

Introduction

Java is a powerful programming language that is widely used for developing a variety of applications, including graphical user interfaces (GUIs). In this project, we used Java Graphics to develop a simple application that demonstrates the capabilities of this programming language.

The goal of this project was to create a graphical representation of a classic game, Tic-Tac-Toe. Our project utilized Java Graphics to create an intuitive user interface, which allows users to interact with the game by clicking on buttons and ask permission for exit or replay the game.

In addition to demonstrating the power of Java Graphics, our project also showcases some of the fundamental programming concepts that are essential to any programming language, such as object-oriented programming. We made different objects including the properties of Game, Players and GameBoard.

Through this project, we hope to provide a valuable resource for anyone looking to learn more about programming with Java Graphics, and to inspire future developers to explore the endless possibilities of this powerful language.

Implementation

For this project, I used Java to develop the Tic-Tac-Toe game as per instructions. The game consists of a 3x3 board and allows two players to take turns placing X's and O's on the board until a win condition is met or the board is full. Here how I implemented the game using four classes :

This code is basically button based as I implemented all the logics and building structure upon the buttons where user clicks along with giving input directly. To develop the game, I first created four classes, **Game**: a class for the game properties, **Player**: a class for the player properties, **GameBoard**: a class for the board operation properties and **TicTacToeGame**: a class to implement all other class properties, game logics and visualize the gameboard to the users.

1. Game.java

The class called "**Game**" contains two array variables *pXset[]* and *pOset[]* both initialized with nine zeros (each refers to a button) who store as well as used to mark the game moves of the players 'X' and 'O' separately. When a user clicks a button it changes the values of these arrays on the consideration of which button is pressed.

Source Code:

```
int[] pXset = {0,0,0,0,0,0,0,0,0};  
int[] pOset = {0,0,0,0,0,0,0,0,0};
```

There are also three methods- one method is *resetGame()* which contains a loop to reset the arrays to 0 when player wants to play again and to help reset the board to play a new game.

Source Code:

```
void resetGame() {
    for(int i=0; i<9; i++){
        pXset[i]=0;
        pOset[i]=0;
    }
}
```

The other two of the methods with Boolean return type are *pXwin()* and *pOwin()* who implement the total 8 combinations of logics for the winning conditions of the button inputs for player 'X' and 'O'. The winning conditions are as follows:

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

b0	b1	b2
b3	b4	b5
b6	b7	b8

Source Code:

```
public boolean pXwin() {
    // Horizontal checks
    if ((pXset[0] == 1 && pXset[1] == 1 && pXset[2] == 1)) {
        return true;
    }
    else if (pXset[3] == 1 && pXset[4] == 1 && pXset[5] == 1) {
        return true;
    }
    else if (pXset[6] == 1 && pXset[7] == 1 && pXset[8] == 1) {
        return true;
    }
}
```

```

    }
    // vertical checks
    else if (pxset[0] == 1 && pxset[3] == 1 && pxset[6] == 1) {
        return true;
    }
    else if (pxset[1] == 1 && pxset[4] == 1 && pxset[7] == 1) {
        return true;
    }
    else if (pxset[2] == 1 && pxset[5] == 1 && pxset[8] == 1) {
        return true;
    }
    // Diagonal checks
    else if (pxset[0] == 1 && pxset[4] == 1 && pxset[8] == 1) {
        return true;
    }
    else if (pxset[2] == 1 && pxset[4] == 1 && pxset[6] == 1) {
        return true;
    }
    return false;
}

public boolean powin() {
    // Horizontal checks
    if ((p0set[0] == 1 && p0set[1] == 1 && p0set[2] == 1)) {
        return true;
    }
    else if (p0set[3] == 1 && p0set[4] == 1 && p0set[5] == 1) {
        return true;
    }
    else if (p0set[6] == 1 && p0set[7] == 1 && p0set[8] == 1) {
        return true;
    }
    // vertical checks
    else if (p0set[0] == 1 && p0set[3] == 1 && p0set[6] == 1) {
        return true;
    }
    else if (p0set[1] == 1 && p0set[4] == 1 && p0set[7] == 1) {
        return true;
    }
    else if (p0set[2] == 1 && p0set[5] == 1 && p0set[8] == 1) {
        return true;
    }
}

```

```

    }

    // Diagonal checks
    else if (pOset[0] == 1 && pOset[4] == 1 && pOset[8] == 1) {
        return true;
    }
    else if (pOset[2] == 1 && pOset[4] == 1 && pOset[6] == 1) {
        return true;
    }
    return false;
}

```

2. Player.java

In the class called "**Player**" I declared and initialized a character variable *nowPlayer* to store the current player 'X' or 'O' which is used toggling the input for the current player between 'X' and 'O'. Initially the current player is 'X'. When current player is 'X', it is used to change the button text to 'X' and toggle *nowPlayer* to 'O' and when current player is 'O', it changes the button text to 'O' and toggle *nowPlayer* to 'X'. Thus the process goes on while playing the game.

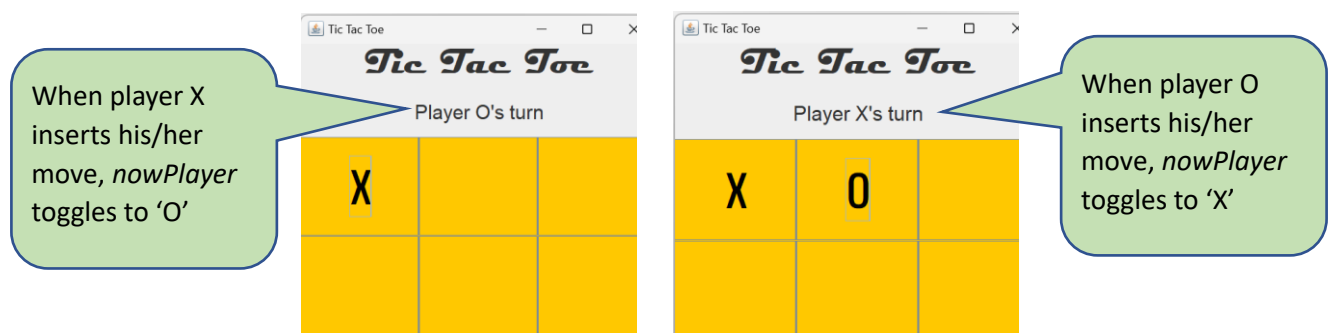
This class also contains a method to reset the variable *nowPlayer* to its normal state 'X'. It confirms that each game starts with the first move of player 'X'.

Source Code:

```

public char nowPlayer = 'X';
void resetPlayer(){
    nowPlayer = 'X';
}

```



2. GameBoard.java

In this class called "**GameBoard**" one integer variable is created called *moves* to track the count of valid moves. Using this variable property of the class it helped to determine whether all 9 valid moves are done or not.

Besides, we also declared and initialized an array of integers named *btnUsed[]* with 0s. It is used to store the used button records in a row. In the game it is used to track the invalid moves. For example, when a user clicks a button the array value changes to non-zero. Thus, by using it we can check the invalid moves by the user and prompt any error message want to show.



The class also contains a method called *resetGameBoard()* which resets the variables to their initial condition again.

Source Code:

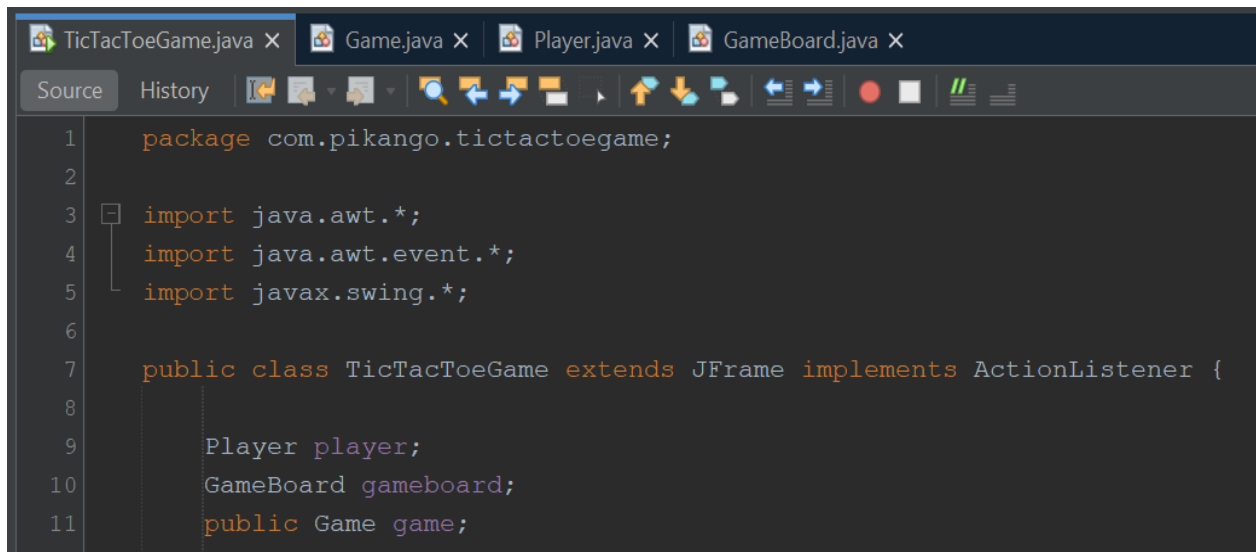
```
int moves = 0;
int[] btnUsed = {0,0,0,0,0,0,0,0,0};
void resetGameBoard() {
    moves=0;
    for(int i=0; i<9; i++){
        btnUsed[i]=0;
    }
}
```

4. TicTacToeGame.java

The class called "**TicTacToeGame**" which extends *JFrame*, implements *ActionListener* and contains the **main** method and the code for the gameboard graphics frame.

Within this class, I used *JFrame*, *JButton*, *JLabel*, *JOptionPane*, *Grid Layout* and *JPanel* to represent the game board visually. I processed the other class objects as the parameter of the constructor to create the instances of the game objects. The constructor adds elements of frame

implementing the ActionListener along with the empty buttons. The gameboard consists total of 9 buttons. Each button reacts upon the click of user based on the applied logical statements through ActionEvent Handling.



```

1 package com.pikango.tictactoe;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class TicTacToeGame extends JFrame implements ActionListener {
8
9     Player player;
10    GameBoard gameboard;
11    public Game game;

```

Next, I built the logic using into ActionListener for each button to read input from two players. For player turns, I used properties of other class objects to prompt the players for their invalid moves and to switch between players after each turn. Then handle invalid moves by the player, check for win conditions through methods from Game class, and show message dialogue using JOptionPane upon confirmation if user wants to play again or not.

To check for win conditions, I used *pXwin()* and *pOwin()* methods of Game class to check for horizontal, vertical, and diagonal matches on the game board. Then to show message after a win or draw I used a method called *playAgainConfirm(char x)* where parameter 'x' represents the current player. The method shows result with messages along with asking the confirmation to play again. If user choose to play again, it resets the game board after gaining confirmation by using the *resetGame()* method in it. Thus the game continues until user confirms to close the game while asking confirmation after a win or a draw. Selecting option 'NO' will close and exit the game window.

Here's some successful results after gameplays of Tic-Tac-Toe:



Overall, I found Java to be a suitable language for developing this Tic-Tac-Toe game. The object-oriented nature of Java allowed me to organize the game logic into separate methods and classes, which made the code easier to read and maintain.

Challenges and Solutions

While making the game I faced some errors. Among the errors the most challenging that I faced was developing the game with the implementation through four different classes. The four classes I made with different methods and field properties were not compositely working in the main class. The exact reason of that error was : I declared and made the instances of the objects of the three classes (Game, Player, GameBoard) into the class field instead of making instances of them through constructor. Moreover, I did not mention the objects in the main class therefore, the game compiled successfully but the game logic was not working properly. Because the instances were not able to be controlled by the main method or class, as a result changes were not figured properly and the result combination also got fractured logically.

To rescue from that situation, I took help from an online AI model called ChatGPT to find out the cases and observed the full code again. After spending times, I found the errors with the constructor and made changes as followings:

1. Only declaring class objects outside the constructor scope.
2. Making the constructor parameterized which will pass the three class objects and assigning them to the class object variables.
3. In the main class, making new instances of the three class objects and pass them through a new instance of TicTacToeGame GUI class as parameters.

Besides, I also faced non-sequential statement error. The problem was appeared while I was trying to add the *turnMessage* which shows which player is in turn to play move and 'GAME OVER' message while game is over. At first, I placed the toggling statement after the checking of winning condition but every time the winning condition is true the game gets reset but the player in turn did not change. As a result, *turnMessage* failed to set to its default state.

To solved the issue, I just changed the sequence of the two statements (toggling and win check) in the code section of each button scope and got the wanted result.

During the process of developing the Tic-Tac-Toe game using Java, I faced two major challenges. One was integrating four different classes, which taught me the importance of code organization and planning. The other challenge was sequencing, which reminded me of the importance of attention to detail. Through these challenges, I learned valuable problem-solving skills and feel more prepared for future coding projects.

Conclusion

In conclusion, the development of the tic tac toe game using Java was a challenging but rewarding project. The implementation of the game allowed for two players to take turns and compete against each other until a win or tie condition was met. The challenges encountered during the project, such as integrating multiple classes and sequencing code properly, helped to develop problem-solving skills and deepen the understanding of Java programming. The tic tac toe game project using Java served as a great introduction to programming fundamentals, particularly object-oriented programming concepts. It provided an opportunity to practice writing classes, methods, and constructors, as well as to gain experience in handling errors and debugging code. While the implementation of the game was relatively simple, it provided a solid foundation for future development of more complex games or applications. The game could be expanded to include additional features such as sound effects, different themes or styles, or even an online multiplayer mode. The introduction of an AI opponent, such as a simple computer algorithm or a more advanced machine learning model, could add to the game's replayability and provide a challenge for players of all skill levels.

Future implementations of the game could include adding additional features such as a scoreboard to keep track of wins and losses, or implementing an AI opponent for a single-player experience. The game could also be expanded to include larger board sizes or different win conditions for added variety. Moreover, a level wise rewarding progress can be developed in the game. Overall, the project served as a valuable learning experience and laid the foundation for potential future developments of the game.

THE END