Advanced Tic Tac Toe Game Development Project Report

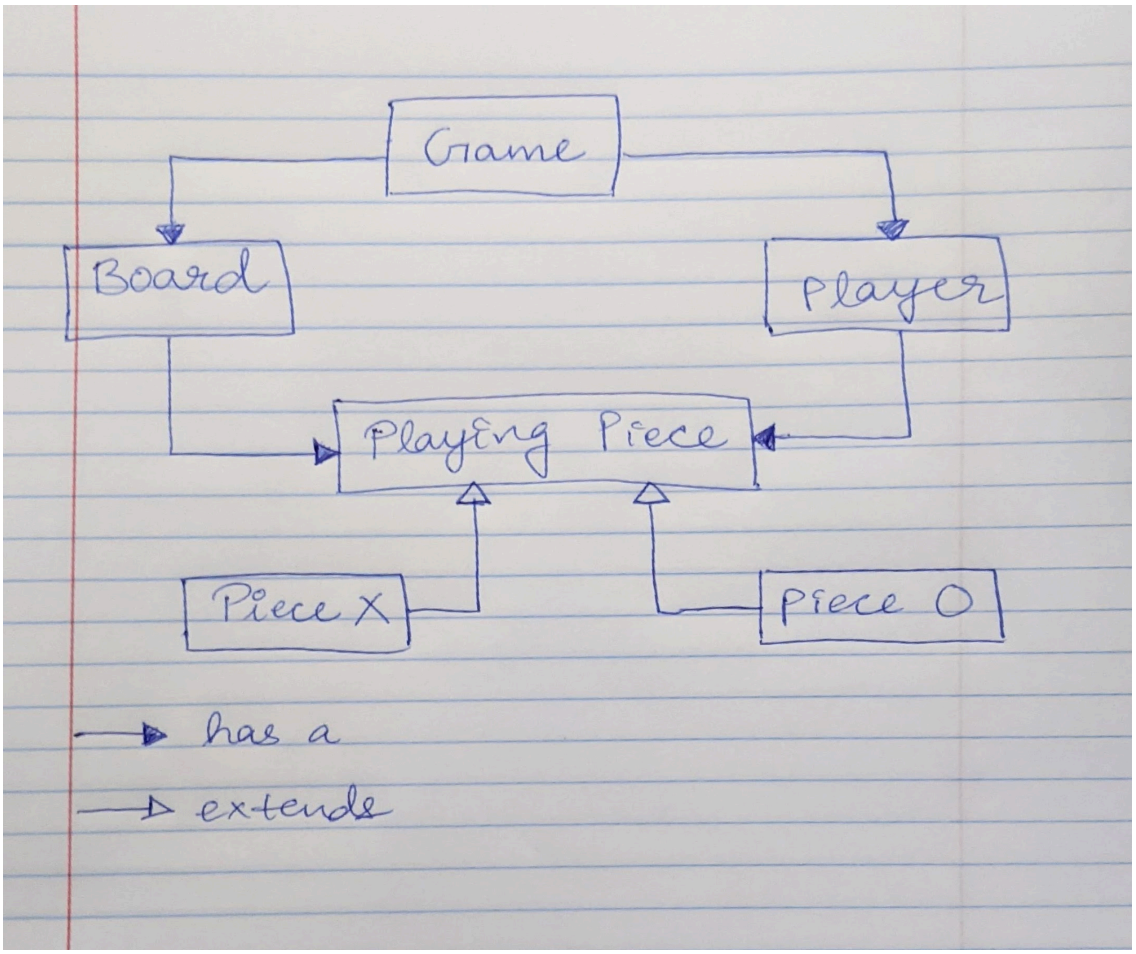
Project Introduction:

Our project is to implement a well-known game, Tic Tac Toe, but having advance features such as Undo and Redo and have implemented the complete game using multiple design patterns.The classic game of Tic Tac Toe is being revolutionized with advanced features aimed at enhancing user engagement and strategic gameplay. This project leverages modern game development techniques and technologies to deliver a more interactive and engaging experience to players.The scope of the project includes introducing new gameplay mechanics and a sophisticated graphical user interface using JavaFX, enriching the game's appeal and complexity.

Objective:

The objective is to develop an advanced version of Tic Tac Toe that incorporates strategic elements like the ability to undo and redo moves, thus providing a dynamic gaming experience that allows players to learn from their actions and adjust strategies in real time.Designed for enthusiasts of puzzles and strategy games of all ages, this version appeals through its user-friendly interface and the ability to alter gameplay dynamically.

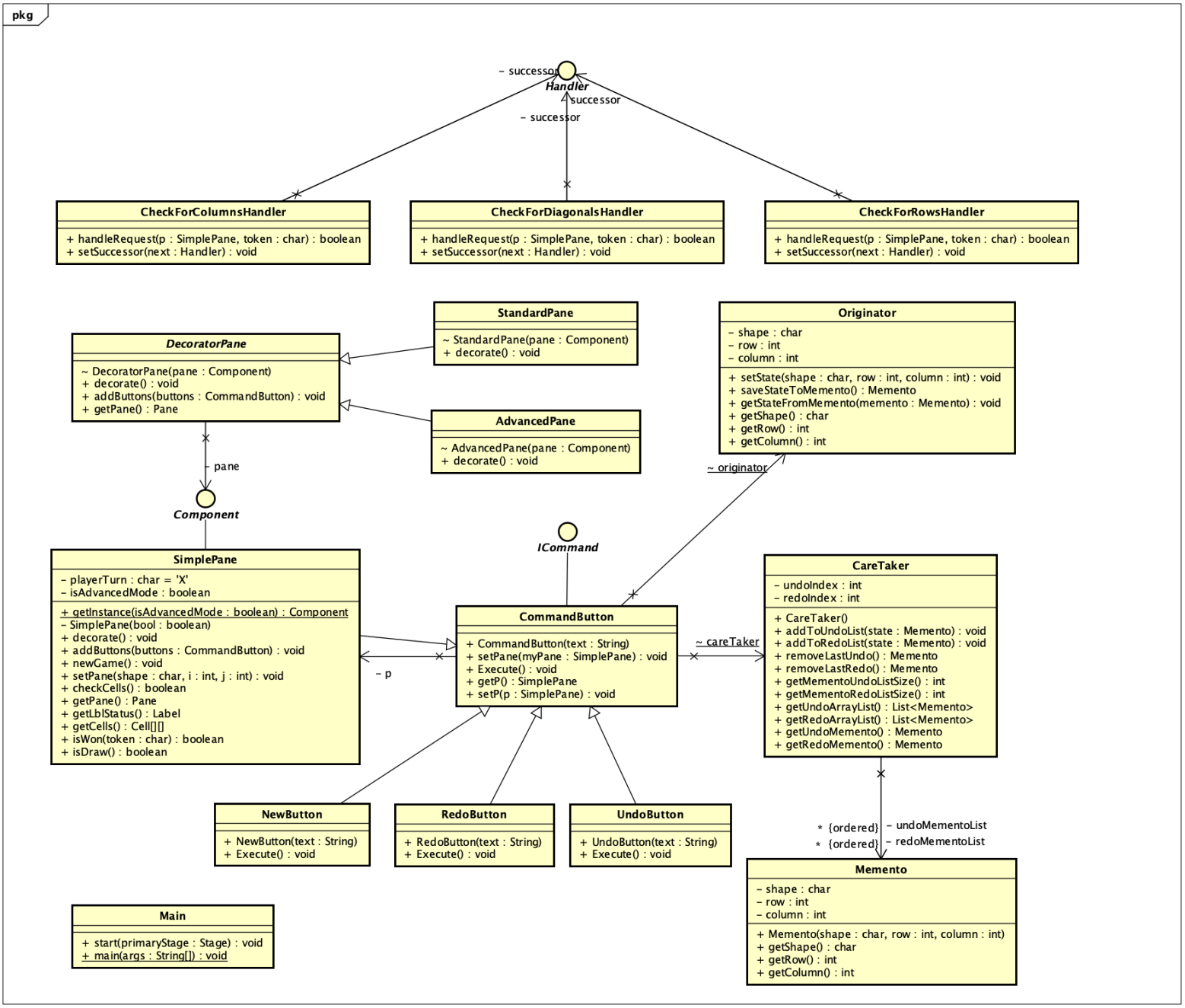
Architecture Overview:



In the Advanced Tic Tac Toe game development project, the architecture is designed around several core components crucial for the functionality and operation of the game. The Game component is central to managing the various states of the game and the turns of the players, ensuring smooth transitions and gameplay mechanics. The Player component represents the participants in the game, each associated with distinct markers and strategies. The Board handles the 2D grid where the game is played, maintaining the state of each cell and determining the placement of the playing pieces. The Playing Pieces themselves, including Piece X and Piece O, are crucial as they represent the markers used by each player to claim spaces on the board.

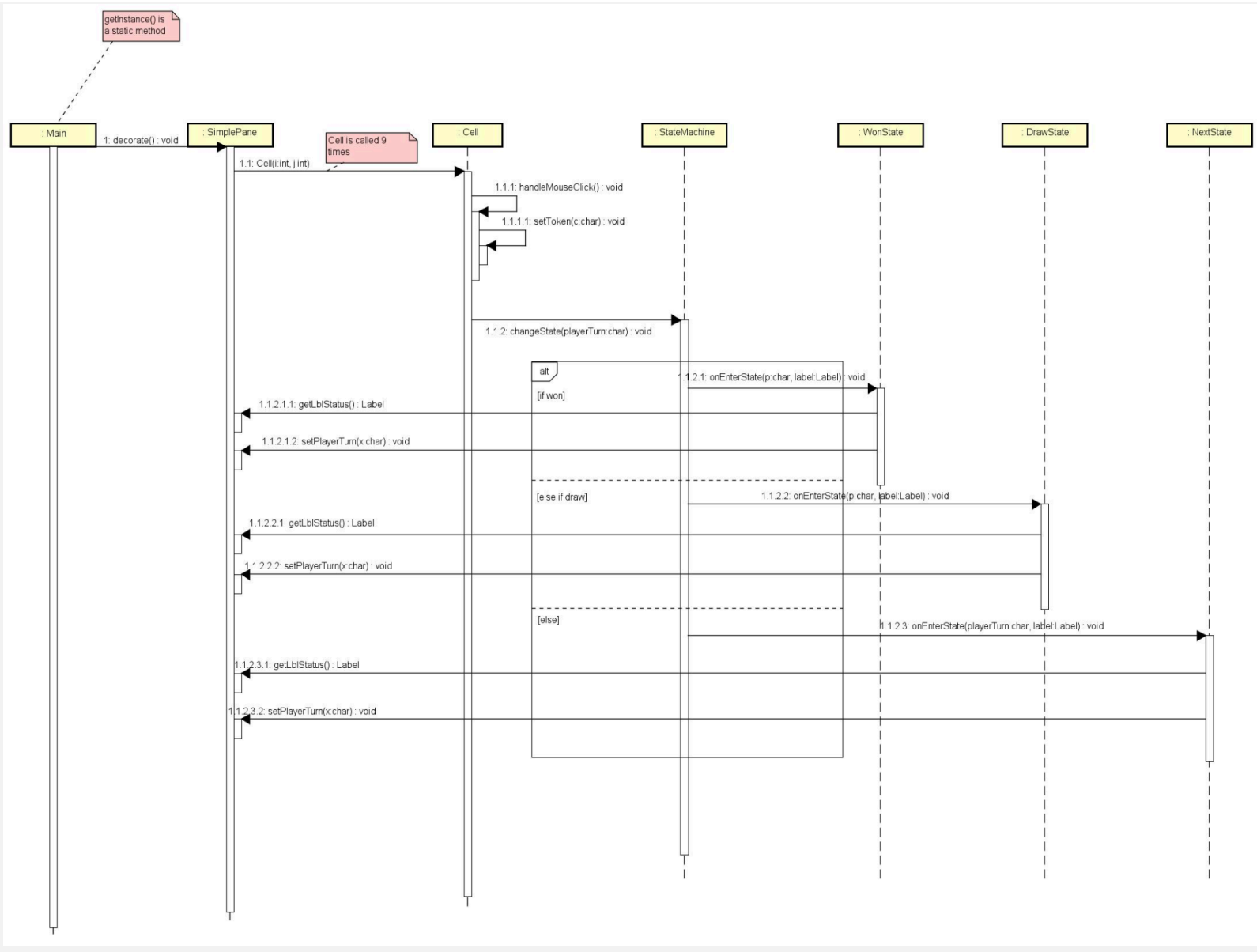
The technology stack for the project is robust, chosen to support the interactive and dynamic nature of the game. The Frontend utilizes JavaFX, a powerful toolkit that enhances the graphical user interface, making it responsive and visually appealing. This choice allows for intuitive interaction with the game elements and smooth animations. The Backend is built with Java, which handles the game logic and state management effectively. Java provides a strong foundation for implementing complex game mechanics and ensures that the game's backend architecture supports extensive gameplay features and real-time game state updates. This combination of technologies ensures that the game is both functional and engaging, providing an enriching user experience.

Class Diagram:

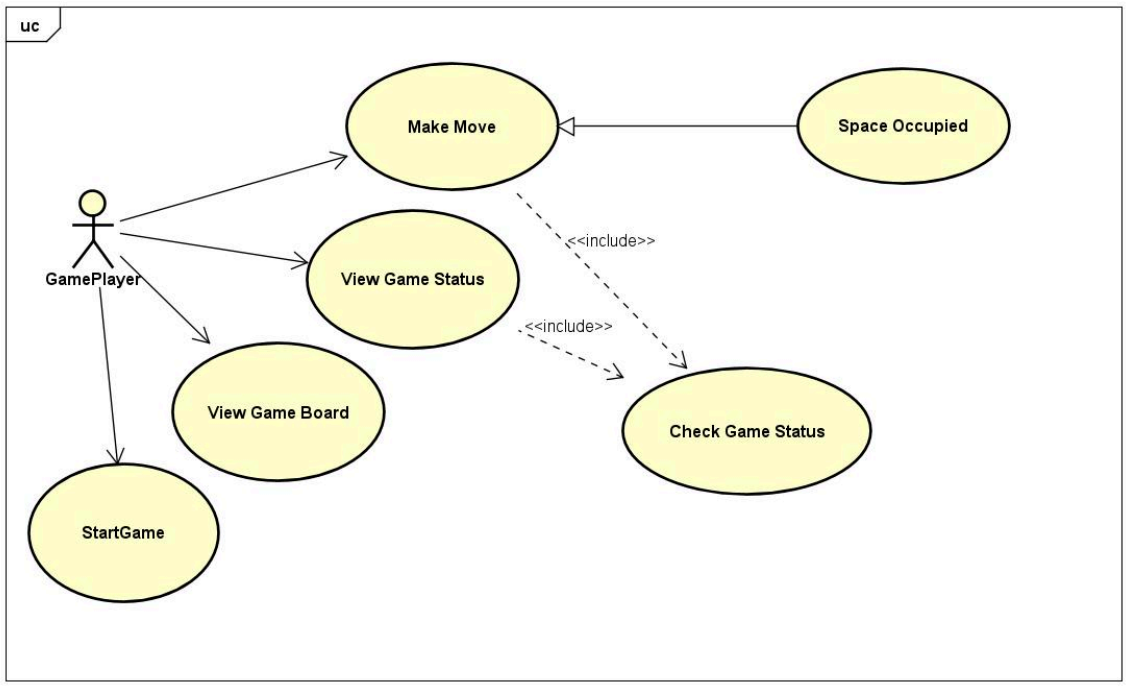


The class diagram illustrates a software application design utilizing multiple design patterns to enhance functionality and maintainability. It features the Chain of Responsibility pattern through handlers like CheckForColumnsHandler, CheckForRowsHandler, and CheckForDiagonalsHandler for processing different types of requests sequentially. The Decorator and Composite patterns are represented by Component and DecoratorPane, allowing dynamic addition of responsibilities to GUI elements. The Command pattern is implemented with ICommand and CommandButton to encapsulate actions as objects, facilitating operation handling. The Memento pattern, through Originator, Memento, and CareTaker, manages state for undo/redo capabilities. Central to the diagram, SimplePane links these functionalities, suggesting its role in GUI management and potentially game-specific logic. This design supports a flexible, extensible architecture, likely for a game or interactive application requiring complex interaction and state management with customizable GUI components.

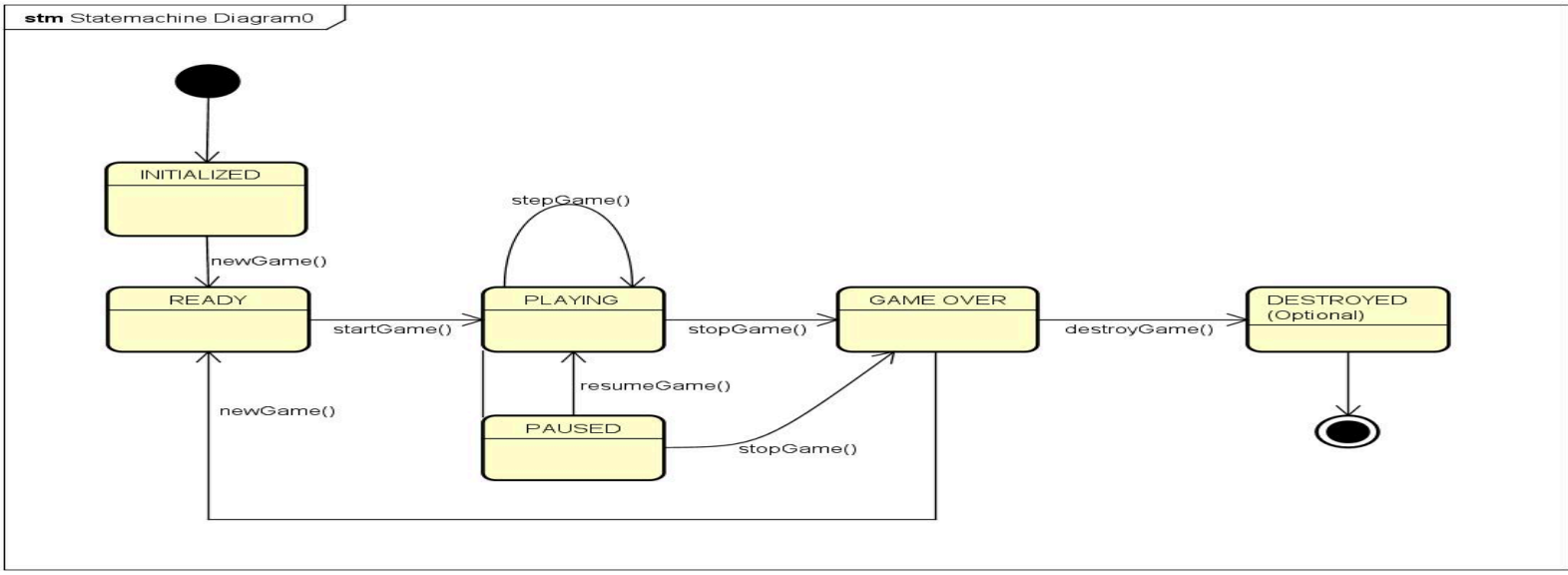
Sequence Diagrams:



UseCase Diagram:



State Diagram:



Design Patterns used in the project:

Strategy Pattern

A behavioral design pattern called strategy enables you to construct a family of algorithms, separate them into distinct classes, and make their objects interchangeable.

Decorator:

decorator is used to attach additional responsibilities to an object dynamically. In this project, we have used Decorator for displaying two types of game modes, Standard Mode and Advance Mode.
In Standard Mode, players can play an ordinary tic tac toe game with only a feature to restart the game. In Advance Mode, Players can undo, redo their moves along with the functionalities of standard mode.

Chain of responsibility

Chain of responsibility is used to avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. The receiving objects are chained and the request is passed amongst the objects until an object handles it.
In this project, we are using this pattern for Checking the winner of the game. So, we have three handlers, where one checks if any row has sequence of X's or O's, if not, then it passes the request to column handler which checks if any column has sequence of X's or O's, and if not, then it passes the request to diagonals handler which checks if the diagonal or anti-diagonal has sequence of X's or O's

Command Pattern

Many turn-based games include an "undo" button to let players reverse mistakes they make during play. This feature becomes especially relevant for mobile game development where the touch may have clumsy touch recognition. Rather than rely on a system where you ask the user "are you sure you want to do this task?" for every action they take, it is much more efficient to let them make mistakes and have the option of easily reverse the action.
So, we have implemented 3 buttons for this game.
New Game button
Undo Button
Redo Button
This Command Pattern comes into picture to run the appropriate logic when the player clicks on any of the above buttons to perform the respective actions.

Memento

Memento is used to capture and externalize an object's internal state so that the object can be restored to this state later.
In this project, we have used Memento along with Command pattern in order to undo-redo the moves of Tic-Tac-Toe.

We can undo and redo multiple times without losing the state of the object. We would be storing the object state everytime the move is made. This allows us to move back to the previous state as we have the details of the previous state.

State Pattern

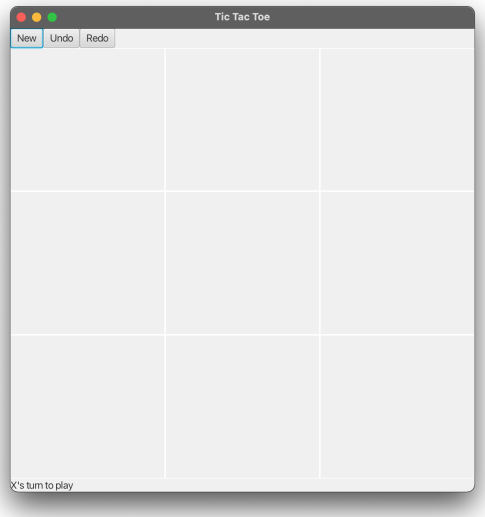
State allows an object to alter its behavior when its internal state changes. The object will appear to change its class.
The Tic-Tac-Toe game can have 3 states based on each move.
Draw
Win
Next Move - Continue
The states continue to work until the game ends. We check the current state and update the state of the machine by checking the game status on every move.

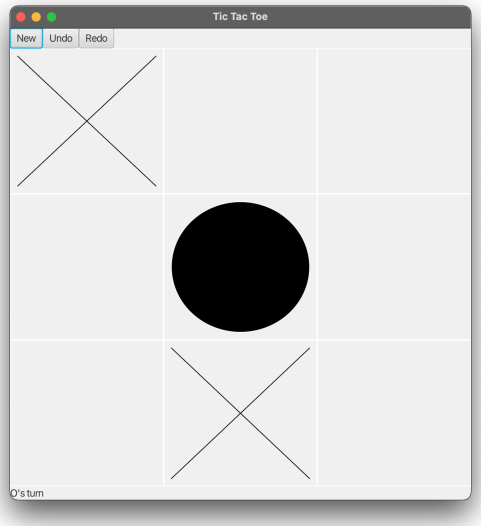
Singleton

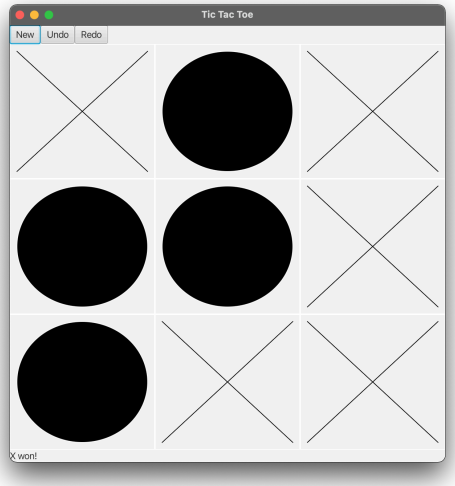
Ensure a class only has one instance, and provide a global point of access to it.
Singleton has a private constructor as we do not want the object to be created multiple times. The global access of the object is done using a static method which checks if the object creation is done and returns the current state of the object. We can only have one instance of the object.

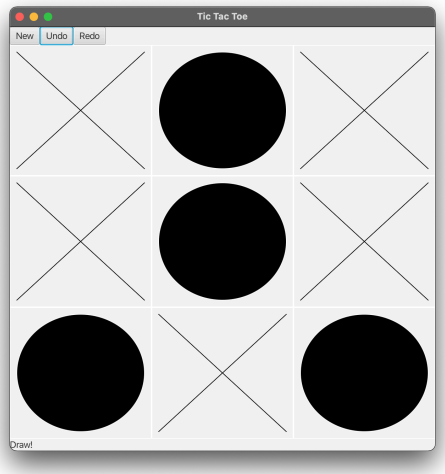
Game screenshots:

Advanced Mode:









References:

1. Bud, J. (2002). Applying Patterns to Develop Extensible and Maintainable Game Engines. IEEE Software.
2. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
3. Riehle, D. (1996). The Event Notification Pattern – Integrating Implicit Invocation with Object-Oriented Systems. Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA).
4. Nystrom, Robert. (2014). Game Programming Patterns.
5. Krasner, G. E., & Pope, S. T. (1988). A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80. Journal of Object-Oriented Programming.