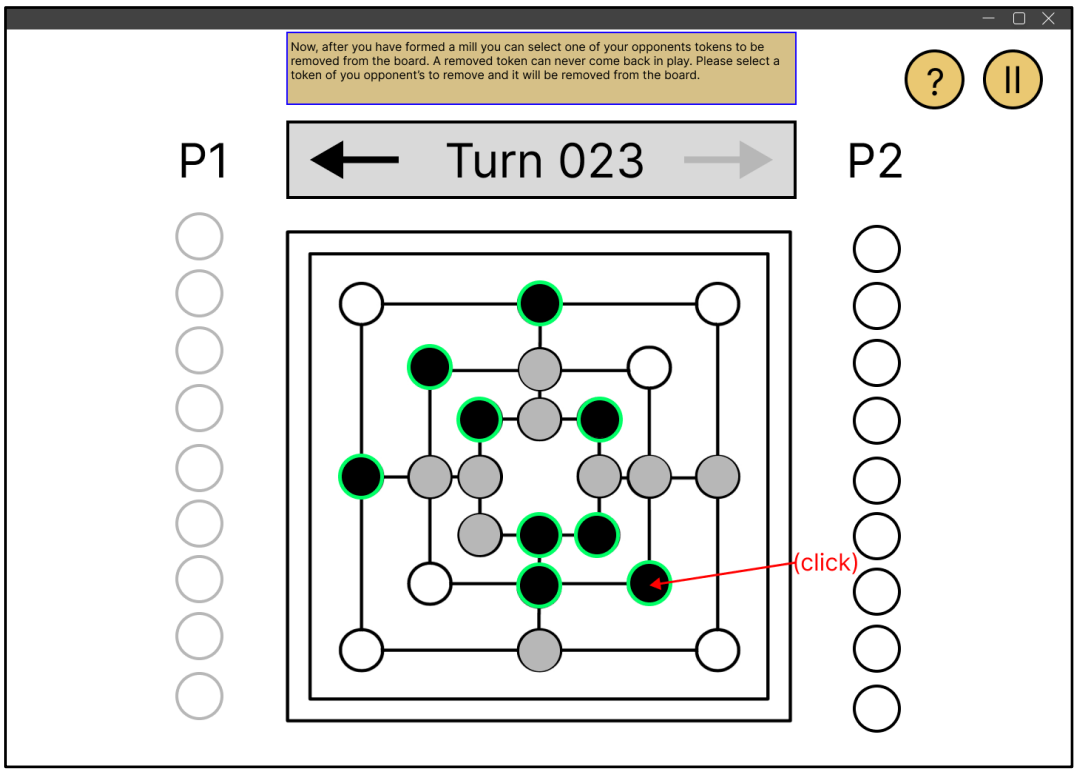
Frame 3: Next the prompt box explains the player what the rectangular box on top of the board indicates. The player clicks next.



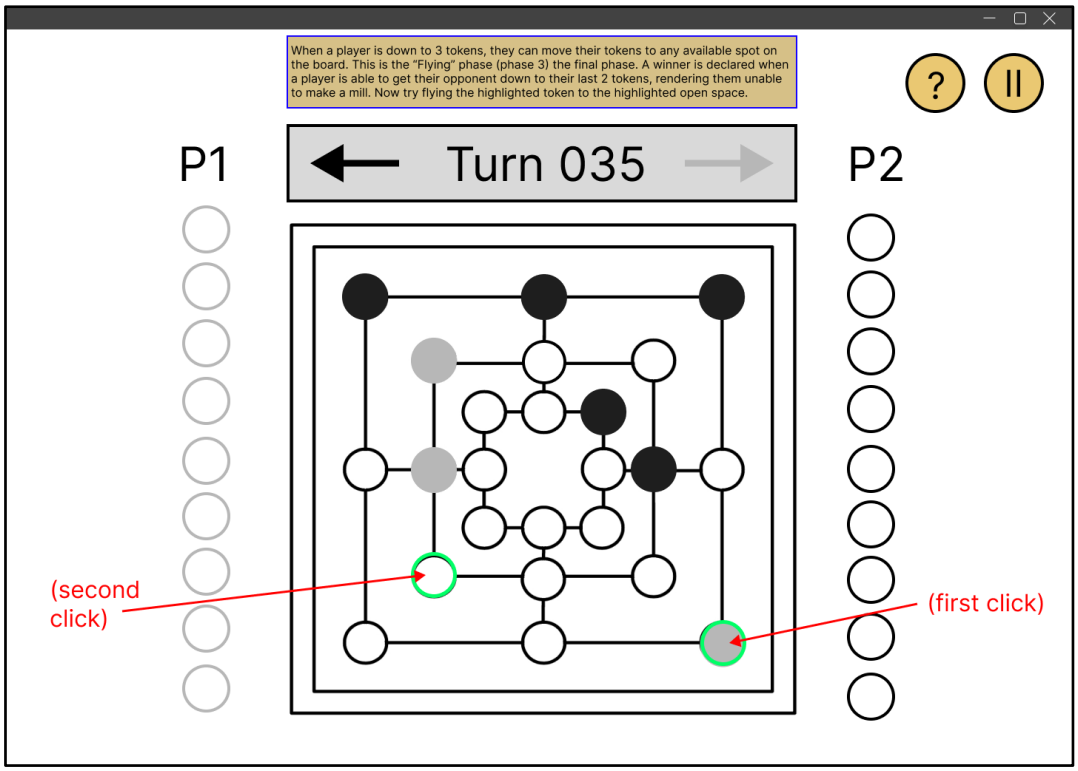
Frame 4: A note box at the top explains the rules gradually. For this instance, the player is told how to place a token. Player follows and places on the board by clicking.



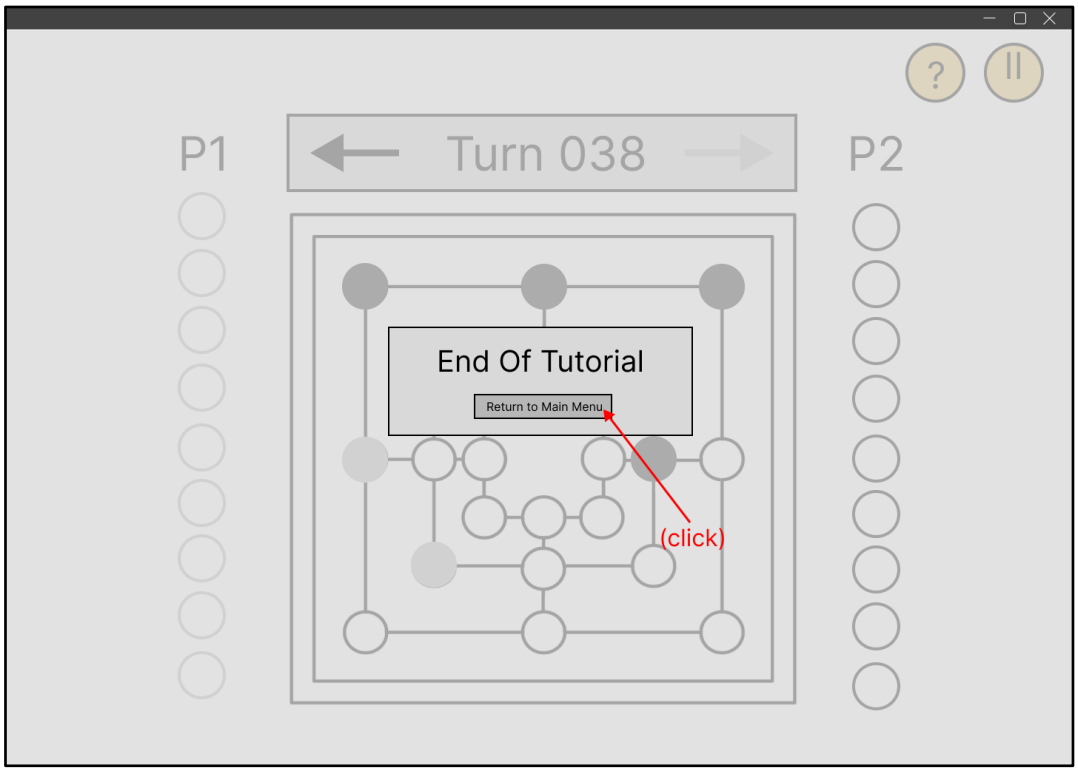
Frame 5: Player is taught how to move a piece to an available adjacent intersection and form a mill. Player follows clicks on the piece to move and clicks on the intersection for it to go on.



Frame 6: Player is taught how to remove an opponent's piece from the board after forming a mill. The player follows and clicks on the opponent's piece that is to be removed.

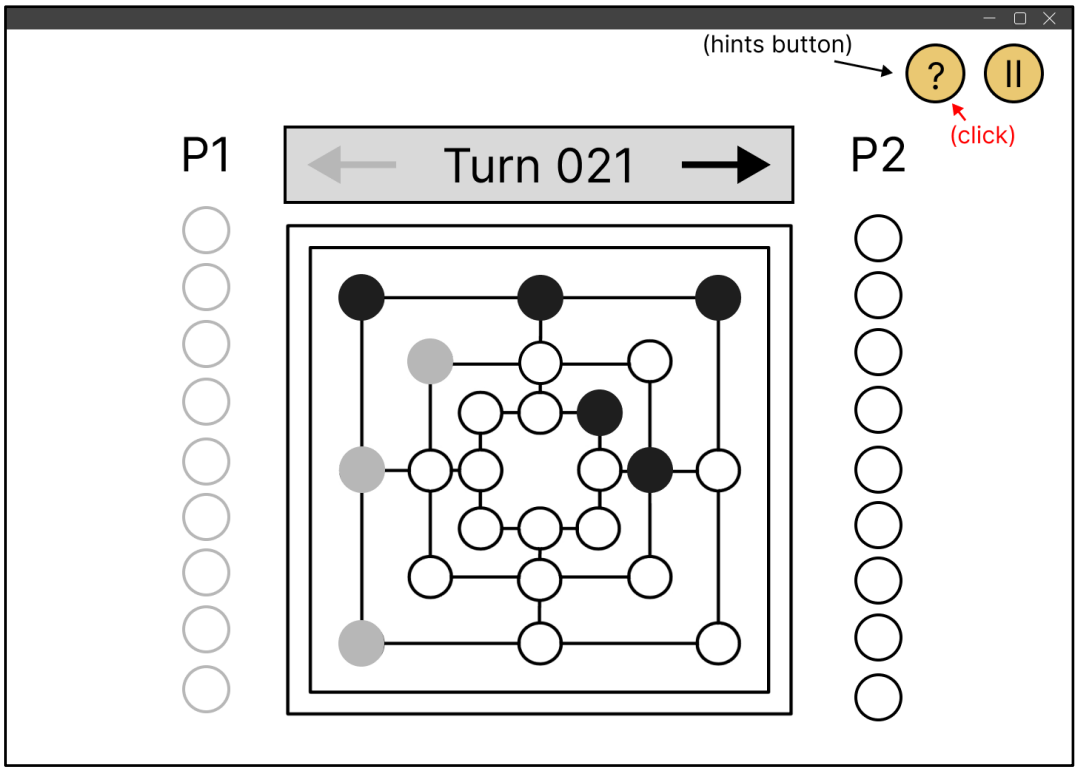


Frame 7: Player is taught how to fly a piece when they are down to three pieces. Player flies the piece by clicking an intersection on the other side of the board as highlighted by the tutorial.

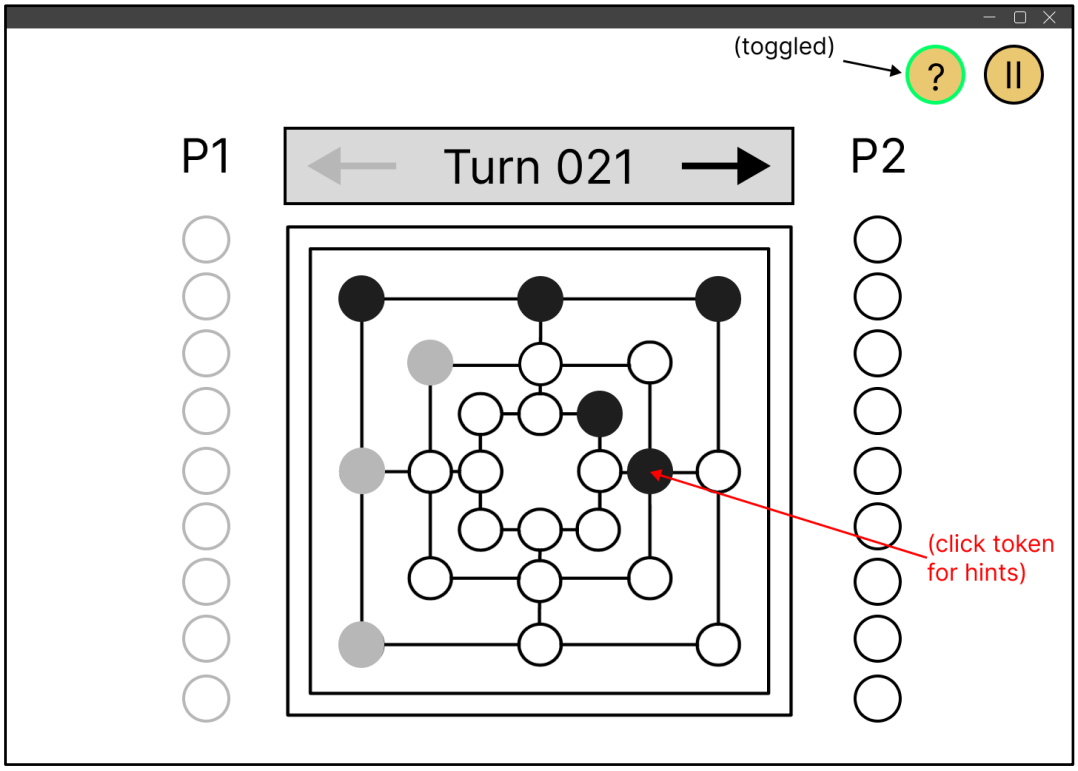


Frame 8: This concludes the tutorial and player returns to the main menu.

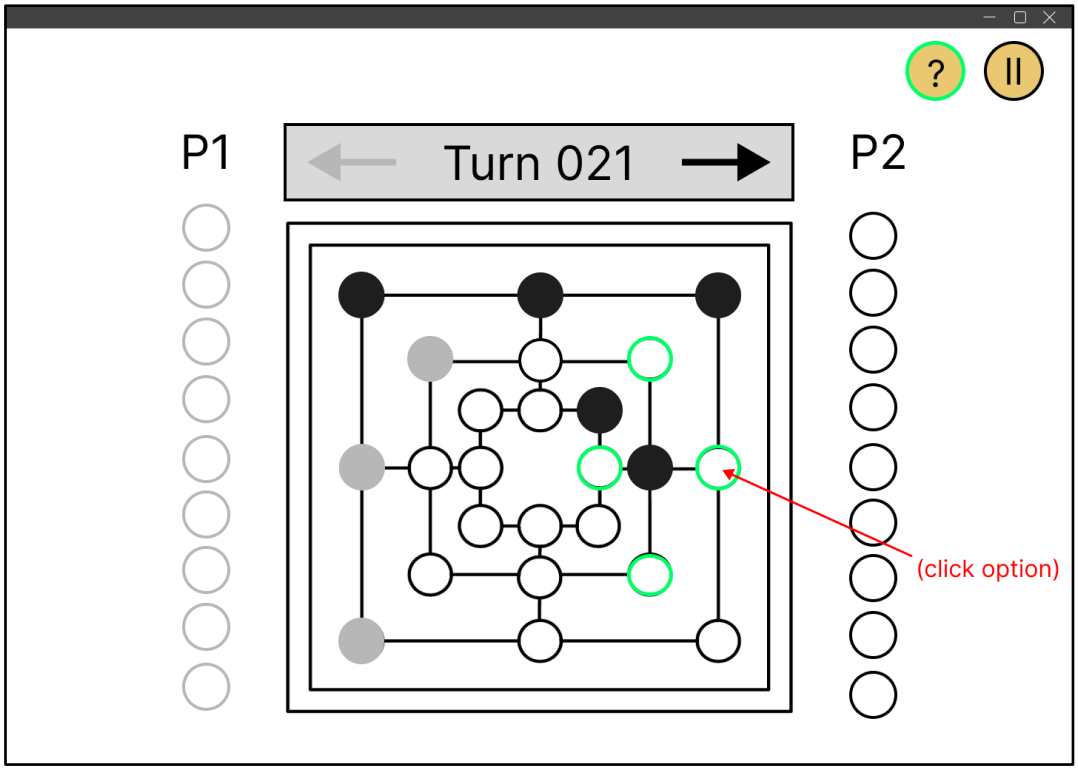
Toggle Hints (Advanced Requirement(a))



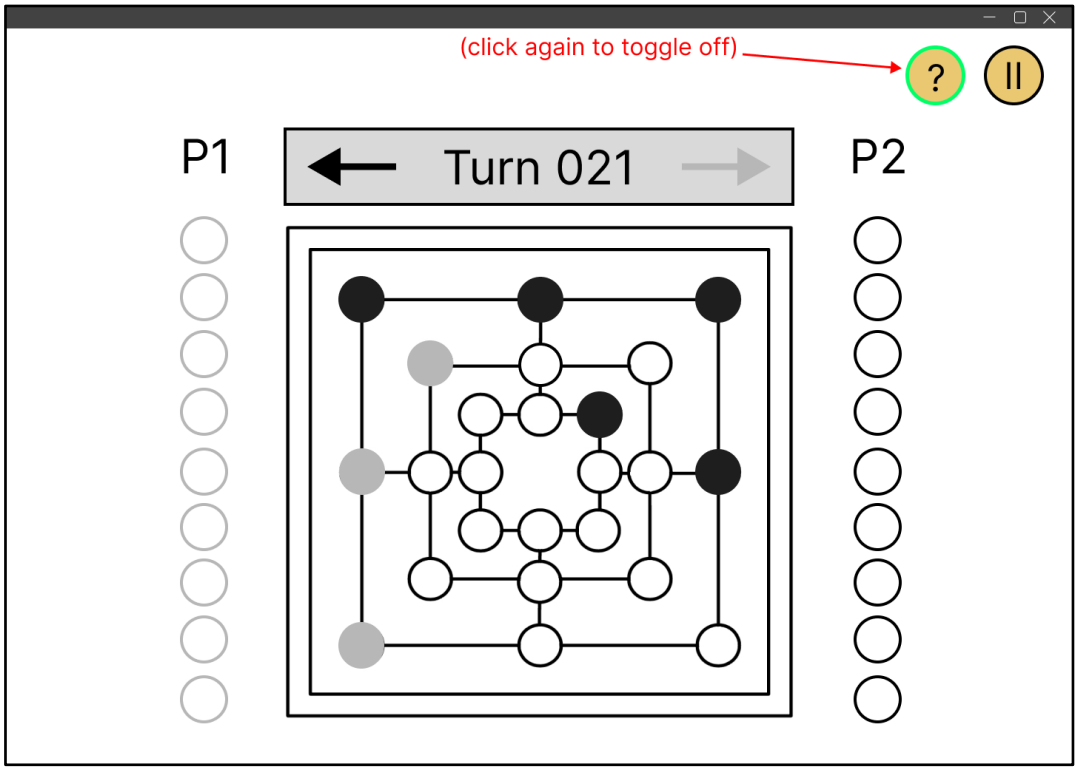
Frame 1: A button with a question mark toggles the hints on/off.



Frame 2: After toggling hints on the player whose turn is it can view all the legal moves of a piece by clicking them.

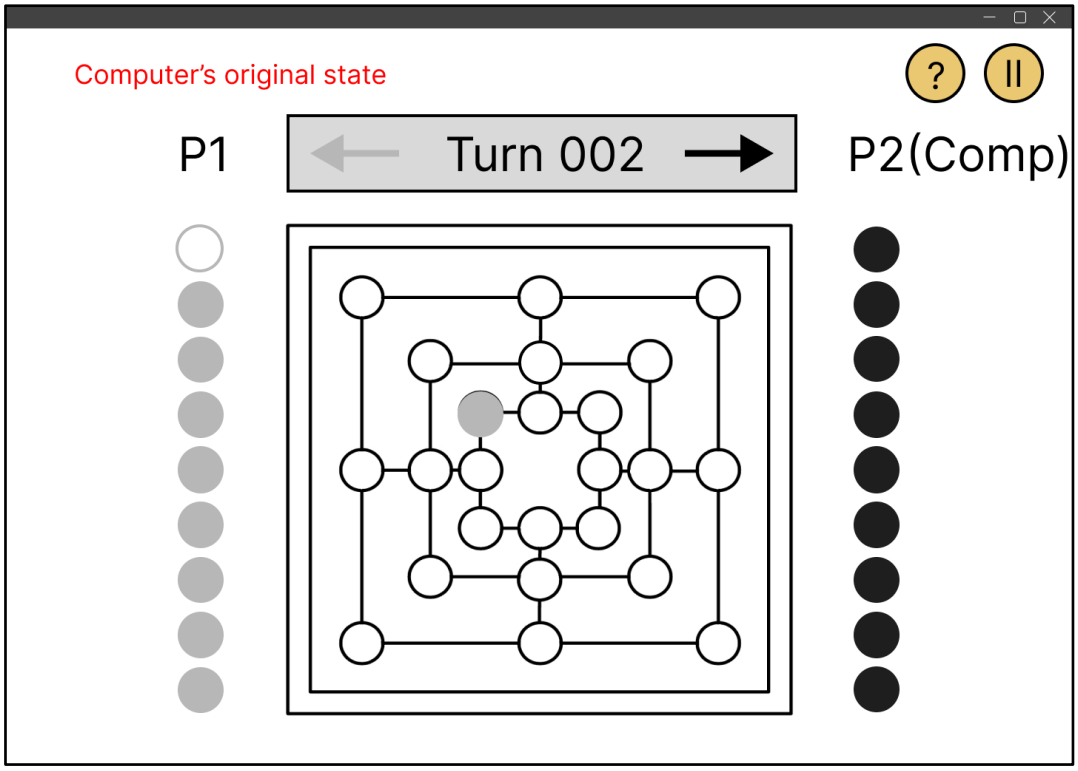


Frame 3: After the player clicks the piece the legal intersections that the piece can be moved to will be highlighted.

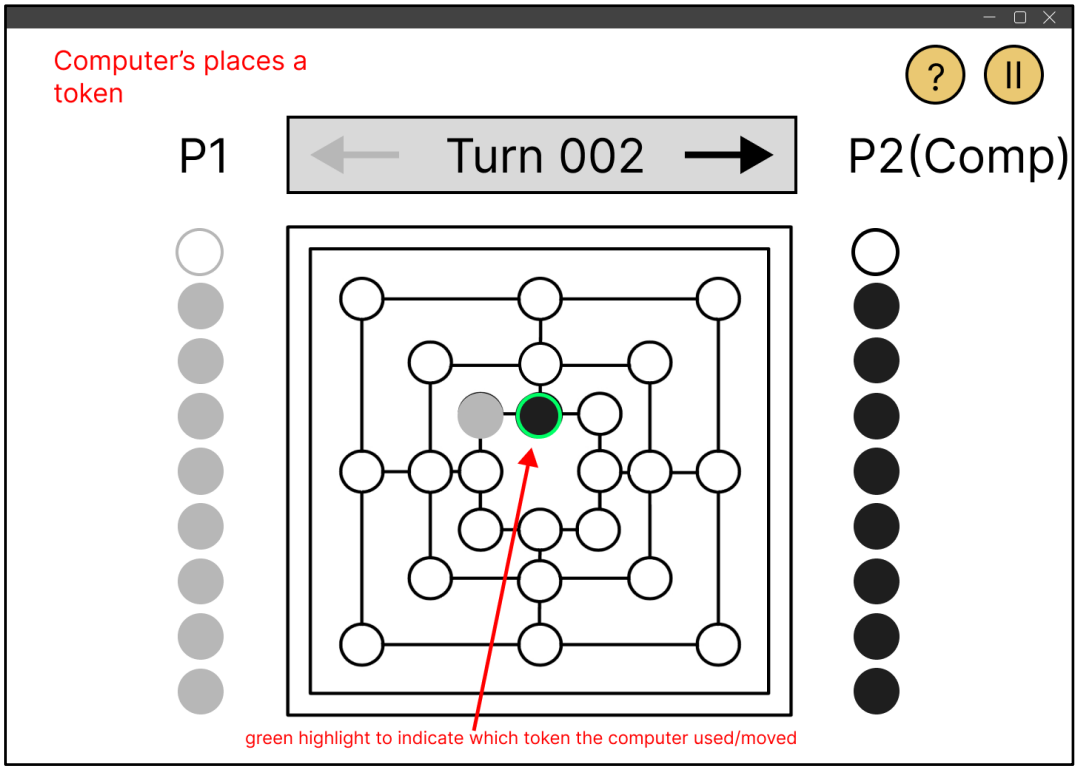


Frame 4: Clicking on the hints button again will toggle the hints off.

Computer Player (Advanced Requirement(c))



Frame 1: Player can play against computer by just clicking on P VS C in the main menu.



Frame 2: After human player plays their turn the computer automatically makes its move (legal) like a normal player.