Tic-Tac-Toe

Java Project

Team-1-

Members:

- Abdallah Helal
- Bassant Ibrahim
- Michael Samir
- Omar Shaker
- Rana Wahid

Supervised by:

Eng. Eman Hesham

Objective:

This project aims to develop a Tic Tac Toe game using Java controlled with gamepads. It mainly consists of developing and implementing two playing modes: Single Player and Multi-Player.

Game Description:

Tic Tac Toe is a two-player game. In this game, there is a board with 3 x 3 squares.

The two players take turns putting marks on a 3x3 board. The goal of Tic Tac Toe game is to be one of the players to get three same symbols in a row - horizontally, vertically or diagonally on a 3 x 3 grid. The player who first gets 3 of his/her symbols in a row wins the game, and the other loses the game.

Game Rules:

1.

a. The 2 players in Multi-Player mode play use “X” and “O”. The player that gets to play first will get the "X" mark (we call him/her player X) and the player that gets to play second will get the "O" mark (we call him/her player O).

b. While in Single Player mode, the player only plays with "X" and the computer plays with "O". The player gets to play first with "X" and it takes turns against the computer.

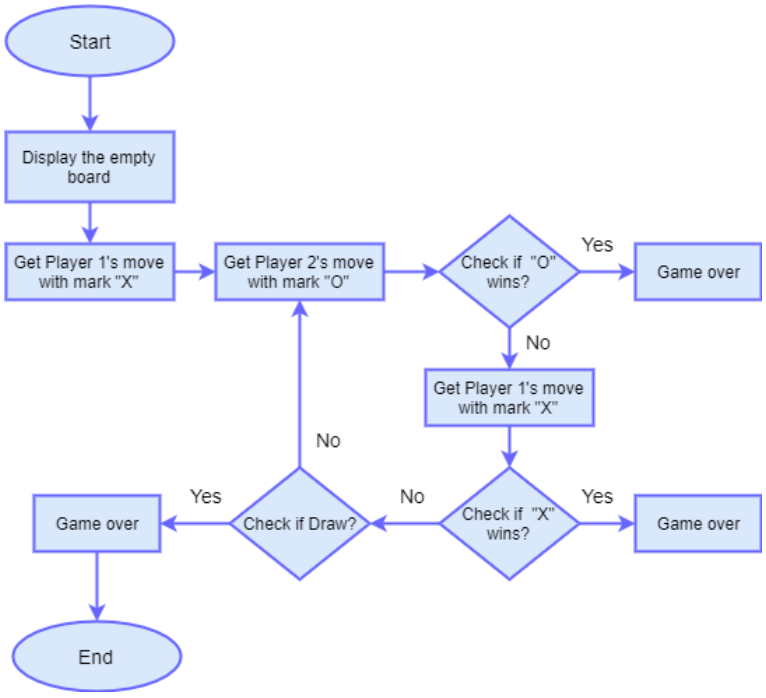
2. A player marks any of the 3x3 squares with his mark (“X” or “O”) and their aim is to create a straight line horizontally, vertically or diagonally with two intensions:

a. One of the players gets three of his/her marks in a row (vertically, horizontally, or diagonally) i.e. that player wins the game.

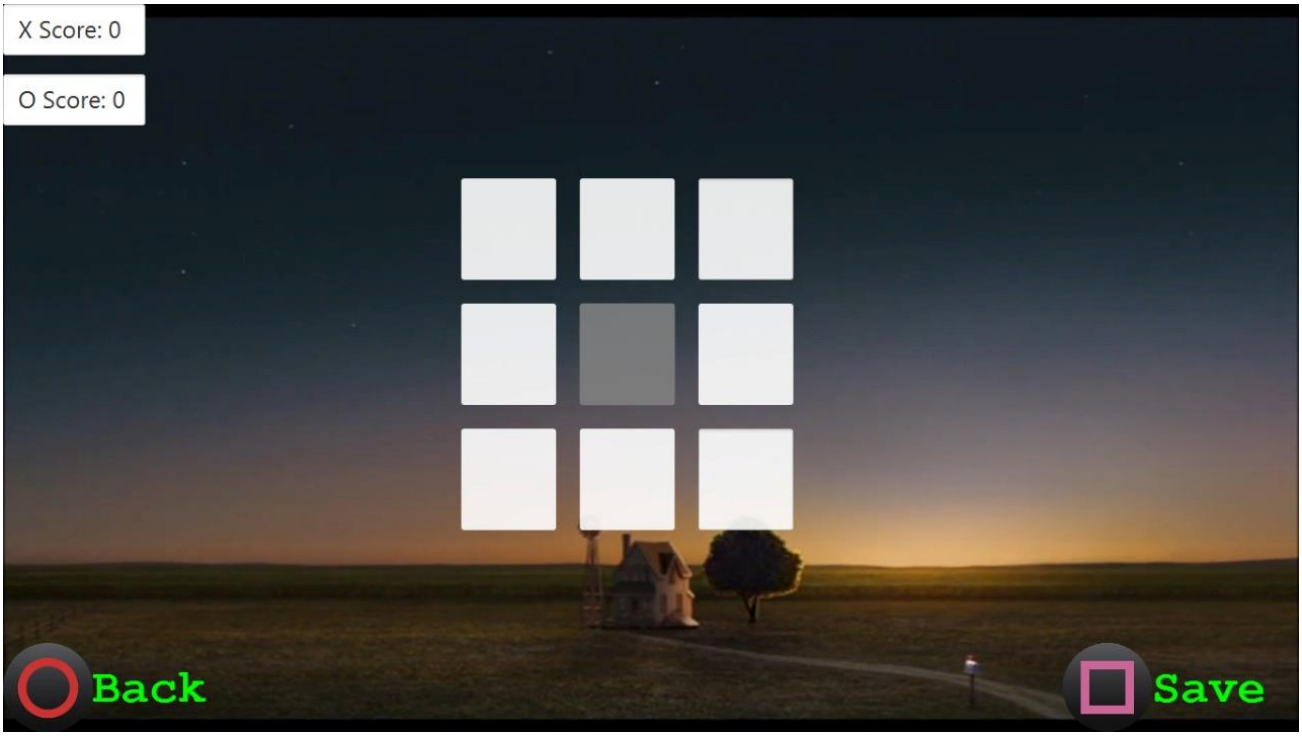
b. If no one can create a straight line with their own mark and all the positions on the board are occupied, then the game ends in a draw/tie.

Implementation Plan:

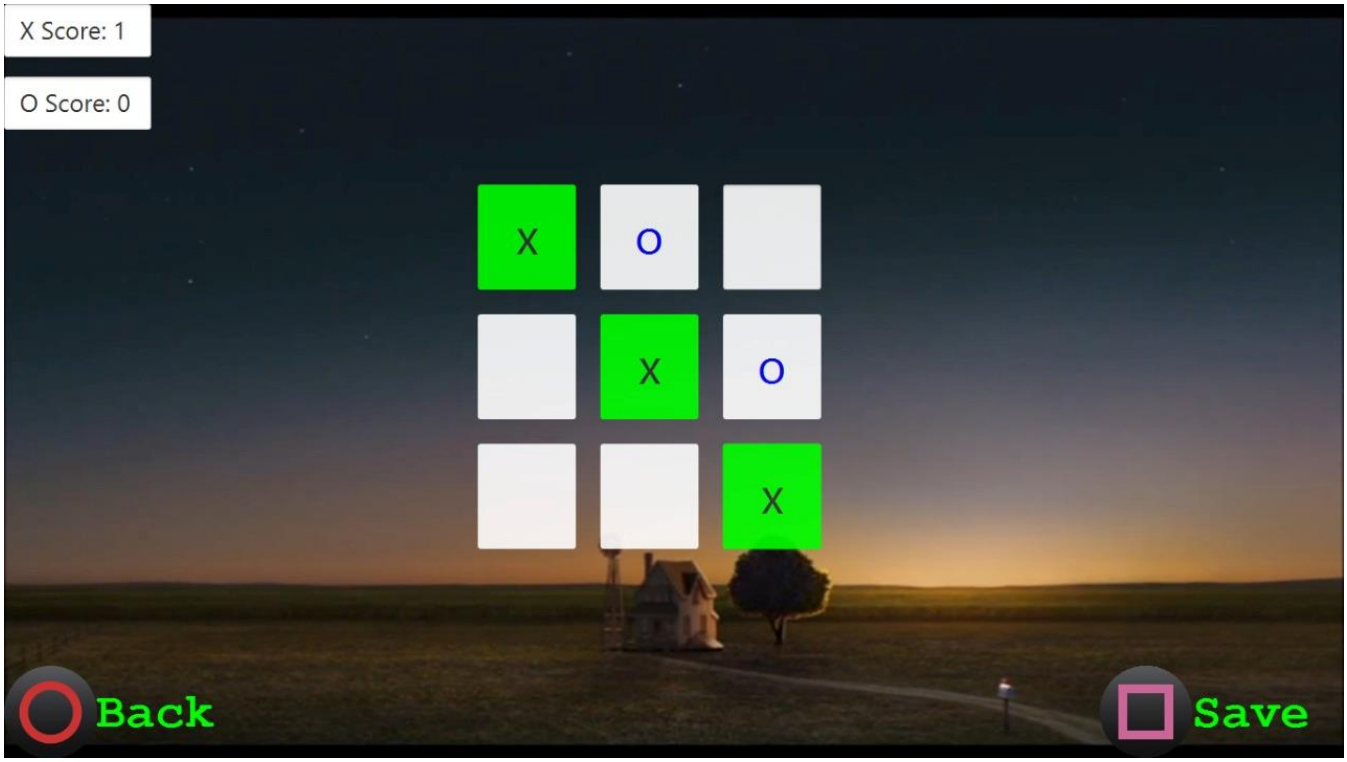
The implementation workflow for this project is as follows:



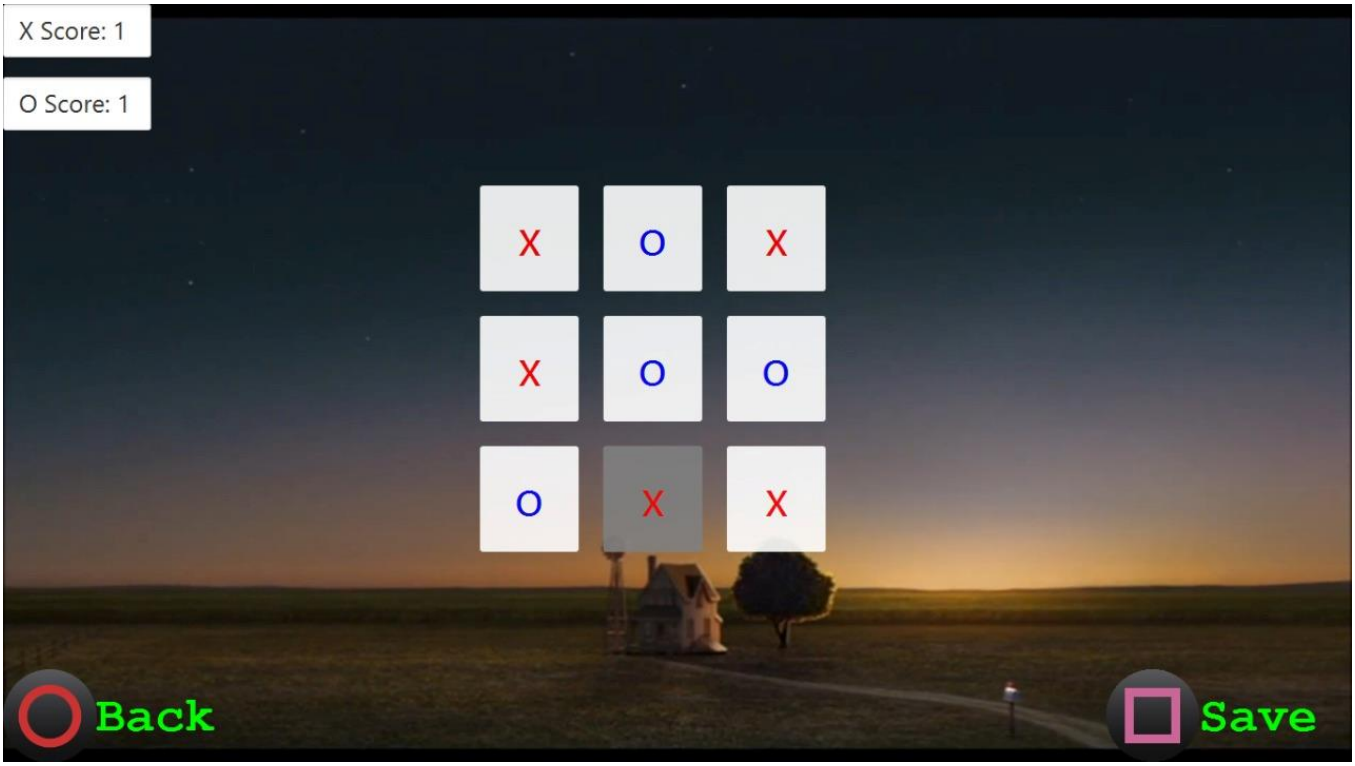
In order to visualize the defined game rules and description, the game is shown in Figures below. First the game will start with empty board.



Then Player X will make his/her first move by playing mark “X” on this board. Then Player O will make his/her move by playing mark “O” on this board. This will keep on continuing until the board is full of marks. Then the program will check if Player X or Player O won and that scenario will be follows: (could be vertically, horizontally or diagonally).



If none of the players win, the program will check for draw.



All this decision making is done by using Minimax algorithm.

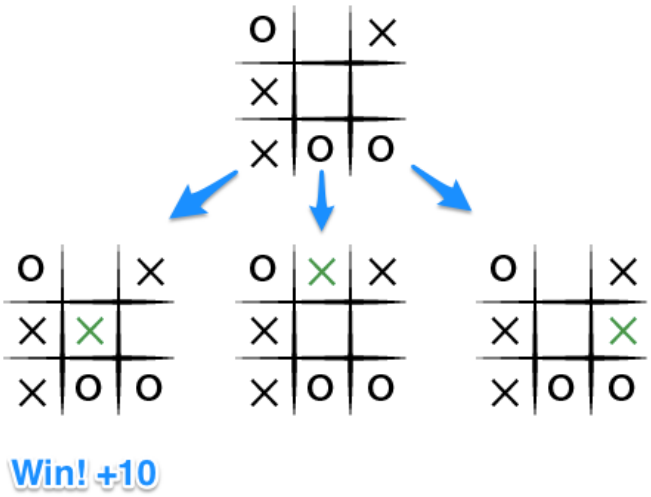
Minimax Algorithm

Minimax is a artificial intelligence algorithm applied to a two player Tic Tac Toe game. This games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and other player loses (-1) or both of anyone not to win (0).

Minimax is a recursive algorithm which is used to choose the best move that leads the Max player to win or not lose (draw). It considers the current state of the game and the available moves at that state, then for each valid move it plays (alternating min and max) until it finds a terminal state - win, draw or lose. Its goal is to minimize the maximum loss i.e. minimize the worst case scenario.

Explanation with an Example

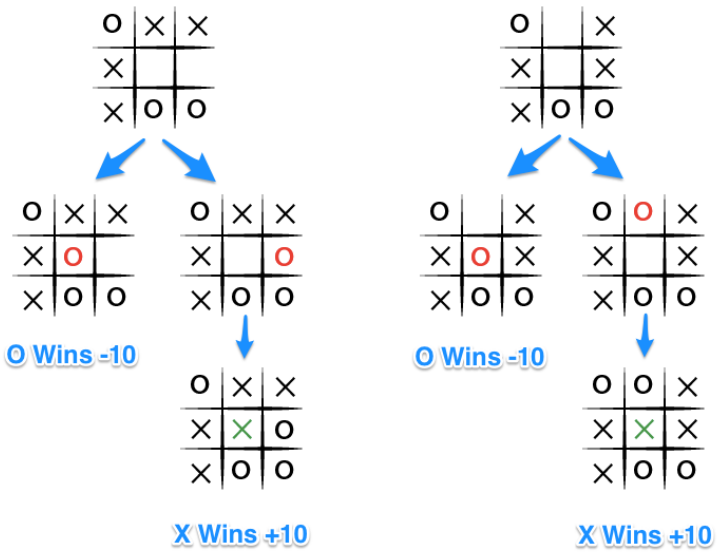
To apply this, let's take an example from near the end of a game, where it is my turn. I am X. My goal here, obviously, is to maximize my end game score.



If the top of this image represents the state of the game when it is my turn, then I have some choices to make, there are three places I can play, one of which clearly results in me wining and earning the 10 points. If I don't make that move, O could very easily win. And I don't want O to win, so my goal here, as the first player, should be to pick the maximum scoring move.

But What About O?

We should assume that O is also playing to win this game, but relative to us, the first player, O wants obviously wants to chose the move that results in the worst score for us, it wants to pick a move that would minimize our ultimate score. Let's look at things from O's perspective, starting with the two other game states from above in which we don't immediately win.



The choice is clear, O would pick any of the moves that result in a score of -10.

Describing Minimax

The key to the Minimax algorithm is a back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.

A description for the algorithm, assuming X is the turn taking player:

* If the game is over, return the score from X's perspective.

* Otherwise get a list of new game states for every possible move.

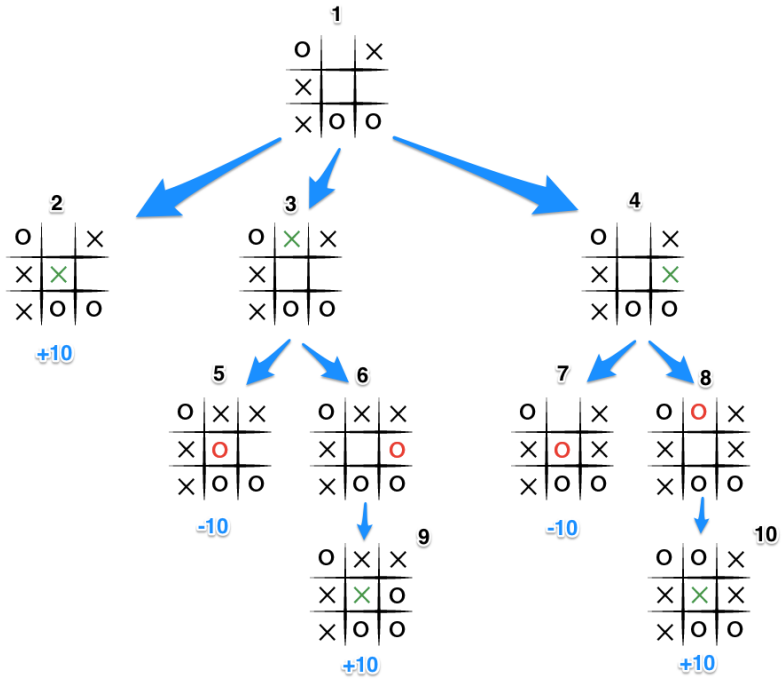
* Create a scores list.

* For each of these states add the minimax result of that state to the scores list.

* If it's X's turn, return the maximum score from the scores list.

* If it's O's turn, return the minimum score from the scores list.

Let's walk through the algorithm's execution with the full move tree, and algorithmically, how the instant winning move will be picked:



* It's X's turn in state 1. X generates the states 2, 3, and 4 and calls minimax on those states.

* State 2 pushes the score of +10 to state 1's score list, because the game is in an end state.

* State 3 and 4 are not in end states, so 3 generates states 5 and 6 and calls minimax on them, while state 4 generates states 7 and 8 and calls minimax on them.

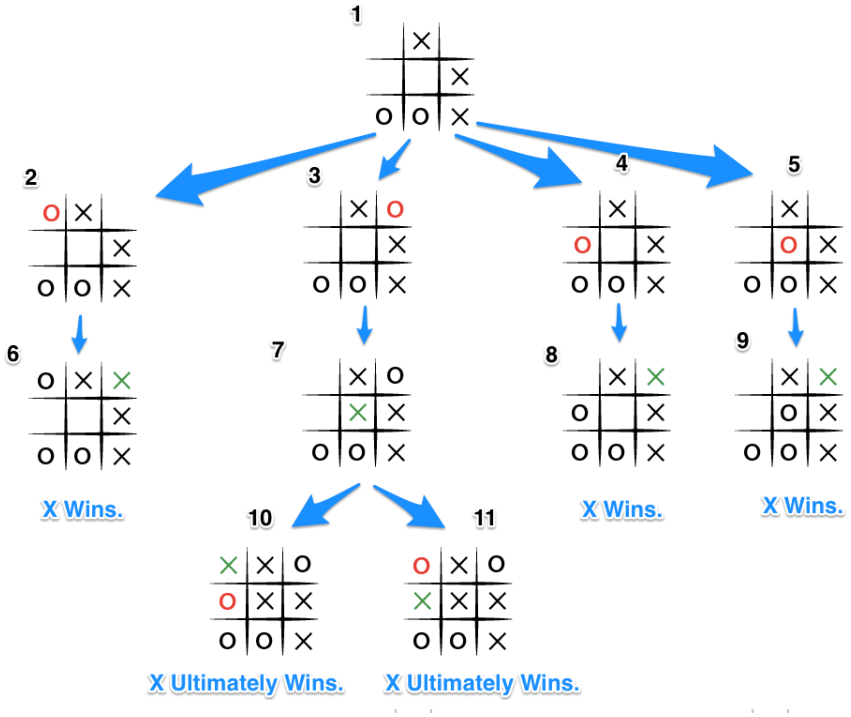
* State 5 pushes a score of -10 onto state 3's score list, while the same happens for state 7 which pushes a score of -10 onto state 4's score list.

* State 6 and 8 generate the only available moves, which are end states, and so both of them add the score of +10 to the move lists of states 3 and 4.

* Because it is O's turn in both state 3 and 4, O will seek to find the minimum score, and given the choice between -10 and +10, both states 3 and 4 will yield -10.

* Finally the score list for states 2, 3, and 4 are populated with +10, -10 and -10 respectively, and state 1 seeking to maximize the score will chose the winning move with score +10, state 2.

Let's see what is happening here by looking through the possible move tree:



* Given the board state 1 where both players are playing perfectly, and O is the computer player. O choses the move in state 5 and then immediately loses when X wins in state 9.

* But if O blocks X's win as in state 3, X will obviously block O's potential win as shown in state 7.

* This puts two certain wins for X as shown in state 10 and 11, so no matter which move O picks in state 7, X will ultimately win.

Another important factor in this algorithm is depth.

The key improvement to this algorithm, such that, no matter the board arrangement, the perfect player will play perfectly, is to take the "depth" or number of turns till the end of the game into account. Basically the perfect player should play perfectly, but prolong the game as much as possible.

So each time we invoke minimax, depth is incremented by 1 and when the end game state is ultimately calculated, the score is adjusted by depth.

Since this is a very complex algorithm, we have a computer to execute this algorithm.

Code Implementation:

We have implemented just one major class ***"Es42_tictactoe.java".***

Those are the major implemented methods and threads:

"clearBoard" method

Looping over rows and columns and inserting blank string.

```
public void clearBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            gameBoxes[i][j].setText("");
            gameBoxes[i][j].setStyle(defaultStyle);
        }
    }
    gameBoxes[yPos][xPos].setStyle("-fx-background-color: grey");
}
```

"initStick" method

Looping over connected controllers and referencing connected gamepads.

```
public void initStick() {
    controllerList = ControllerEnvironment.getDefaultEnvironment().getControllers();
    int controllersCount = 0;
    for (int i = 0; i < controllerList.length; i++) {
        if (controllerList[i].getType() == Controller.Type.STICK) {
            if (controllerList[i].poll()) {
                if (controllersCount == 0) {
                    gamePad1 = controllerList[i];
                    queue1 = gamePad1.getEventQueue();
                    controllersCount++;
                } else {
                    gamePad2 = controllerList[i];
                    queue2 = gamePad2.getEventQueue();
                    controllersCount++;
                }
            }
        }
    }
    if (controllersCount == 1) {
        gamePad2 = gamePad1;
        queue2 = gamePad1.getEventQueue();
    }
}
```

"init" method

Setting roots' nodes and running gamepads listener thread.

"start" method

Setting and showing stage icon, title and scene and making it full screen.

```
@Override
public void start(Stage primaryStage) {

    primaryStage.setTitle("TicTacToe");
    primaryStage.setScene(currentScene);
    primaryStage.setFullScreen(true);
    primaryStage.getIcons().add(new Image("file:images/icon.png"));
    currentStage = primaryStage;
    primaryStage.show();
}
```

GamePad Thread

If any button is pressed, it checks button name, value and current root ID. Then, takes the appropriate action.