**Project Sprint #4**

Implement all the features that support a player (**human or computer**) to play a simple or general SOS game against another player (**human or** **computer**). The minimum features include **choosing human or computer for red and/or blue players**, **choosing the game mode (simple or general)**, **choosing the board size**, **setting up a new game**, **making a move (in a simple or general game)**, and **determining if a simple or general game is over**. The computer component must be able to play complete simple and general games. You are encouraged to consider basic strategies for winning simple or general games (e.g., against a poor human player). Optimal play is not required.

The following is a sample GUI layout. You should use a class hierarchy to deal with the computer opponent requirements. If your current code has not yet considered class hierarchy, it is time to refactor your code.

|  |  |  |
| --- | --- | --- |
| SOS Icon  Description automatically generated Simple game Icon  Description automatically generated General game Board size  8 | | |
| Blue player  Icon                          Description automatically generated Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon                          Description automatically generated Computer | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon  Description automatically generated Computer |
|  | Current turn: blue (or red) | New Game |

Figure 1. Sample GUI layout of the working program for Sprint 3

**Total points: 24**

1. **Demonstration (8 points)**

Submit a video of no more than five minutes, clearly demonstrating that you have implemented the computer opponent features and written some automated unit tests.

1. A complete simple game where the blue player is a human, the red player is the computer, and there is a winner
2. A complete general game where the blue player is the computer, the red player is a human, and there is a winner
3. A complete simple game where both sides are played by the computer
4. A complete general game where both sides are played by the computer
5. Some automated unit tests for the computer opponent.

In the video, you must explain what is being demonstrated.

1. **User Stories for the Computer Opponent Requirements (1 points)**

* **User Story Template**: As a <role>, I want <goal> [so that <benefit>]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** |
| 8 | Computer Play a Game of a SOS | As a User, I want to be able play a game of SOS against a computer or computer player against another computer player, so that I can if I don’t have another user to play with or I don’t want to play with another user, I can play against a computer component or watch the computer play against the computer. | 8 | 2 hours |

1. **Acceptance Criteria (AC) for the Computer Opponent Requirements (4 points)**

Add or delete rows as needed.

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inProgress)** |
| 1 Choose Game Mode | 1.1 | AC 1.1 <Option Game Mode: Simple>  Given User selects game mode  When User selects “Simple Game” option  Then the System Starts a new game under the mode Simple with a clear game from scratch. | Complete |
| 1.2 | AC 1.2 <Option Game Mode: General>  Given User selects game mode  When User Selects “General Game” option  Then the System Starts a new game under the mode General with a clear game from scratch. | Complete |
| 2 Choose the Board Size | 2.1 | AC 2.1<Board Size>  Given User selects the board size  When User enters “Board Size”  Then the System starts with an adjusted size of the board for the game to match the value of the board size. | Complete |
| 3 Setup New Game | 3.1 | AC 3.1 <New Game >  Given User in middle of Game or Game Over, wants new game  When User selects the “New Game” option  Then the system starts clears the board and has user to select a game mode. | Complete |
| 4 Make a Move Simple Game | 4.1 | AC 4.1 <User wants to place an S or O on the board >  Given User is in a current game and it is there turn  When User Selects an empty square and places an S or an O  Then System adds an S or an O onto the empty square the user selected and inputs. Next Users turn. | Complete |
|  | 4.2 | AC 4.2 <User wants to place an S or O on the board and selects invalid placement outside of board or tries to place on occupied cell>  Given User is in a current game and it is their turn, if user selects a position outside the board or on a cell that is already occupied, then nothing will happen and it will remain same User’s turn. | Complete |
| 5 Make a Move General Game | 5.1 | AC 5.1 <User wants to place an S or O on the board >  Given User is in a current game and it is there turn  When User Selects an empty square and places an S or an O  Then System adds an S or an O onto the empty square the user selected and inputs. If SOS created, keep users turn, else, next Users turn. | Complete |
| 5.2 | AC 5.2 <User wants to place an S or O on the board and selects invalid placement outside of board or tries to place on occupied cell>  Given User is in a current game and it is their turn, if user selects a position outside the board or on a cell that is already occupied, then nothing will happen and it will remain same User’s turn. | Complete |
|
| 6 Determine if Simple Game is Over | 6.1 | AC 6.1 <Check Game Over: Simple>  Given User is in the middle of a Simple Game  When A User has an S-O-S connected together  Then the System ends the game and determines winner. If same number, then it’s a draw. | Complete |
| 7 Determine if General Game is Over | 7.1 | AC 7.1 <Check Game Over: General>  Given User is in the middle of a General Game  When all the squares are filled.  Then the System ends the game and determines the winner based off who has the most S-O-S’s connected. If same number, then it’s a draw. | Complete |
| 8 Computer Play a Game of SOS | 8.1 | AC 8.1 <User wants to play a Computer Opponent>  Given User is not in a game and wants to start a new game  When User selects to begin a new game with the selected blue and red players as their given human or computer player and starts a new game  Then the system starts a new game, with the selected game mode, selected board size and a clear board with the red and blue players as the selected human and computer player and plays a game of SOS. | Complete |
| 8.2 | AC 8.2 <User wants to have a Computer Opponent Play another Computer Opponent>  Given User is not in a game and wants to start a new game  When User selects the red and blue player as computer players and starts a new game  Then the system starts a new game, with the selected game mode, selected board size and a clear board with the red and blue players as computer players and plays out a game of SOS. | Complete |

1. **Summary of All Source Code (1 points)**

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| sosGame.h | Production code | 47 |
| sosGUI.h | Production code | 84 |
| main.cpp | Production code | 12 |
| sosGame.cpp | Production code | 268 |
| sosGUI.cpp | Production code | 408 |
| tst\_testemptyboard.cpp | Test code | 72 |
| tst\_testsosgeneralgame.cpp | Test code | 215 |
| tst\_testsosgui.cpp | Test code | 136 |
| tst\_testsossimplegame.cpp | Test code | 181 |
| sosGameAI.h | Production Code | 22 |
| sosGameAI.cpp | Production Code | 172 |
| tst\_testsosgameai.cpp | Test Code | 97 |
| Total | | 1714 |

**You must submit all source code to get any credit for this assignment.**

1. **Production Code vs New User stories/Acceptance Criteria (2 points)**

Summarize how each of the new user story/acceptance criteria is implemented in your production code (class name and method name etc.)

|  |  |
| --- | --- |
| **User Story ID** | **User Story Name** |
| 1 | Choose a board size |
| 2 | Choose the game mode of a chosen board |
| 3 | Start a new game of the chosen board size and game mode |
| 4 | Make a move in a simple game |
| 5 | Make a move in a general game |
| 6 | A simple game is over |
| 7 | A general game is over |
| 8 | Computer Play of a Game of SOS |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Summary Code** |
| 1 Choose Game Mode | 1.1 | sosGame,  sosGUI | void newGame(int boardSize, int gameMode);  void newSOSGame(sosGame newGame);  void setGameStart();  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. On the GUI, when a Game Mode of Simple is selected, the sosGame within sosGUI class is updated when a first move is made, newGame which takes an input of a board size and game mode and creates a new game and sets GUI elements to match and display it. setGameStart sets the GUI elements on start of a new Game. |
|  | 1.2 | sosGame,  sosGUI | void newGame(int boardSize, int gameMode);  void newSOSGame(sosGame newGame);  void setGameStart();  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. On the GUI, when a Game Mode of General is selected and a first move is made, the sosGame within sosGUI class is updated to that game mode using the method call newGame which takes an input of a board size and game mode and creates a new game and sets GUI elements to match and display it. setGameStart sets the GUI elements on start of a new Game. |
| 2 Choose the Board Size | 2.1 | sosGame,  sosGUI | void initGameBoard();  void newGame(int boardSize, int gameMode);  void newSOSGame(sosGame newGame);  void setGameStart();  void drawGameGrid();  void drawGameBoard();  void onBoardSizeSpinBoxValueChange(int); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. On the GUI, when a Board Size of the Spin Box is chosen, it sets Board Size to update or can create a new game instantly as well using new Game to set board size and game mode. Whenever the value of the spinbx for board size is changed, it updates the drawGameGrid to change to the size of the board. setGameStart sets the elements for the new game of game board GUI when a new game of board size and mode is made. |
| 3 Setup New Game | 3.1 | sosGame,  sosGUI | int getRows();  int getCols();int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  void initGameBoard();  void updateStatus();  void setGameStart();  void newSOSGame(sosGame newGame);  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. |
| 4 Make a Move Simple Game | 4.1 | sosGame,  sosGUI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  void makeMove(int row, int col, int letter);  void updateStatus();  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. Checks if an SOS is connected and if the game is over. As long as a valid move, then the move is made and updated to the game. This then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the Simple Game as long as it is a valid cell and is empty, then updates status of the GUI game. |
|  | 4.2 | sosGame,  sosGUI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  void makeMove(int row, int col, int letter);  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. This then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the Simple Game as long as it is a valid cell and is empty, then updates status of the GUI game to the other players turn to make a move. If it is an invalid move of the Simple Game, nothing happens and it continues to be the same users turn. |
| 5 Make a Move General Game | 5.1 | sosGame,  sosGUI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  void makeMove(int row, int col, int letter);  int getCell(int row, int col);  void updateStatus();  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. Checks if an SOS is connected and if the game is over. As long as a valid move, then the move is made and updated to the game. If the player making the move completes an SOS, it remains their turn. This then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the General Game as long as it is a valid cell and is empty, then updates status of the GUI game. |
|  | 5.2 | sosGame,  sosGUI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  void makeMove(int row, int col, int letter);  int getCell(int row, int col);  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is an invalid move of the General Game, nothing happens and it continues to be the same users turn. |
| 6 Determine if Simple Game is Over | 6.1 | sosGame,  sosGUI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  void makeMove(int row, int col, int letter);  int getCell(int row, int col);  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard();  void vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void restartGame();  bool sosConnected(int currentTurn, int selRow, int selCol, int letter);  void addSOSConnected(int currentTurn, int row1, int col1, int row2, int col2, int row3, int col3);  bool checkAllCellsFilled();  void checkSimpleGameOver();  void newGameStart(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. Checks if an SOS is connected with sosConnected, stores connected sos’s with addSOSConnected and if the simple game is over with checkSimpleGameOver and checkAllCellsFilled to see if all cells are filled. As long as a valid move, then the move is made and updated to the game. If the game is over, game state is updated for status. This then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the Simple Game. Uses getSOSConnected and getSOSConnectedNum to get the conneceted SOS’s from the game logic to draw lines for connected SOS’s. If game is over, then a push button for to restart to a new game is shown and if clicked, restartGame is called to restart game logic and newGameStart is called to restet the GUI to start a new game. |
| 7 Determine if General Game is Over | 7.1 | sosGame,  sosGUI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  void makeMove(int row, int col, int letter);  int getCell(int row, int col);  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard();  void vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void restartGame();  bool sosConnected(int currentTurn, int selRow, int selCol, int letter);  void addSOSConnected(int currentTurn, int row1, int col1, int row2, int col2, int row3, int col3);  bool checkAllCellsFilled();  void checkGeneralGameOver();  void newGameStart(); | Complete | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. Checks if an SOS is connected with sosConnected, stores connected sos’s with addSOSConnected and if the general game is over with checkGeneralGameOver and checkAllCellsFilled to see if all cells filled. As long as a valid move, then the move is made and updated to the game. If the player making the move completes an SOS, it remains their turn. If the game is over, game state is updated for status. This then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the General Game. Uses getSOSConnected and getSOSConnectedNum to get the conneceted SOS’s from the game logic to draw lines for connected SOS’s. If game is over, then a push button for to restart to a new game is shown and if clicked, restartGame is called to restart game logic and newGameStart is called to restet the GUI to start a new game. |
| 8 | 8.1 | sosGame,  sosGUI,  sosGameAI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  void makeMove(int row, int col, int letter);  int getCell(int row, int col);  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard();  void vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void restartGame();  bool sosConnected(int currentTurn, int selRow, int selCol, int letter);  void addSOSConnected(int currentTurn, int row1, int col1, int row2, int col2, int row3, int col3);  bool checkAllCellsFilled();  void checkGeneralGameOver();  void newGameStart();  q |  | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. Checks if an SOS is connected with sosConnected, stores connected sos’s with addSOSConnected and if the general game is over with checkGeneralGameOver or checkSimpleGameOver and checkAllCellsFilled to see if all cells filled. As long as a valid move, then the move is made and updated to the game. Then there is check for if it a computer player’s turn. If it is, it passes the game board to sosGameAI which then determines a move to be made by the computer and is returned back to the sosGUI and the move is updated to the board. This then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the General Game. Uses getSOSConnected and getSOSConnectedNum to get the conneceted SOS’s from the game logic to draw lines for connected SOS’s. If game is over, then a push button for to restart to a new game is shown and if clicked, restartGame is called to restart game logic and newGameStart is called to restet the GUI to start a new game. |
|  | 8.2 | sosGame,  sosGUI,  sosGameAI | int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  void makeMove(int row, int col, int letter);  int getCell(int row, int col);  int getPlayerTurnLetter();  void mousePressEvent(QMouseEvent \*event);  void drawGameGrid();  void drawGameBoard();  void vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void restartGame();  bool sosConnected(int currentTurn, int selRow, int selCol, int letter);  void addSOSConnected(int currentTurn, int row1, int col1, int row2, int col2, int row3, int col3);  bool checkAllCellsFilled();  void checkGeneralGameOver();  void newGameStart();  void getAIMove(sosGame currentGame);  void calcGameMove(); |  | sosGUI class creates a default GUI for SOS Game and calls sosGame class to create an SOS Game that hold the game information. drawGameGrid() draw the SOS Game board and drawGameBoard draws the current SOS Game board with the current state of the game. When setting up a new game to be called with newSOSGame being applied to the sosGUI class, it calls getRows, getCols, getGameMode, getTurn, and getCell in order to get the sosGame information and to update the GUI layout to match the current state of the game. On mouePressEvent, checks for validation of an SOS move within a current game and valid cell. If it is valid move in a playing game, makeMove is called to check for the move and if the cell is empty or not. Checks if an SOS is connected with sosConnected, stores connected sos’s with addSOSConnected and if the general game is over with checkGeneralGameOver or checkSimpleGameOver and checkAllCellsFilled to see if all cells filled. As long as a valid move, then the move is made and updated to the game. Then there is check for if it a computer player’s turn. If it is, it passes the game board to sosGameAI which then determines a move to be made by the computer and is returned back to the sosGUI and the move is updated to the boardThis then in turn updates the game board GUI with drawGameGrid and drawGameBoard for the General Game. Uses getSOSConnected and getSOSConnectedNum to get the conneceted SOS’s from the game logic to draw lines for connected SOS’s. If game is over, then a push button for to restart to a new game is shown and if clicked, restartGame is called to restart game logic and newGameStart is called to restet the GUI to start a new game. |

1. **Tests vs New User stories/Acceptance Criteria (2 points)**

Summarize how each of the new user story/acceptance criteria is tested by your test code (class name and method name) or manually performed tests.

6.1 Automated tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Class Name (s) of the Test Code** | **Method Name(s) of the Test Code** | **Description of the Test Case (input & expected output)** |
| 1 Choose Game Mode | 1.1 | TestSOSSimpleGame | testEmptySimpleGame() | Test case takes a new game input of a board size and an input of simple game mode. It compares that the game mode is a simple game mode and compares every cell of the game to make sure a new simple game of all empty cells were created. Output pass as long as all cells are empty of a new game. |
|  | 1.2 | TestSOSGeneralGame | testEmptyGeneralGame() | Test case takes a new game input of a board size and an input of general game mode. It compares that the game mode is a general game mode and compares every cell of the game to make sure a new general game of all empty cells were created. Output pass as long as all cells are empty of a new game. |
| 2 Choose the Board Size | 2.1 | TestSOSGUI | testBoardSizeGUI() | Test case takes an input for a board size. Output passes as long as it compares that the board size input matches the board size of the game board. |
| 3 Setup New Game | 3.1 | TestSOSGUI | testNewGameBoardGUI() | Test case takes an input for a board size and a game mode to create a new game board. Output passes as long as the input board size and input game mode matches the current game board and size, and checks to make sure all of the cells of the new board size are empty. |
| 4 Make a Move Simple Game | 4.1 | TestSOSSimpleGame | testEmptySimpleGame(),  testMoveVacantCellSimpleGame()  testSimpleGameSOSConnected() | Test case takes a new game input of a board size and an input of simple game mode. It compares that the game mode is a simple game mode and compares every cell of the game to make sure a new simple game of all empty cells were created. Output pass as long as all cells are empty of a new game.  Test case takes an input for a new simple game and input move for a Letter S at one location and a Letter O at another location. Output passes as long as compares that game mode is simple, and that there is a Letter S and Letter O in corresponding locations.  Test case takes inputs for a simple game for letters to form an SOS. Output passes as long as compares that before the last letter is placed there are is not any connected SOS’s and after there is a connected SOS. |
|  | 4.2 | TestSOSSimpleGame | testInvalidRowSimpleGame(), testInvalidColumnSimpleGame(),  testSimpleGameNonVacantCell() | Test case compares an input cell location outside of the valid rows size and outputs as passed as long as it matches as null.  Test case compares an input cell location outside of the valid columns size and outputs as passed as long as it matches as null.  Test case compares input cell that is already occupied with a move and outputs as passed as long as nothing changes and current cell remains the same. |
| 5 Make a Move General Game | 5.1 | TestSOSGeneralGame | testEmptyGeneralGame(),  testMoveVacantCellSimpleGame()  testGeneralGameSOSConnected() | Test case takes a new game input of a board size and an input of general game mode. It compares that the game mode is a general game mode and compares every cell of the game to make sure a new general game of all empty cells were created. Output pass as long as all cells are empty of a new game.  Test case take an input for a new general game and input move for a Letter S at one location and a Letter O at another location. Output passes as long as compares that game mode is general, and that there is a Letter S and Letter O in corresponding locations.  Test case takes inputs for a general game for letters to form an SOS. Output passes as long as compares that before the last letter is placed there are is not any connected SOS’s and after there is a connected SOS. |
|  | 5.2 | TestSOSGeneralGame | testInvalidRowGeneralGame(), testInvalidColumnGeneralGame(),  testGeneralGameNonVacantCell() | Test case compares an input cell location outside of the of valid rows size and outputs as passed as long as it matches as null.  Test case compares an input cell location outside of the valid columns size and outputs as passed as long as it matches as null.  Test case compares input cell that is already occupied with a move and outputs as passed as long as nothing changes and current cell remains the same. |
| 6 Determine if Simple Game is Over | 6.1 | TestSOSSimpleGame | testSimpleGameOverP1Wins()  testSimpleGameOverP2Wins()  testSimpleGameOverDraw() | Test case compares a simple game inputs for letters SOS and passes as long as compare matches game state as player one as winner and a connected SOS.  Test case compares a simple game inputs for letters SOS and passes as long as compare matches game state as player two as winner and a connected SOS.  Test case compares a simple game inputs to completely fill all cells of the game board and passes as long as compare matches game state as a draw, all cells filled, and there are no connected SOSs. |
| 7 Determine if General Game is Over | 7.1 | TestSOSGeneralGame | testGeneralGameOverP1Wins()  testGeneralGameOverP2Wins()  testGeneralGameOverDrawNoSOS()  testGeneralGameOverDrawWithSOS() | Test case compares a general game inputs to completely fill all cells of the game board and to make an SOS. It passes as long as compare matches all cells filled, game stare winner as player one, and there is a connected SOS.  Test case compares a general game inputs to completely fill all cells of the game board and to make an SOS. It passes as long as compare matches all cells filled, game stare winner as player two, and there is a connected SOS.  Test case compares a general game inputs to completely fill all cells of the game board and to makes no SOS. It passes as long as compare matches all cells filled, game stare winner as draw, and there is no connected SOSs.  Test case compares a general game inputs to completely fill all cells of the game board and makes two connected SOSs for each player. It passes as long as compare matches all cells filled, game stare winner as draw, and there is the same number of connected SOSs for each player. |
| 8 | 8.1 | TestSOSAI | testAIMovesEmptyBoard()  testAICompleteSOSAvailable  MoveOnBoardSimpleGame()  testAICompletesMoveThat  DoesntSetupSOSForOpponent  GeneralGame() | Test case take empty board for a new general game and places consecutive moves by computer.  Test case takes an empty board for a simple game, places an S and O consecutively and takes the computer move to place and S to create an SOS and win the game.  Test case takes an empty board for a general game and places S and O letters on the board. With no SOS available to be completed, Computer makes a move that doesn’t setup an SOS to be completed by the next players turn. |
|  | 8.2 | TestSOSAI | testAIMovesEmptyBoard()  testAICompleteSOSAvailable  MoveOnBoardSimpleGame()  testAICompletesMoveThat  DoesntSetupSOSForOpponent  GeneralGame() | Test case take empty board for a new general game and places consecutive moves by computer.  Test case takes an empty board for a simple game, places an S and O consecutively and takes the computer move to place and S to create an SOS and win the game.  Test case takes an empty board for a general game and places S and O letters on the board. With no SOS available to be completed, Computer makes a move that doesn’t setup an SOS to be completed by the next players turn. |

6.2 Manual tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** |
| 1 Choose Game Mode | 1.1 | Select Simple Game. | Sets Game Mode to selected Simple Game |  |
|  | 1.2 | Select General Game | Sets Game Mode to selected General Game |  |
| 2 Choose Board Size | 2.1 | Select Board size on Spin Box for Board Size | Sets Board Size to selected board size |  |
| 3 Setup New Game | 3.1 | Sets up a new game with selected board size and selected game mode | New Game started, board size adjusted, board mode set to chosen options |  |
| 4 Make a Move Simple Game | 4.1 | Place an S and O for each users within a valid empty cell for a simple game | S and O are placed within the selected empty cells |  |
|  | 4.2 | Make a move outside valid board size cells of Simple Game | Nothing happens, remains current users turn |  |
| 5 Make a Move General Game | 5.1 | Place an S and O for each users within a valid empty cell for a general game | S and O are placed within the selected empty cells |  |
|  | 5.2 | Make a move outside valid board size cells of General Game | Nothing happens, remains current users turn |  |
| 6 Determine if Simple Game is Over | 6.1 | Connected SOS for blue  Connected SOS for red  No connected SOS for draw | Simple game over, blue line SOS, restart game appears, no more moves can be made, blue wins  Simple game over, red line SOS, restart game appears, no more moves can be made, red wins  Simple game over, no SOS, all cells filled, restart game appears, no more moves can be made, draw |  |
| 7 Determine if General Game is Over | 7.1 | Connected SOS for blue, all cells filled  Connected SOS for red, all cells filled  No connected SOS for draw, all cells filled  Same number connected SOSs for draw, all cells filled | General game over, all cells filled, blue line SOSs, restart game appears, no more moves can be made, blue wins  General game over, all cells filled, red line SOSs, restart game appears, no more moves can be made, red wins  General game over, all cells filled, no SOSs, restart game appears, no more moves can be made, draw  General game over, all cells filled, blue and red line SOSs, restart game appears, no more moves can be made, draw |  |
| 8 | 8.1 | Places consecutive moves by computer on empty general Game  Computer places an S to complete an SOS on a simple game board with S and O setup  General Game with S and O letters placed on board with no SOS available to be completed, Computer makes a move that doesn’t setup an SOS to be completed by next players turn | General Game Board with two moves made by computer  Simple Game board, game over, SOS completed by Computer, restart game appears, no more moves can be made  General Game Board, multiple S and O letters placed on board in selected spaces, Computer places move that doesn’t setup an SOS to be completed by next player |  |
|  | 8.2 | Places consecutive moves by computer on empty general Game  Computer places an S to complete an SOS on a simple game board with S and O setup  General Game with S and O letters placed on board with no SOS available to be completed, Computer makes a move that doesn’t setup an SOS to be completed by next players turn | General Game Board with two moves made by computer  Simple Game board, game over, SOS completed by Computer, restart game appears, no more moves can be made  General Game Board, multiple S and O letters placed on board in selected spaces, Computer places move that doesn’t setup an SOS to be completed by next player |  |

6.3 Other automated or manual tests not corresponding to the acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number** | **Test Input** | **Expected Result** | **Class Name of the Test Code** | **Method Name of the Test Code** |
| 1 | Click Restart Game Push Button | Start of New Game GUI, resets all GUI components back to start to choose a new game | sosGUI  sosGame | onNewGameButtonClicked()  newGameStart()  restartGame() |
|  |  |  |  |  |

1. **Present the class diagram of your production code (3 points) and describe how the class hierarchy in your design deals with the computer opponent requirements (3 points)**?

**Class Diagram:**

Graphical user interface, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

My class hierarchy is setup in a way that for the computer opponent requirements just requires the input of the game board layout that the gui class is using. Within the gui class, whenever there needs to be a move made by the computer opponent, a copy of the game board is passed to the gameAI class. Then a move is generated based off what the board looks like and is returned to the gui class where the move is then applied to the game board within the gui class. This way the gameAI class isn’t making the changes to the game board that is being used for play by the gui class directly. It just decides what move should be made and returns it back to prevent an the actual game board that is used for play from being fully passed back and forth as the main board, only a copy of the board is passed.