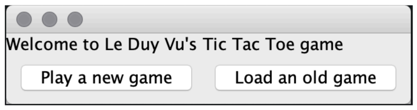
# ANALYSIS PHASE

## Functional Specification

Welcome to my Tic Tac Toe program. It allows a user to play a Tic Tac Toe game with a computer through a graphical user interface. Before playing a game, the user can choose the mark they want to play as (X or O) and the difficulty (easy or hard). In the easy mode, the computer always makes random moves. In the hard mode, it first checks if there is any next possible move that brings the victory to either side. Otherwise, if no one can win in the next turn, it makes a random move. The user makes a play by clicking on the cell on the Tic Tac Toe Board. Besides, the program saves finished games into the database, then allows you to load the old games and watch each move from both sides.

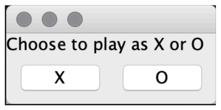
## User Manual with Graphical User Interface

*(1)* *Welcome Screen*


Welcome to Le Duy Vu's Tic Tac Toe game

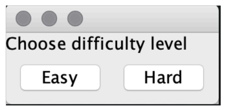
- Choose "Play a new game" to start playing —> go to *(2)*
- Choose "Load an old game" to view a past game —> go to *(6)*

*(2)* *Choose Your Mark*


Choose to play as X or O

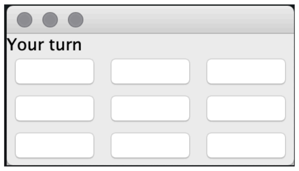
Choose to play as X or O —> go to *(3)*

*(3)* *Choose Difficulty*


Choose difficulty level

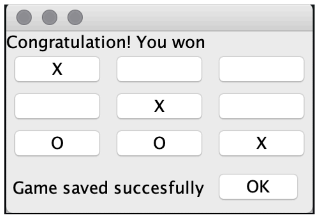
Choose to easy or hard mode to play against —> go to *(4)*

*(4)* *Game Screen*



Click on any cell you want to place your mark on. The computer will play it turn and the game continue until either someone wins or draw —> go to *(5)*

*(5)* *Declare Winner*

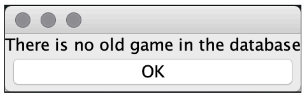

Congratulation! You won

Game saved succesfully

An example of post game screen. Here you play as X and win. The screen shows the winner and notifies whether the game has been successfully saved or not. Click OK to go back to the Welcome Screen —> go to *(1)*

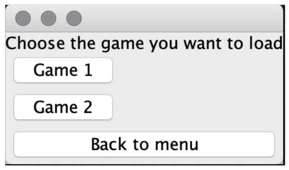
*(6)* *Load Game*

- If there is no game yet in the database


There is no old game in the database

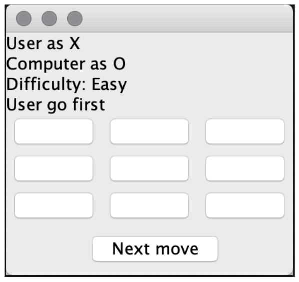
+ Click OK to go back to the Welcome Screen —> go to *(1)*

- If there is/are saved game(s):


Choose the game you want to load

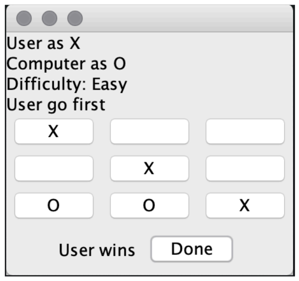
+ Choose the game to want to load —> go to *(7)*
+ Click Back to Menu to go back to the Welcome Screen —> go to *(1)*

*(7)* *Show Old Game*


User as X
Computer as O
Difficulty: Easy
User go first

Click Next move each time to display the next move the players made. Repeat until the end of the game —> go to *(8)*

*(8)* *Summary*


User as X
Computer as O
Difficulty: Easy
User go first

| | | |
|---|---|---|
| X | | |
| | X | |
| O | O | X |

User wins

When all moves have been shown, declare the winner. Click Done to go back to the Load Game screen —> go to *(6)*

# Use Cases

## Start A New Game

1. Click button "Play a new game".
2. Choose any mark as desired (X or O).
3. Choose any difficulty as desired (easy or hard)

## Play Game

1. See Start A New Game procedure. The system will decide randomly who will play first.
2. Click any cell you want to mark. After each play, the system will check if anyone wins yet.
3. The computer plays its turn.
4. Keep playing until there's a winner.

5. The system will declare the winner (or draw) and notify if the game has been saved successfully.
6. Click OK to go back to the Welcome Screen.

VARIATION 1:

1. During step 4, the user clicks on a cell that has been marked.
2. Nothing will happen. The system and interface remains the same until the user makes a valid move (click an unmarked cell).

VARIATION 2:

1. In step 5, an unexpected error happens that the system can't save the game into a file.
2. Instead of showing "Game saved succesfully", the system shows "Unexpected error. Game can't be saved".
3. Continue with step 6.

VARIATION 3:

1. Before step 5, the users closes the window before finishing the game.
2. Any work in progress will be discarded and nothing will be saved.

## Load An Old Game

1. Click button "Load an old game".
2. Choose a game you want to load by clicking its respective button or click button "Back to menu" to go back to the Welcome Screen.
3. The screen shows properties of the chosen game, an empty board, and "Next move" button.
4. Click button "Next move" to show the next move made turn by turn.
5. When there is no move left, the system declares the winner.
6. Click button "Done" to go back to the Load Game screen.

VARIATION 1:

1. After step 1, if there is no saved game in the database, the screen will notify.
2. Click button "OK" to go back to the Welcome Screen.

VARIATION 2:

1. In step 2, an unexpected error happens that the system can't open the file containing the data of the chosen game.
2. The button representing that game will show "Error: Can't open file".
3. Repeat step 2 with a different game, or go back to the menu.

# DESIGN PHASE

## Identify Classes

- Player (Interface)
- User
- AbstractAI (Abstract)
- EasyAI
- HardAI
- Board
- Log
- ControlSystem

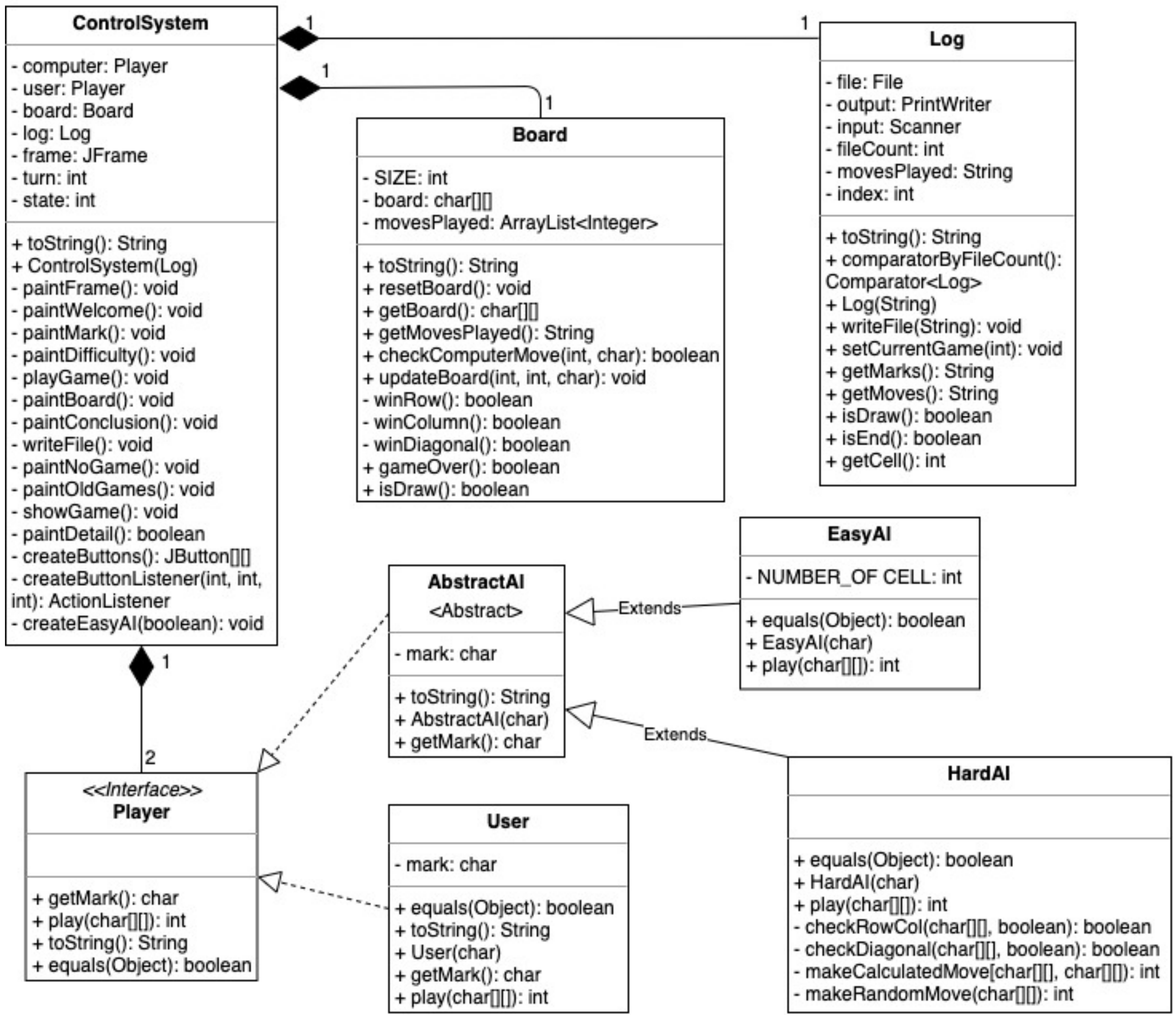
## Class Responsibility

- **Player** contains the common methods of User and AI such as getMark() and play().
- **User** contains the unique mark that they choose.
- **AbstractAI** contains the leftover mark based on the choice of **User**.
- **EasyAI** and **HardAI** contains different version of method play(), making random and calculated moves, respectively.
- **Board** saves the moves made by the players, checks the validity of the moves, and determines after every move if the game is over.
- **Log** has the responsibility of writing data into, loading data from files, and processing said data.
- **ControlSystem** designs the GUI for the program, controls the flow of the gameplay, and makes decision based on the choice of the players.

## Class Relationship

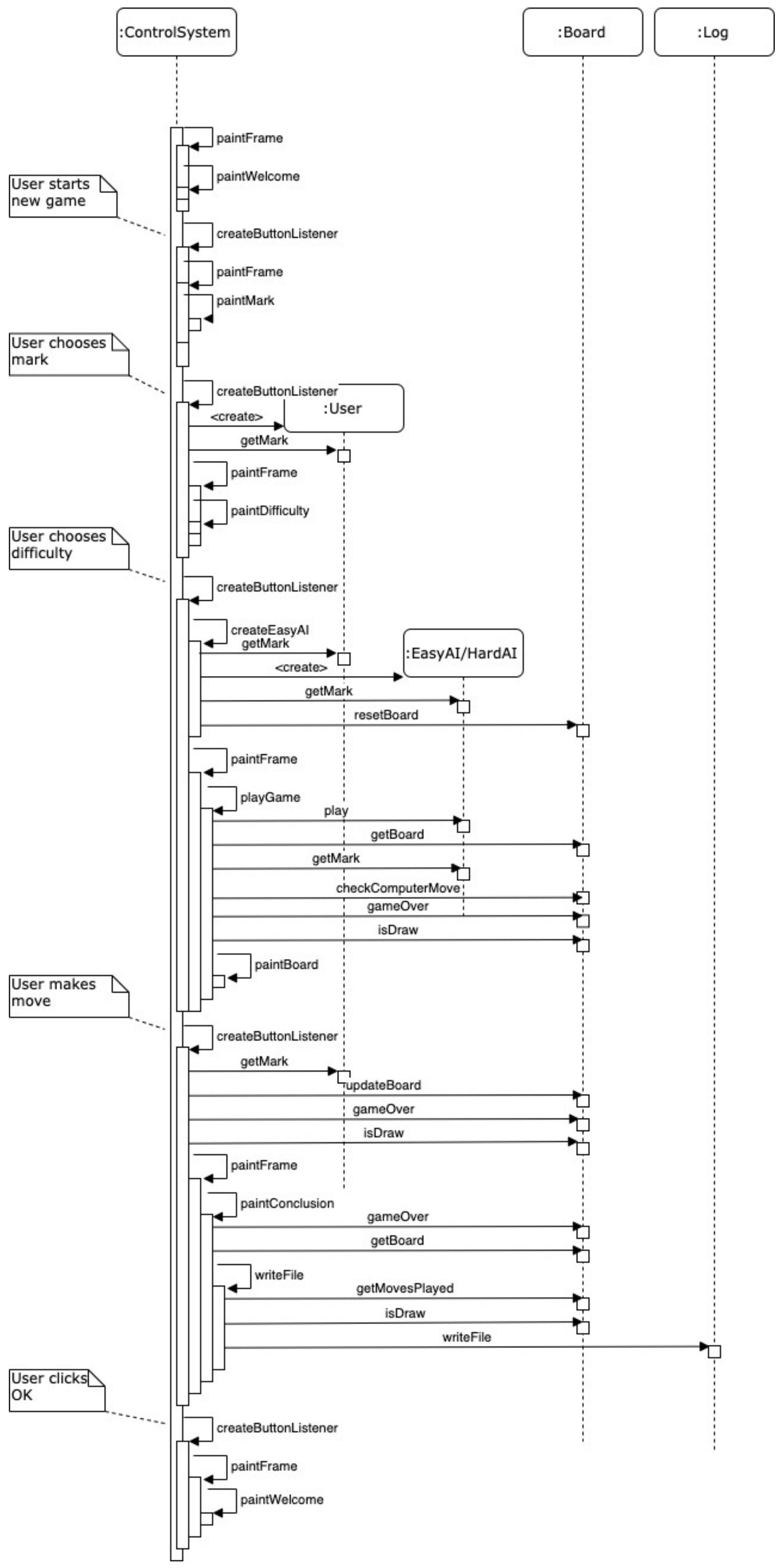
- **ControlSystem** "has" 1 **Board** (composition)
- **ControlSystem** "has" 1 **Log** (composition)
- **ControlSystem** "has" 2 **Player**s (composition)
- A **User** "is" a **Player** (inheritance)
- An **AbstractAI** "is" a **Player** (inheritance)
- An **EasyAI** "is" an **AbstractAI** (inheritance)
- A **HardAI** "is" an **AbstractAI** (inheritance)

## UML Diagram

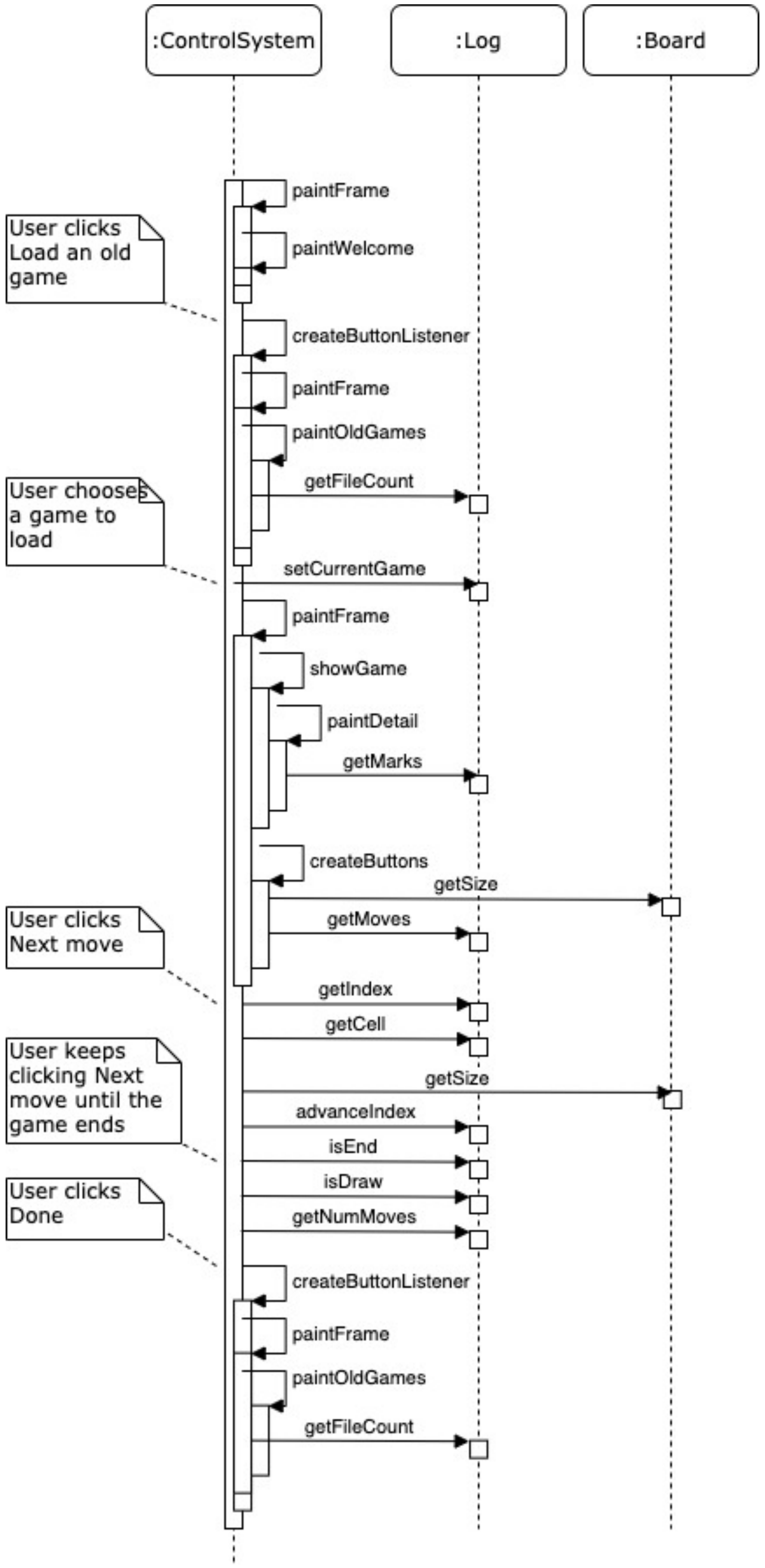

ControlSystem
- computer: Player
- user: Player
- board: Board
- log: Log
- frame: JFrame
- turn: int
- state: int
+ toString(): String
+ ControlSystem(Log)
- paintFrame(): void
- paintWelcome(): void
- paintMark(): void
- paintDifficulty(): void
- playGame(): void
- paintBoard(): void
- paintConclusion(): void
- writeFile(): void
- paintNoGame(): void
- paintOldGames(): void
- showGame(): void
- paintDetail(): boolean
- createButtons(): JButton[][]
- createButtonListener(int, int, int): ActionListener
- createEasyAI(boolean): void
1
1
1
1
Board
- SIZE: int
- board: char[][]
- movesPlayed: ArrayList<Integer>
+ toString(): String
+ resetBoard(): void
+ getBoard(): char[][]
+ getMovesPlayed(): String
+ checkComputerMove(int, char): boolean
+ updateBoard(int, int, char): void
- winRow(): boolean
- winColumn(): boolean
- winDiagonal(): boolean
+ gameOver(): boolean
+ isDraw(): boolean
Log
- file: File
- output: PrintWriter
- input: Scanner
- fileCount: int
- movesPlayed: String
- index: int
+ toString(): String
+ comparatorByFileCount(): Comparator<Log>
+ Log(String)
+ writeFile(String): void
+ setCurrentGame(int): void
+ getMarks(): String
+ getMoves(): String
+ isDraw(): boolean
+ isEnd(): boolean
+ getCell(): int
EasyAI
- NUMBER_OF CELL: int
+ equals(Object): boolean
+ EasyAI(char)
+ play(char[][]): int
Extends
AbstractAI
<Abstract>
- mark: char
+ toString(): String
+ AbstractAI(char)
+ getMark(): char
Extends
1
2
<<Interface>>
Player
+ getMark(): char
+ play(char[][]): int
+ toString(): String
+ equals(Object): boolean
User
- mark: char
+ equals(Object): boolean
+ toString(): String
+ User(char)
+ getMark(): char
+ play(char[][]): int
HardAI
+ equals(Object): boolean
+ HardAI(char)
+ play(char[][]): int
- checkRowCol(char[][], boolean): boolean
- checkDiagonal(char[][], boolean): boolean
- makeCalculatedMove[char[][], char[][]): int
- makeRandomMove(char[][]): int

## Sequence Diagram

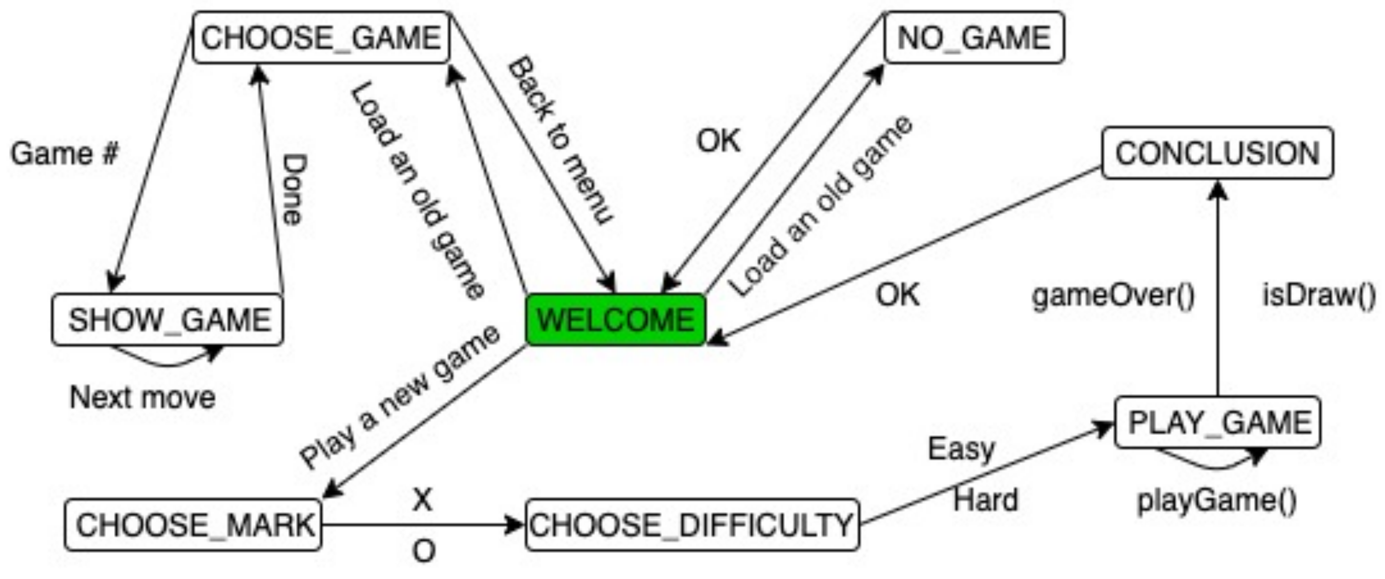
### Play Game



## Load An Old Game



## State Diagram


CHOOSE_GAME
NO_GAME
CONCLUSION
SHOW_GAME
WELCOME
PLAY_GAME
CHOOSE_MARK
CHOOSE_DIFFICULTY
Game #
Done
Load an old game
Back to menu
OK
Load an old game
OK
gameOver()
isDraw()
Next move
Play a new game
Easy
Hard
playGame()
X
O