Contributors:

Abdurrahman Mohammad

Eduardo Fragoso

Design Phase

Interface: GameInterface

Responsibilities:

* Interface to implement a game storage for Tic Tac Toe
* Contains the implementation for **static** methods to save and load games from files
* Forces its implementor to implement specific methods essential in running the game

Fields: None  
Methods:

* public void saveMove(char move, int row, int col)
  + Method to save a move to the game storage
* public char readMove(int row, int col)
  + Method to read a move from the game storage
* public static void saveGame(String filename, Game game) throws IOException
  + Method to save game to a file
* public static Game loadGame(String filename) throws FileNotFoundException
  + Method to load a game from an existing file

Class: Game

Responsibilities:

* Stores the Tic Tac Toe game
* Lets the user save a move to the game storage
* Lets the user read a spot on the game storage
* Lets the user compare game storage objects
* Overrides Object class methods

Fields:

* private char[][] game
  + Array of chars to represent and store the Tic Tac Toe game board and keep track of moves

Methods:

* public Game()
  + Creates a new tic tac toe game storage object
* public void saveMove(char move, int row, int col)
  + Method to store a move in the Tic Tac Toe game
* public char readMove(int row, int col)
  + Method to read a move on the game board
* public static Comparator<Game> comparatorByFill()
  + Compare by how filled a board is (compare by fill)
* public static Comparator<Game> comparatorByFirst()
  + Compare by first move
* public static Comparator<Game> comparatorByX()
  + Compare by how many X's are in the board
* public static Comparator<Game> comparatorByO()
  + Compare by how many O's are in the board
* public String toString()
  + Converts a game storage object to a string
* public boolean equals(Object otherObject)
  + Determines if two games are equal
* public int hashCode()
  + Hashes the game

Class: Board

Responsibilities:

* Maintain an active GUI for a tic tac toe game
* Return the button pressed when requested by the user
* Set the text of the buttons based on user’s input
* Prints strings onto the GUI
* Clears the text of all the buttons when requested by the user
* Prints a tic tac toe game onto the GUI buttons

Fields:

* private int click = 0
  + Represents spot (0, 0) on the tic tac toe GUI
* private Game game
  + Stores the last printed game just in case if we need to save it
* private JTextField textField
  + A text field for input/output
* private JButton button1
  + Represents spot (0, 0) on the tic tac toe GUI
* private JButton button2
  + Represents spot (0, 1) on the tic tac toe GUI
* private JButton button3
  + Represents spot (0, 2) on the tic tac toe GUI
* private JButton button4
  + Represents spot (1, 0) on the tic tac toe GUI
* private JButton button5
  + Represents spot (1, 1) on the tic tac toe GUI
* private JButton button6
  + Represents spot (1, 2) on the tic tac toe GUI
* private JButton button7
  + Represents spot (2, 0) on the tic tac toe GUI
* private JButton button8
  + Represents spot (2, 1) on the tic tac toe GUI
* private JButton button9
  + Represents spot (2, 2) on the tic tac toe GUI

Methods:

* public Board()
  + Constructs and initializes the GUI
* public int getButton()
  + Method to get the button clicked by the user
* public void setButton(int num, String input)
  + Sets the text of a button
* public void printString(String output)
  + Prints string onto the GUI
* public String getString()
  + Get a string from the text field
* public void clearBoard()
  + Clear the string of the buttons in the GUI (board)
* public void printGame(Game game)
  + Prints the game onto the GUI (board)
* private void saveGame()
  + Saves the last printed game (last game)

Interface: AI

Responsibilities:

* Forces implementor to implement important methods related to a tic tac toe AI
* Contains the definition of certain static methods

Fields: None  
Methods:

* public Game makeMove(Game game)
  + Method for the AI to make a move in a game
* public static char checkGame(Game game)
  + Method to check if the game is over
* private static char checkDraw(Game game)
  + Helper method for checkGame()
  + Check if the game is a draw (no spots with ' ')
* private static char checkRows(Game game)
  + Helper method for checkGame()
  + Check to see if anyone has 3 in a row and return the winner
* private static char checkColumns(Game game)
  + Helper method for checkGame()
  + Check to see if anyone has 3 in a column and return the winner
* private static char checkDiagonals(Game game)
  + Helper method for checkGame()
  + Check to see if anyone has 3 in a diagonal and return the winner

Class: AIEasy

Responsibilities:

* Construct a Tic Tac Toe AI for Tic Tac Toe with easy difficulty
* Defines the implementation for makeMove() mentioned in the interface AI
  + Makes random moves using random numbers
* Compares 2 AIEasy objects

Fields:

* private final char piece
  + The AI's playing piece ('X or 'O')

Methods:

* public AIEasy(char piece)
  + Constructs a Tic Tac Toe AI for Tic Tac Toe with easy difficulty
* public Game makeMove(Game game)
  + This method makes the AI select its move on the game board
* public char getPiece()
  + Getter for the AI's playing piece
* public static Comparator<AIEasy> comparatorByPiece()
  + Compare multiple AI by playing piece

Class: AIMedium

Responsibilities:

* Construct a Tic Tac Toe AI for Tic Tac Toe with medium difficulty
* Defines the implementation for makeMove() mentioned in the interface AI
  + Makes somewhat smart moves: first it tries to win the game, then it tries to prevent itself from losing, then it does a random move
* Compares 2 AIMedium objects

Fields:

* private final char opponentPiece
  + Keeps track of the opponent's piece. Piece is set as final.
* private boolean isMove
  + Keeps track if move has been made. It is a global variable used to make a move.

Methods:

* public AIMedium(char piece)
  + Constructs a Tic Tac Toe AI for Tic Tac Toe with medium difficulty
* public Game makeMove(Game game)
  + This method makes the AI select its move on the game board
* private Game analyzeRows(Game game, char piece)
  + Helper method for makeMove()
  + Analyzes the rows and attains a win by rows or prevents a loss by rows based on the piece passed in
* private Game analyzeCols(Game game, char piece)
  + Helper method for makeMove()
  + Analyzes the columns and attains a win by rows or prevents a loss by columns based on the piece passed in.
* private Game analyzeDiagonal1(Game game, char piece)
  + Helper method for makeMove()
  + Analyzes the diagonal from top left to bottom right and attains a win by this diagonal or prevents a loss by this diagonal based on the piece passed in.
* private Game analyzeDiagonal2(Game game, char piece)
  + Helper method for makeMove()
  + Analyzes the diagonal from bottom left to top right and attains a win by this diagonal or prevents a loss by this diagonal based on the piece passed in.
* public static Comparator<AIMedium> comparatorByOpponentPiece()
  + Compare multiple AI by opponent's piece

Class: AIHard

Responsibilities:

* Construct a Tic Tac Toe AI for Tic Tac Toe with easy difficulty
* Defines the implementation for makeMove() mentioned in the interface AI
  + Makes moves based on strategies: https://www.wikihow.com/Win-at-Tic-Tac-Toe

Fields: None  
Methods:

* public AIHard(char piece)
  + Constructs a Tic Tac Toe AI for Tic Tac Toe with hard difficulty
* public Game makeMove(Game game)
  + This method makes the AI select its move on the game board
* private Game moveX(Game game, int count)
  + Helper method for makeMove()
  + Tries to win or draw the game if the AI goes first
* private Game moveO(Game game, int count)
  + Helper method for makeMove()
  + Tries to win or draw the game if the AI goes second
* private Game takeCorner(Game game)
  + Helper method for makeMove()
  + Lets the AI select the corners and marks them in the game board storage

Class: TicTacToe

Responsibilities:

* Combines all the components to make a tic tac toe game
* Lets the user pick the mode of the game and starts it
* Lets the user load a game then play it
* Lets the user save the game
* Lets the user select who goes first
* Sets up the AI
* Manages the GUI

Fields:

* Board board
  + Make a GUI for the Tic Tac Toe game

Methods:

* public void initiate()
  + Method to initiate the game
* private void saveGame(Game game)
  + Helper method for initiate()
  + Method to save a game
* private char playFirst()
  + Helper method for initiate()
  + Lets user pick who starts the game
* private int getGameSelection()
  + Helper method for initiate()
  + Method to get the game mode/type: Easy, medium, player vs. player
* private Game playerMove(Game game, char piece)
  + Helper method for initiate()
  + Method for player to make his move
* private void printWinner(Game game)
  + Helper method for initiate()
  + Prints the winner of the game
* private Game playMultiplayer(Game game)
  + Helper method for initiate()
  + Method to play Tic Tac Toe with another player
* private AIEasy AIsetup(int AIselection)
  + Helper method for initiate()
  + Determines which AI to use and returns it
* private Game playAI(Game game, AIEasy AIobject)
  + Helper method for initiate()
  + User plays with the chosen AI

Class Relationships

* Board uses Game
* AI uses Game
* AIEasy uses Game
* AIMedium uses Game
* AIHard uses Game
* TicTacToe uses Game
* TicTacToe uses AI
* TicTacToe uses AIEasy
* TicTacToe uses AIMedium
* TicTacToe uses AIHard
* TicTacToe contains Board
* AIEasy implements AI
* AIMedium extends AIEasy
* AIHard extends AIMedium