**Tic Tac Toe Game**

**Abstract**

Tic Tac Toe Game is a very famous game that aims to entertain players while at the same time enhance their mental abilities. The game is mainly composed of a table with nine cells where only two players compete against each other. The players have to fill the cells with one of two values, either an “X” or “O”, each player is assigned a specific value in advance. They play in turns, in which the main goal is to create a horizontal, vertical or diagonal line of three consecutive cells that contain the same value either an “X” or “O”.

**Functional Requirements**

The following points illustrate the main requirement of the project:

**System Functional Requirements:**

* The game shall allow no more than two players at a time.
* The game shall assign a name for each player if not already entered by him.
* The game shall allow each participant to play ONLY on his turn, using a specific value either "X" or "O".
* The game shall provide a score for each player to state the winner.
* The game shall declare the winner to be the first one who creates a horizontal, vertical or diagonal line of three consecutive cells that contain the same value either an “X” or “O”.
* The game shall reset the table cells whenever a player win.
* The game shall accept a predefined range of positive values from 1 to 9 ONLY.
* The game shall never allow a player to place a value in a cell that is already taken.
* The game shall always assign a value to each player, either an "X" or "O".

**User Functional Requirements:**

* Each player shall fill the cells with one of two values, either an “X” or “O”.
* Each player shall be assigned a specific value in advance, either an “X” or “O”.
* Each player shall play ONLY on his turn.

**System Main Components**

**Classes:**

* Main Classes:
  + TicTacToe.
  + Game.
* Support Classes:
  + Arrays.
  + InputMismatchException.
  + Scanner.
* Test Class:
  + GameTest.
  + GameParameterizedTest.
  + GameTestSuite.
  + TestRunner.

**Methods:**

* public String getPlayerNames():
  + Gets the players’ names if entered by users, otherwise assigns the names “Player one, Player two”.
* public void populateEmptyBoard():
* Fills the cells with the correct values from 1 to 9.
* public void printBoard():
* Displays a table with nine cells.
* public String determineWhoseTurn():
  + Swaps between players, so they can play in turns.
* public String checkWinner():
  + Checks the rows of the tables for a horizontal, vertical or diagonal line of three consecutive cells that contain the same value either an “X” or “O to determine the winner.
* public boolean checkInput():
  + Ensures that the entered value lies between 1 to 9.
* public boolean placeValue():
  + Ensures that the selected cell where a value will be placed is not already taken.
* public String checkPlayerTurn():
  + Assignes a specific value either "X" or "O" to each player.

**Matching Table**

|  |  |  |
| --- | --- | --- |
| **System Requirement** | **Programmed Method Name** | **Test Case Name** |
| The game shall assign a name for each player if not already entered by them | public String getPlayerNames(){} | public void testGetPlayerOneName(){} |
| public void testGetPlayerTwoName(){} |
| The game shall allow each participant to play ONLY on his turn, using a specific value either "X" or "O" | public String determineWhoseTurn(){} | public void testDetermineWhoseTurnIncorrect(){} |
| public void testDetermineWhoseTurnX(){} |
| public void testDetermineWhoseTurnO(){} |
| The game shall declare the winner to be the first one who creates a horizontal, vertical or diagonal line of three consecutive cells that contain the same value either an “X” or “O” | public String checkWinner(){} | public void testCheckWinner(){} |
| The game shall reset the table cells whenever a player win | public void populateEmptyBoard(){} | public void testPopulateEmptyBoardX(){} |
| public void testPopulateEmptyBoardO(){} |
| The game shall accept a predefined range of positive values from 1 to 9 ONLY | public boolean checkInput(){} | public void checkInputNegativeValue(){} |
| public void checkInputInRange(){} |
| public void checkInputLessthanRange(){} |
| public void testCheckInputGreaterThanRange(){} |
| The game shall never allow a player to place a value in a cell that is already taken | public boolean placeValue(){} | public void testPlaceValueValidCell(){} |
| public void testPlaceValueInalidCell(){} |
| The game shall always assign a value to each player, either an "X" or "O" | public String checkPlayerTurn(){} | public void testCheckPlayerXTurn(){} |
| public void testCheckPlayerOTurn(){} |

**Test Scenarios' Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Scenario** | **Test Steps** | **Test Data / Input** | **Expected Output** | **Actual Output** | **Used Test Case** | **Pass / Fail** | **Notes** |
| Check first player's name | 1.Start the game.  2. Enter a space instead of an exact name.  3. Click enter. | A space | Player One | Player One | public void testGetPlayerOneName(){} | Pass | - |
| Check second player's name | 1.Start the game.  2. Enter a space instead of an exact name.  3. Click enter. | A space | Player Two | Player Two | public void testGetPlayerTwoName(){} | Pass |  |
| Check that the game is correctly swapping between players | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player plays hist turn  5. Click enter. | Current turn :X | Next turn: X | Next turn: O | public void testDetermineWhoseTurnIncorrect(){} | Pass | This test has been ignored because already enough tests have been used for the method (determineWhoseTurn) – in this test we have used assertNotEquals() |
| Check that the game is correctly swapping between players | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player plays hist turn  5. Click enter. | Current turn: X | Next turn: O | Next turn: O | public void testDetermineWhoseTurnX(){} | Pass |  |
| Check that the game is correctly swapping between players | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player plays hist turn  5. The second player plays his turn. | Current turn: O | Next turn: X | Next turn: X | public void testDetermineWhoseTurnO(){} | Pass |  |
| Check that the game declares the winner correctly each time a player win | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. Each player plays in his turn.  5. The second player completes a line of "OOO". | A board with filled cells | "O" wins | "O" wins | public void testCheckWinner(){} | Pass | We used Parameterized test to provide different values each time |
| Check that the game resets the board whenever a player win | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. Each player plays in his turn.  5. The first player completes a line of "XXX".  6. The first player wins the round. | A board with filled cells (with a condition that there are three consecutive cells that contain the same value, "X" | An empty board | An empty board | public void testPopulateEmptyBoardX(){} | Pass |  |
| Check that the game resets the board whenever a player win | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. Each player plays in his turn.  5. The second player completes a line of "OOO".  6. The second player wins the round. | A board with filled cells (with a condition that there are three consecutive cells that contain the same value, "O" | An empty board | An empty board | public void testPopulateEmptyBoardO(){} | Pass |  |
| Checks that the entered value is not a negative one | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player enters a negative value. | -9 | An exception should be thrown | An exception is thrown | public void testCheckInputNegativeValue() { | Pass | We have used Exception testing |
| Checks that the entered value is not a negative one | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player enters an acceptable value. | 6 | true | true | public void checkInputInRange(){} | Pass |  |
| Checks that the entered value is not a negative one | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player enters a value greater than the maximum allowed one. | 10 | false | false | public void testCheckInputGreaterThanRange(){} |  |  |
| Checks that the entered value is not a negative one | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player enters a value less than the minimum allowed one. | 0 | false | false | public void checkInputLessthanRange(){} | Pass |  |
| Checks that the user enters a valid cell index | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player enters an acceptable value (available cell). | Turn: X.  A board with the first two cells felled (1 and 2).  User input for the cell to be occupied:3. | True | True | public void testPlaceValueValidCell(){} | Pass |  |
| Checks that the user enters a valid cell index | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name.  4. The first player enters an unacceptable value (cell is taken already). | Turn: X.  A board with the first two cells felled (1 and 2).  User input for the cell to be occupied:2. | False | False | public void testPlaceValueInalidCell(){} | Pass |  |
| Check that each player is given a specific turn during the game | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name. | Current turn: X.  Player One: Shoroog.  Player Two: Samar. | Shoroog | Shoroog | public void testCheckPlayerXTurn(){} | Pass |  |
| Check that each player is given a specific turn during the game | 1.Start the game.  2. Enter the first player's name.  3. Enter the second player's name. | Current turn: O.  Player One: Shoroog.  Player Two: Samar. | Samar | Samar | public void testCheckPlayerOTurn(){} | Pass |  |