MAY 2023

Game Engine

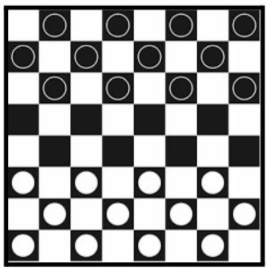
Programming Paradigms Project

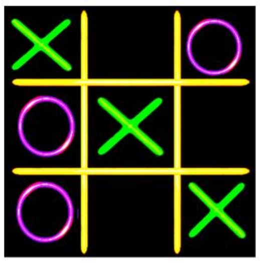
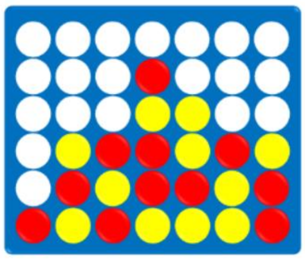
Contributors

Ahmed Hesham Abdelrahman Wael Mahmoud Gouda Mohamed Amin

Overview

The objective of this project is to explore the distinction between the characteristics of **object-oriented** and **functional programming** paradigms by implementing a generic game engine. The engine will be utilized for drawing game boards, implemented two times, once using JavaScript and once using Scala. Six games will be supported by the engine: *Tic-Tac-Toe, Connect-4*, *Checkers, Chess, Sudoku, and 8-Queens*. Additionally, the engine should be easily extendable to draw any other board game.



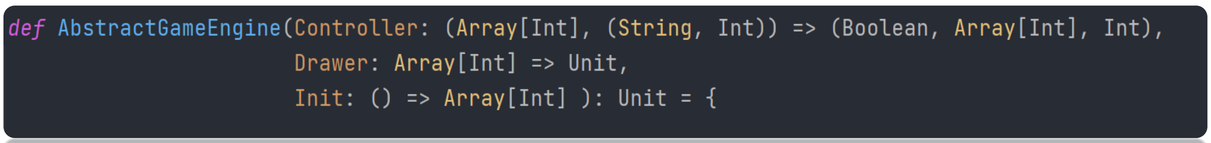



The engine's primary purpose will be to handle two tasks: drawing the board and pieces and enforcing the rules of moving the pieces. This report will examine how the implementation of the game engine differs depending on the programming paradigm utilized, and the advantages and disadvantages of each approach.

Main Features

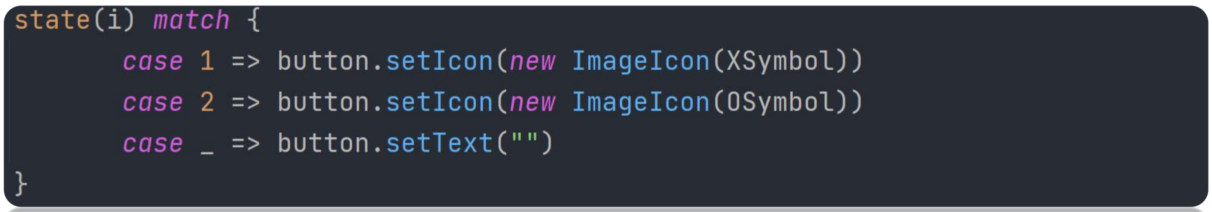
Scala Implementation

Higher Order Function => Our abstract game engine functions takes 3 functions as parameters which are Drawer, Controller and Init:



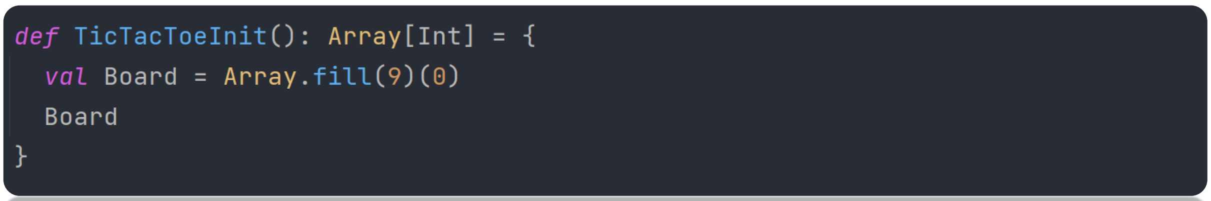
```
def AbstractGameEngine(Controller: (Array[Int], (String, Int)) => (Boolean, Array[Int], Int),
                       Drawer: Array[Int] => Unit,
                       Init: () => Array[Int] ): Unit = {
```

Pattern Matching => We used pattern matching multiple time through out the project to match the pieces to values as in TicTacToe we matched the state value to 3 values:



```
state(i) match {
       case 1 => button.setIcon(new ImageIcon(XSymbol))
       case 2 => button.setIcon(new ImageIcon(OSymbol))
       case _ => button.setText("")
}
```

Pure Functions => Each game has an initializer which generates the same board for every new game (Except sudoku) and takes no arguments:

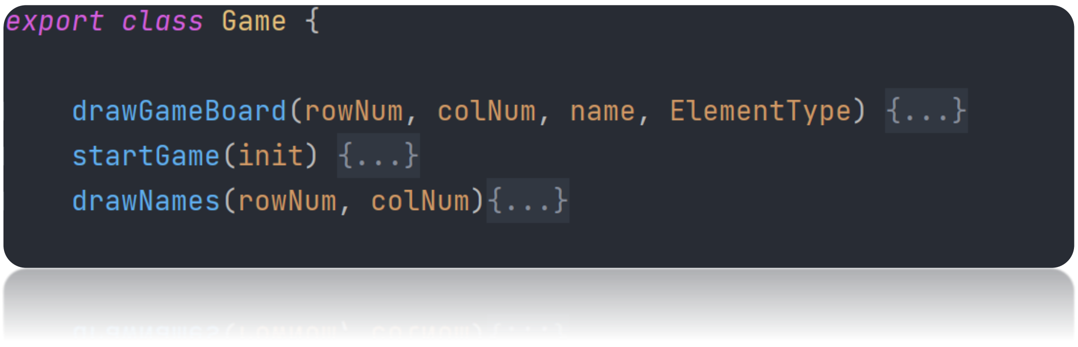


```
def TicTacToeInit(): Array[Int] = {
  val Board = Array.fill(9)(0)
  Board
}
```

Main Features

Javascript Implementation

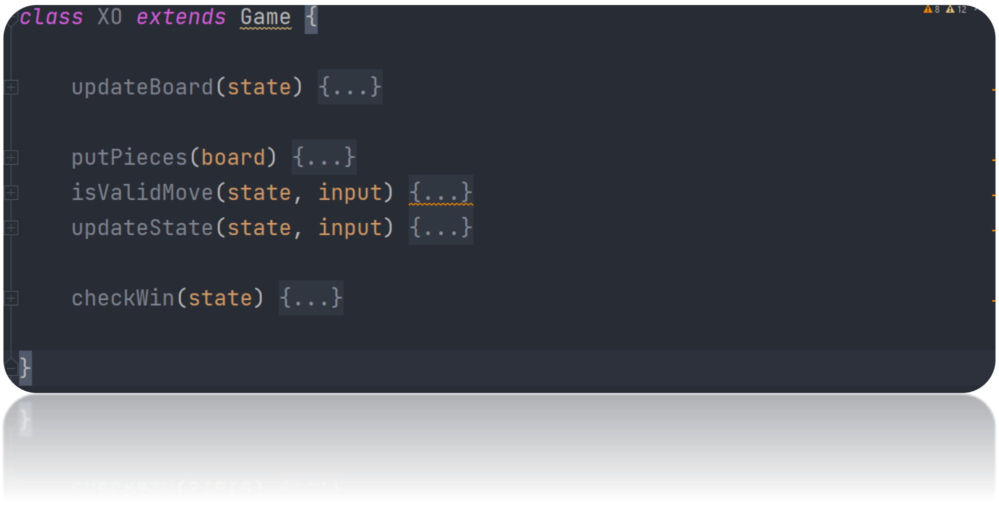
Inheritance => It provides reuseablity of functions of common logic such as sequence in game (Start game, drawGameBoard, drawNames …) in classes that extend from the parent class (Game).



```
export class Game {

    drawGameBoard(rowNum, colNum, name, ElementType) {...}
    startGame(init) {...}
    drawNames(rowNum, colNum){...}
```

Overriding => This feature is used which enables each game class to provide different implementation for a method that is already defined in the parent class. (updateBoard, updateState, isValidMove, …)



```
class XO extends Game {

    updateBoard(state) {...}

    putPieces(board) {...}
    isValidMove(state, input) {...}
    updateState(state, input) {...}

    checkWin(state) {...}
}
```

Comparisons

Main Diffrences:

In ***Scala***, the program is a sequence of functions which call each other. Whereas in ***Javascript***, there is a parent class which is extended by each game where each game is a class having its own set of functions.

Pros and Cons:

POC	Pros	Cons
Scala	1) Programs are stateless 2) Well-suited for parallelism	1) No code reuseability 2) Not suitable for all programs
Javascript	1) We can reuse the code multiple times using class 2) Inherit the class to subclass for data redundancy	1) It required a lot of time to create 2) Complex Structure