TicTacToe Functional Specification

Grace Jiang

Introduction and Overview:

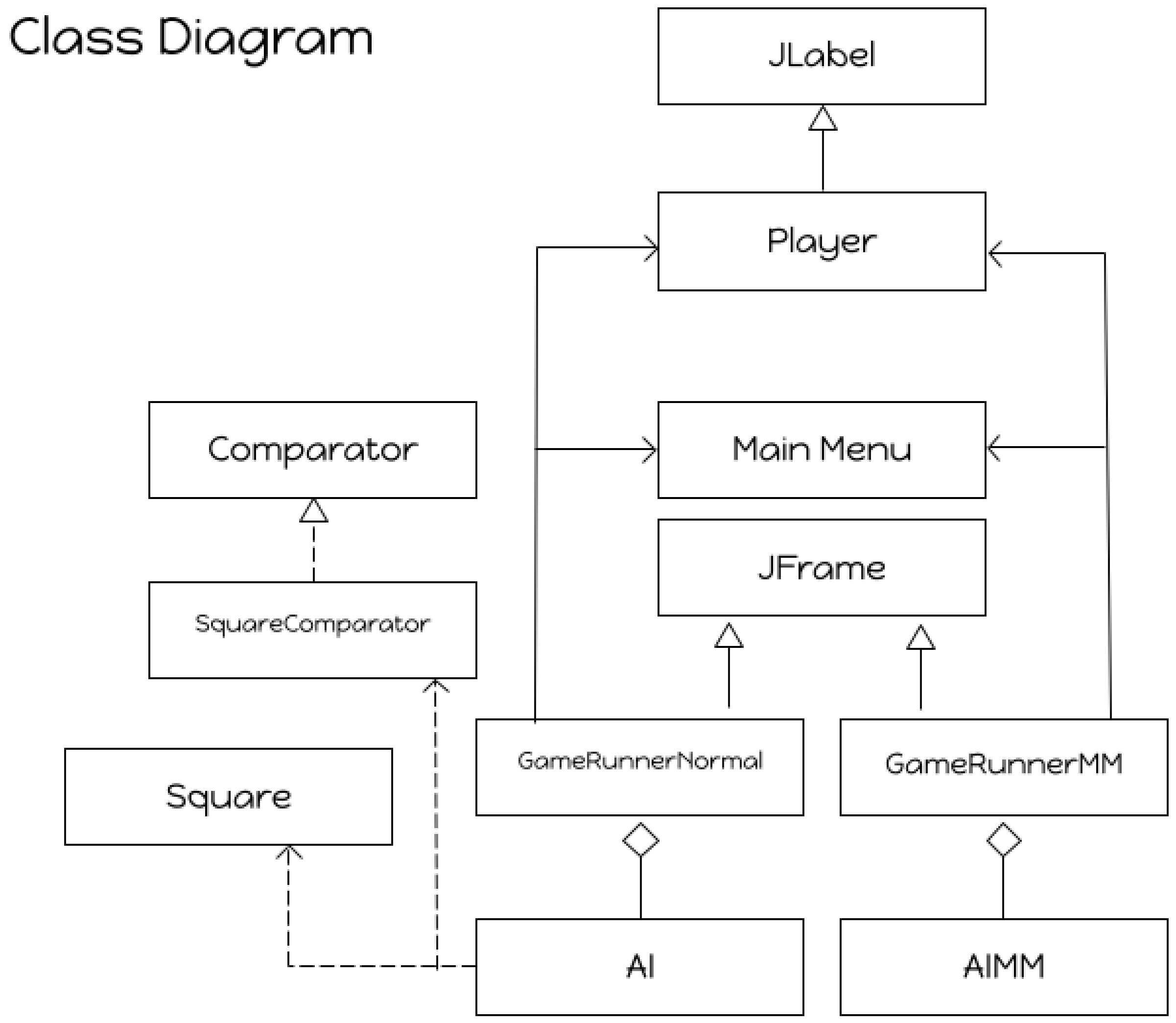
This TicTacToe program starts from a main menu with two buttons, a normal level and unbeatable level button. They each lead to a new screen with different games. The human player is “X” and the computer is “O.” The human player chooses a spot and then the computer chooses a spot. This goes on back and forth until someone wins or there is a tie. Keep in mind that the normal level is not perfect and does not cover all possible situations, so it lets the human win sometimes. On the contrary, the unbeatable level is unbeatable, hence the name. To make it seem like the computer “thinks” after the human chooses a spot, a thread is created in the constructor at the start of the game that constantly checks if the human just chose a spot. If so, the boolean *aiIsThinking* is toggled and the thread sleeps for one second before returning back to normal gameplay.

Structural Design:

Type	Purpose
Priority Queue	The normal level AI uses this to choose moves when other strategies don’t complete. Each square on the gameboard has a “priority number” according to how many winning combinations it’s a part of. For example, the middle square (5th), is part of 4 winning combinations so its priority number is 4; the first (1th) is part of 3 combinations so its priority number is 3. In the

	priority queue, an ascending comparator is used, so higher priority numbers are favored.
2D int array with recursion	The int[][] board represents the constantly-updating current state of the unbeatable game level. It is present in the GameRunnerMM.

Object-Oriented Design



- **Player** extends JLabel - a JLabel that either displays “X” or “O,” depending on who claims the square. There are nine of these on the board in a GridLayout. They each have a black border, giving the illusion of grid lines.
- **GameRunnerNormal** extends JFrame - this makes a JFrame and fills it with Player JLabels. It takes care of mouse input to set human-chosen squares to “X.” Essentially it takes care of the rules: no one can choose an occupied square, the player cannot choose while the computer is “thinking,” etc.
- **AI** - this is the AI for the normal level. It uses a bunch of if statements to choose a square according to these rules:
    1. Check the board for possible computer wins. If there exists one, take it and win.
    2. Check the board to block possible human wins. If there exists one, take it and prevent the human from winning.
    3. Check for “forks.” For example, if the computer is “O:”
       ```
       [   ][O][ 3]
       [   ][X][O]
       [   ][X][   ]
       ```
       Choose square 3 because the “O” squares would form a “fork.”
    4. If none of the above executes, a move is chosen from the priority queue.

- **GameRunnerMM** extends JFrame - essentially the same as GameRunnerNormal -- it runs the unbeatable level of the game
- **AIMM** - recursively creates a tree of all possible game outcomes and chooses the best move, assuming the human also plays optimally
- **Square** - represents one square on the game board
- **SquareComparator** - provides comparator for the Priority Queue that AI (sometimes) uses to choose move.

- **MainMenu** - main method is here. It is the main menu page with its two buttons to choose game level.