Documentation for Algorithm and Programming Final Project: “TicTacToe”

Lecturer:
Mr. Jude Joseph Lamug Martinez, MCS.

Report done by member of Class L1CC:

Micky Malvino Kusandiwinata **2602174522**

Binus School of Computer Science Undergraduate Program
Universitas Bina Nusantara
Jakarta
2023

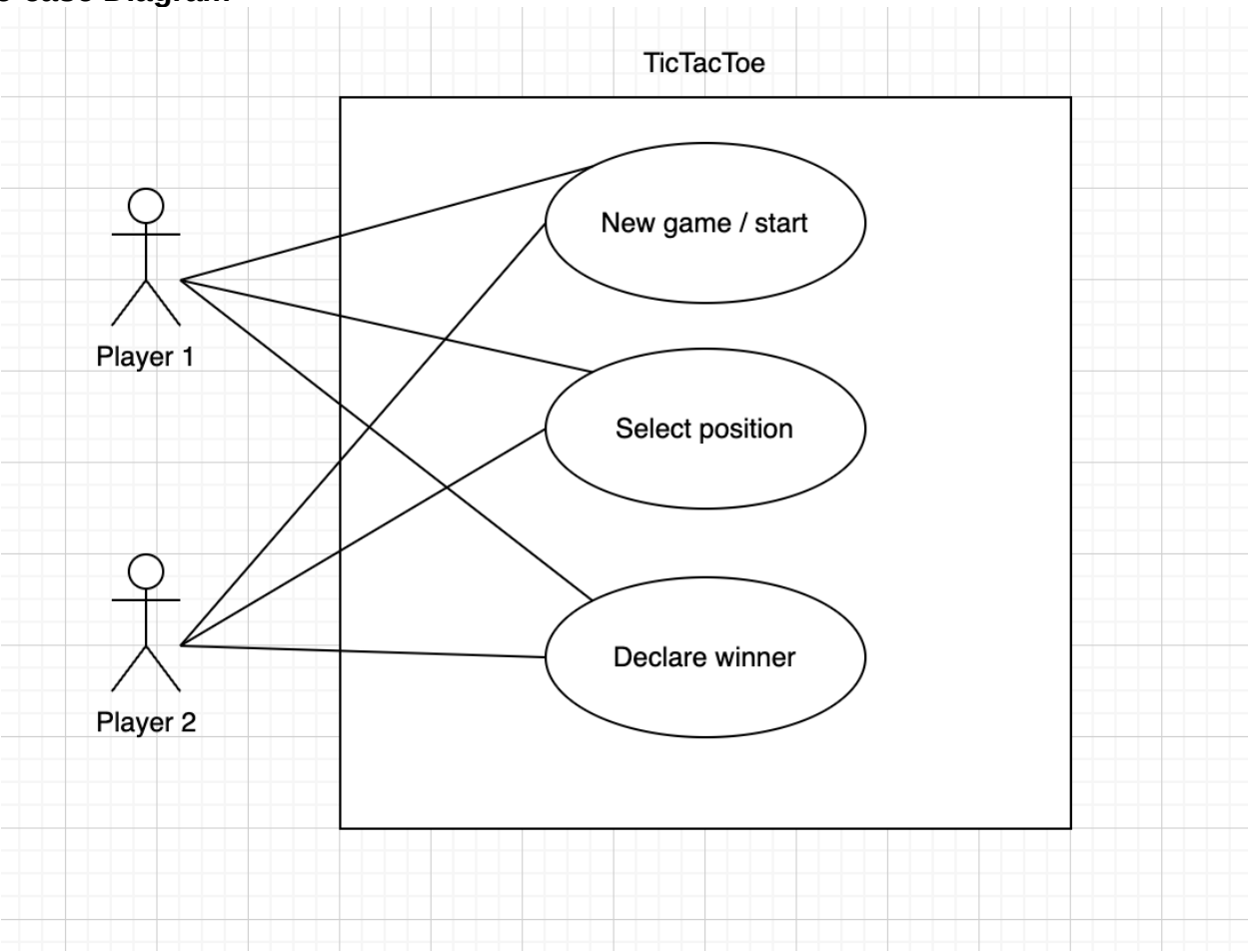
Project Documentation

Brief Description

For this final project, I made something we all know and love, Tic-Tac-Toe. Tic-tac-toe is one of the oldest game in the books, a simple game for two players, X and O, who take turns marking the spaces in a 3x3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. The game can also be a draw if neither player is able to get three in a row and all the spaces are filled. One of the main reasons I chose this is because of its simplicity and my love for the game, I used to grow up playing it all the time with my friends with pencil and paper, so naturally, once I realized I can put it into code using python, I was looking forward to it.

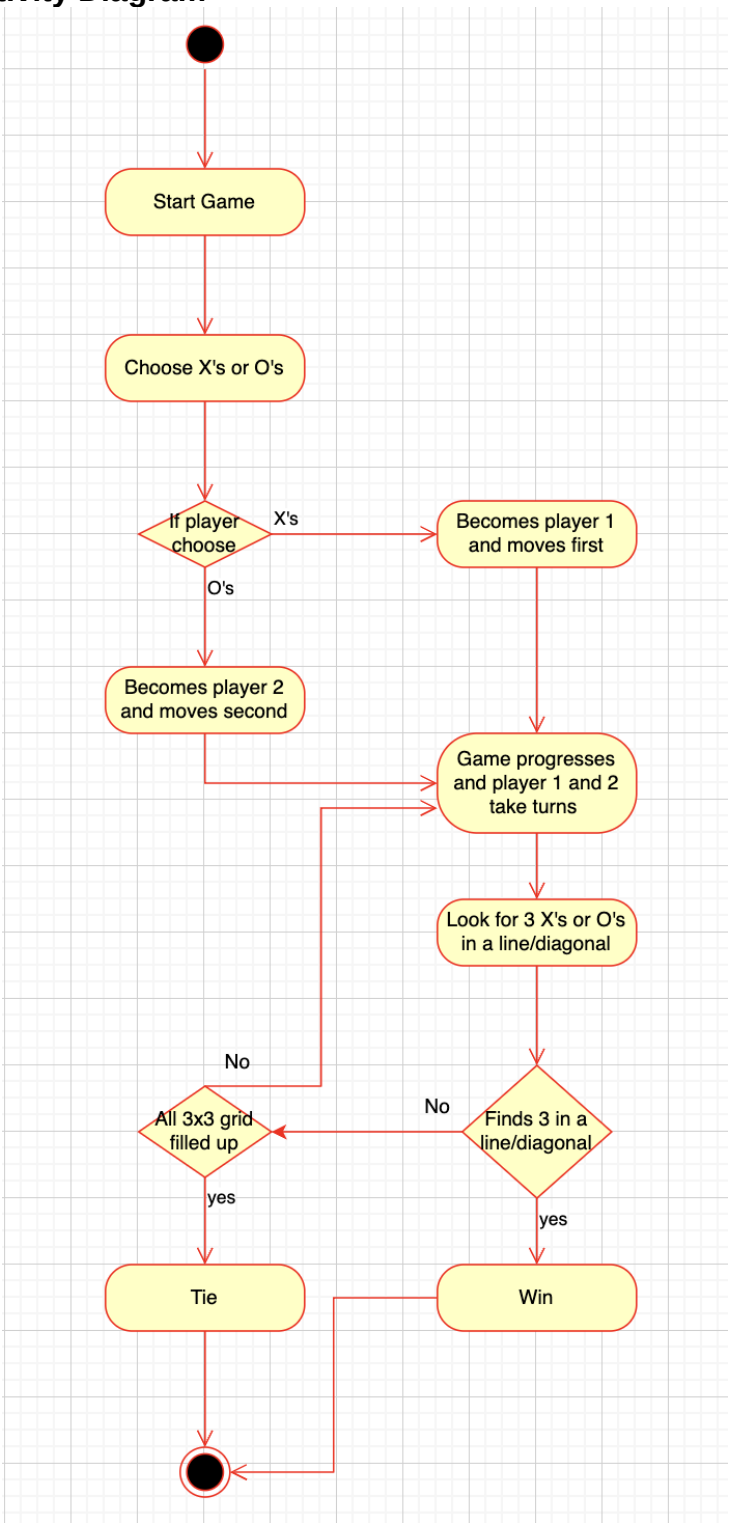
With this program, there are 3 modes, 1) is the 2-player mode where you can play against another player making it the traditional turn-based Tic-Tac-Toe game. 2) is where you play against a computer as your opponent and 3) is the impossible mode where you won't be able to win at all, just ties or losses :(

Use-case Diagram

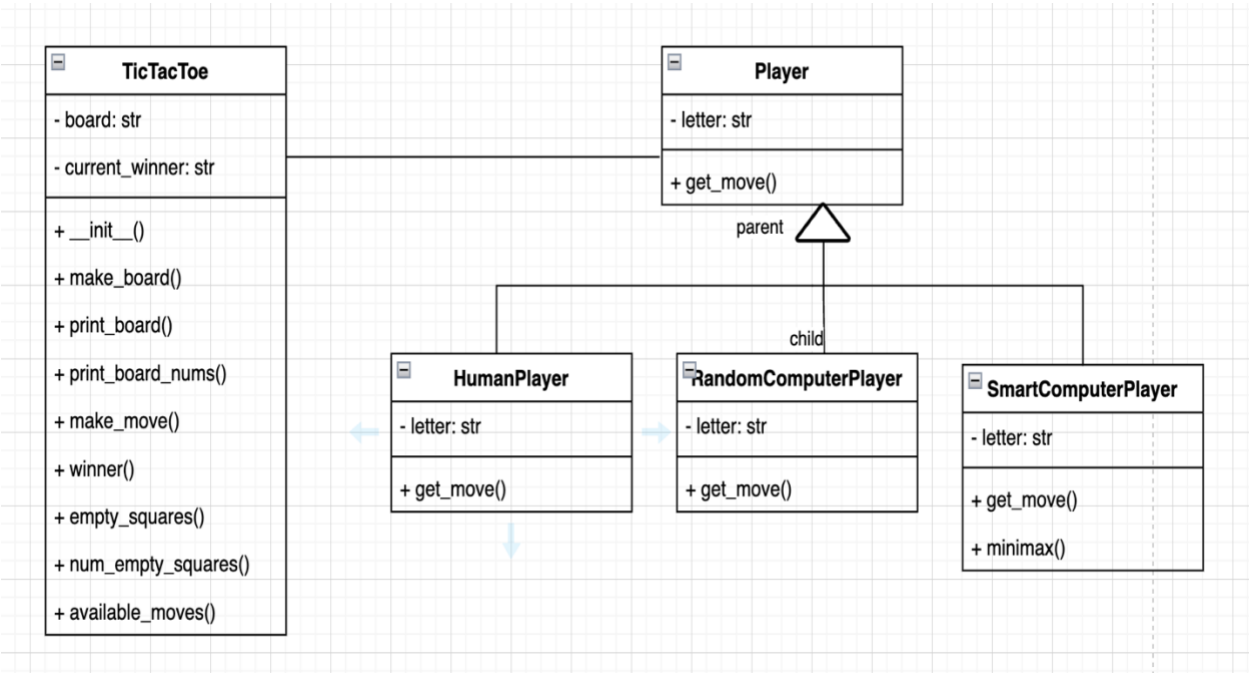


- 2 players as actors, can be human player or robot
- System is the box with TicTacToe up top
- Game will start after pressing play on vscode or typing python3 main.py on the terminal
- After the game start, player 1(X) will move first(can be changed who moves first) and player 2(O) will move after
- After 3 X's or O's have been found either stacked horizontally, vertically or diagonally, game will stop and declare the winner

Activity Diagram



Class Diagram



The TicTacToe class contains the main game logic, including the game board, the logic to make moves, check for a winner, and check for a tie. The class also has a method to print the game board and a method to print the board's numbers.

The Player class is an abstract class with a single method “get_move()” that is implemented by subclasses: HumanPlayer, RandomComputerPlayer, and SmartComputerPlayer.

HumanPlayer is a subclass of Player and it allows a human player to make a move by prompting the user to input a square number.

RandomComputerPlayer is a subclass of Player and it allows the computer player to make a move by randomly selecting a square.

SmartComputerPlayer is a subclass of Player and it uses the Minimax algorithm to make a move, this algorithm makes the computer player to make an optimal move, it simulates the game to the end and chooses the move that results in the best outcome for the computer player.

Modules

Import math – mainly used for the impossible mode to work, also to check where we are on the 3x3 grid on the current turn

Import time – (put in a 0.8s time delay to make sure there's a little bit of time between the press of the key and output on the screen, without this the player's output will pop out too quickly and the game might progress too fast to process what the last turn was resulting in confusion) I used "time.sleep()" function to pause the game for a short period of time between turns to make the game more realistic.

Import random – (generate random moves for the computer player/robot)

Essential Algorithms

Detecting if 3 X's or O's are stacked in a vertical, diagonal, or horizontal manner, if it is then the player(either X or O) will be declared the winner. And if there's no 3 straight X's or O's stacked until all boxes in the 3x3 grid are filled then it will print out a tie

The Minimax algorithm, which is used by the SmartComputerPlayer class to make an optimal move. The minimax algorithm is a decision-making algorithm that is used in game theory and artificial intelligence. It can be used to determine the best move in a two-player game such as Tic-Tac-Toe, where one player is trying to maximize their score and the other player is trying to minimize it.

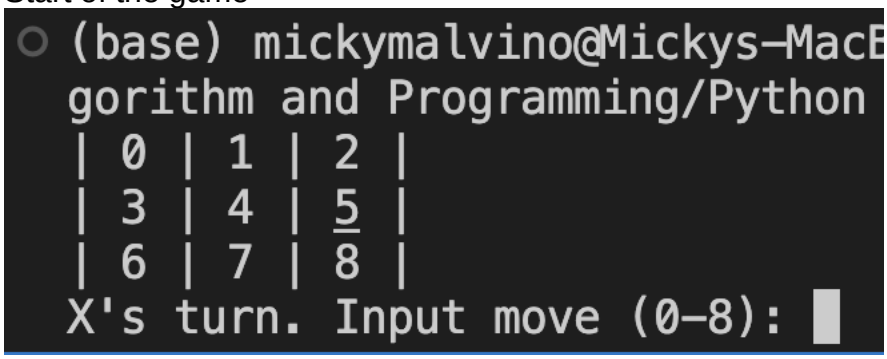
The algorithm simulates the game by recursively exploring all possible future moves and outcomes. It starts by assuming that the current player is the maximizing player and chooses the move that leads to the highest score. Then it assumes that the next player is the minimizing player and chooses the move that leads to the lowest score. It repeats this process until the end of the game is reached.

In the code, the SmartComputerPlayer's "minimax()" method is used to simulate the game by recursively exploring all possible future moves and outcomes, it considers the current state of the game and the player's turn to decide which move will lead to the best outcome. It uses the "state.current_winner", "state.empty_squares()", "state.num_empty_squares()", and "state.available_moves()" to check the current state of the game, then it returns the optimal next move by choosing the move that leads to the best outcome for the current player.

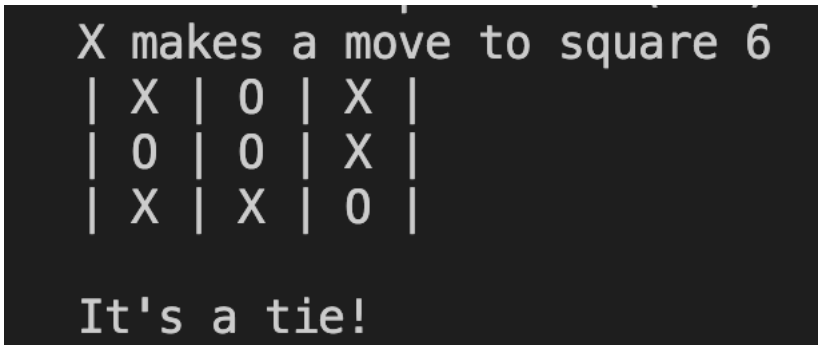
In summary, the Minimax algorithm is used in this code to make the computer player to make an optimal move by simulating the game to the end and choosing the move that results in the best outcome for the computer player.

Screenshot of the application

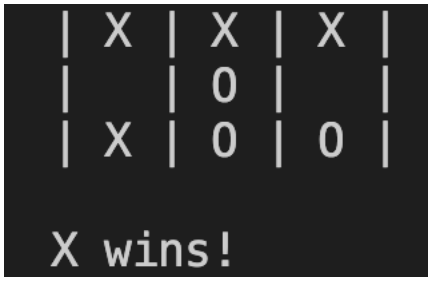
Start of the game



When it's a tie



When X wins



When O wins

```
O makes a move to square 5
| X | O | X |
| O | O | O |
| X |   | X |

O wins!
```

Choosing the player
Playing against robot

```
#pick the type of player you want to play against and which turn you'll go in
if __name__ == '__main__':
    x_player = HumanPlayer('X')
    o_player = RandomComputerPlayer('O')
    t = TicTacToe()
    play(t, x_player, o_player, print_game=True)
```

Playing 1v1

```
x_player = HumanPlayer('X')
o_player = HumanPlayer('O')
```

Playing on impossible mode

```
x_player = HumanPlayer('X')
o_player = SmartComputerPlayer('O')
```

Lesson learned/Reflection
Something I learned about is time management, and to spend time working on projects earlier so I wouldn't have to undergo sleepless nights closer to the deadline trying to understand the code. Unfortunately, I decided to spend my holidays on another project and spent the rest of my free time relaxing instead of working on this one. From the code perspective, I learned that it is very important to comment on blocks of code to make sure not only the makers, but other people who read the code can understand just as well and continue developing the code further. Another thing is the importance of debugging, while lots of source codes are available online, not many of them are working, and when you do find one that's nice it's not gonna be 100% accurate with what you're looking for. Instead, there's still lots of bugs so you still have to change key points to make the program run to your liking.

From the code there's a lot to get like basic input and output operations in Python, Understanding basic programming concepts such as loops, conditionals, tuples and function calls. use conditional statements to check for winning and losing conditions.

Code aside, from this I learned the game logic behind TicTacToe and how to be unbeatable by learning the movement pattern of the AI, the key is the middle box, whichever tile we pick on the first turn, the robot will pick the middle tile and go from there. I thought this was pretty interesting. On offense, the best move for the first player is to pick one of the 4 most corners, if the defense places in the middle, it's a tough spot, but if the 2nd player places anywhere else you can win the game. Then you need to create a trap by placing another X in another corner to create two different ways in which you can win, then go on from there.

Overall, it's good that this course has a final project I can work on, it has been a journey learning a lot of things in python which is implemented in the project. In short, the project is a Tic-Tac-Toe game implementation that allows the user to play against the computer with three different types of computer players: HumanPlayer(1v1), RandomComputerPlayer(player vs robot), and SmartComputerPlayer(impossible mode).