FIT3077: SOFTWARE ENGINEERING: ARCHITECTURE AND DESIGN

Sprint 4 (30%)

Design Rationale

Done By

Shillong Xiao (Norman)	**(30257263)**
Sok Ear	**(32442688)**
Kisara Batugedara	**(30912539)**

Faculty of Information Technology

Note for the project

The working game prototype can be seen under **FX branch**. This program is designed for a system with more than 27 inches. If the screen size is below this one way to fix the problem is by adjusting the resolution of the screen.

For example, for Macbook, this could be done by going to system preferences, display, then select “more space”.

Design Rationale

Advanced Requirement - Tutorial Mode and Hint Button Feature

Implementation of Tutorial Mode through a seperate TutorialGame class.

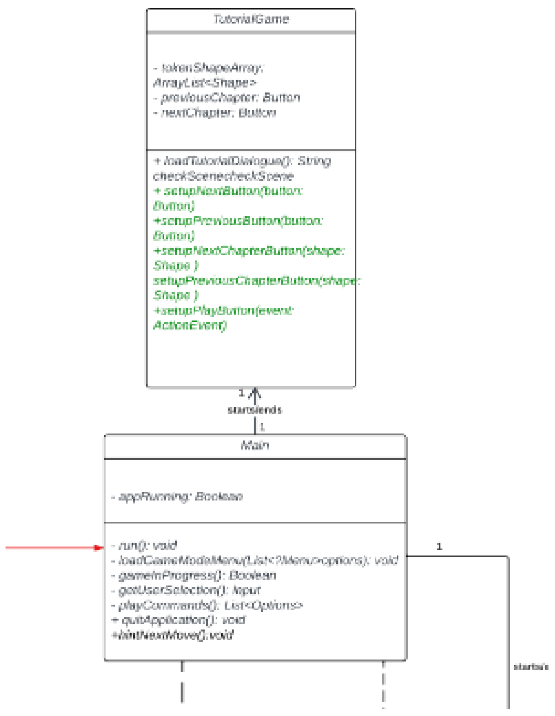
To implement the tutorial mode, we ensured that a class called TutorialGame handled all the functionalities for it. The methods inside this class implement the tutorial in a slideshow format where there are different chapters that depict a certain scenario in the game and within a certain chapter, there are different pages that explain multiple rules and conditions of the 9 Men’s Morris Game. For each chapter that depicts a certain scenario, the tokens were hardcoded to represent the scenario accurately for better visualisation and understanding.The reason why we went with this approach is because it helps the user understand all the possible scenarios in the game which in turn makes them understand the rules better with visual representation.

Implementing this feature of the game via a new class did not trouble us in terms of changing the architecture of the game as all we had to do was create a new class and have it handle the tutorial functionalities through its methods.

The method of implementation of this advanced feature was finalised in our first sprint itself as our group had agreed having one class handle the functionalities of the tutorial mode was far more effective in terms of implementation and as well as making it easier for users to understand the rules of the game better. In terms of the difficulty level for implementing this feature, it was quite easy to integrate it to the current system as all that was required was to implement a class and its respective methods to handle the tutorial slideshow. This ensures that our programming practices also abides by basic Object-Oriented Principles such as the **Open/Close Principle** where the software is open for extension and closed for modification of source code.

Advantages	Disadvantages
- **Improved Code Organization**: Implementing a separate class TutorialGame, helps make the code more organised in terms of all its functionalities. - **Reusability:** Since this feature is encapsulated through a separate class, it can be reused again if in the future, more game modes are introduced and tutorials are required for each game mode.	- **Increased Complexity:** Introducing another class entity adds an additional layer of complexity to the codebase. This can lead to larger class sizes, increased cognitive load, and potentially more challenging debugging and maintenance.

Conclusion: After careful consideration, despite it being complex in terms of debugging and maintenance we decided it was best to go ahead and implement the tutorial mode in the form of a slideshow that is handled by a class called TutorialGame.


TutorialGame
- tokenShapeArray: ArrayList<Shape>
- previousChapter: Button
- nextChapter: Button
+ loadTutorialDialogue(): String
checkScenecheckScene
+ setupNextButton(button: Button)
+setupPreviousButton(button: Button)
+setupNextChapterButton(shape: Shape)
setupPreviousChapterButton(shape: Shape)
+setupPlayButton(event: ActionEvent)
1
starts/ends
1
Main
- appRunning: Boolean
- run(): void
- loadGameModeMenu(List<?Menu>options): void
- gameInProgress(): Boolean
- getUserSelection(): Input
- playCommands(): List<Options>
+ quitApplication(): void
+hintNextMove():void
1
starts/s

Implementation of the Hint Button feature by adding methods in existing classes.

To implement the hint button feature, we decided the best way to go about this was to have a button exist on the right bottom corner of the application. When a user clicks on the hint button, the game highlights the positions that the user can place their respective tokens at in a purple colour. This was implemented by adding methods to existing classes such as FxController, Display and Game.

Decision 1: The Hint button must highlight the specific positions that the user can place their tokens at when they're unsure of what to do next. Therefore, should Hint be implemented as a class or just a method?

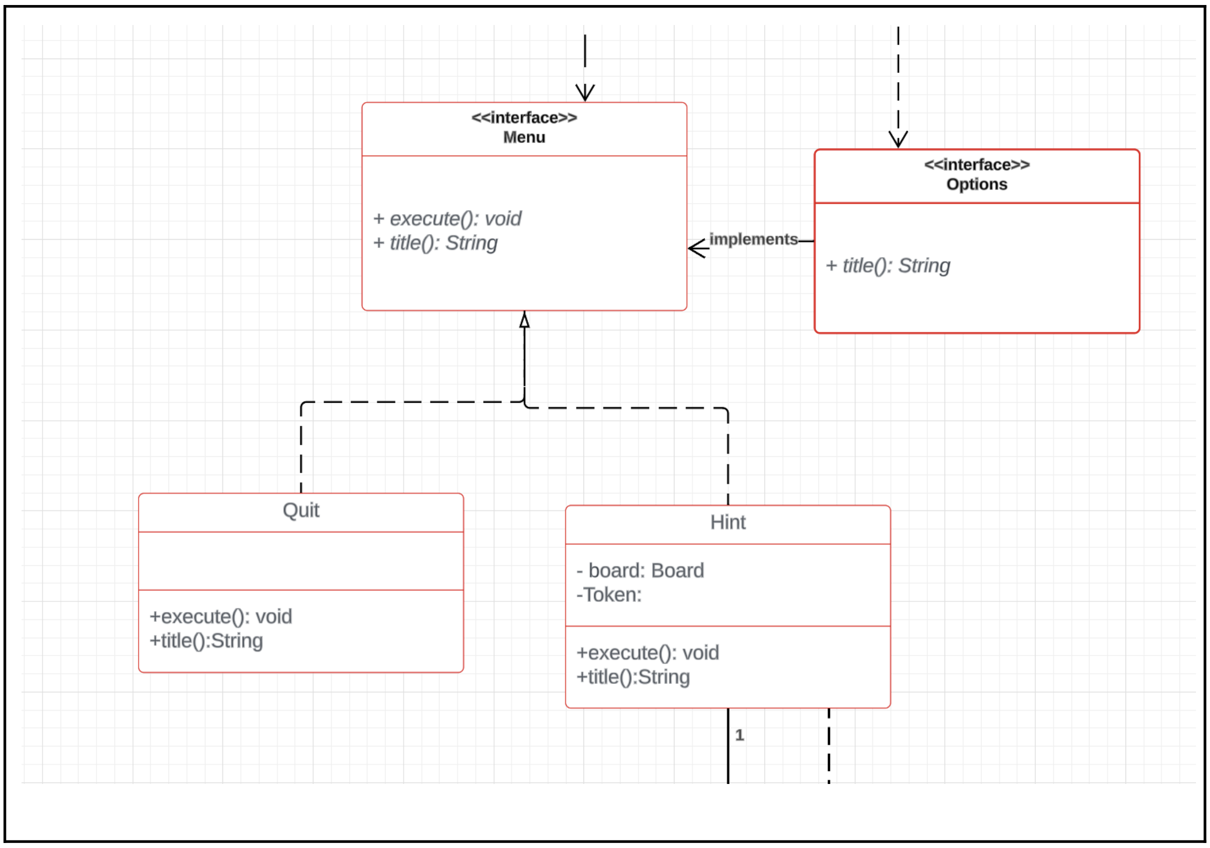
Alternative 1 Hint is made into a class that handles all functionalities regarding hints.	**Alternative 2** Hint is made into a method handled by Game class
Advantages - It has one single responsibility which is highlighting all the positions that the player can move their token into. Therefore it conforms to the Single Responsibility Principle - It increases readability as all the functions related to hint actions are encapsulated within a single class. **Disadvantages** - Making it a class can lead to decreased performance and introduce more complexity - It will require more development time as it requires integration with other classes. - It will lead to an increased level of dependency in the codebase since the Hint Action will require dependencies on the Board class, Display class, and Token class, which is undesirable.	**Advantages:** - It ensures simplicity - It ensures better integration. When the FxController class receives an on-click event on the Hint button, it will call the Game class to execute the hint. Since the game already has dependencies on the Board to find the legal moves and the Display to display the glowing effect on the Board, it will enhance efficiency and prevent unnecessary complexity that can complicate implementation and maintenance in the future - The current game architecture involves the interaction between FxController (controlling the UI) and the rest of the game logic to be done through Game class. Not having a separate Hint Action class ensures that there remains a single entry point, which contributes to a higher level of maintainability in the code. **Disadvantages** - This will increase the responsibility of the Game class, which may lead to an unbalanced workload distribution among the classes.
Decision	

After careful consideration, alternative 2 is selected due to better integration with the rest of the game. Moreover, this approach promotes code organisation and maintainability. By having a single entry point, the Game class maintains its role as the central hub for managing game logic, including hint generation. This centralised control enhances the maintainability of the codebase, making it easier to understand and update.

Key Change in Architecture: Removal of Menu Interface

One crucial modification is the elimination of the Menu interface. The Menu interface was originally designed to handle the on-click event of Hint and Quit buttons, with the intention of sharing the implementation of this interface between the two game modes: Tutorial game mode and PVP game mode. However, due to our team's specification changes, tutorial game mode no longer requires any of these functionalities. Therefore, the Menu interface is no longer needed.

Decision Remove Menu Interface
Advantages - It allows the menu functionalities such as Quit and Hint buttons to be handled by FxController instead of another interface. This ensures that all the UI event handling stays encapsulated in the FxController class. This can promote better encapsulation which endorses maintainability in the future. - Given the current implementation of the game where Game mode and Tutorial Game mode share no common menu functionality, the implementation of a menu interface is redundant. **Disadvantages** - It may lead to future code repetition if there were to be other game screens or game modes in the future that require the usage of Quit or Hint Buttons. This repetition can violate the "Don't Repeat Yourself" principle.
Decision Upon careful consideration, this decision is adopted. Given the current stage of the sprint, the menu interface does not provide any meaningful value. JavaFX already allows for seamless integration of buttons onto the screen without the requirement of an interface for the functionality to work properly. Additionally, considering that the game solely consists of Tutorial mode and PVP Game Mode, each with distinct menu interfaces, the presence of a generic menu interface becomes redundant and serves little purpose for the game's specific requirements. interface, the codebase becomes more streamlined and focused, ensuring efficient and purposeful implementation of the game's functionalities. Consequently, this design change will remove the need of another interface: Option, and 2 classes: Quit and Hint.

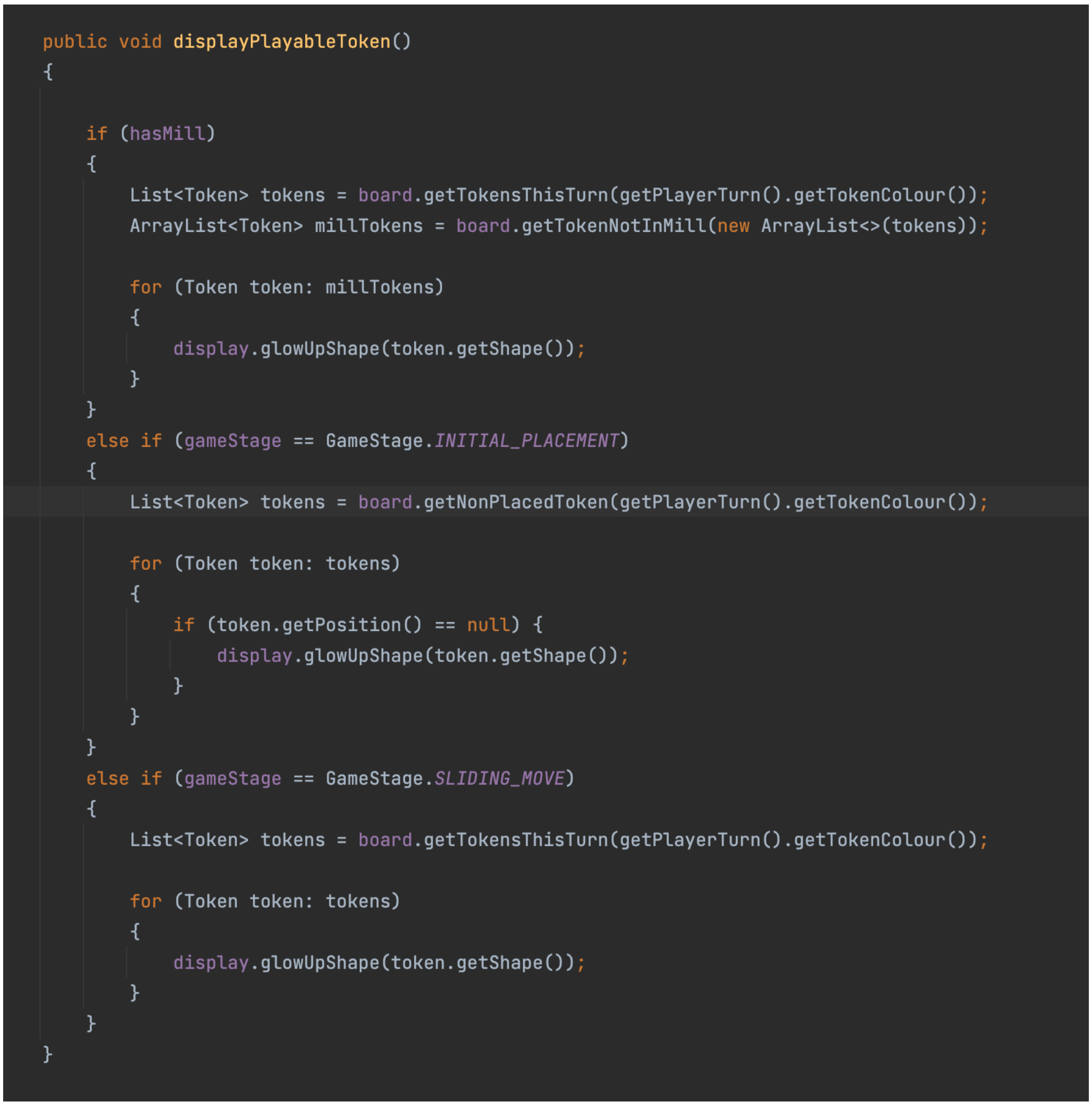


The implementation of Hint button action required no refactoring on the previous code base. However, it required a few functionalities to be added to JavaFx class, Game class and Display class.

- In the FXController class, a new attribute named hintButton: Shape needs to be added to represent the hint button present on the game screen. Consequently, a new method setupHintButton(hintButton: Button) has been implemented to handle the on-click event of this button.
- Within the Game class, two additional methods have been introduced to the Display class:
  - displayPlaceablePositionFor(token: Token)
  - displayPlayableToken()
  - These methods are responsible for displaying the appropriate tokens or positions to provide hints during gameplay.
- In the Display class, two new methods have been added:
  - glowUpShape(Shape shape)
  - resetGlowUp()
  - The glowUpShape() method is used to display the hints by highlighting specific shapes, while the resetGlowUp() method is responsible for reverting the hints back to their original state after the move has been made.

Difficulty in implementing the advanced requirements

Most of the existing functionalities already present in the code greatly facilitated the implementation of these additional functionalities, eliminating the need to create new ones.



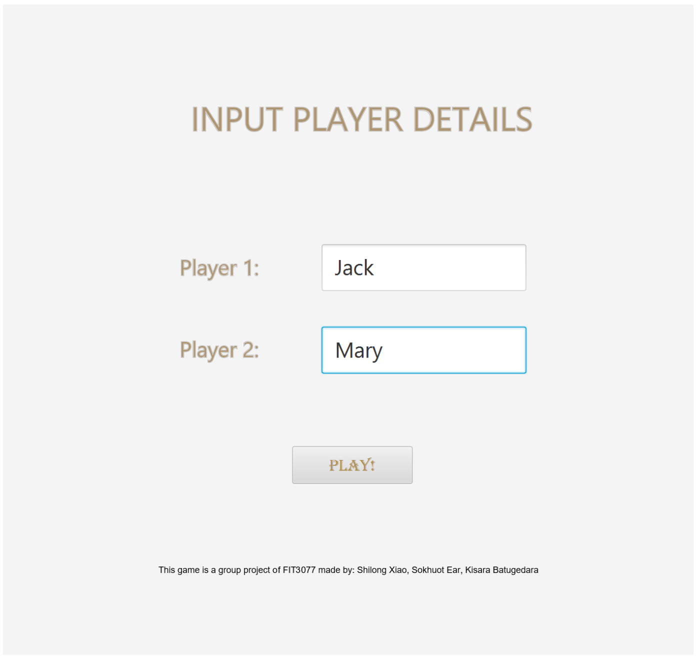
```java
public void displayPlayableToken()
{

    if (hasMill)
    {
        List<Token> tokens = board.getTokensThisTurn(getPlayerTurn().getTokenColour());
        ArrayList<Token> millTokens = board.getTokenNotInMill(new ArrayList<>(tokens));

        for (Token token: millTokens)
        {
            display.glowUpShape(token.getShape());
        }
    }
    else if (gameStage == GameStage.INITIAL_PLACEMENT)
    {
        List<Token> tokens = board.getNonPlacedToken(getPlayerTurn().getTokenColour());

        for (Token token: tokens)
        {
            if (token.getPosition() == null) {
                display.glowUpShape(token.getShape());
            }
        }
    }
    else if (gameStage == GameStage.SLIDING_MOVE)
    {
        List<Token> tokens = board.getTokensThisTurn(getPlayerTurn().getTokenColour());

        for (Token token: tokens)
        {
            display.glowUpShape(token.getShape());
        }
    }
}
```

For instance, functionalities like board.getNonPlacedToken() and board.getTokenPlayerThisTurn() have already been defined and implemented. This pre-existing setup made the implementation process easier and more streamlined. It highlights the modularity of the methods within the board class, which are not only useful but also well-defined. This modularity enables easy reuse of these methods and contributes to the overall support and future development and extension of the code.

One potential improvement is to use an enum type for setting up game stages. Although the current implementation handles different game scenarios using if-else conditions, this approach may become less extensible if additional game modes are introduced in the future. However, considering the current requirements only involve three game stages, introducing more abstractions to cover these stages could unnecessarily complicate the game, given the current requirements.

Screenshots

Input player name:

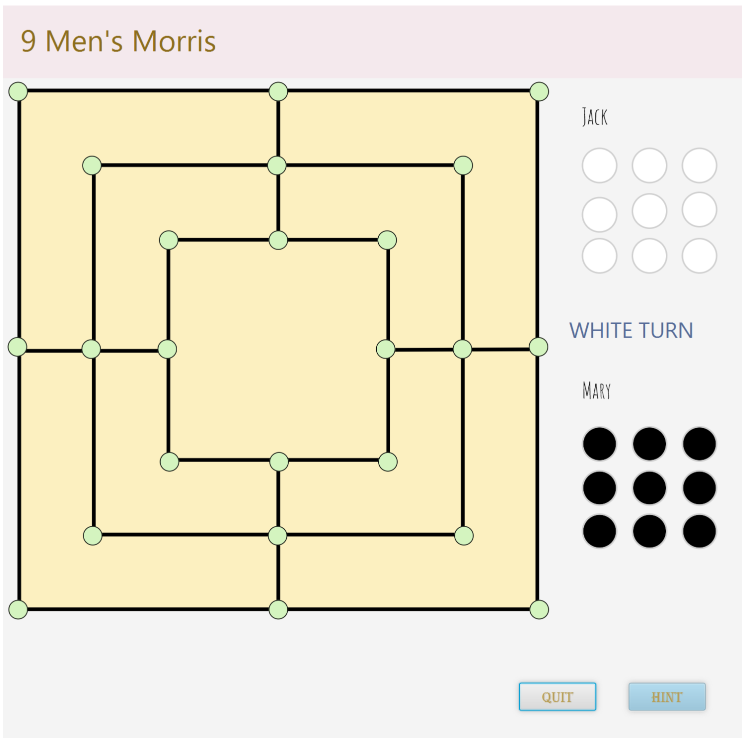

INPUT PLAYER DETAILS

Player 1: Jack

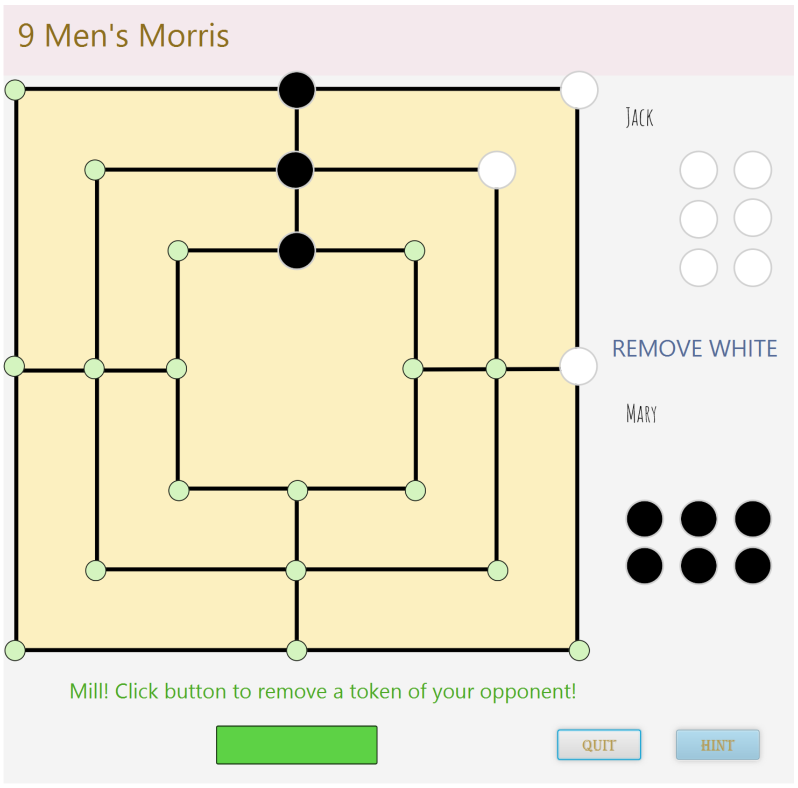
Player 2: Mary

This game is a group project of FIT3077 made by: Shilong Xiao, Sokhuot Ear, Kisara Batugedara

Empty board

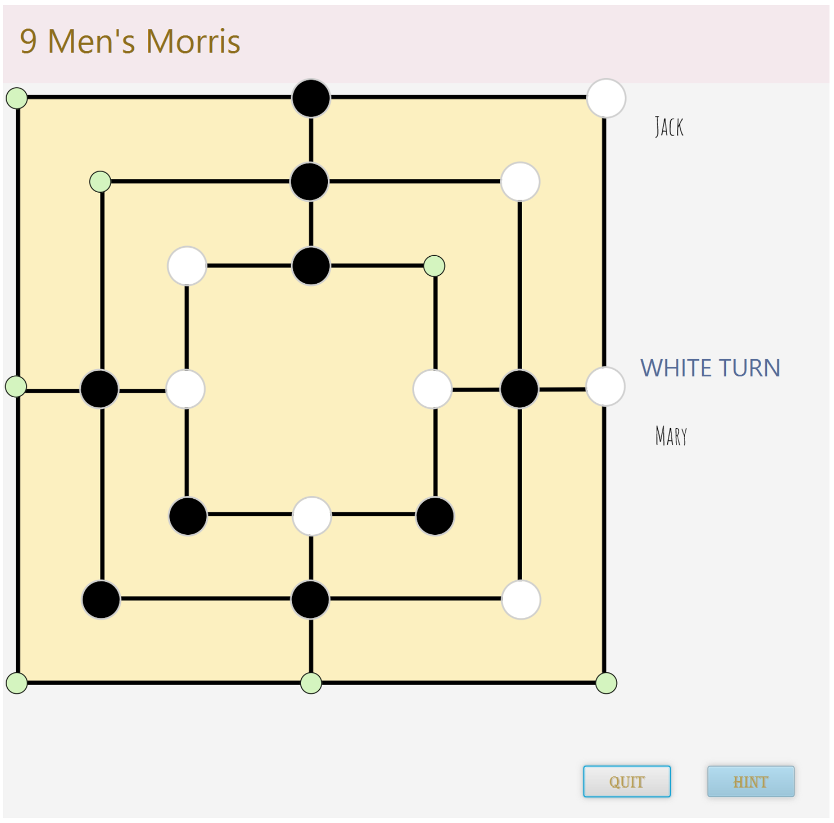


Mill happens:

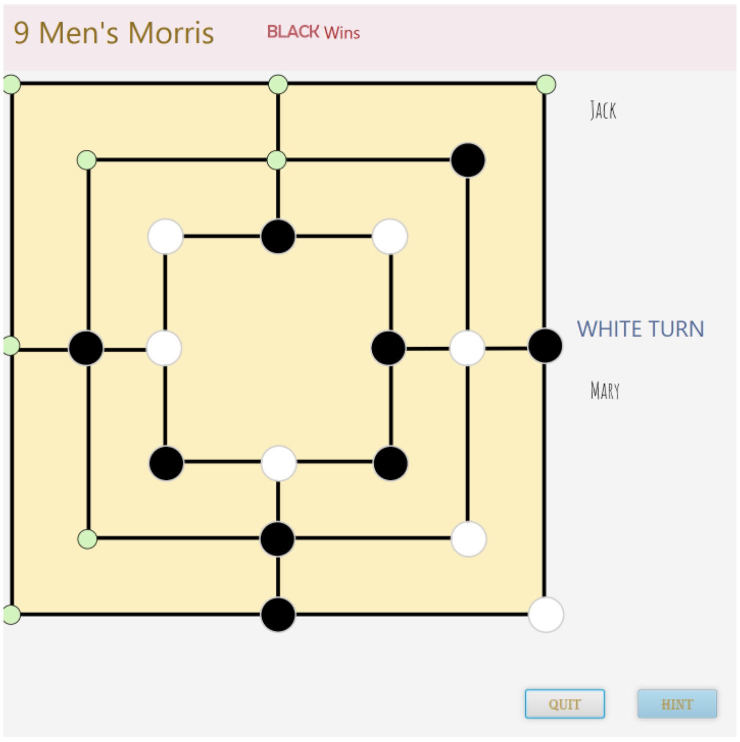


Mill! Click button to remove a token of your opponent!

Slide Move starts:

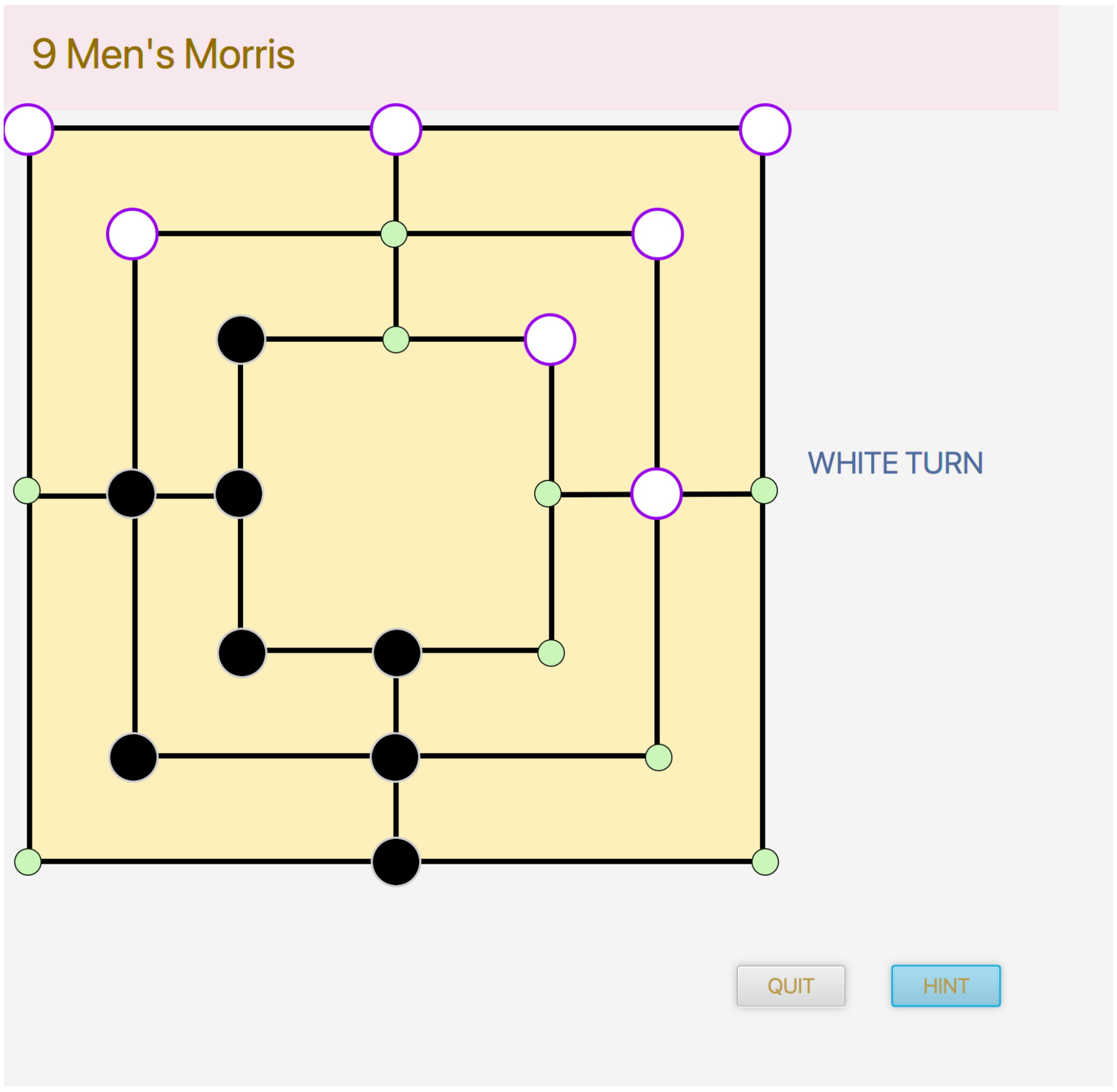


Winning condition: No available moves

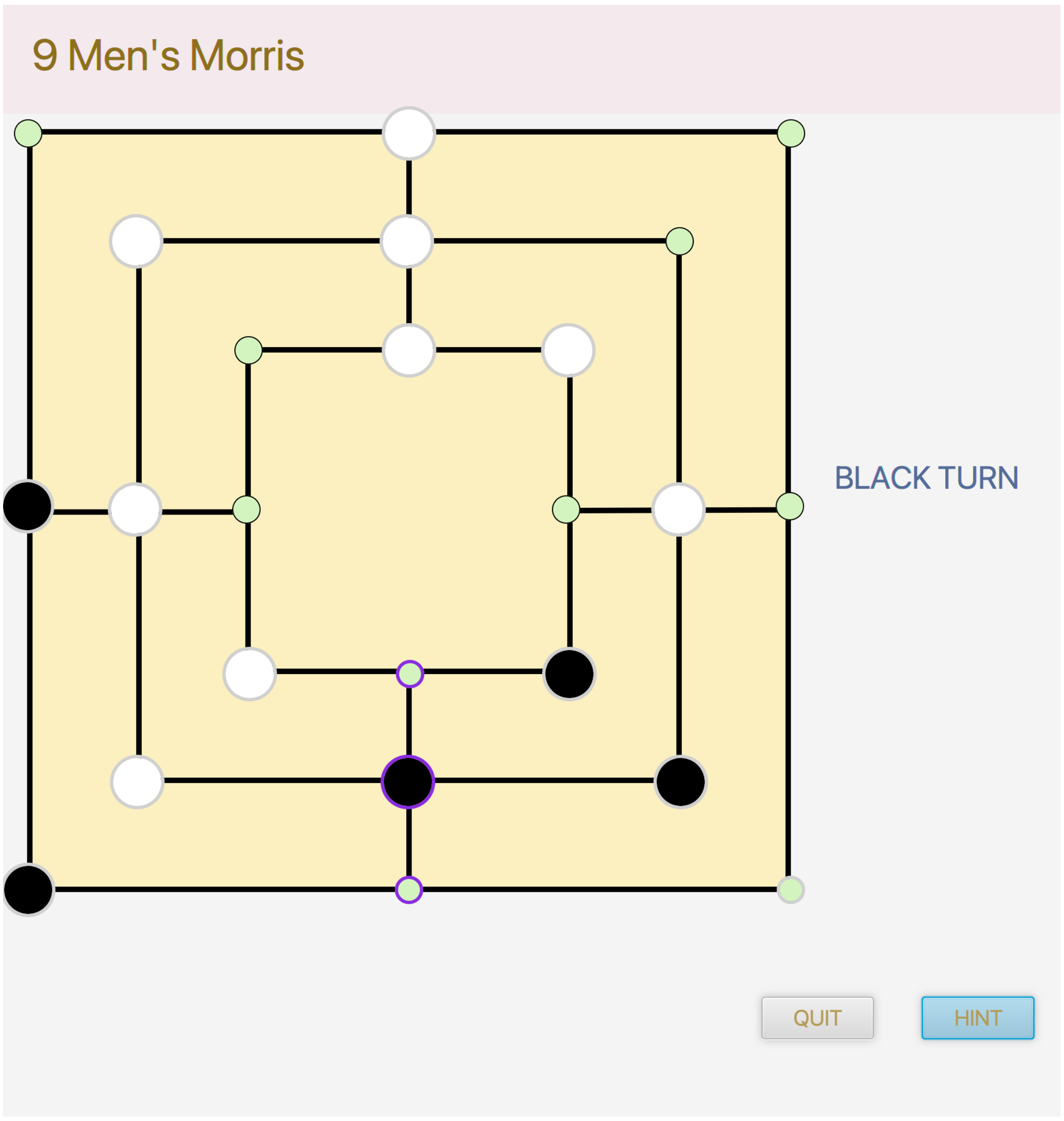


We have demonstrated the winning condition of two tokens left in the video :)

Hint button when pressed with no token selected

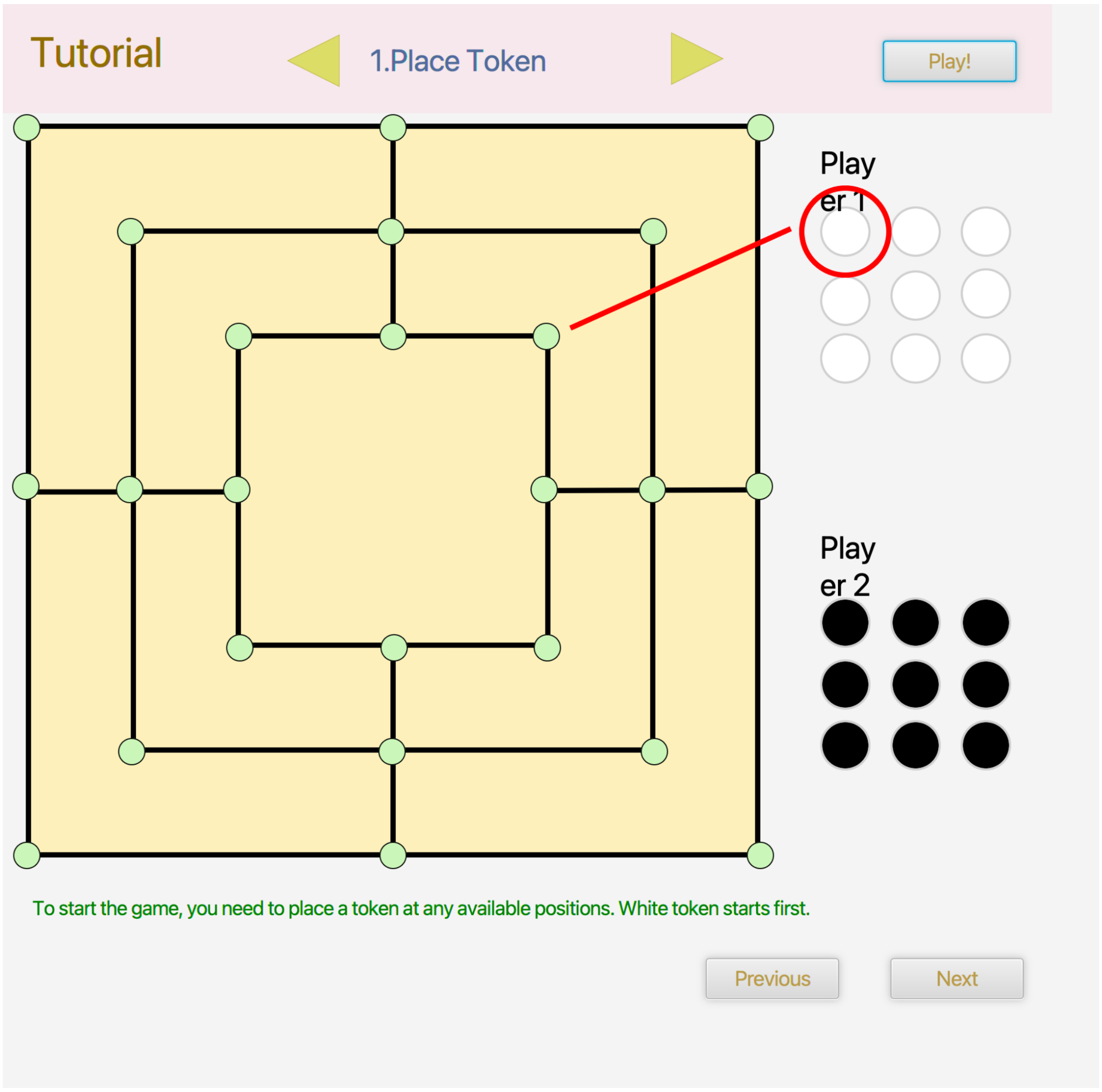


Hint button when clicked after a token is clicked

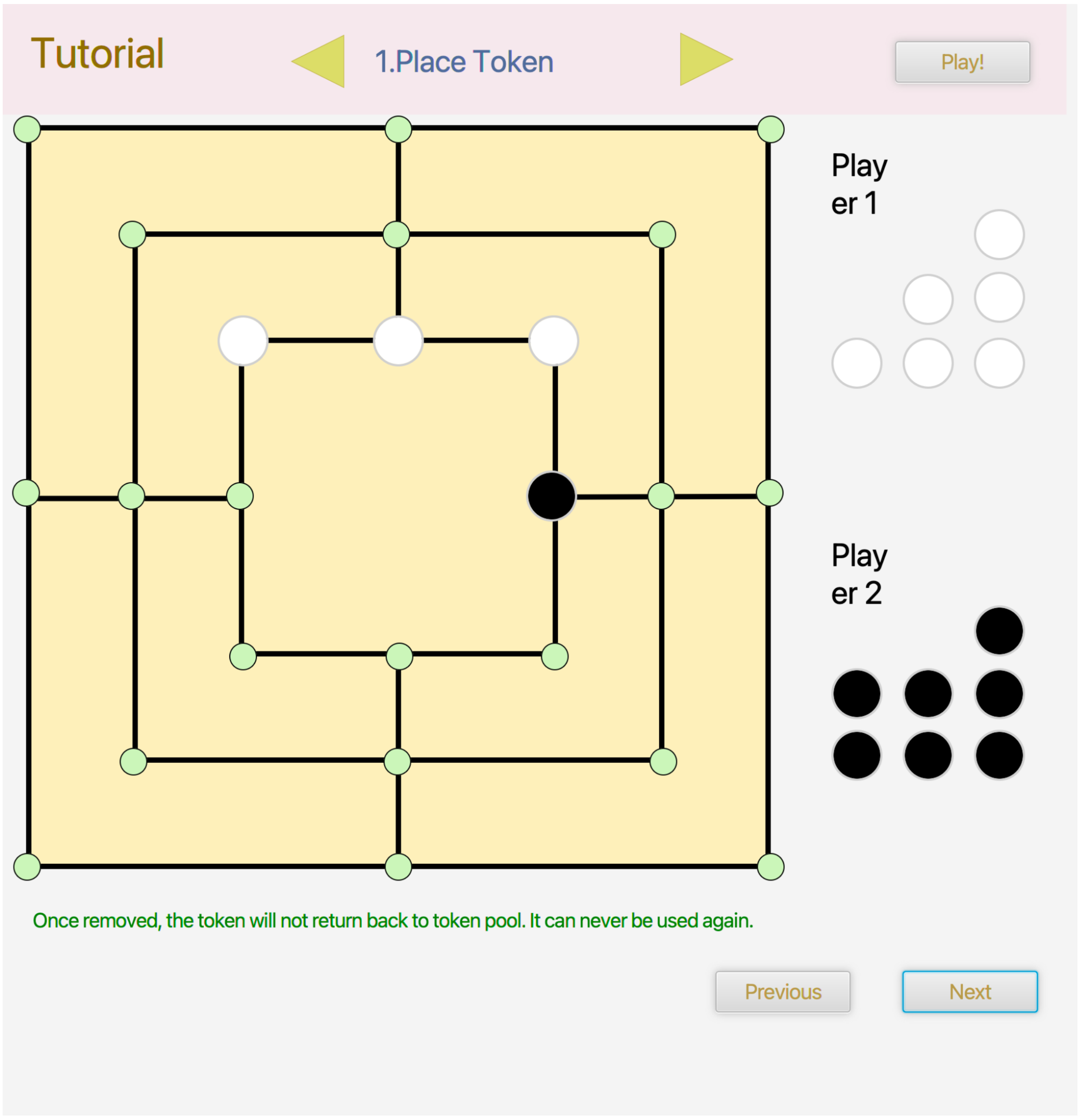


Tutorial:

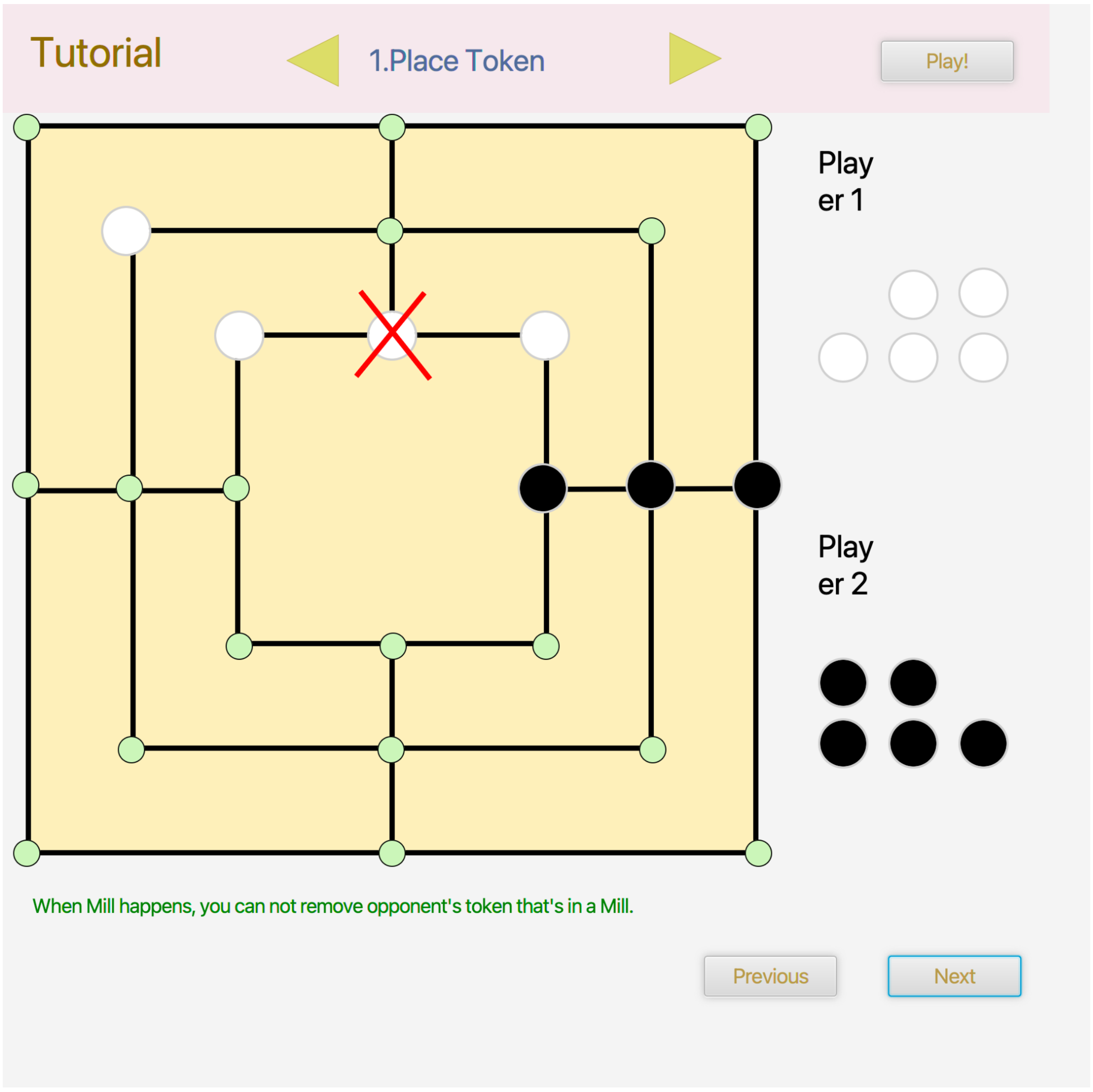
One screenshot for each chapter only as there are many scenarios



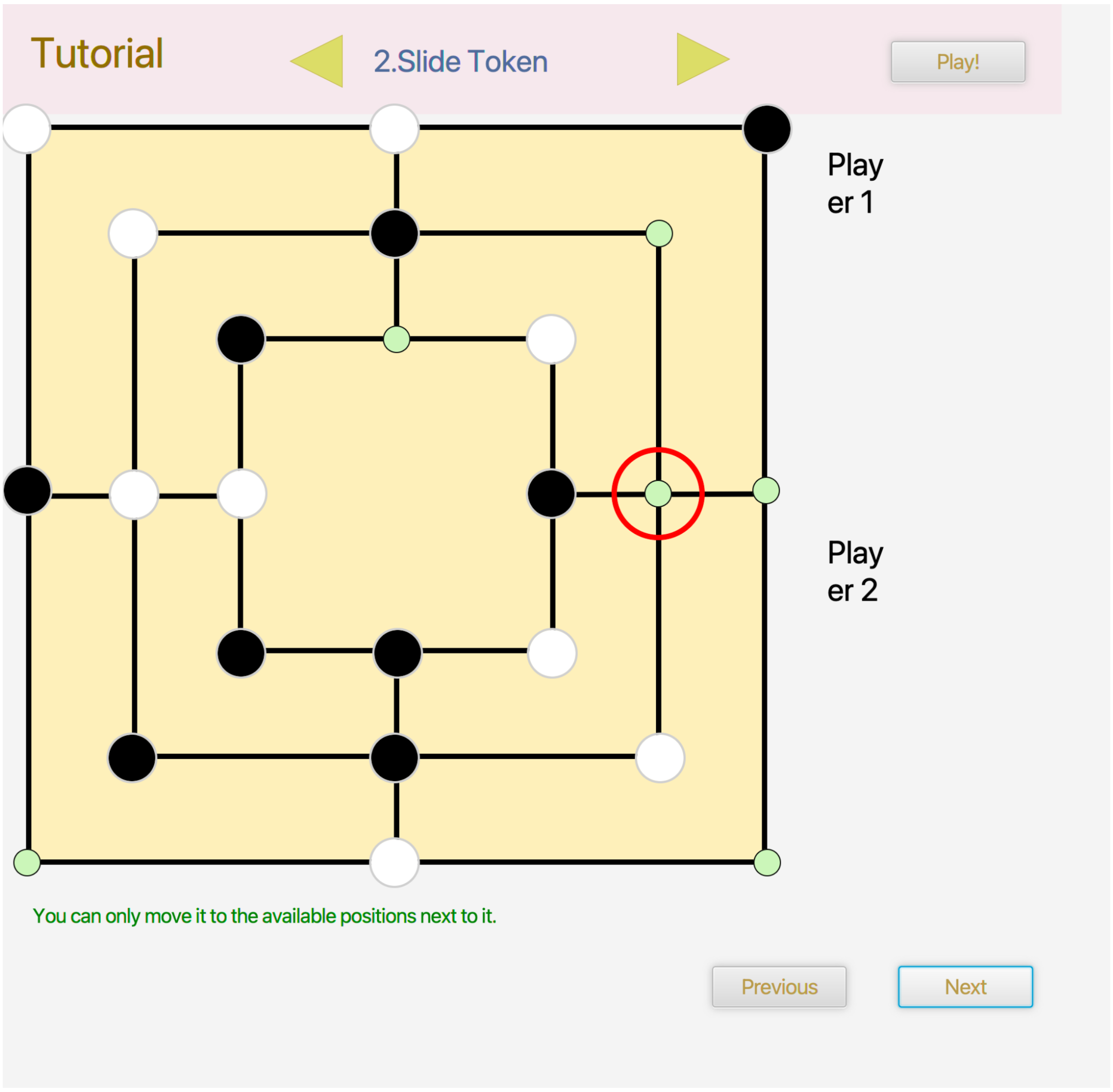
To start the game, you need to place a token at any available positions. White token starts first.



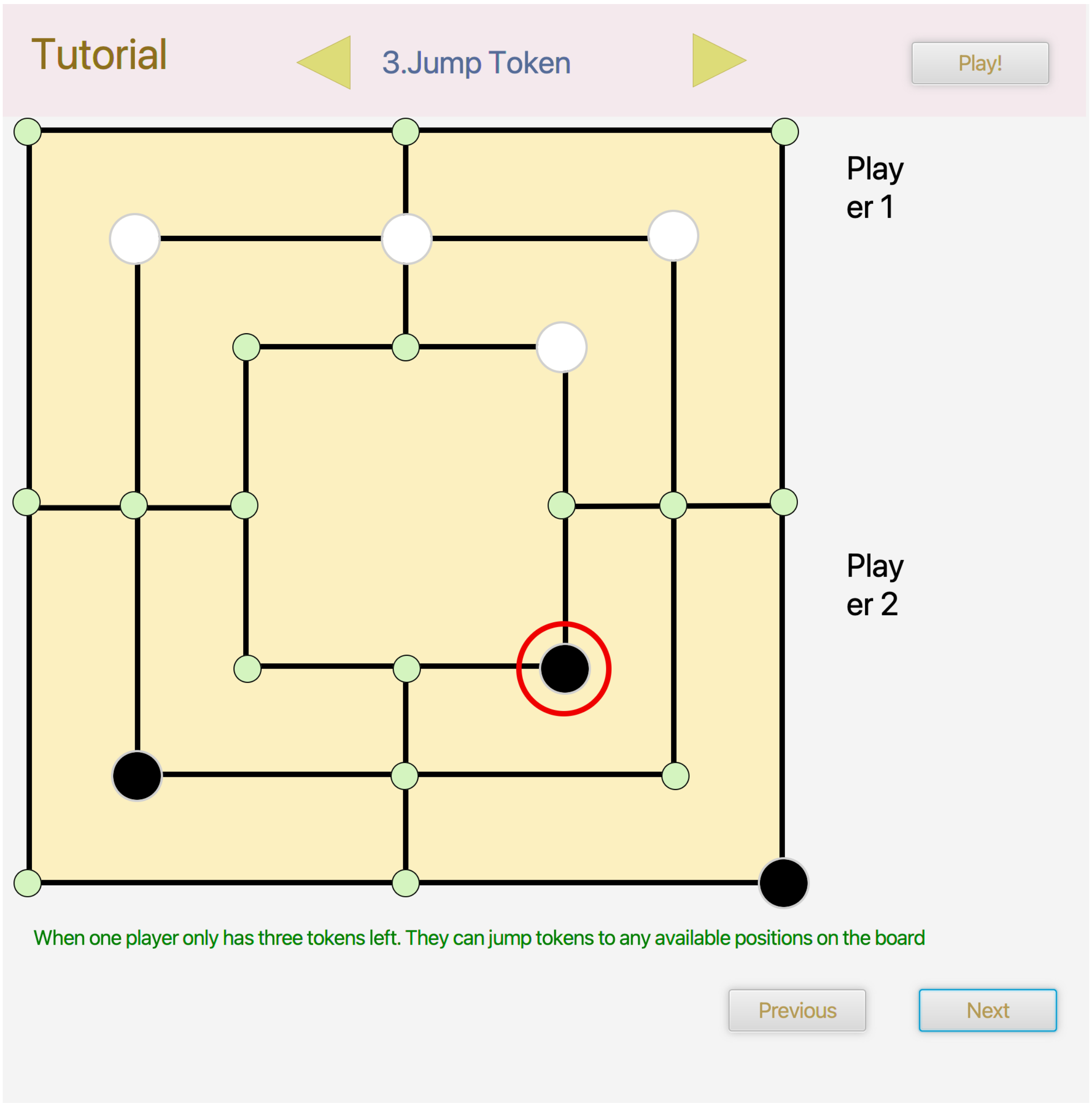
Once removed, the token will not return back to token pool. It can never be used again.



When Mill happens, you can not remove opponent's token that's in a Mill.



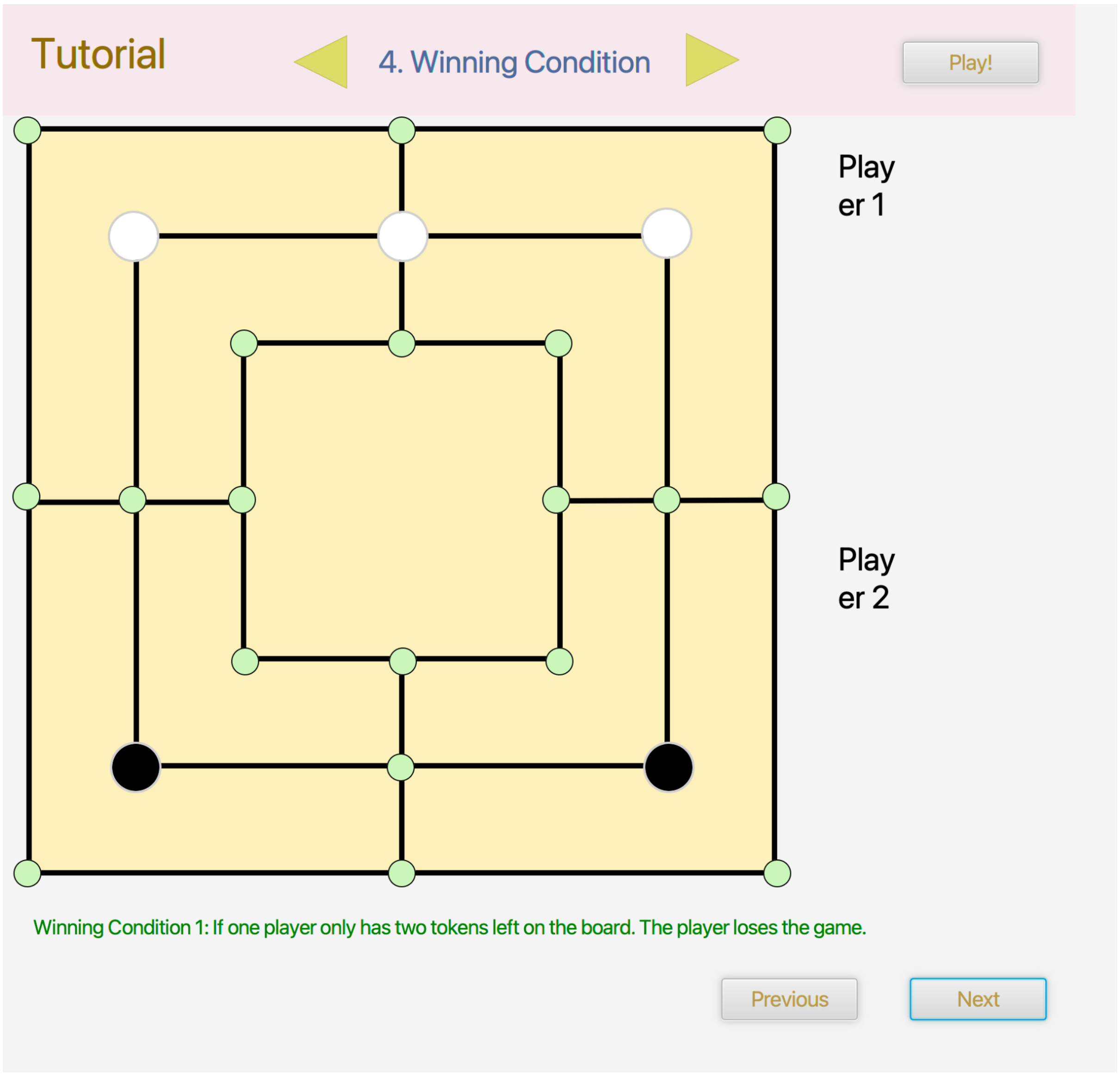
You can only move it to the available positions next to it.



Tutorial

3.Jump Token

When one player only has three tokens left. They can jump tokens to any available positions on the board



Winning Condition 1: If one player only has two tokens left on the board. The player loses the game.