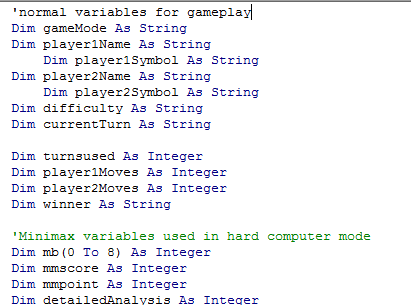
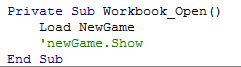
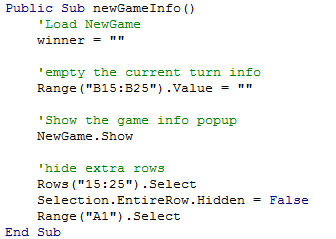
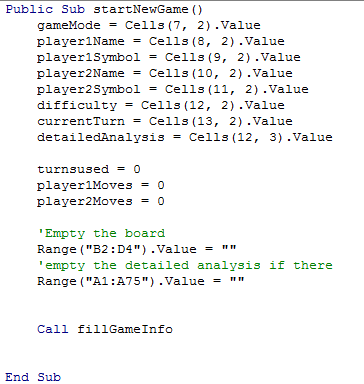
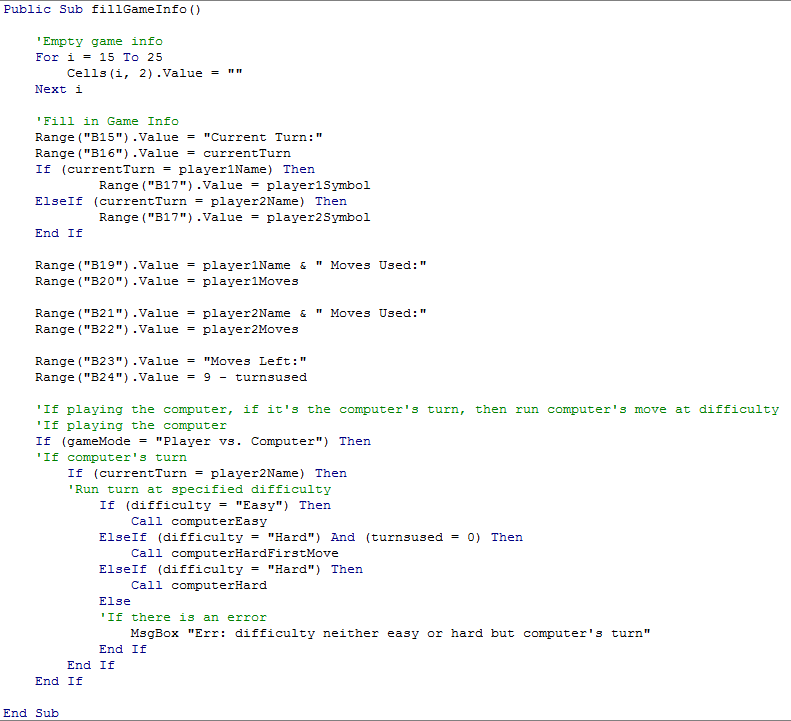
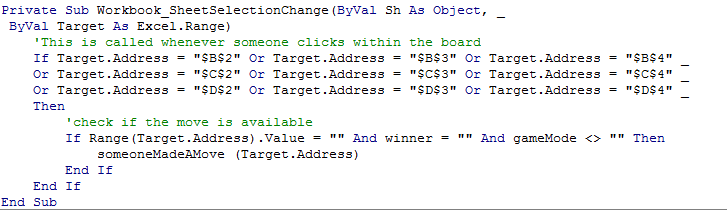
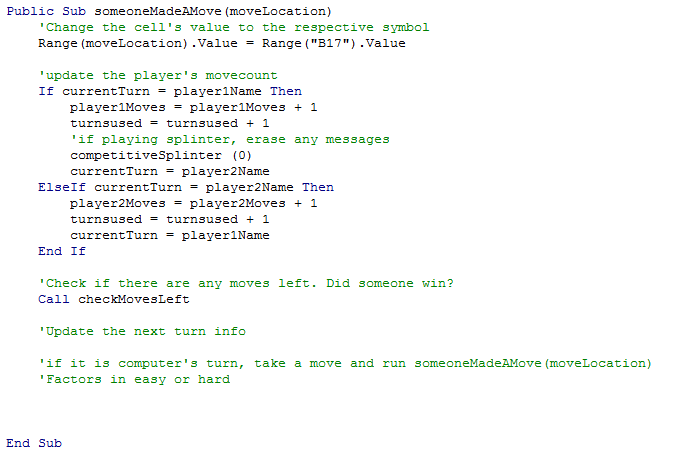
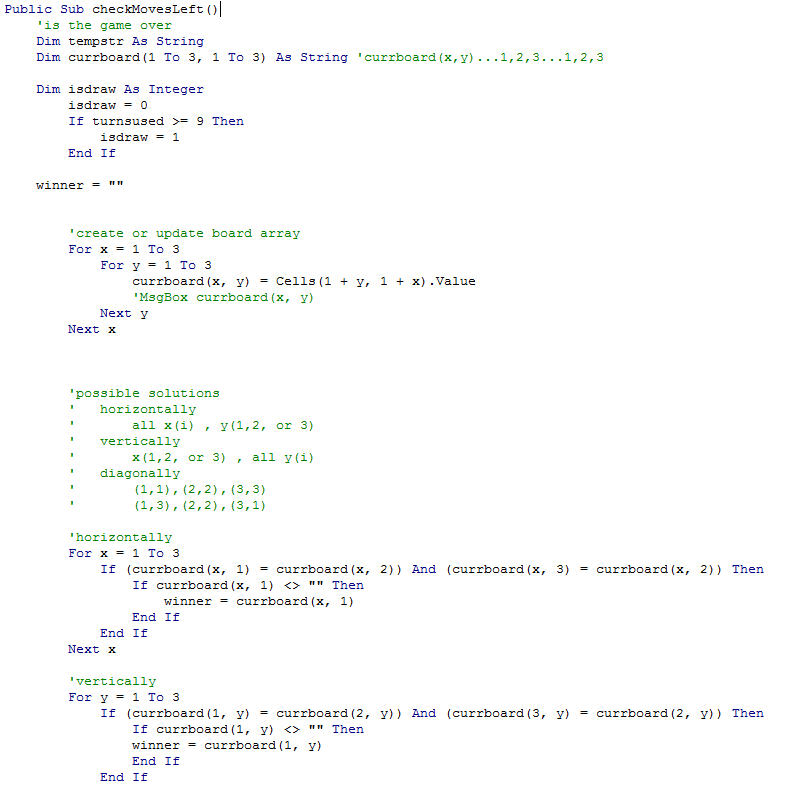
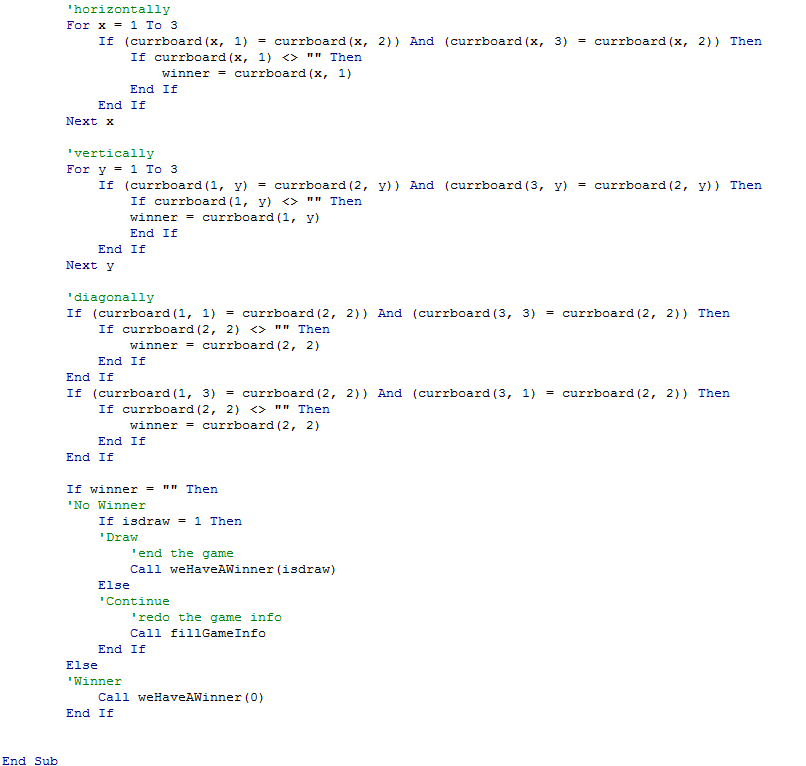
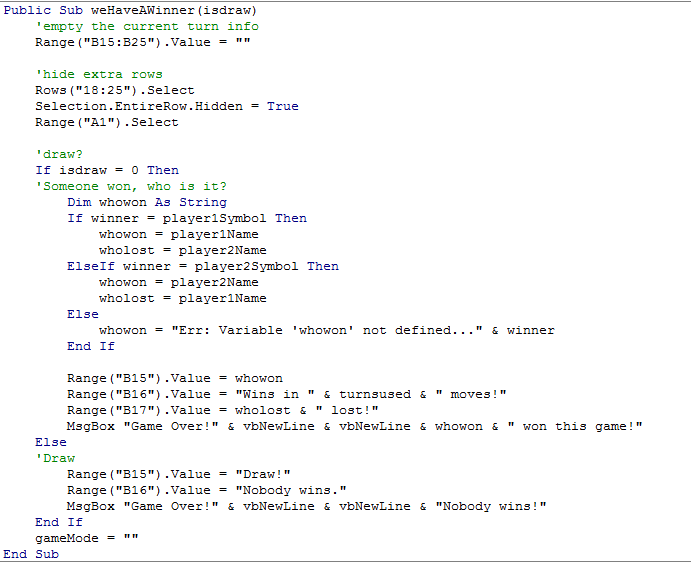
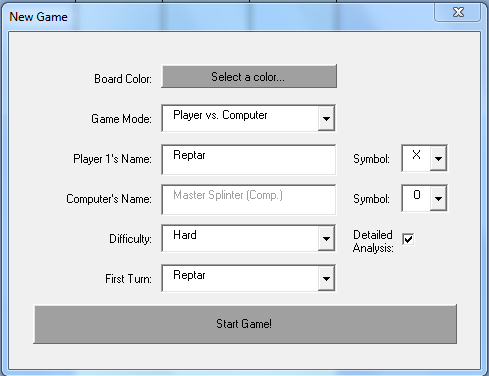
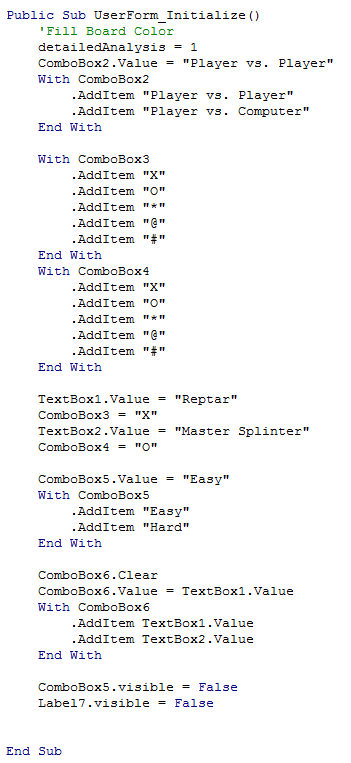
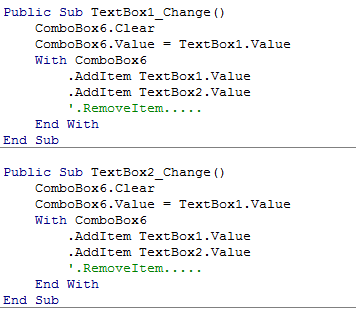
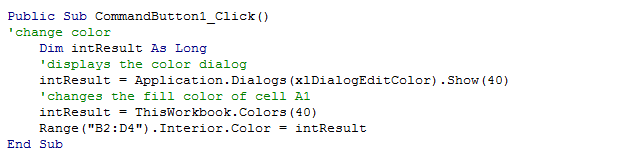
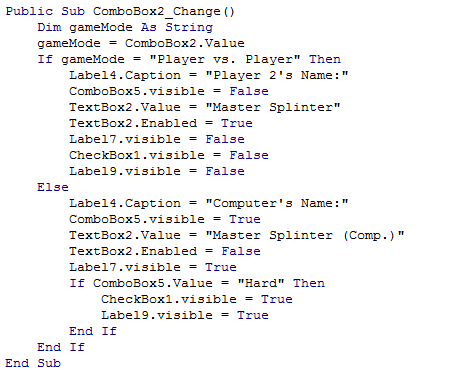
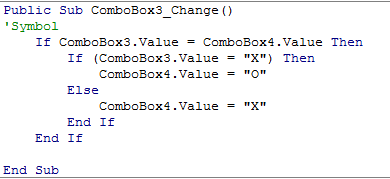
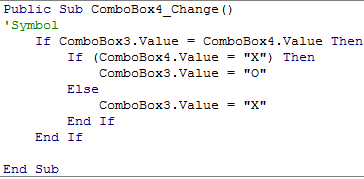
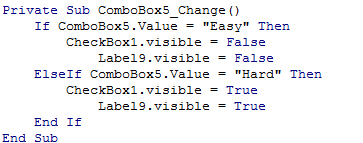
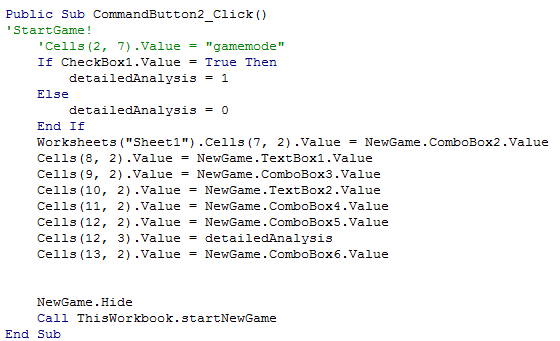
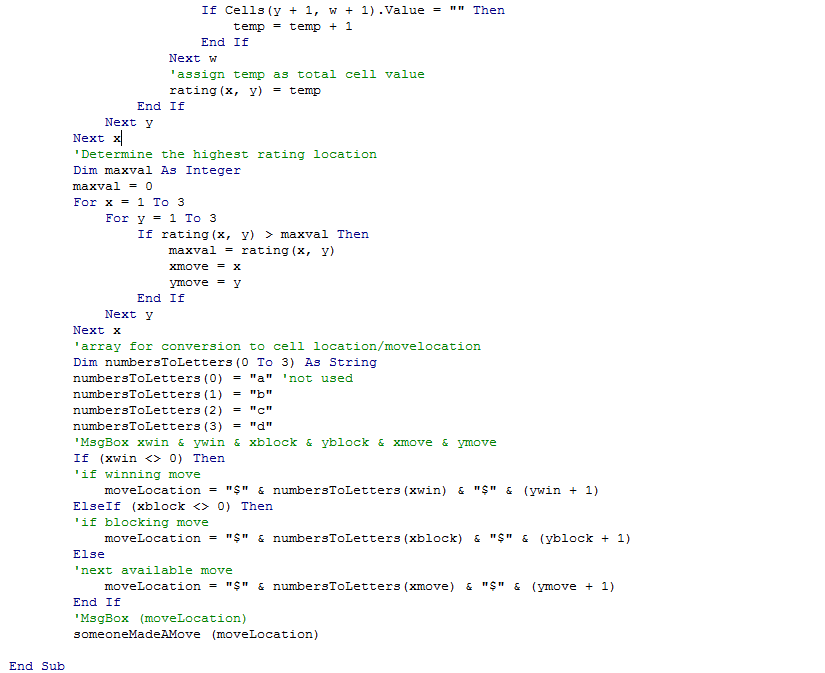
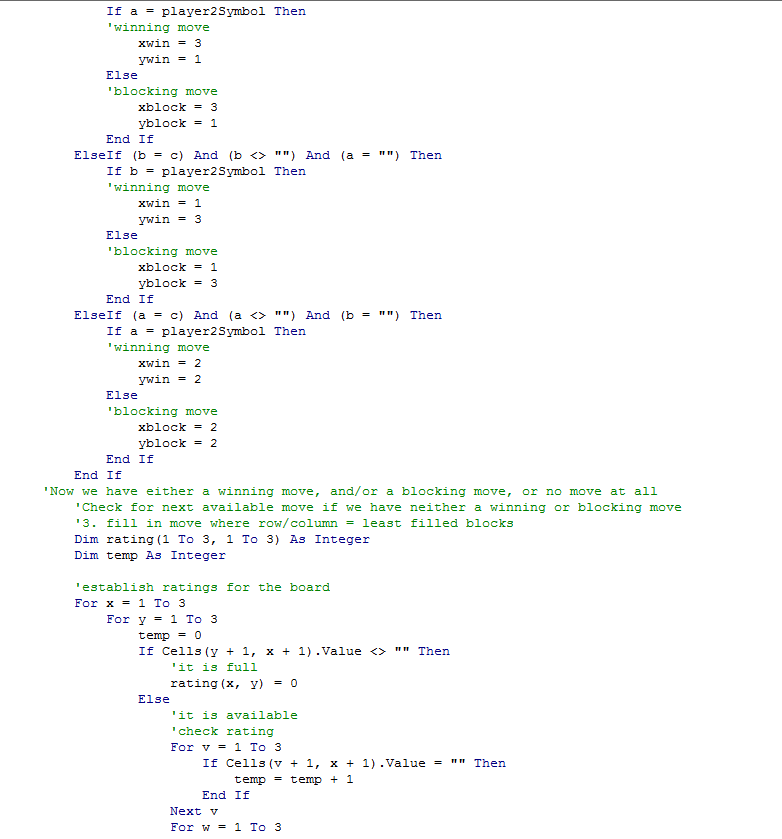
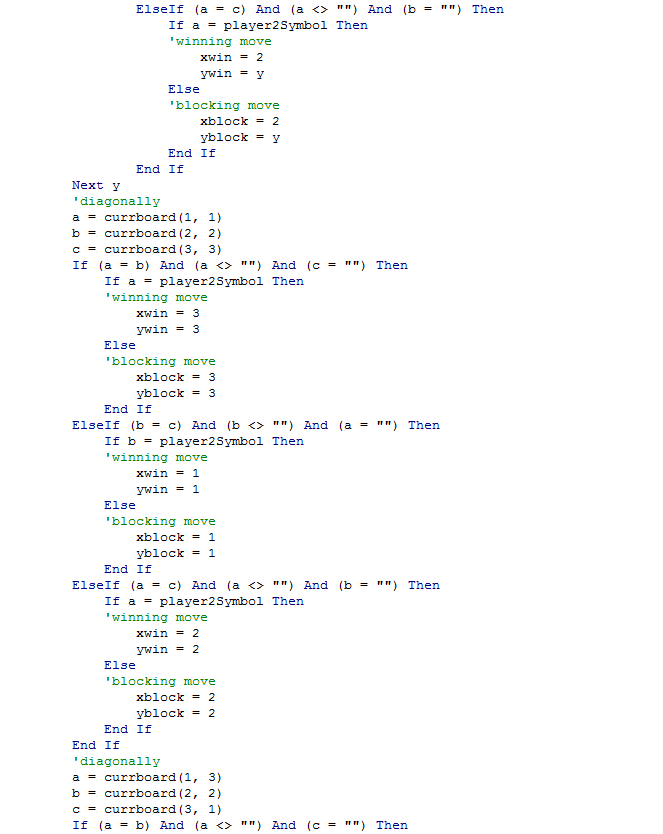
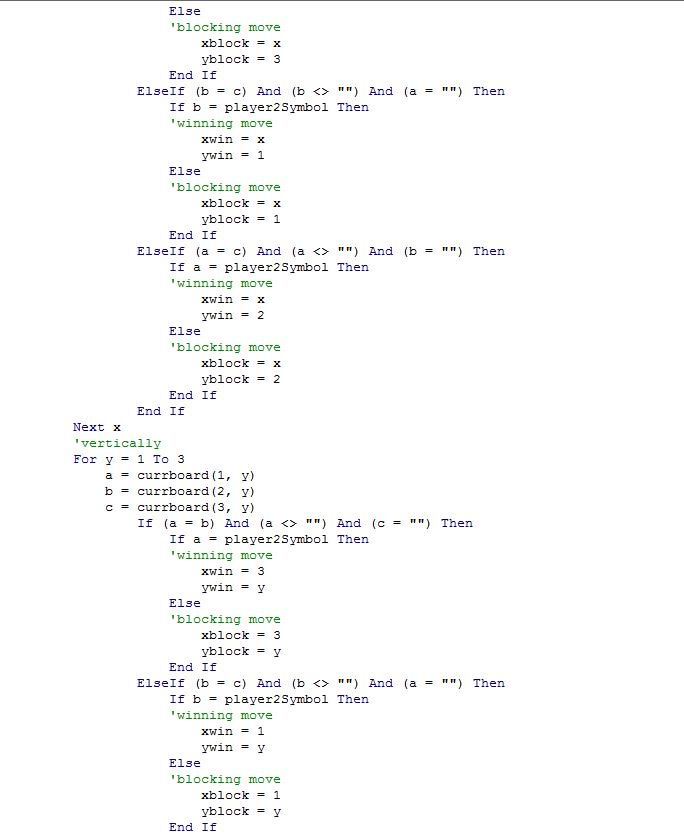
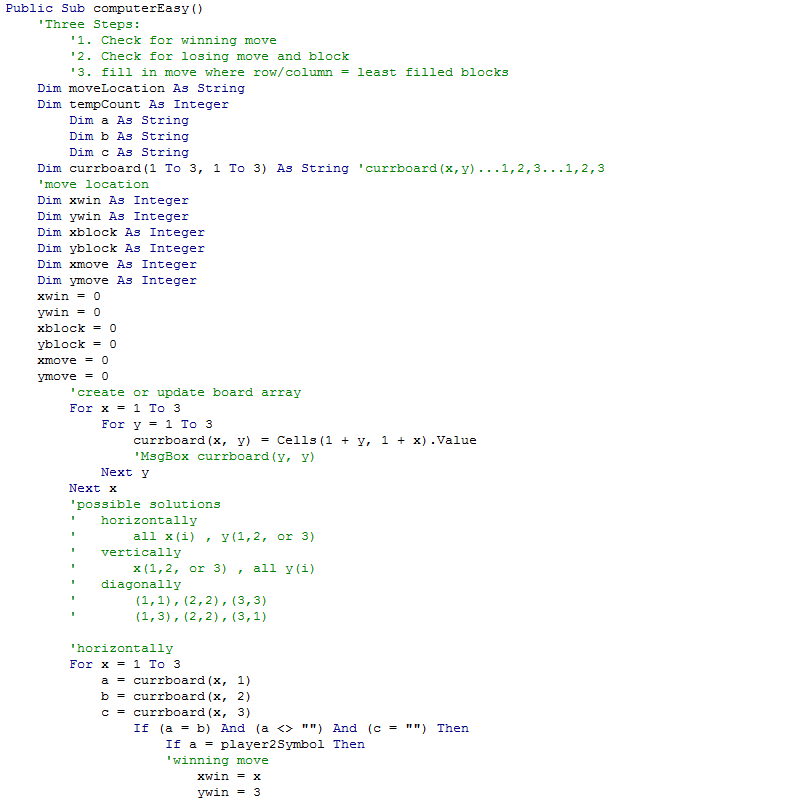
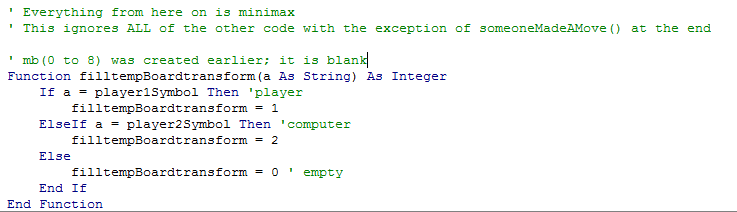
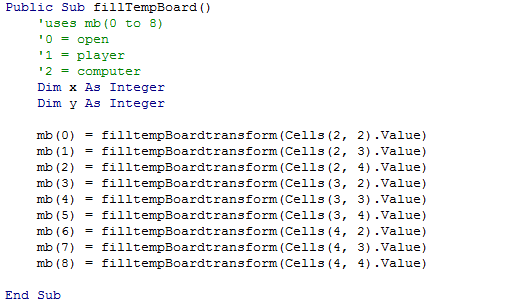
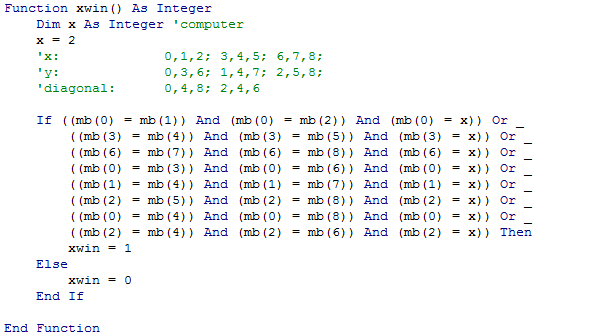
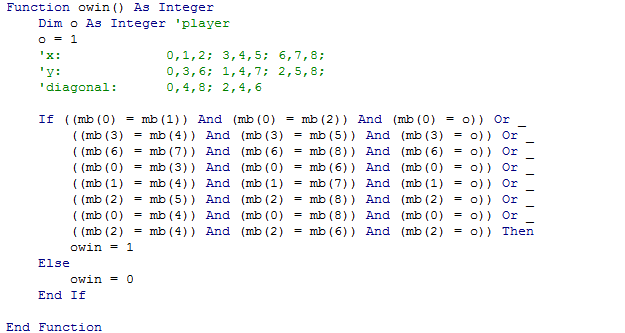
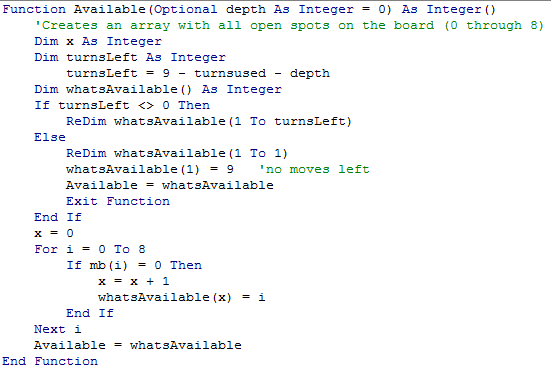
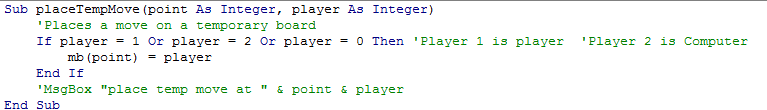
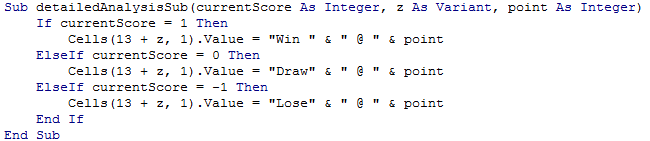
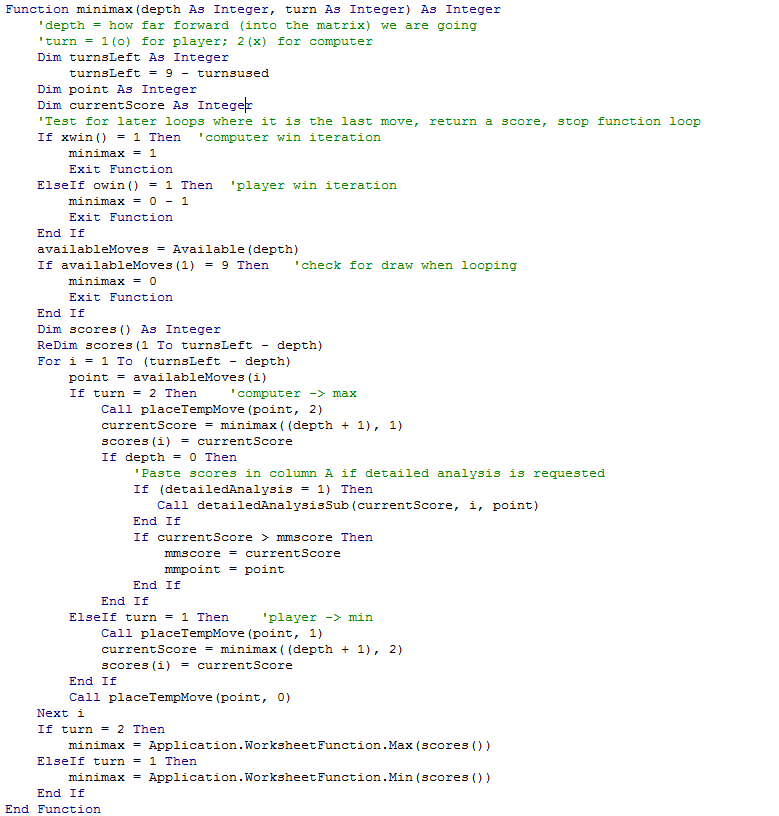
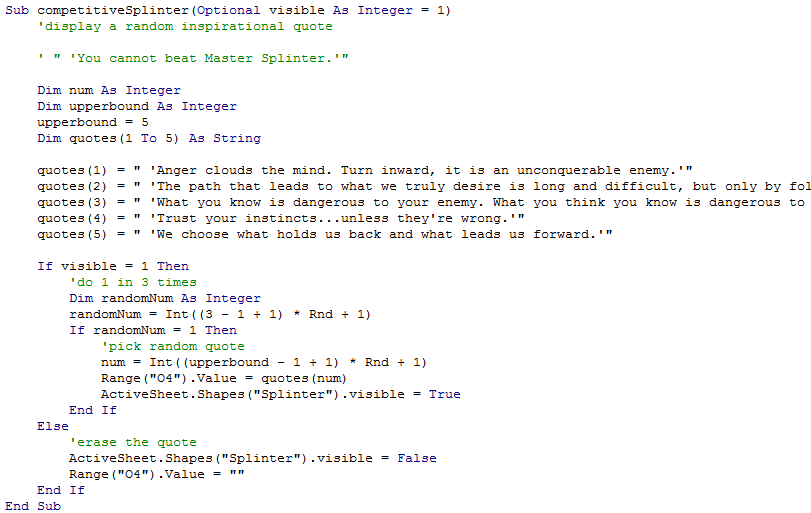
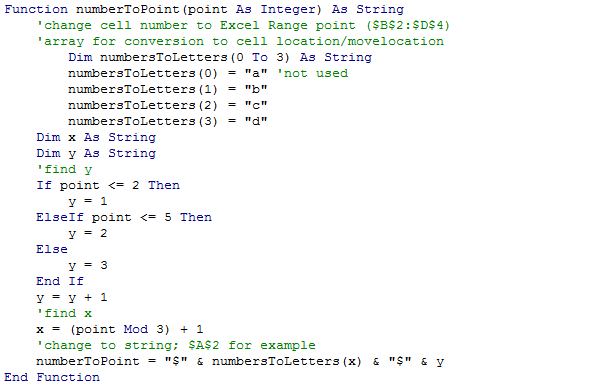
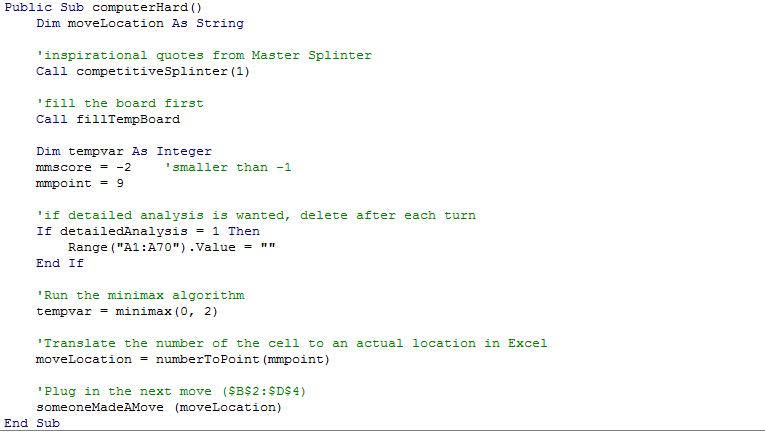
Tic-Tac-Toe Technical Report Team Awesome

Revision 1.0 – 12/1/2015

1. The code is broken down into four sections: the basic gameplay and rules of the game, the form that asks for user input, the easy mode against a computer, and the hard mode against a computer. The code that runs the game is outlined and commented below:
   1. Basic Gameplay and Rules
      1. We begin by setting the variables that are for use within the entire program and not in a specific subroutine. Minimax (hard mode) variables are set below separate from others.
         1. 
      2. We begin by loading a form when the user opens the program.
         1. 
      3. The next section focuses on starting a new game. It takes the information from the form and inputs it into the spreadsheet in hidden and visible cells.
         1. 
      4. Then we create a new game. This sets all of the variables that were defined in the beginning of the program. Each new game resets these variables.
         1. 
      5. We call this sub many times throughout the game. Each time there is a move played, this is called to fill in the new information to keep the players updated. Information such as the number of moves used and moves left are updated and filled in here.
         1. 
      6. This subroutine is used to determine whether a move is being made by the player. If a cell is clicked on within the range, it will run the subroutine someoneMadeAMove() with the address of the cell in it. It also checks to make sure that the given move is legal and still available.
         1. 
      7. This is the subroutine that is called when someone makes a move. It fills the move in on the board and checks to see if there are any moves left. It also resets many of the variables for the next turn.
         1. 
      8. This long section of code is used to determine if there are any moves available to either player. If it determines that the game is over, it will call a function to display and announce the winner and total moves used. If not, the code will continue with the next move by running fillGameInfo()
         1. 
         2. 
      9. This subroutine announces the winner. It clears the board of information for the next turn and announces the winner or a draw via a message box. It also labels the winner below the board.
         1. 
   2. User Input Form
      1. This is used to determine how the player wants to play. It asks questions such as names, the game move to use, which symbols belong to whom, the difficulty, and who plays first.
         1. 
      2. This code starts the form. When opening, it needs to populate each drop down. It also needs to select defaults for each one. It also needs to hide certain boxes until they are needed, such as the difficulty…we will not need that if the person is playing another person.
         1. 
      3. When names are changed, they need to be updated in the dropdown that allows the player to choose who gets to go first. That way, it does not show old names when they have been changed.
         1. 
      4. This allows the user to select the color of the game board. Of course, I would choose to play with a fuschia background, but maybe you would disagree.
         1. 
      5. This is run upon a change in the game mode. Certain game modes require certain textboxes and options to show. This will hide and show them as needed. This is used when the player decides to play another player or the computer.
         1. 
      6. These two areas of code prevent two people from having the same symbol. If player 2 is X and player 1 chooses X, it will change player 2 to O, for example. Try it out!
         1. 
         2. 
      7. This code is used to show more options depending on the level of difficulty chosen. ComboBox5 is only shown when the player wants to play the computer.
         1. 
      8. This will be called upon submitting the form. When submitted, the data is transferred to the spreadsheet by placing the cells in an area hidden to the user. The code in the spreadsheet can then access that data by grabbing values from the cells. Here, we place those values in the spreadsheet.
         1. 
   3. Easy Mode vs. Computer
      1. This code is used to run a computer’s move on easy. A set of steps was followed. First, it checks if there is a move that it can play that will win the game. If so, it will make the move. Next, it will check if there is a move that needs to be played to prevent a direct win (such as the other player having two X’s in a row and waiting to play the third). Finally, it will check which spot on the board has the least filled spots in the row and column. These were given directly for the directions from our project syllabus. It is a very effective way of playing and can put up quite the fight. It ends the subroutine by playing the move that it deems to be the best option.
         1. 
   4. Hard Mode vs. Computer (Minimax)
      1. We begin the minimax theorem (the hard mode against the computer) here. This small section of code works well with the next to fill in a temporary game board used in the minimax code. This board will be changed multiple times during each turn as the computer determines the best way to win or draw. The computer cannot lose. It converts X’s and O’s to numbers for the computer to read in the board.
         1. 
      2. The board is populated here multiple times as the computer progresses in its determining of the best move to use.
         1. 
      3. This code is used to determine if X wins (the computer for the code’s purposes).
         1. 
      4. This code is used to determine if O wins (the player for the code’s purposes).
         1. 
      5. This code creates an array with a list of numbers relating to open cells. If a cell is open to be played on, it is added to an array that is passed on for the minimax theorem to use. This is called multiple times per move as the computer determines the best move to play.
         1. 
      6. This section is used to place a temporary move as the computer fills in every possible play that is can use. As it fills in every board combination, it will add or subtract the move from the board. Here it adds it.
         1. 
      7. This line of code is for the player to see what the computer is thinking. It will establish every possible move that it can play and whether it can win, draw, or lose at that point. With the Detailed Analysis enabled during a game, the player can watch the computer thinking.
         1. 
      8. The minimax theorem is the heart of this program. It searches through every possible move to determine how to win. It will use a list of available tiles to and run through each iteration by rerunning the minimax function. Through each run, it places a temporary tile on the board to determine the next move. For example, it will fill the third point on a board, then fill in the fourth with a random user’s play, and then it will place the fifth one. It will remove the fifth one if that is a losing move and it will play the fifth at another spot. It will do this over and over again until it finds a winning move or runs out of spots. If the only possible outcome is a loss, it will mark that as a bad spot to play and move on to the next. It will accept a draw, but only if there is no possibility of a win. If there is a possibility for a win, it will take it. It is impossible to beat the minimax theorem because it will never pick a move that has a chance at losing. After it runs through each new iteration, it removes the piece from the board and begins searching through the next. Each time, it sends its score to the program to determine if it can win. If it can win, it send a 1, if it can lose, it sends a -1, and if a draw is most likely, it sends a 0. Many thousands of iterations can be done in a mere second or two. Once an optimal move is found, it will return the variable to the function that calls it.
         1. 
      9. Master Splinter decided to drop by with some motivational quotes. This code randomly displays master splinter and a random quote of his to motivate the player who honestly cannot win the game.
         1. 
      10. This code is used to convert the numbers that the program was used to seeing into columns and rows for the computer to place the point on the excel table. It returns the point that the game is expected to play. For example, 5 will play at point (3,2).
          1. 
      11. This is the function that puts everything together. It will ask Master Splinter to show up if he wants to. It will fill up a temporary board array. It will set preliminary scores for the optimal moves to play. If will remove any random data in order to run a detailed analysis. It will run the actual minimax algorithm. It will convert a chosen move to a point to play. And finally, it will play that move and wait for the next move.
          1. 
      12. This is required if the computer is expected to play first. It covers a number of bugs that arose if the computer plays first and isn’t ready. It sets a random move so will never play the first move in the same spot. Regardless of the move played, the computer cannot lose. No minimax algorithm is run here.
          1. 