**Functional Requirements:**

* As a player, I can choose how many players will play the game, so that I can set a certain amount of players to play the game.
* As a player, I will be notified if the amount of players I entered is too small or too high, so that the game will not have an invalid amount of players.
* As a player, I can choose what symbol represents my character, so that each players’ characters are different
* As a player, I will be notified if a symbol I choose to represent my character is already taken, so that I can’t have the same symbol as another player.
* As a player, I can choose how many rows the tic tac toe board will have, so that the board will be my customized row number.
* As a player, I will be notified if the amount of rows I choose is invalid, so that the board will have a valid amount of rows.
* As a player, I can choose how many columns the tic tac toe board will have, so that the board will be my customized column number.
* As a player, I will be notified if the amount of columns I choose is invalid, so that the board will have a valid amount of columns.
* As a player, I can choose how markers in a row will win the game for a player, so that the amount of markers in a row to win will be to my customized number.
* As a player, I will be notified if the amount in a row to win the game I choose is invalid, so that the game will have a valid number to win variable.
* As a player, I can choose to implement a fast or memory efficient implementation of the game, so that the game will either be fast or memory efficient.
* As a player, I will be notified if the implementation I choose is invalid, so that the game will use a correct implementation.
* As a player, I can choose a position to move, in order to continue playing the tic tac toe game.
* As a player, my move is alternated with the other player(s), so that the other player(s) and I each get a turn.
* As a player, I can mark positions with my marker symbol, so that the other player(s) and I can see where we moved.
* As a player, I am not allowed to mark spaces that are already taken, so that each space of the board can only be occupied by a maximum of one player marker.
* As a player, I will be notified if a space is not available if it is occupied or out of bounds, so that I know that a space is not available for me to go to.
* As a player, I will be notified if there is a draw, so that the other player(s) and I can know if the game has ended in a draw.
* As a player, I can place my marker in a row horizontally the amount of times set to produce a winner to win the game, so that I know I can win the game by connecting a certain amount of my markers horizontally.
* As a player, I can place my marker in a row vertically the amount of times set to produce a winner to win the game, so that I know I can win the game by connecting a certain amount of my markers vertically.
* As a player, I can place my marker in a row diagonally the amount of times set to produce a winner to win the game, so that I know I can win the game by connecting a certain amount of my markers diagonally.
* As a player, I will be notified if a player has won, so that the other player(s) and I can know if there is a winner.
* As a player, I will be notified which player has won, so that the other player(s) and I know who the victor is.
* As a player, I have the choice to play again once the game is over, so that the other player and I can play a new game.
* As a player, I can play every new game on an empty board, so that the other player and I can play on a fresh board.
* As a player, I will be notified if my move is not in range of the board, so that I can only make legal moves on the board.
* As a player, I have

**Nonfunctional Requirements:**

* System must be coded in Java
* System must run in Unix
* System board must be created using a 2D array or HashMap
* System code must be maintainable with JavaDocs comments and contracts for each function
* System must follow player move instructions correctly
* System contracts must be formally stated when possible
* System Javadocs must include invariants for the classes
* System code must be easy to read
* System must not have magic numbers
* System must be easy to add onto
* System must override toString method in BoardPosition class
* System must be modifiable
* System must use best practices discussed in class
* System must override the toString method in GameBoard and GameBoardMem class
* System must override the equals method in BoardPosition class
* System must check to see if there is a winner after every move
* System must check to see if there is a draw after every move
* System must run on either a fast or memory efficient implementation
* System must have a customizable player number, board row number, board column number, and number to win the game
* System must only take in a valid player number, board row number, board column number, and number to win the game
* System fast implementation checks the entire row/column/diagonal to see if there is winner
* System memory implementation checks the spaces around the last move to see if there is winner
* System fast implementation uses a 2D board array
* System memory efficient implementation uses a HashMap.
* System must be able to print out the board
* System must use java swing
* System must display board through java swing