**Project Sprint #3**

The SOS game is described in CS449HomeworkOverview.docx. You should read the description very carefully.

Your submission must include the GitHub link to your project and you must ensure that the instructor has the proper access to your project. You will receive no points otherwise.

**GitHub link:** [**https://github.com/Pavan698/sprint3/tree/main**](https://github.com/Pavan698/sprint3/tree/main)

Implement all the features that support a human player to play a simple or general SOS game against a human opponent and refactor your existing code if necessary. The minimum features include **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 following is a sample GUI layout. It is required to use a class hierarchy to deal with the common requirements of the Simple Game and the General Game. **If your code for Sprint 2 has not 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 S  Icon  Description automatically generated O | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated S  Icon  Description automatically generated O |
|  | Current turn: blue (or red) | New Game |

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

**Deliverables: expand and improve your submission for sprint 2.**

1. **Demonstration (9 points)**

Submit a video of no more than five minutes, clearly demonstrating the following features.

1. A simple game that the blue player is the winner
2. A simple draw game with the same board size as (a)
3. A general game that the red player is the winner, and the board size is different from (a)
4. A general draw game with the same board size as (c)
5. Some automated unit tests for the simple game mode
6. Some automated unit tests for the general game mode

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

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Main | Production code | 6 |
| Game | Production code | 176 |
| GUI | Production code | 429 |
| GeneralGame | Production code | 15 |
| SimpleGame | Production code | 21 |
| GameTest | Test Code | 351 |
|  | Total | 998 |

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

1. **Production Code vs User stories/Acceptance Criteria (3 points)**

Summarize how each of the 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 | A simple game is over |
| 6 | Make a move in a general game |
| 7 | A general game is over |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID and**  **Name** | **AC**  **ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes**  **(optional)** |
| 1 Choose a board size | 1.1 | GUI  Game | boolean handleBoardSizeTextChange(String txtBoardSize) public void setBoardSize(int boardSize) | Complete |  |
| 1.2 | GUI  Game | boolean handleBoardSizeTextChange(String txtBoardSize)  boolean isBoardSizeTextNumeric(String txtBoardSize)  boolean isBoardSizeGreaterThanTwo(int newSize) { | Complete |  |
| 2 Choose the game mode of a chosen board | 2.1 | GUI  SimpleGame | public void itemStateChanged(ItemEvent e)  public SimpleGamee(int boardSize) | Complete |  |
|  | 2.2 | GUI  GeneralGame | public void itemStateChanged(ItemEvent e)  GeneralGame(int boardSize) | Complete |  |
| 3 Start a new game of the chosen board size and game mode | 3.1 | GUI  Game | private void newGameButtonClicked()  public void reset(int boardSize, boolean isSimpleGame) | Complete |  |
| 4 Make a move in a simple game | 4.1 | GUI  SimpleGame | private void cellClicked(JButton button, int finalI, int finalJ)  public void makeMove(int row, int col,  String symbol) | Complete |  |
|  | 4.2 | GUI  GUI  GUI  GUI  GUI  GUI.CustomButton  GUI.CustomButton  GUI.CustomButton  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game | private boolean  checkSequenceAndDrawLine(int row, int col, String symbol, Color color)  public void drawVerticalLine(Color color, int startRow, int col)  public void drawHorizontalLine(Color color, int row, int startCol)  public void  drawDiagonalLineFromTopLeft(Color color, int startRow, int startCol)  public void drawDiagonalLineFromTopRight(Color color, int startRow, int startCol)  public void addTopLeftToBottomRightLine(Color color)  public void addTopRightToBottomLeftLine(Color color)  public void addCenterHorizontalLine(Color color) public void addCenterVerticalLine(Color color)  public void addCenterVerticalLine(Color color)  public boolean checkTopVerticalSOS(int row, int col)  public boolean checkBottomVerticalSOS(int row, int col)  public boolean checkForwardHorizontalSOS(int row, int col)  public boolean checkBackwardHorizontalSOS(int row,  int col)  public boolean checkDiagonalSOSFromBottomLeft(int  row, int col)  public boolean checkDiagonalSOSFromBottomRight(int  row, int col)  public boolean checkDiagonalSOSFromTopLeft(int  row, int col)  public boolean checkDiagonalSOSFromTopRight(int  row, int col)  public void setBluePlayersTurn(boolean bluePlayersTurn) | Complete |  |
| 5 A simple game is over | 5.1 | GUI  SimpleGame  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game | private void cellClicked(JButton button, int finalI, int finalJ)  public boolean isGameOver()  boolean checkSequenceAt(int row, int col)  public boolean checkTopVerticalSOS(int row, int col)  public boolean checkBottomVerticalSOS(int row, int col)  public boolean checkForwardHorizontalSOS(int row, int col)  public boolean checkBackwardHorizontalSOS(int row,  int col)  public boolean checkDiagonalSOSFromBottomLeft(int  row, int col)  public boolean checkDiagonalSOSFromBottomRight(int  row, int col)  public boolean checkDiagonalSOSFromTopLeft(int  row, int col)  public boolean checkDiagonalSOSFromTopRight(int  row, int col)  public int getRedPlayerSOSCount()  public int getBluePlayerSOSCount() | Complete |  |
|  | 5.2 | GUI  SimpleGame  Game | private void cellClicked(JButton button, int finalI, int finalJ)  public boolean isGameOver()  public boolean isBoardFull() | Complete |  |
| 6 Make a move in general game | 6.1 | GUI  GeneralGame | private void cellClicked(JButton button, int finalI, int finalJ)  public void makeMove(int row, int col, String symbol) | Complete |  |
|  | 6.2 | GUI  GUI  GUI  GUI  GUI  GUI.CustomButton  GUI.CustomButton  GUI.CustomButton  GUI.CustomButton  Game  Game  Game  Game  Game  Game  Game  Game  Game | private boolean  checkSequenceAndDrawLine(int row, int col, String symbol, Color color)  public void drawVerticalLine(Color color, int startRow, int col)  public void drawHorizontalLine(Color color, int row, int startCol)  public void  drawDiagonalLineFromTopLeft(Color color, int startRow, int startCol)  public void drawDiagonalLineFromTopRight(Color color, int startRow, int startCol)  public void addTopLeftToBottomRightLine(Color color)  public void addTopRightToBottomLeftLine(Color color)  public void addCenterHorizontalLine(Color color) public void addCenterVerticalLine(Color color)  public void addCenterVerticalLine(Color color)  public boolean checkTopVerticalSOS(int row, int col)  public boolean checkBottomVerticalSOS(int row, int col)  public boolean checkForwardHorizontalSOS(int row, int col)  public boolean checkBackwardHorizontalSOS(int row,  int col)  public boolean checkDiagonalSOSFromBottomLeft(int  row, int col)  public boolean checkDiagonalSOSFromBottomRight(int  row, int col)  public boolean checkDiagonalSOSFromTopLeft(int  row, int col)  public boolean checkDiagonalSOSFromTopRight(int  row, int col)  public void setBluePlayersTurn(boolean bluePlayersTurn) |  |  |
|  | 6.3 | Game | public void setBluePlayersTurn(boolean Complete bluePlayersTurn) | Complete |  |
| 7 A General game is over | 7.1 | GUI  GeneralGame  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game | private void cellClicked(JButton button, int finalI, int finalJ)  public boolean isGameOver() public boolean isBoardFull()  boolean checkSequenceAt(int row, int col)  public boolean checkTopVerticalSOS(int row, int col)  public boolean checkBottomVerticalSOS(int row, int col)  public boolean checkForwardHorizontalSOS(int row, int col)  public boolean checkBackwardHorizontalSOS(int row, int col)  public boolean checkDiagonalSOSFromBottomLeft(int  row, int col)  public boolean checkDiagonalSOSFromBottomRight(int  row, int col)  public boolean checkDiagonalSOSFromTopLeft(int  row, int col)  public boolean checkDiagonalSOSFromTopRight(int  row, int col)  public int getRedPlayerSOSCount()  public int getBluePlayerSOSCount() | Complete |  |
|  | 7.2 | GUI  GeneralGame  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game  Game | private void cellClicked(JButton button, int finalI, int finalJ)  public boolean isGameOver() public boolean isBoardFull()  boolean checkSequenceAt(int row, int col)  public boolean checkTopVerticalSOS(int row, int col)  public boolean checkBottomVerticalSOS(int row, int col)  public boolean checkForwardHorizontalSOS(int row, int col)  public boolean checkBackwardHorizontalSOS(int row,  int col)  public boolean checkDiagonalSOSFromBottomLeft(int row, int col)  public boolean checkDiagonalSOSFromBottomRight(int  row, int col)  public boolean checkDiagonalSOSFromTopLeft(int  row, int col)  public boolean checkDiagonalSOSFromTopRight(int  row, int col)  public int getRedPlayerSOSCount()  public int getBluePlayerSOSCount() | Complete |  |

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

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

|  |  |
| --- | --- |
| **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 | A simple game is over |
| 6 | Make a move in a general game |
| 7 | A general game is over |

4.1 Automated tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User**  **Story ID and Name** | **AC**  **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 a board size | 1.1 | GameTest | testChooseBoardSize() | Test setting and getting the board size, expecting the set size to match the get size.  Input: Set board size to 10. Expected Output: Get board size should return 10. |
| 1.2 | GameTest | testBoardSizeEnteredIsNumeric | Test checking if a string is numeric, expecting false for a non-numeric string and true for a numeric string.  Input: "a". Expected Output: isBoardSizeTextNumeric should return false.  Input: "5". Expected Output: isBoardSizeTextNumeric should return true |
|  |  |  | testBoardSizeEnteredIsGreaterThanTwo | Test checking if an entered number is greater than 2, expecting false for a number less than 2 and true for a number greater than 2. |
| 2 Choose the game mode of a chosen board | 2.1 | GameTest | testSimpleGameMode | This test case verifies whether a new game instance created in "SimpleGame" mode is indeed  an instance of the  "SimpleGame" class.  Input: A new game instance is created with a board size of 8, initialized in "SimpleGame" mode.  Expected Output: The test should return true |
|  | 2.2 | GameTest | testGeneralGameMode | This test case verifies whether a new game instance created in "GeneralGame" mode is indeed  an instance of the  "GeneralGame" class.  Input: A new game instance is created with a board size of 8, initialized in "GeneralGame" mode.  Expected Output: The test should return true |
| 3 Start a  new game  of the chosen board size and game mode | 3.1 | GameTest | testStartGameWithChosenSizeAndMode | This test case checks whether starting a new game with a chosen board size and mode  works correctly in a  "SimpleGame" instance.  Input: A new "SimpleGame" instance is created with an initial board size of 8.   * The board size is changed to 9 using the ‘reset’ method. - The test verifies the board size, checks if the instance is not of type "GeneralGame," and examines the content of the game board.   Expected Output: The board size is expected to be 9 after calling ‘reset’, and this is verified using the  ‘assertEquals’ method.  Since the game was initially created as a "SimpleGame," the test expects that it is not an instance of "GeneralGame," and this is checked using ‘assertEquals’ with ‘false’. - The last ‘assertEquals’ checks if the content of the game board at position [0][0] is an empty string, indicating that the board is empty at the start of the game. |
| 4 Make a move in a simple game | 4.1 | GameTest | testMakeSMoveInSimpleGame | Test making an 'S' move in a simple game, expecting the board to have 'S' at the specified position.  Input: Reset game with board size 8 and simple game mode, then make an 'S' move at [5][0]. Expected Output: The board should have 'S' at [5][0]. |
|  | 4.2 | GameTest | testMakeOMoveInSimpleGame | Test making an 'O' move in a simple game, expecting the board to have 'O' at the specified position.  Input: Reset game with board size 8 and simple game mode, then make an 'O' move at [0][2]. Expected Output: The board should have 'O' at [0][2]. |
| 5 A simple game is over | 5.1 | GameTest | testIsSimpleGameOver\_SOSWinner | Test checking the "isGameOver" method's behavior in a simple game mode when there is a winning "SOS" sequence  Input: A SimpleGame instance with a board size of 8.  Three moves are made to form an "SOS" sequence on the first row (0, 0, "S"; 0, 1, "O"; 0, 2, "S").  Expected Output: The  “isGameOver” method returns true, indicating that the game is over because a winning "SOS" sequence has been formed |
|  | 5.2 | GameTest | testIsSimpleGameOver\_BoardFull | This test case aims to validate the "isGameOver" method's behavior in a simple game mode when the game is not over, and no "SOS" sequence has been formed.  Input: A SimpleGame instance with a board size of 8.  Three moves are made on the  first row (0, 0, "S"; 0, 1, "S"; 0, 2, "S"), but no "SOS" sequence is formed.  Expected Output: The  "isGameOver" method returns false, indicating that the game is not over as no winning "SOS" sequence is present.  This test case verifies the behavior of the "isGameOver" method in a simple game mode when the game is over because the game board is ful).  Input: A SimpleGame instance with a board size of 3.  Sequential moves are made to fill the entire 3x3 game board with "S" marks.  Expected Output: The  "isGameOver" method returns true, indicating that the game is over due to the entire board being filled with moves and no "SOS" sequence formed. |
| 6 Make a move in a  general  game | 6.1 | GameTest | testMakeSMoveInGeneralGame | Test making an 'S' move in a general game, expecting the board to have 'S' at the specified position.  Input: Reset game with board size 8 and general game mode, then make an 'S' move at [5][0]. Expected Output: The board should have 'S' at [5][0]. |
|  | 6.2 | GameTest | testMakeOMoveInGeneralGame | Test making an 'O' move in a general game, expecting the board to have 'O' at the specified position.  Input: Reset game with board size 8 and general game mode, then make an 'O' move at [0][2]. Expected Output: The board should have 'O' at [0][2]. |
| 7 A general game is over | 7.1 | GameTest | testIsGeneralGameOver\_BluePlayerWins | This test case assesses the "isGameOver" method within a general game mode when the blue player wins by forming one SOS sequence.  Input:A GeneralGame instance with a board size of 3. Moves are made to fill the majority of the 3x3 board with "S" moves and one "O" move, forming a single SOS sequence for the blue player.  Expected Output: The  "isGameOver" method returns true, indicating that the game is over because the blue player has formed an SOS sequence and won. It further ensures that the count of SOS sequences for the blue player is greater than the count for the red player, verifying that the blue player has won. |
|  | 7.2 | GameTest | testIsGeneralGameOver\_BoardFull | This test case examines the functionality of the  "isGameOver" method in a general game mode when the game is over due to the board being full.  Input:A GeneralGame instance with a board size of 3. Moves are made to fill the  entire 3x3 board with "S" moves,  Expected Output: The  "isGameOver" method returns true, indicating that the game is over. In this case, the game ends because the entire board is filled, and no more moves can be made. It further ensures that the count of SOS sequences for the blue and red players is the same, verifying that the game is a draw. |

4.2 Manual tests directly corresponding to some acceptance criteria

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story**  **ID and**  **Name** | **AC**  **ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** | |
| 1 Choose a board size | 1.1 | Set board size to 10. | A new 10\*10 game board is displayed | Test passed | |
| 1.2 | Input: "a"  Input: "5" | Error message indicating invalid input  Error message indicating only numbers can be entered for board size. | Test passed  Test passed | |
| 2 Choose the game mode of a chosen board | 2.1 | Set game mode to simple. | Simple Game radio button is selected and General Game radio button deselected | Test passed | |
| 2.2 | Set game mode to general. | Simple Game radio button is deselected and General Game radio button selected | Test passed | |
| 3 Start a  new game  of the chosen board size and game mode | 3.1 | Board size = 9,  Game mode =  General Game  Board size =8,  Game mode =  Simple Game | When ‘New Game’ button is pressed a message indicating the new game start is shown and players are allowed to make moves in a General game of 9\*9 game board.  When ‘New Game’ button is pressed a message indicating the new game start is shown and players are allowed to make moves in a Simple game of 8\*8 game board. | Test passed  Test passed | |
| 4 Make a move in a simple game | 4.1 | During player’s turn in a Simple Game, player selects ‘S’ from his side of radio button and clicks on an empty square  During player’s turn in a Simple Game, player selects ‘O’ from his side of radio button and clicks on an empty square  During player’s turn in a Simple Game, player selects ‘S’ from his side of radio | The square which was clicked on is now marked as ‘S’  The square which was clicked on is now marked as ‘O’  An error message is displayed showing player should click on an empty square. | Test passed  Test passed  Test passed | |
|  | 4.2 | During player’s turn in a Simple Game, player selects ‘S’ from his side of radio button and clicks on an empty square  During player’s turn in a Simple Game, player selects ‘O’ from his side of radio button and clicks on an empty square  During player’s turn in a Simple Game, player selects ‘S’ from his side of radio button and clicks on a non empty square  During player’s turn in a Simple Game, player selects ‘O’ from his side of radio button and clicks on a non empty square | The square which was clicked on is now marked as ‘S’ and turn is altered  The square which was clicked on is now marked as ‘O’ and turn is altered  An error message is displayed showing player should click on an empty square and turn is not altered  An error message is displayed showing player should click on an empty square and turn is not altered | Test passed  Test passed  Test passed  Test passed | |
| 5 A simple game is over | 5.1 | Make a SOS sequence by blue player in a simple  game | SOS sequence is marked with a blue line and a Game Over message is displayed declaring the blue player as the winner | Test passed | |
|  | 5.2 | Fill the board without making any SOS sequence | Game Over message is displayed declaring it’s a draw | Test passed | |
| 6 Make a move in a  general  game | 6.1 | During player’s turn in a General Game, player selects ‘S’ from his side of radio button and clicks on an empty square  During player’s turn in a General Game, player selects ‘O’ from his side of radio button and clicks on an empty square  During player’s turn in a General Game, player selects ‘S’ from his side of radio button and clicks on a non empty square  During player’s turn in a General Game, player selects ‘O’ from his side of radio button and clicks on a non empty square | The square which was clicked on is now marked as ‘S’  The square which was clicked on is now marked as ‘O’  An error message is displayed showing player should click on an empty square.  An error message is displayed showing player should click on an empty square | Test passed  Test passed  Test passed  Test passed | |
|  | 6.2 | During player’s turn in a General Game, player selects ‘S’ from his side of radio button and clicks on an empty square  During player’s turn in a General Game, player selects ‘O’ from his side of radio button and clicks on an empty square  During player’s turn in a General Game, player selects ‘S’ from his side of radio button and clicks on a non empty square  During player’s turn in a General Game, player selects ‘O’ from his side of radio button and clicks on a non empty square | The square which was clicked on is now marked as ‘S’ and turn is altered  The square which was clicked on is now marked as ‘O’ and turn is altered  An error message is displayed showing player should click on an empty square and turn is not altered  An error message is displayed showing player should click on an empty square and turn is not altered | Test passed  Test passed  Test passed  Test passed |
| 7 A general game is over | 7.1 | Play until the board is full ensuring only one SOS sequence is created and its by the blue player | SOS sequence is marked with a blue line and a Game Over message is displayed declaring the blue player as the winner | Test passed |
|  | 7.2 | Fill the board without making any SOS sequence | Game Over message is displayed declaring it’s a draw | Test passed |

4.3 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 | - | Initial board size is 8 | GameTest | testInitialBoardSize |
| 2 | - | Game  board is created and its empty | GameTest | testInitializeBoard |
| 3 | - | Initial player is Blue Player | GameTest | testInitialPlayerTurn |
| 4 | Make a move in simple game and reset to a new game with a 10 \* 10 game board | Game  board size is 10.  A game board is created and its empty | GameTest | testResetGame |
| 5 | Make 3 moves without making any SOS sequences  in a Simple  Game | Game is not over | GameTest | testIsSimpleGameOver\_NotOver |
| 6 | Make 3 moves making an SOS sequence  in a General  Game | Game is not over | GameTest | testIsGeneralGameOver\_SOSSequence |
| 7 | Game instance (SimpleGame, board size: 8); Moves:  (0,1,"S"),  (0,2,"O"),  (0,3,"S") | True, a  SOS  sequence is made horizontally | GameTest | testCheckSimpleGameForwardHorizontalSOS |
| 8 | Game instance (SimpleGame, board size: 8); Moves:  (0,3,"S"),  (0,2,"O"),  (0,1,"S") | True, a  SOS  sequence is made horizontally | GameTest | testCheckSimpleGameBackwardHorizontalSOS |
| 9 | Game instance (SimpleGame, board size: 8); Moves:  (2,0,"S"),  (1,0,"O"),  (0,0,"S") | True, a  SOS  sequence is made  vertically | GameTest | testCheckSimpleGameTopVerticalSOS |
| 10 | Game instance (SimpleGame, board size: 8); Moves:  (0,0,"S"),  (1,0,"O"),  (2,0,"S") | True, a  SOS  sequence is made  vertically | GameTest | testCheckSimpleGameBottomVerticalSOS |
| 11 | Game instance (SimpleGame, board size: 5); Moves:  (0,0,"S"),  (1,1,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckSimpleGameDiagonalSOSFromTopLeft |
| 12 | Game instance (SimpleGame, board size: 5); Moves:  (0,4,"S"),  (1,3,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckSimpleGameDiagonalSOSFromTopRight |
| 13 | Game instance (SimpleGame, board size: 5); Moves:  (4,0,"S"),  (3,1,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckSimpleGameDiagonalSOSFromBottomLeft |
| 14 | Game instance (SimpleGame, board size: 5); Moves:  (4,4,"S"),  (3,3,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckSimpleGameDiagonalSOSFromBottomRight |
| 15 | Game instance (GeneralGame, board size: 8); Moves:  (0,1,"S"),  (0,2,"O"),  (0,3,"S") | True, a  SOS  sequence is made horizontally | GameTest | testCheckGeneralGameForwardHorizontalSOS |
| 16 | Game instance (GeneralGame, board size: 8); Moves:  (0,3,"S"),  (0,2,"O"),  (0,1,"S") | True, a  SOS  sequence is made horizontally | GameTest | testCheckGeneralGameBackwardHorizontalSOS |
| 17 | Game instance (GeneralGame, board size: 8); Moves:  (2,0,"S"),  (1,0,"O"),  (0,0,"S") | True, a  SOS  sequence is made  vertically | GameTest | testCheckGeneralGameTopVerticalSOS |
| 18 | Game instance (GeneralGame, board size: 8); Moves:  (0,0,"S"),  (1,0,"O"),  (2,0,"S") | True, a  SOS  sequence is made  vertically | GameTest | testCheckGeneralGameBottomVerticalSOS |
| 19 | Game instance (GeneralGame, board size: 5); Moves:  (0,0,"S"),  (1,1,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckGeneralGameDiagonalSOSFromTopLeft |
| 20 | Game instance (GeneralGame, board size: 5); Moves: | True, a  SOS  sequence is | GameTest | testCheckGeneralGameDiagonalSOSFromTopRight |
|  | (0,4,"S"),  (1,3,"O"),  (2,2,"S") | made diagonally |  |  |
| 21 | Game instance (GeneralGame, board size: 5); Moves:  (4,0,"S"),  (3,1,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckGeneralGameDiagonalSOSFromBottomLeft |
| 22 | Game instance (GeneralGame, board size: 5); Moves:  (4,4,"S"),  (3,3,"O"),  (2,2,"S") | True, a  SOS  sequence is made diagonally | GameTest | testCheckGeneralGameDiagonalSOSFromBottomRight |

1. **(Part 1) Describe how the class hierarchy in your design deals with the common and different requirements of the Simple Game and the General Game.**

**At least 1/2 page, excluding screenshots/diagrams (12pt, single-spaced), is required.** **(3 Points)**

I have implemented a class hierarchy for a SOS game, consisting of an abstract base class ‘Game’ and two concrete subclasses: ‘SimpleGame’ and ‘GeneralGame’. The base class ‘Game’ contains the common logic for both types of games, while the subclasses implement game mode specific behavior. Here is how the class hierarchy deals with the common and different requirements of the Simple Game and General Game.

The ‘Game’ class contains the shared properties and methods that are common to both game types. These include properties like ‘boardSize’, ‘gameBoard’ etc.. The handling of player turns is also common, using the ‘isBluePlayersTurn’ property to determine the current player's turn. The ‘bluePlayerSOSCount’ and ‘redPlayerSOSCount’ properties are common, keeping track of the number of SOS sequences for each player.

Methods like ‘makeMove’, ‘isGameOver’, and ‘isBoardFull’ are shared among both game types. The ‘reset’ method in the ‘Game’ class initializes the game board and other properties. This is also common to both game types, ensuring they start with a clean slate. Both game types also share the ‘checkSequenceAt’ and other sequence-checking methods. These methods are used to detect sequences in the game board. ‘SimpleGame’ and ‘GeneralGame’ are subclasses that implement game-specific behavior. In ‘SimpleGame’, the ‘isGameOver’ method checks for presence of a SOS sequence and if not whether the board is full to declare that the game is over. In contrast, ‘GeneralGame’ simply checks if the board is full to determine the game is over.

**(Part 2) Demonstrate how you use LLM (ChatGPT or other) to analyze how well your code adheres to the design principles discussed in class - modularity, cohesion, coupling, and encapsulation – using the definitions provided in class. Provide screenshots of your interactions with the LLM, showing the prompts used and the responses received.**  **Interpret the LLM’s feedback, refine your prompts if needed to obtain more relevant responses, and discuss any changes made to your code based on the analysis. Explain how these refinements improve adherence to the design principles.**  **At least 1/2 page is required, excluding screenshots (12pt, single-spaced).** **(2 Points)**

Regarding modularity, while the initial separation into base and derived game classes was sound, the LLM noted the Game class contained excessive responsibilities - managing both board state and SOS validation logic. This violated the Single Responsibility Principle. Following its suggestion, I extracted all directional SOS checks (horizontal, vertical, diagonal) into a dedicated SOS Validator utility class, significantly improving modular organization. For cohesion, ChatGPT observed that the original Game class mixed core game mechanics with mode-specific behaviors. By moving all win-condition logic to the respective SimpleGame and GeneralGame subclasses, each class became more focused and internally consistent.

The analysis of coupling proved particularly insightful. While inheritance coupling between base and derived classes was appropriately loose, ChatGPT identified tight coupling in the makeMove() implementation, where the base class made assumptions about subclass behavior. I refactored this using the Template Method pattern - the base class now handles the universal move validation while delegating sequence checking to abstract methods implemented by subclasses. For encapsulation, beyond fixing the array exposure issue, the LLM recommended making the gameBoard field private and providing controlled access methods, which I implemented along with input validation in makeMove().These changes collectively produced a more robust architecture. The improved modularity makes the system easier to extend (new game modes can be added without modifying existing code). Enhanced cohesion means each class's purpose is immediately clear to maintainers. Reduced coupling minimizes ripple effects from changes, and stricter encapsulation prevents accidental state corruption. The LLM's ability to cross-reference these principles helped identify non-obvious relationships - like how the original SOS check methods, while functionally correct, actually weakened cohesion by mixing validation with state management. This analysis would have been significantly harder to perform manually.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer test

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.