Game 1

Tic-Tac-Toe

Tic-Tac-Toe ~ Introduction

- Tic-Tac-Toe, also called Noughts and Crosses, is a classic game which consists of two players and a 3x3 grid. One player is X and the other is O.
- The players take turns placing their respective symbol onto an empty cell in the grid.
- The objective of the game is to connect a straight line of three of the same consecutive symbols.
- If the grid is full and no player has won the game, it is a draw.

Tic-Tac-Toe ~ AI Explanation (1)

- AI Type: Heuristic
  - Technique designed to solve a problem faster than using classical methods to compute.
  - Examples:
    - Choose the most constrained move (least legal moves available).
    - Do the most difficult thing first.
    - Try the most promising thing first (estimate).
      - Used in Greedy Best First Search.
- This type of AI technique helps the user to play against an opponent which makes useful decisions, rather than random ones, thus making it feel like the opponent is a real person which is making intelligent moves.

Tic-Tac-Toe ~ AI Explanation (2)

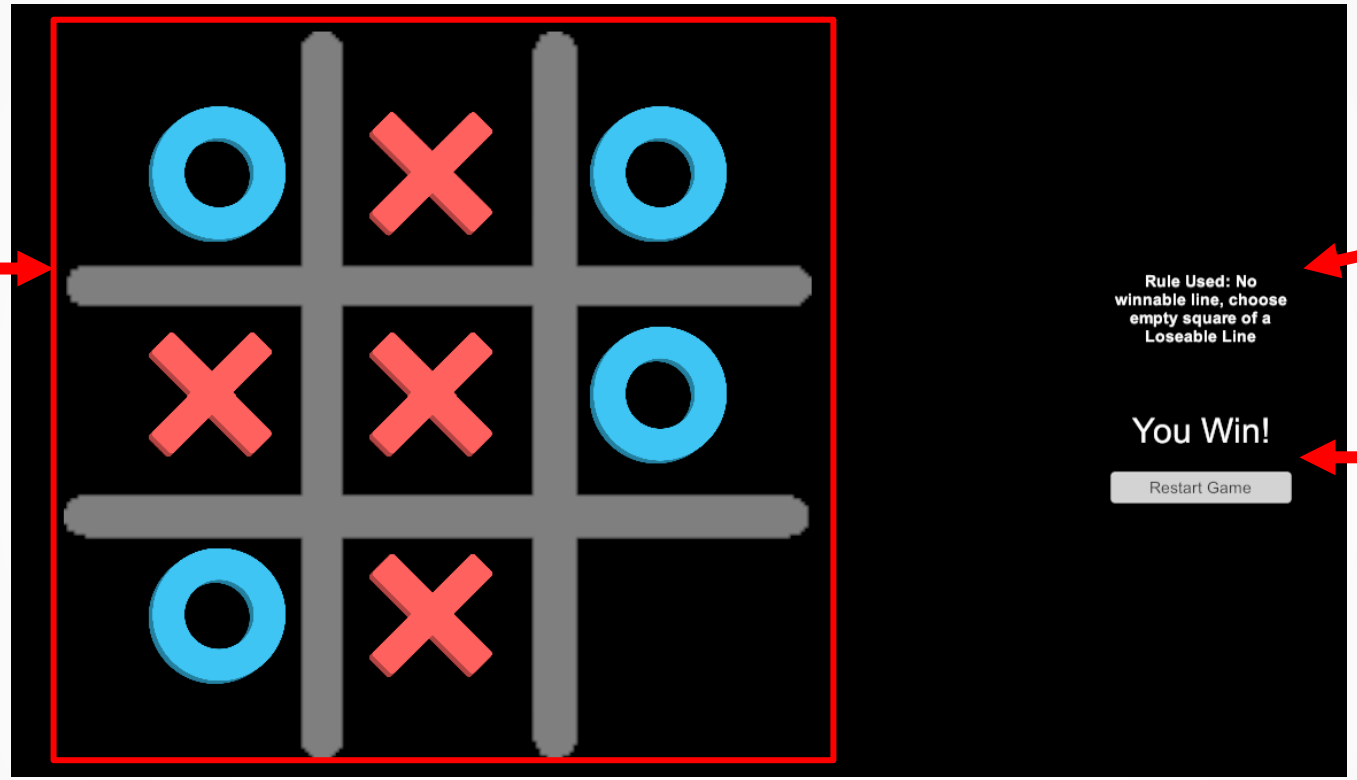
- Pseudocode
  - Move 1:
    - Go in a random corner
  - Move 2:
    - Go to cardinally opposite corner if available (horizontal or vertical).
  - Move 3:
    - If there is a winning condition (2 Os and one empty cell), choose the empty cell.
    - Else if there is a losing condition (2 Xs and one empty cell), choose the empty cell.
    - Else go in a free corner.

Tic-Tac-Toe ~ AI Explanation (3)

- Pseudocode
  - Move 4:
    - If there is a winning condition (2 Os and one empty cell), choose the empty cell.
    - Else if there is a losing condition (2 Xs and one empty cell), choose the empty cell.
    - Else go in a free corner.
  - Move 5:
    - Go to remaining free cell.

Tic-Tac-Toe ~ Mini-Game Implementation (1)

Playable Area:



3x3 Grid

Information about rule used

Game State and Restart Button

Tic-Tac-Toe ~ Mini-Game Implementation (2)

- Implementation of the Mini Game was inspired from [1,2], and sprites used to create the game were used from [3].
- PlayerTurn Script – Handles player input.
  - Variables
    - gameFinished – If true, game is finished (win, loss or draw), if false, game is ongoing.
    - type – If 0, cell is empty, if 1, cell is X, if 2, cell is O.
  - Functions
    - SetSquareType() – Set sprite's image and type for a player's move.

Tic-Tac-Toe ~ Mini-Game Implementation (3)

- GameScript Script – Handles the AI as well as checking the game state.
  - Variables
    - aiMoveCount – Counts the number of moves made by the AI.
    - aiFirstMove – Holds the position of the first cell played by the AI.

Tic-Tac-Toe ~ Mini-Game Implementation (4)

- GameScript Script - Handles the AI as well as checking the game state.
  - Functions
    - AITurn() – Decide which cell to play, set its type and sprite.
    - isLineWinnable() – Returns the position of the empty cell if a line obeys the winning / losing condition, otherwise it returns a -1.
    - checkWin() – Checks if a winning / losing or draw condition has been met.
    - decideWinner() – Sets the game state according to who the winner is.
    - endGame() – Sets the gameFinished variable for each cell's PlayerTurn script to true.

Tic-Tac-Toe ~ Mini-Game Implementation (5)

- RestartGame Script – Handles restarting the scene.
  - Functions
    - Restart() – Restarts the current scene.

Tic-Tac-Toe ~ Exercise (1)

- Replace the dummy program with the functionality mentioned earlier.
- Solution guide provided under the AITurn() function.

```
public void AITurn() {
    //Find the first empty square available and play it
    ruleName.text = "Rule Used: Choose first square available";
    int square = 0;
    for (int tempNumber = 0; tempNumber < 9; tempNumber++) {
        if (squares[tempNumber].GetComponent<PlayerTurn>().type == 0) {
            square = tempNumber;
            break;
        }
    }
    //Setting the square's image
    squares[square].GetComponent<PlayerTurn>().type = 2;
    squares[square].GetComponent<PlayerTurn>().spriteRenderer.sprite = images[1];
    //Call function to check if the game has ended
    checkWin();
}
```

Tic-Tac-Toe ~ Exercise (2)

Now it's your turn to Code ! – Let's implement a Tic-Tac-Toe heuristic ☺

Open the GameScript script and implement the following code in the AITurn() method

1. If aiMoveCount is 0, generate a random number between 1 and 4, depending on the number generated, set the square at that location to type 2 and set its sprite to images[1]. Also set aiFirstMove to the position of the square. For instance, position 8 is the bottom right corner and position 0 is the top left corner.
2. If aiMoveCount is 1, check if the vertically opposite corner of aiFirstMove is not of type 1, if it is not, set its type to 2 and sprite to images[1]. Otherwise, set the horizontally opposite corner of aiFirstMove to type 2 and its sprite to images[1].

Tic-Tac-Toe ~ Exercise (3)

3. If aiMoveCount is 2 or 3, for each possible line format (there are eight in total), check if the line is not winnable for the ai. To do this, call the function isLineWinnable(), with the first three parameters being the positions of the squares, and the fourth boolean variable being true for the ai's case, and false for the player's case. For instance, to check for the top line we would write something like:

   if (isLineWinnable(0, 1, 2, true) != -1)

   where -1 means that the line is not winnable. In the if statement, set a temporary variable to the return of the function call, and set the type of the position to 2 and its sprite to images[1]. After this, we check if a line is losable, to do this we just pass false instead of true to the isLineWinnable() function. Finally, in the else statement, we just check each of the corners, and if there is a free space, meaning its type is 0, we set its type to 2 and its sprite to images[1].

Tic-Tac-Toe ~ Exercise (4)

4. If aiMoveCount is 4, loop through each square, and if its type is 0, set its type to 2 and its sprite to images[1].
5. After all the conditional statements, increment aiMoveCount by one so that it is increased after a move is played. Also, call the checkWin() function, to check if the game has concluded or not.

Tic-Tac-Toe ~ Conclusion

- Heuristics in games similar to Tic-Tac-Toe are relatively straight-forward and easy to implement. It is also important to decide the appropriate heuristic to use in the code beforehand.
- The disadvantage of some heuristics is that they may require repetition in lines of code and constant conditional statement checks. Heuristics may not necessarily provide the best possible move either.

Tic-Tac-Toe ~ References

[1] - “Java Graphics tutorial,” *Tic-tac-toe AI - Java Game Programming Case Study*. [Online]. Available: https://www3.ntu.edu.sg/home/ehchua/programming/java/javagame_tictactoe_ai.html [Accessed: 26-Mar-2023].

[2] - “Special blog- tic tac toe game in unity.,” *Special Blog- Tic Tac Toe Game in Unity*. [Online]. Available: https://unfragilecoding.blogspot.com/2018/08/special-blog-creating-tick-tack-toe-game.html [Accessed: 26-Mar-2023].

[3] - "iOS Tic Tac Toe with an Unbeatable AI," *PNGKey*. [Online]. Available: https://www.pngkey.com/detail/u2w7w7i1u2i1t4w7_ios-tic-tac-toe-with-an-unbeatable-ai/ [Accessed: May 18, 2023].