AlphaXO Documentation

Created by

Jetnipat Lapsuwannawong 6432026221

2110215 Programming Methodology
Semester 2 Year 2021
Chulalongkorn University

AlphaXO

Introduction

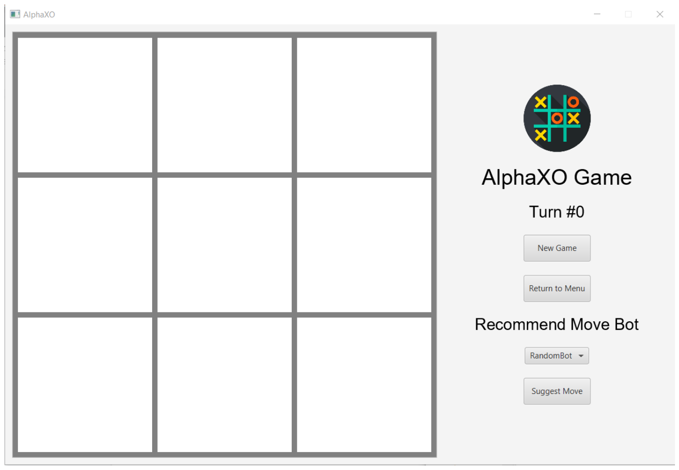
AlphaXO is a tic-tac-toe game. Each player takes turns to place a mark on the board. First player to have a number of connected marks will be a winner. This game can be drawn. Players can also play against bot to practice their skill, with AI move recommendations to help understanding the game.

Rule

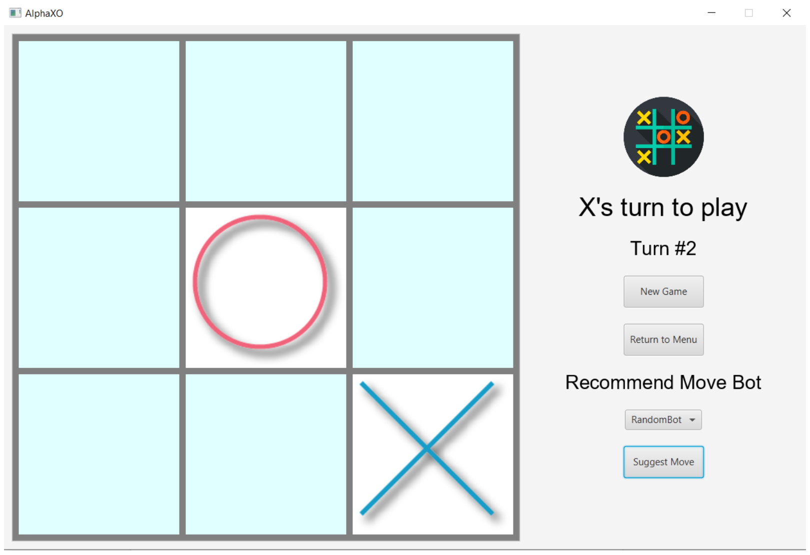
- This game require 2 players
- Each player take turn to put the mark on the N X N board
- In a classic mode, first player to have a N connected marks will be a winner
- In a speed mode, first player to have 5 connected marks will be a winner
- If a player cannot place a mark on the board anymore, and still doesn't have a winner. The game will be drawn.

Example : Classic Mode

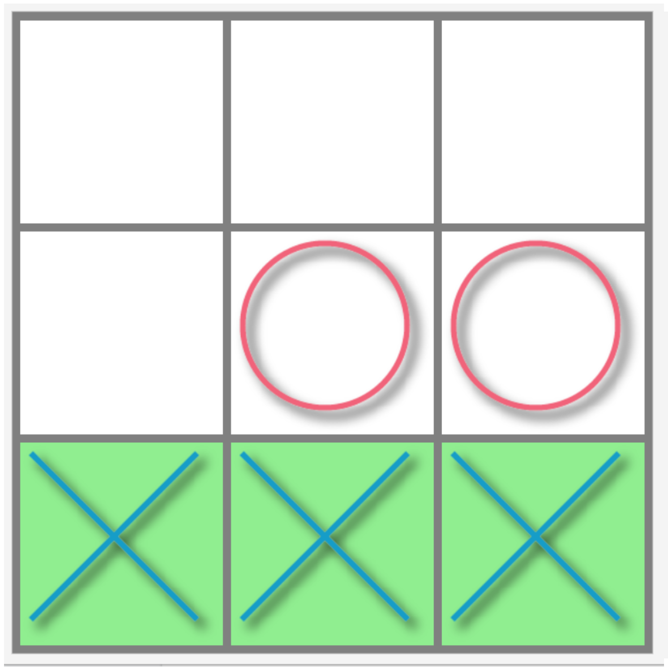
- This is an example of 3 by 3 board, with 9 square

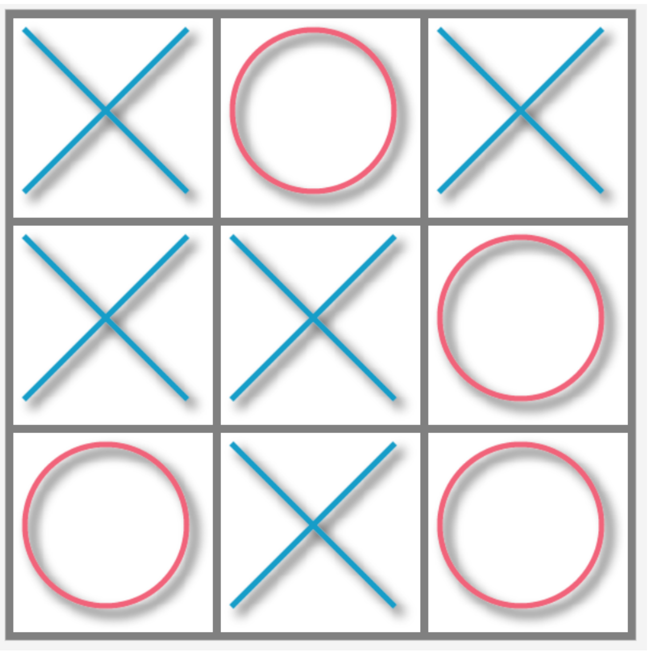


- Each player takes turns placing their symbol, the player who goes first will have the X symbol and another player will have the O symbol. The blue area is a valid move that player can play (Can only play on empty spaces)



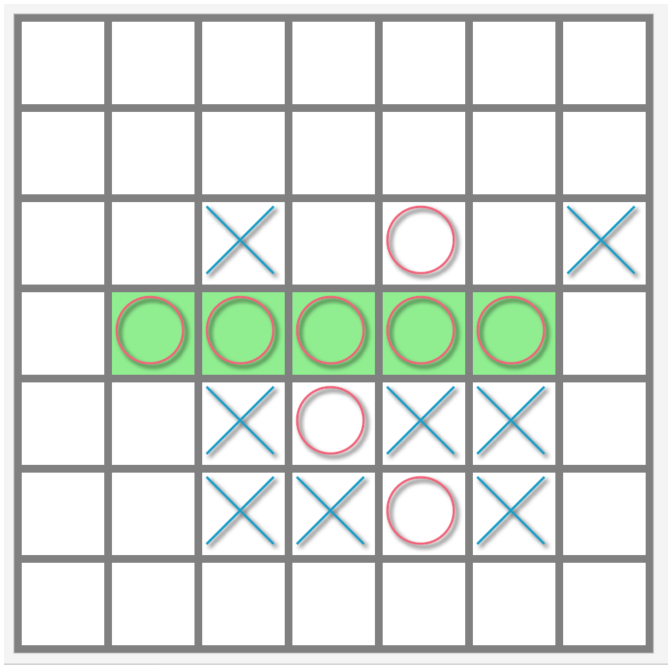
- The game ended with one of the players getting N connection marks (in this case N = 3). If a player cannot place a new mark anymore (Board is full) and still doesn't have a winner the game will be drawn.





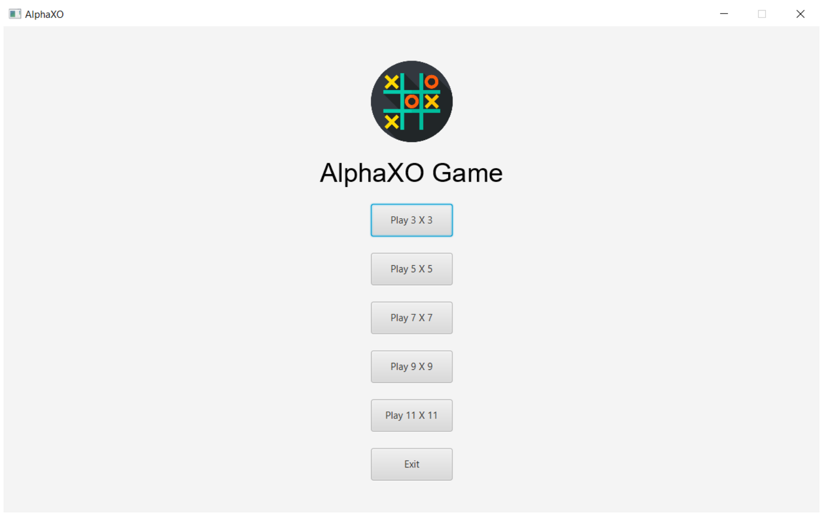
Example: Speed Mode

- In speed mode, most rules of classic mode still apply. The only difference is the win conditions. In this mode you only need to get 5 connected marks in order to win. Don't rely on board size (Even 100 X 100 board, You still only need 5 to win)

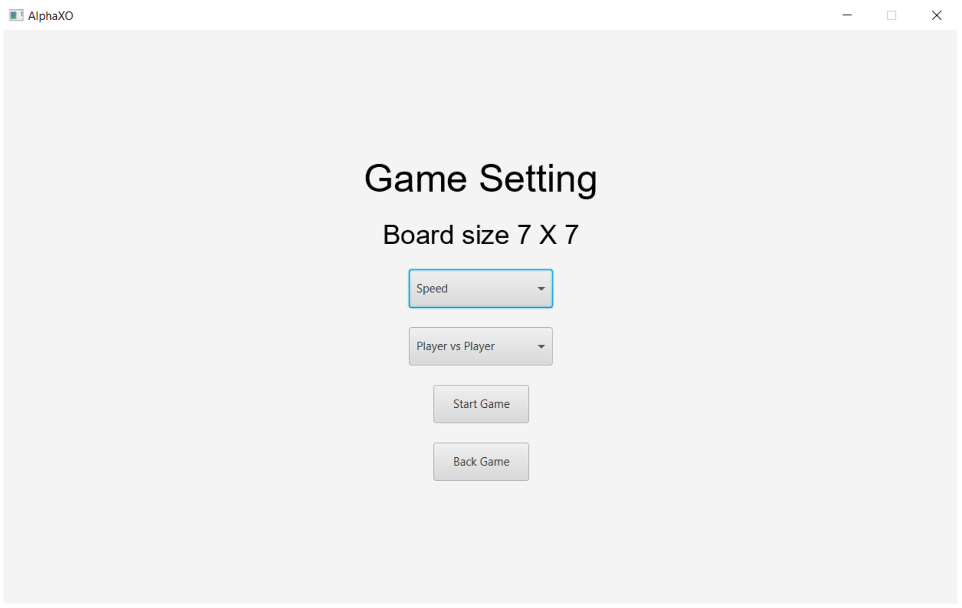


User Interface

- Main menu, You can exit the game or select the board size you want to play



- Game setting, when you finish selecting board size. It will lead you to this page to set your game. First option is game mode. You can play classic or speed depending on board size. Second option is play type. You can play with 2 players or play with AI

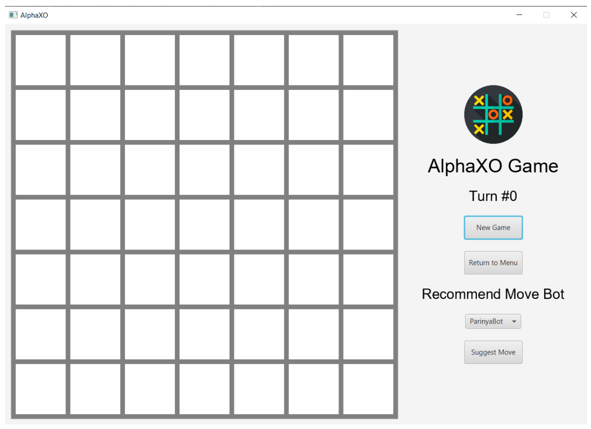


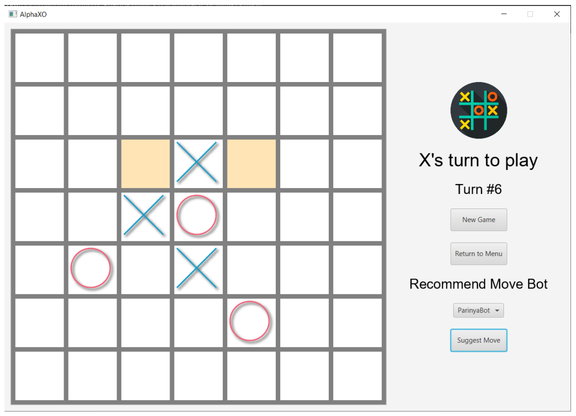
Board size 7 X 7

Speed

Player vs Player

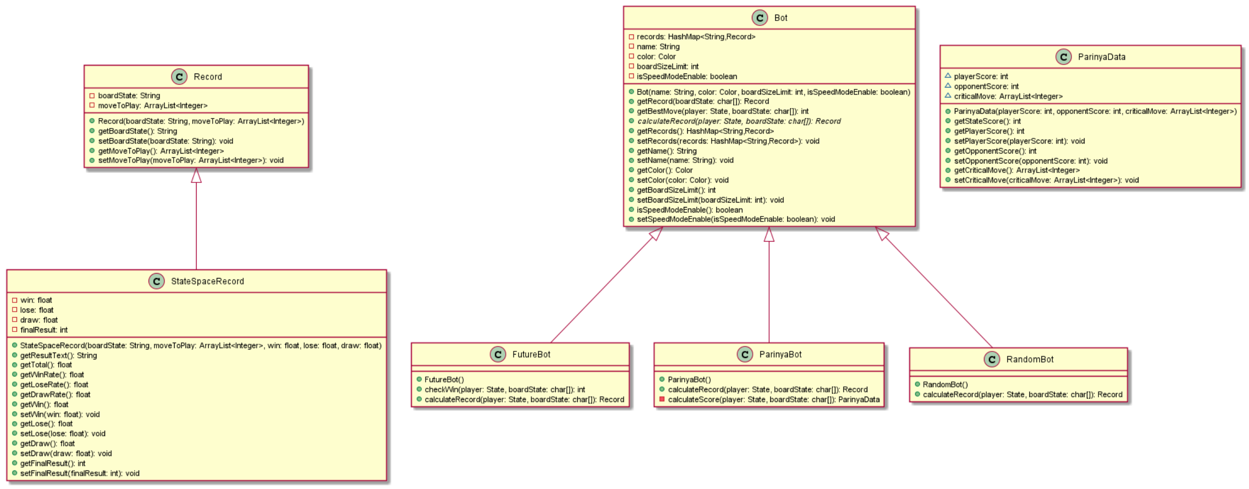
- Main game page, this page contains most of the gameplay. You can click the board to place your mark. Choose the AI and click suggest move to get a move recommendation. If you play with AI. It will respond to your move automatically. You can go back to the menu or new game at any time.





Class diagram

1.Package bot


Record
boardState: String
moveToPlay: ArrayList<Integer>
Record(boardState: String, moveToPlay: ArrayList<Integer>)
getBoardState(): String
setBoardState(boardState: String): void
getMoveToPlay(): ArrayList<Integer>
setMoveToPlay(moveToPlay: ArrayList<Integer>): void
StateSpaceRecord
win: float
lose: float
draw: float
finalResult: int
StateSpaceRecord(boardState: String, moveToPlay: ArrayList<Integer>, win: float, lose: float, draw: float)
getResultText(): String
getTotal(): float
getWinRate(): float
getLoseRate(): float
getDrawRate(): float
getWin(): float
setWin(win: float): void
getLose(): float
setLose(lose: float): void
getDraw(): float
setDraw(draw: float): void
getFinalResult(): int
setFinalResult(finalResult: int): void
Bot
records: HashMap<String,Record>
name: String
color: Color
boardSizeLimit: int
isSpeedModeEnable: boolean
Bot(name: String, color: Color, boardSizeLimit: int, isSpeedModeEnable: boolean)
getRecord(boardState: char[]): Record
getBestMove(player: State, boardState: char[]): int
calculateRecord(player: State, boardState: char[]): Record
getRecords(): HashMap<String,Record>
setRecords(records: HashMap<String,Record>): void
getName(): String
setName(name: String): void
getColor(): Color
setColor(color: Color): void
getBoardSizeLimit(): int
setBoardSizeLimit(boardSizeLimit: int): void
isSpeedModeEnable(): boolean
setSpeedModeEnable(isSpeedModeEnable: boolean): void
ParinyaData
playerScore: int
opponentScore: int
criticalMove: ArrayList<Integer>
ParinyaData(playerScore: int, opponentScore: int, criticalMove: ArrayList<Integer>)
getStateScore(): int
getPlayerScore(): int
setPlayerScore(playerScore: int): void
getOpponentScore(): int
setOpponentScore(opponentScore: int): void
getCriticalMove(): ArrayList<Integer>
setCriticalMove(criticalMove: ArrayList<Integer>): void
FutureBot
FutureBot()
checkWin(player: State, boardState: char[]): int
calculateRecord(player: State, boardState: char[]): Record
ParinyaBot
ParinyaBot()
calculateRecord(player: State, boardState: char[]): Record
calculateScore(player: State, boardState: char[]): ParinyaData
RandomBot
RandomBot()
calculateRecord(player: State, boardState: char[]): Record

1.1 Abstract Bot

1.1.1 Fields

- HashMap<String, Record>	Storing calculation result for dynamic programming
- String name	Name of bot
- Color color	Color of bot, use for move recommendation ui
- int boardSizeLimit	Max board size that this bot can use on
- boolean isSpeedModeEnable	Is this bot supported speed mode

1.1.2 Constructor

+ Bot (String name, Color color, int boardSizeLimit, boolean isSpeedModeEnable)	Store all parameter in the class

1.1.3 Methods

+ Record getRecord(char[] boardState)	Get record if already calculated, if not calculate and then return after finish
+ int getBestMove(State player, char[] boardState)	Random a move from list of available moves
+ abstract Record calculateRecord(State player, char[] boardState)	Abstract methods for calculating available move from a board positions
+ getter/setter	

1.2 FutureBot extends Bot

1.2.1 Constructor

+ FutureBot()	Use super constructor - Name = “ FutureBot” - Color = LIGHT PINK - Max board size = 3 - isSpeedModeEnable = false

1.2.2 Methods

+ int checkWin(State player, char[] boardState)	Check is any player win from positions - Return 1 if player win - Return -1 if opponent win - Return 0 if draw
+ Record calculateRecord(State player, char[] boardState)	A recursive function to calculate best move from a game state, using minimax + state space search

1.3 ParinyaBot extends Bot

1.3.1 Constructor

+ ParinyaBot()	Use super constructor - Name = “ ParinyaBot” - Color = MOCCASIN - Max board size = 13 - isSpeedModeEnable = true

1.3.2 Methods

+ int checkWin(State player, char[] boardState)	Check is any player win from positions - Return 1 if player win - Return -1 if opponent win - Return 0 if draw
+ Record calculateRecord(State player, char[] boardState)	Calculate best move by evaluating board state with a

	math formula
- ParinyaData calculateScore(State player, char[] boardState)	Calculate score for each move using math formula and some brute force cases

1.4 RandomBot extends Bot

1.4.1 Constructor

+ RandomBot()	Use super constructor - Name = “ RandomBot” - Color = LIGHTCYAN - Max board size = 999 - isSpeedModeEnable = true

1.4.2 Methods

+ Record calculateRecord(State player, char[] boardState)	Use all valid moves as a best move

1.5 ParinyaData

1.5.1 Fields

- int playerScore	Storing calculation result for dynamic programming
- int opponentScore	Name of bot
- ArrayList<Integer> criticalMove	Color of bot, use for move

	recommendation ui

1.5.2 Constructor

+ ParinyaData(int playerScore, int opponentScore, ArrayList<Integer> criticalMove)	Store all parameter in the class

1.5.3 Methods

+ int getStateScore()	return 1 + playerScore - opponentScore
+ getter/setter	

1.6 Record

1.6.1 Fields

- String boardState	Board State of this record
- ArrayList<Integer> moveToPlay	Best move to play on this state

1.6.2 Constructor

+ Record(String boardState, ArrayList<Integer> moveToPlay)	Store all parameter in the class

1.6.3 Methods

+ getter/setter	

1.7 StateSpaceRecord extends Record

1.7.1 Fields

- float win	Store win count in state space
- float lose	Store losecount in state space
- float draw	Store draw count in state space
- int finalResult	Store final result of state space

1.7.2 Constructor

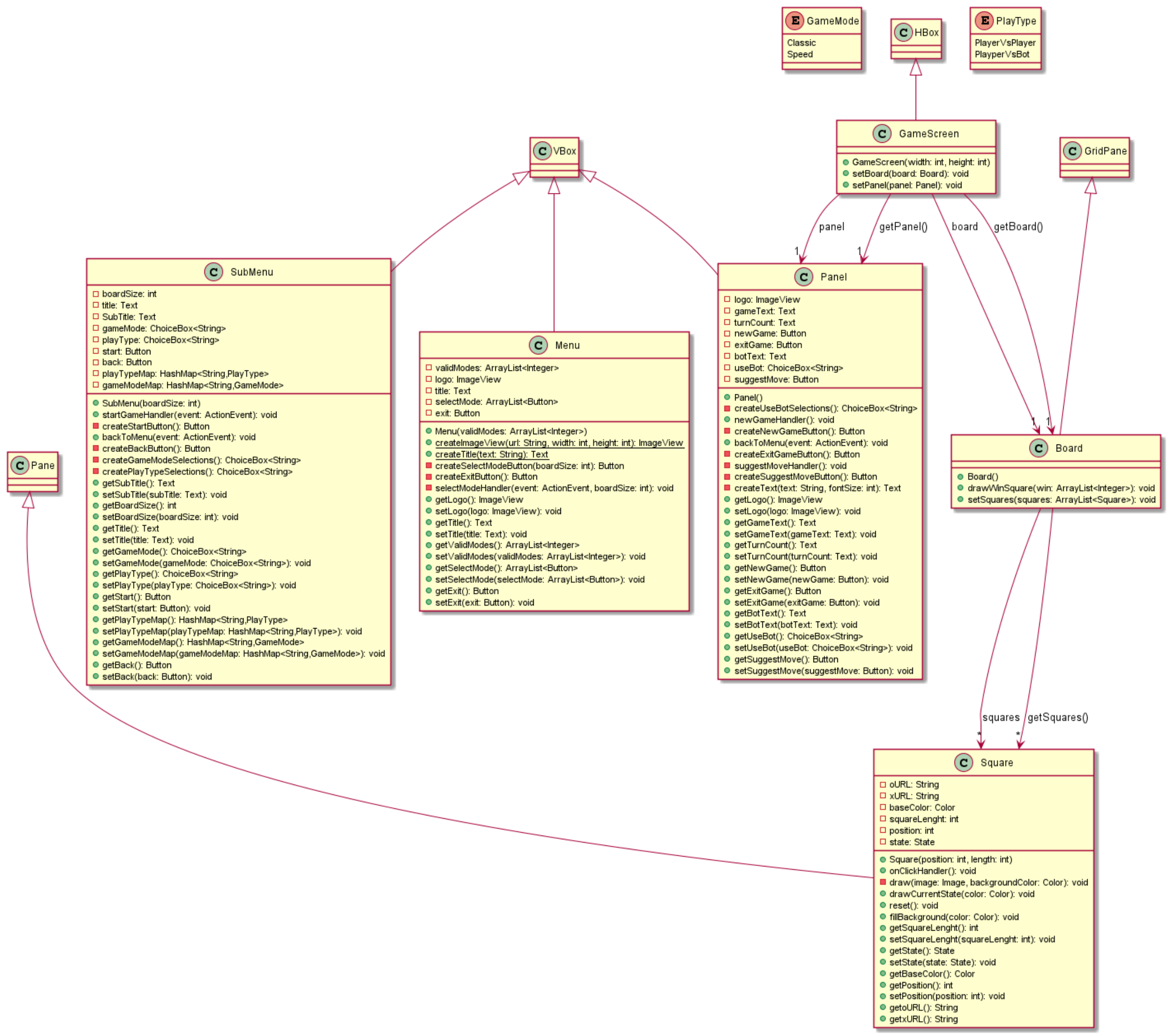
+ Record(String boardState, ArrayList<Integer> moveToPlay)	- Super Constructor and - Store all parameter in the class

1.7.3 Methods

+ String getResultText()	Return a win, lose, draw as a text for bot display
+ float getTotal()	Win + lose + draw
+ float getWinRate()	Win / total

+ float getLoseRate()	Lose / total
+ float getDrawRate()	Draw / total
+ getter/setter	

2.Package gui


GameMode
Classic
Speed
HBox
PlayType
PlayerVsPlayer
PlayperVsBot
VBox
GameScreen
GameScreen(width: int, height: int)
setBoard(board: Board): void
setPanel(panel: Panel): void
GridPane
panel
getPanel()
board
getBoard()
1
1
1
1
SubMenu
boardSize: int
title: Text
SubTitle: Text
gameMode: ChoiceBox<String>
playType: ChoiceBox<String>
start: Button
back: Button
playTypeMap: HashMap<String,PlayType>
gameModeMap: HashMap<String,GameMode>
SubMenu(boardSize: int)
startGameHandler(event: ActionEvent): void
createStartButton(): Button
backToMenu(event: ActionEvent): void
createBackButton(): Button
createGameModeSelections(): ChoiceBox<String>
createPlayTypeSelections(): ChoiceBox<String>
getSubTitle(): Text
setSubTitle(subTitle: Text): void
getBoardSize(): int
setBoardSize(boardSize: int): void
getTitle(): Text
setTitle(title: Text): void
getGameMode(): ChoiceBox<String>
setGameMode(gameMode: ChoiceBox<String>): void
getPlayType(): ChoiceBox<String>
setPlayType(playType: ChoiceBox<String>): void
getStart(): Button
setStart(start: Button): void
getPlayTypeMap(): HashMap<String,PlayType>
setPlayTypeMap(playTypeMap: HashMap<String,PlayType>): void
getGameModeMap(): HashMap<String,GameMode>
setGameModeMap(gameModeMap: HashMap<String,GameMode>): void
getBack(): Button
setBack(back: Button): void
Menu
validModes: ArrayList<Integer>
logo: ImageView
title: Text
selectMode: ArrayList<Button>
exit: Button
Menu(validModes: ArrayList<Integer>)
createImageView(url: String, width: int, height: int): ImageView
createTitle(text: String): Text
createSelectModeButton(boardSize: int): Button
createExitButton(): Button
selectModeHandler(event: ActionEvent, boardSize: int): void
getLogo(): ImageView
setLogo(logo: ImageView): void
getTitle(): Text
setTitle(title: Text): void
getValidModes(): ArrayList<Integer>
setValidModes(validModes: ArrayList<Integer>): void
getSelectMode(): ArrayList<Button>
setSelectMode(selectMode: ArrayList<Button>): void
getExit(): Button
setExit(exit: Button): void
Panel
logo: ImageView
gameText: Text
turnCount: Text
newGame: Button
exitGame: Button
botText: Text
useBot: ChoiceBox<String>
suggestMove: Button
Panel()
createUseBotSelections(): ChoiceBox<String>
newGameHandler(): void
createNewGameButton(): Button
backToMenu(event: ActionEvent): void
createExitGameButton(): Button
suggestMoveHandler(): void
createSuggestMoveButton(): Button
createText(text: String, fontSize: int): Text
getLogo(): ImageView
setLogo(logo: ImageView): void
getGameText(): Text
setGameText(gameText: Text): void
getTurnCount(): Text
setTurnCount(turnCount: Text): void
getNewGame(): Button
setNewGame(newGame: Button): void
getExitGame(): Button
setExitGame(exitGame: Button): void
getBotText(): Text
setBotText(botText: Text): void
getUseBot(): ChoiceBox<String>
setUseBot(useBot: ChoiceBox<String>): void
getSuggestMove(): Button
setSuggestMove(suggestMove: Button): void
Board
Board()
drawWinSquare(win: ArrayList<Integer>): void
setSquares(squares: ArrayList<Square>): void
Pane
squares
getSquares()
*
*
Square
oURL: String
xURL: String
baseColor: Color
squareLenght: int
position: int
state: State
Square(position: int, length: int)
onClickHandler(): void
draw(image: Image, backgroundColor: Color): void
drawCurrentState(color: Color): void
reset(): void
fillBackground(color: Color): void
getSquareLenght(): int
setSquareLenght(squareLenght: int): void
getState(): State
setState(state: State): void
getBaseColor(): Color
getPosition(): int
setPosition(position: int): void
getoURL(): String
getxURL(): String

2.1 Board extends GridPane

2.1.1 Fields

- ArrayList<Square> squares	List of square on a board

2.1.2 Constructor

+ Board()	- Do the styling - Create Square as board size into squares fields

2.1.3 Methods

+ drawWinSquare	Draw a green background to a winning square when game ended
+ getter/setter	

2.2 enum GameMode

2.2.1 Constructor

+ enum GameMode	- Classic - Speed

2.3 GameScreen extends HBox

2.3.1 Fields

- Board board	Store a Board

- Panel panel	Store a Panel

2.3.2 Constructor

+ GameScreen(int width, int height)	- Do the styling - Create new board - Create new Panel

2.3.3 Methods

+ getter/setter	

2.4 GuiTool

2.4.1 Methods

+ static ImageView createImageView(String url, int width, int height)	Create an ImageView from parameter
+ static Text createTitle(String text)	Create Text from String

2.5 Menu extends VBox

2.5.1 Fields

- ArrayList<Integer> validModes	Storing valid board size (mode)
- ImageView logo	Game logo

- Text title	Game title
- ArrayList<Button> selectMode	Button to select board size (mode)
- Button exit	Exit program

2.5.2 Constructor

+ Menu(ArrayList<Integer> validModes)	- Do the styling - Store all parameter - Create selectMode button from a validModes

2.5.3 Methods

- Button createSelectModeButton(int boardSize)	- Create a button for board size selection - setOnAction to selectModeHandler
- Button createExitButton()	- Create Exit button - When click exit program
- selectModeHandler(ActionEvent event, int boardSize)	- Open sub menu page - Passing board size to sub menu to do game setting
+ getter/setter	

2.6 Panel extends VBox

2.6.1 Fields

- ImageView logo	Game Logo
- Text gameText	Show game status
- Text turnCount	Show turn count
- Button newGame	Run newGame in Game to reset everything
- Button exitGame	Return to menu
- Text botText	Show text from AI
- ChoiceBox<String> useBot	Show current using AI and can change AI
- Button suggestMove	Suggest move using current AI in useBot ChoiceBox

2.6.2 Constructor

+ Panel()	- Do the styling - Set all parameter with default value

2.6.3 Methods

- ChoiceBox<String> createUseBotSelections()	- Create ChoiceBox for useBot selection - Get list of bot from Game.instance()
+ void newGameHandler()	Click to reset a game and start a new game
- Button	- Create a new game

createNewGameButton()	button - setOnAction to newGameHandler
+ backToMenu(ActionEvent event)	Go back to menu page
- Button createExitGameButton()	- Button to return to menu - setOnAction to backToMenu
- void suggestMoveHandler()	- Get currently use AI - Get the record of current game state - Show recommend moves
- Button createSuggestMoveButton()	Click to make AI suggesting moves
- Text createText(String text, int fontSize)	- Create Text from String and font size
+ getter/setter	

2.7 enum PlayType

2.7.1 Constructor

+ enum PlayType	- PlayerVsPlayer - PlayperVsBot

2.8 Square extends Pane

2.8.1 Fields

- final String oURL	Set to “o.png”
- final String xURL	Set to “x.png”
- final Color baseColor	Set to Color.WHITE
- int squareLength	Height and width of square
- int position	Position on the board 0 - N^2 depends on board size
- State state	Current state of the square

2.8.2 Constructor

+ Square(int position, int length)	- Do the styling - Store all parameter - Set Event handler mouse click to onClickHandler

2.8.3 Methods

+ void onClickHandler()	- If isGameEnd or State != Empty return - Set state to currentTurn - Update game state in Game
- void draw(Image image, Color backgroundColor)	Draw a square with image and background in the parameter
+ void drawCurrentState(Color color)	Call draw function to draw a current state of square

+ void reset()	- Set state to State.EMPTY - Fill background as base color
+ void fillBackground(Color color)	- Fill background of square with color in parameter
+ getter/setter	

2.9 SubMenu extends Vbox

2.9.1 Fields

- int boardSize	Board size
- Text title	Title
- Text SubTitle	Subtitle
- ChoiceBox<String> gameMode	Choice box to select GameMode
- ChoiceBox<String> playType	Choice box to select PlayType
- Button start	Start the game with all current setting
- Button back	Go back to menu page
- HashMap<String, PlayType> playTypeMap	HashMap to map display String value to actual enum value
- HashMap<String, GameMode> gameModeMap	HashMap to map display String value to actual enum value

2.9.2 Constructor

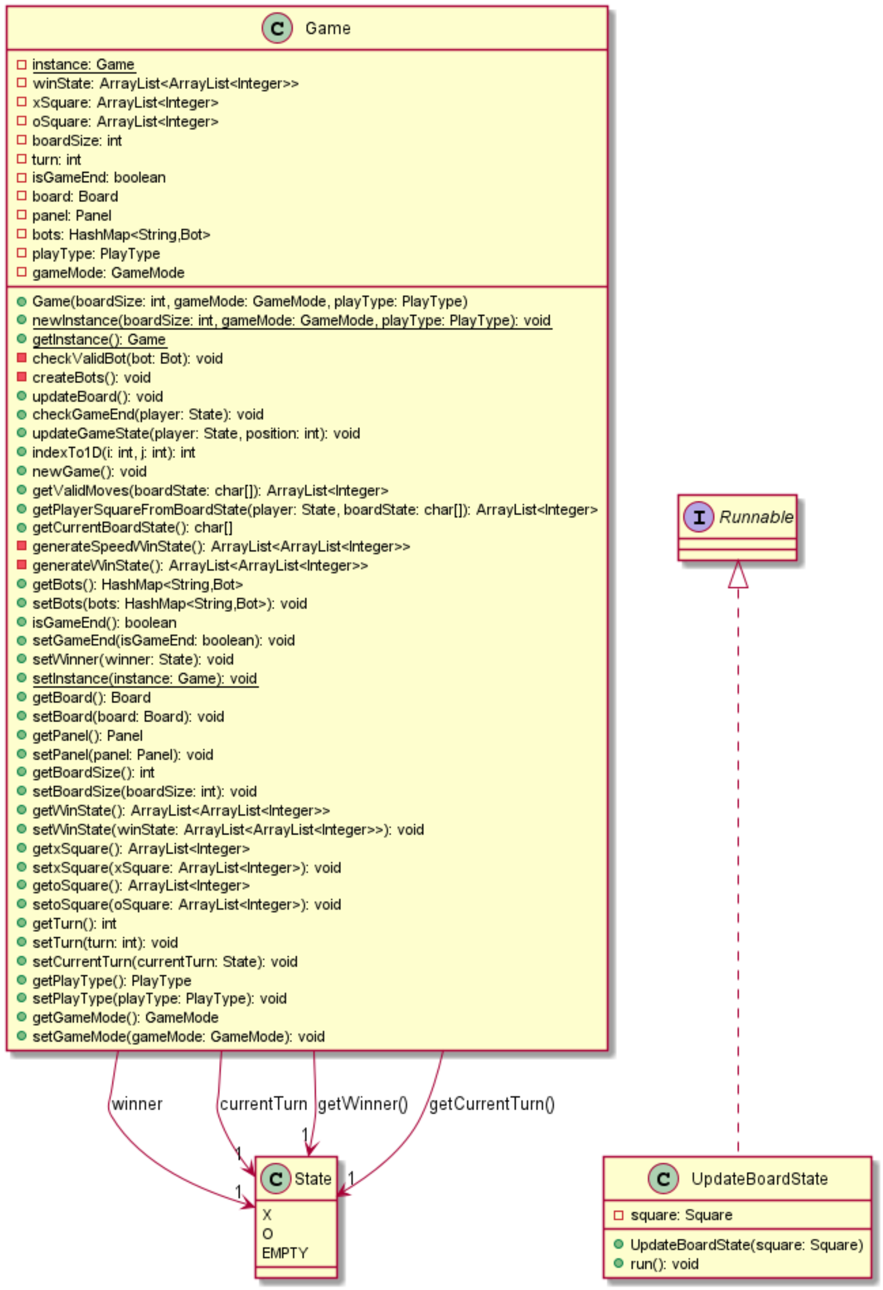
+ SubMenu(int boardSize)	- Do the styling - Store all parameter - Create every UI elements

2.9.3 Methods

+ void startGameHandler(ActionEvent event)	- Get all the current setting - Go to game page with all the setting
- Button createStartButton()	Create Button to go to game page by using startGameHandler
+ void backToMenu(ActionEvent event)	Go back to menu page
- Button createBackButton()	- Create Back Button - setOnAction to backToMenu
- ChoiceBox<String> createGameModeSelections()	- Create a choice box - Add Classic and speed to HashMap - Add Classic and Speed option to choiceBox
- ChoiceBox<String> createPlayTypeSelections()	- Create a choice box - Add PlayerVsPlayer and PlayperVsBot to HashMap

	- Add PlayerVsPlayer and PlayperVsBot option to choiceBox
+ getter/setter	

3.Package logic


Game
instance: Game
winState: ArrayList<ArrayList<Integer>>
xSquare: ArrayList<Integer>
oSquare: ArrayList<Integer>
boardSize: int
turn: int
isGameEnd: boolean
board: Board
panel: Panel
bots: HashMap<String,Bot>
playType: PlayType
gameMode: GameMode
Game(boardSize: int, gameMode: GameMode, playType: PlayType)
newInstance(boardSize: int, gameMode: GameMode, playType: PlayType): void
getInstance(): Game
checkValidBot(bot: Bot): void
createBots(): void
updateBoard(): void
checkGameEnd(player: State): void
updateGameState(player: State, position: int): void
indexTo1D(i: int, j: int): int
newGame(): void
getValidMoves(boardState: char[]): ArrayList<Integer>
getPlayerSquareFromBoardState(player: State, boardState: char[]): ArrayList<Integer>
getCurrentBoardState(): char[]
generateSpeedWinState(): ArrayList<ArrayList<Integer>>
generateWinState(): ArrayList<ArrayList<Integer>>
getBots(): HashMap<String,Bot>
setBots(bots: HashMap<String,Bot>): void
isGameEnd(): boolean
setGameEnd(isGameEnd: boolean): void
setWinner(winner: State): void
setInstance(instance: Game): void
getBoard(): Board
setBoard(board: Board): void
getPanel(): Panel
setPanel(panel: Panel): void
getBoardSize(): int
setBoardSize(boardSize: int): void
getWinState(): ArrayList<ArrayList<Integer>>
setWinState(winState: ArrayList<ArrayList<Integer>>): void
getxSquare(): ArrayList<Integer>
setxSquare(xSquare: ArrayList<Integer>): void
getoSquare(): ArrayList<Integer>
setoSquare(oSquare: ArrayList<Integer>): void
getTurn(): int
setTurn(turn: int): void
setCurrentTurn(currentTurn: State): void
getPlayType(): PlayType
setPlayType(playType: PlayType): void
getGameMode(): GameMode
setGameMode(gameMode: GameMode): void
Runnable
winner
currentTurn
getWinner()
getCurrentTurn()
1
1
1
1
State
X
O
EMPTY
UpdateBoardState
square: Square
UpdateBoardState(square: Square)
run(): void

3.1 Game

3.1.1 Fields

- static Game instance	Current Game instance
- ArrayList<ArrayList<Integer>> winState	Store all possible win state depending on board size and gameMode
- ArrayList<Integer> xSquare	Store all X’s Square
- ArrayList<Integer> oSquare	Store all O’s Square
- int boardSize	Board size
- int turn	Turn count
- boolean isGameEnd	Check if game ended
- State winner	Winner - State.O if O win - State.X if X win - State.EMPTY if draw
- State currentTurn	Current player turn - State.O if O to play - State.X if X to play
- Board board	Store Current Board
- Panel panel	Store Current Panel
- HashMap<String, Bot> bots	HashMap to map display String value to actual Bot object
- PlayType playType	PlayerVsPlayer or

	PlayerVsBot
- GameMode gameMode	Classic or Speed

3.1.2 Constructor

+ Game(int boardSize, GameMode gameMode, PlayType playType)	- Do the styling - Store all parameter - Generate win state depending on board size and gameMode - Use newGame method to reset a game status

3.1.3 Methods

+ drawWinSquare	Draw a green background to a winning square when game ended
+ static void newInstance(int boardSize, GameMode gameMode, PlayType playType)	Create a new instance of game using a parameter
- void checkValidBot(Bot bot)	Check if bot can be use for this board size and gameMode
- void createBots()	- Create a bot object - Use CheckValidBot to each bot to check if it can be use in this game or not

+ void updateBoard()	Update the board UI to the most recent one
+ Int checkGameEnd(State player)	Check if there is any winner or if the game ended. If yes set gameEnd to true and set the result of the game
+ void updateGameState(State player, int position)	- Update the game state when the player makes a play. - Progressing the turn - Check if there any winner using checkGameEnd - Call the UI update function
+ int indexTo1D(int i, int j)	2d array index to 1d using formula i * board size + j
+ void newGame()	Reset all parameter to default and ready for a new game
+ ArrayList<Integer> getValidMoves(char[] boardState)	Return all empty square that are available in the board state
+ ArrayList<Integer> getPlayerSquareFromBoardState(State player, char[] boardState)	Return all square that are own by a specific player in the board state
+ char[] getCurrentBoardState()	Return current board state as a char array
- ArrayList<ArrayList<Integer>> generateSpeedWinState()	Generate all win state for Speed gameMode depending on board size

- ArrayList<ArrayList<Integer>> generateWinState()	Generate all win state for Classic gameMode depending on board size
+ getter/setter	

3.2 enum GameMode

3.2.1 Constructor

+ enum State	- O - X - EMPTY

3.3 UpdateBoardState implements Runnable

3.3.1 Fields

- Square square	Storing Square

3.3.2 Constructor

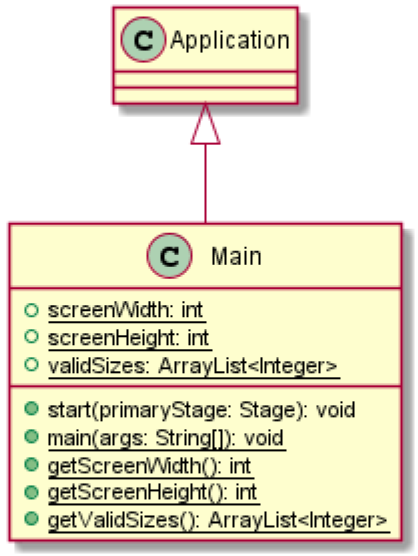
+ UpdateBoardState(Square square)	- Store all parameter - Super constructor

3.3.3 Methods

+ void run()	- Update the board UI to current State - Update game state in

	Game

4.Package Main


Application
Main
screenWidth: int
screenHeight: int
validSizes: ArrayList<Integer>
start(primaryStage: Stage): void
main(args: String[]): void
getScreenWidth(): int
getScreenHeight(): int
getValidSizes(): ArrayList<Integer>

4.1 Main extends Application

3.1.1 Fields

- static final int screenWidth	Application screenWidth
- static final int screenHeight	Application screenHeight
- static final ArrayList<Integer> validSizes	Store all valid board size

3.1.2 Methods

+ void start(Stage primaryStage) throws Exception	Create new Menu and display it

+ static void main(String[] args)	Call launch(args)
+ getter/setter	