**Project Sprint #5**

The main tasks of this assignment are:

1. Adding the feature of recording a game into a text file. The user story and acceptance criteria of both record and replay are required, but the implementation of replay is for extra credit (up to 2 points in the weighted total).
2. Conducting a code review exercise.
3. Summarizing the lessons learned from Sprint 0 through Sprint 5.

The following is a sample GUI layout of the final product, where “Replay” is optional.

|  |  |  |
| --- | --- | --- |
| 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  Replay |
| Record game | Current turn: blue (or red) | New Game |

Figure 1. Sample GUI layout of the final product

**Total points: 16**

1. **Demonstration (6 points)**

Submit a video of no more than 8 minutes, clearly demonstrating that you have implemented all the features in the following table. In the video, you must explain what is being demonstrated.

|  |  |
| --- | --- |
|  | **Feature** |
| 1 | A complete simple game of two human players is recorded |
| 2 | A complete general game of two human players is recorded |
| 3 | A complete simple game of human-computer players is recorded |
| 4 | A complete general game of human-computer players is recorded |
| 5 | A complete simple game of computer-computer players is recorded |
| 6 | A complete general game of computer-computer players is recorded |

If you have implemented the “replay” feature for extra credit, you should include its demonstration in the video.

1. **User Stories and Acceptance Criteria for the Record/Replay Requirements (1 points)**

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

Add or delete rows as needed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** |
| 9 | Record a game of SOS into a Text File | As a User, I want to be able to record a played game of SOS into a text file so that I can look back at how the game played out and the moves that were made by the players. | 9 | 2 hours |
| 10 | Replay an SOS Game from the GUI | As a User, I want to be able to replay a played game of SOS that was just played out so that I can watch the game at how it was played out on the Graphical User Interface and see what the moves that occurred by the players throughout the game. | 10 | 1 hour |

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inProgress)** |
| 9 Record a game of SOS into a Text File | 9.1 | AC 9.1 <User wants to record an SOS game into a text file>  Given User is not in a game and wants to record a game of SOS  When User selects the “Record Game” option  Then the Board Size, Game Mode, Player One player type, Player Two player type, and all game moves (Move Number, Player Turn, Row, Column, Letter) that occur between the two players will be stored. When the game of SOS is complete, all the information for the game and winner will be output and saved to a text file in the order of occurrence that can be viewed to see how the game of SOS was played. | Complete |
| 10 Replay an SOS Game from the GUI | 10.1 | AC 10.1 <User wants to replay an SOS game from the GUI>  Given User just finished an SOS game and recorded the SOS game  When User selects the “Replay Game” option  Then the played game of SOS that was just recorded into a text file will be replayed move by move from beginning to end and all of the information about the game (Board Size, Game Mode, Player One player type, Player Two player type) that occurred displayed onto the GUI, | Complete |

1. **Code Review (2 points)**

Apply source code review to one or two most important classes (and other classes if time permits) and report the findings. In addition to looking for bugs, the review should check: (1) whether the entire project has followed the coding standard in a consistent manner, (2) whether the project has followed the design principles introduced in class, and (3) whether there are code smells that indicate the need for refactoring. The following checklists provide basic guidelines. You may add new items to each of the checklists.

Make sure your answers resulted from the code review exercise. If there is no finding for an entry, you should provide an explanation. For example, if your answer to “Are the naming conventions violated?” is no, you should describe a naming convention and present an example. You will receive no credit for this part if your answers are simply yes or no without additional information.

Classes that have been reviewed: sosGUI, sosGame

Date/time duration of the code review exercise: 05/09, 2 hours

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Are the naming conventions violated? | Class names don’t start with Capital Letter  -sosGUI, sosGame  Variable names don’t include underscores, not all lowercase  -Examples: cellSize, boardSize, rectX, rectY  Constant names don’t have leading letter like ‘k’  -Ex. outputDrawWinnerText, fixedWidthWindow, fixedHeightWindow | |
| Is the ordering convention of method arguments violated? | Declaration Order for Class are grouped in proper order  Should be: Types, non-static data members, static constansts, factory functions, constructors, destructor, all other functions, all other data members  Mine is: Constructors, Destructors, Types, static constants, non static data members, all other data members, all other functions  Ex.ample:  sosGame();  ~sosGame();  enum GameMode{NONE, SIMPLE, GENERAL};  enum GameState{NOTPLAYING, PLAYING, DRAW, PLAYERONE\_WON, PLAYERTWO\_WON};  enum Cell{EMPTY, LETTERS, LETTERO};  enum Turn{P1, P2};  int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void newGame(int boardSize, int gameMode);  void restartGame();  void makeMove(int row, int col, int letter); | |
| Any comments meaningless or inconsistent with the code? | Overall classes (sosGUI, sosGame) don’t have comments with examples  Functions comments are too obvious, not all the proper information like inputs/outputs, arguments, etc.  Example:  //Get Current Players Turn Letter thats checked  int sosGUI::getPlayerTurnLetter() | |
| Any code block has an inconsistent formatting style? | All my code blocks have same format style of it being blocked, indendented, nested, spaced, etc.  Example:  void sosGUI::newSOSGame(sosGame newGame)  {  game = newGame;  setGameStart();  }  void sosGUI::paintEvent(QPaintEvent \*)  {  drawGameGrid();  drawGameBoard();  } | |
| Any indentations inconsistent? | Change in indentations on extended lines  Example:  if ((game.getTurn() == game.P1 && bluePlayerS->isChecked() == true)  || (game.getTurn() == game.P2 && redPlayerS->isChecked() == true))  Versus:  currentPlayerTurn->move((fixedWidthWindow/2)-(currentPlayerTurn->size().width()/2),  (((rectY+cellSize)+fixedHeightWindow)/2)-(currentPlayerTurn->size().height())); | |
| Design Principles | Any class/method not well-modularized? | My setGameStart() method repetitive with if/else statements of just changing UI items that repeated throughout other methods  Example:  int tempBluePlayerType;  int tempRedPlayerType;  boardSize = game.getRows();  boardSizeSpinBox->setValue(boardSize);  boardSizeSpinBox->setEnabled(false);  if (game.getGameMode() == game.GENERAL) {  generalGame->setChecked(true);  generalGame->setEnabled(true);  simpleGame->setChecked(false);  simpleGame->setEnabled(false);  }  else  {  simpleGame->setChecked(true);  simpleGame->setEnabled(true);  generalGame->setChecked(false);  generalGame->setEnabled(false);  }  if (bluePlayerHuman->isChecked() == true) {  bluePlayerHuman->setEnabled(true);  bluePlayerHuman->setChecked(true);  bluePlayerComputer->setEnabled(false);  bluePlayerComputer->setChecked(false);  tempBluePlayerType = HUMAN;  }  else {  bluePlayerComputer->setEnabled(true);  bluePlayerComputer->setChecked(true);  bluePlayerHuman->setEnabled(false);  bluePlayerHuman->setChecked(false);  tempBluePlayerType = COMPUTER;  }  if (redPlayerHuman->isChecked() == true) {  redPlayerHuman->setEnabled(true);  redPlayerHuman->setChecked(true);  redPlayerComputer->setEnabled(false);  redPlayerComputer->setChecked(false);  tempRedPlayerType = HUMAN;  }  else{  redPlayerComputer->setEnabled(true);  redPlayerComputer->setChecked(true);  redPlayerHuman->setEnabled(false);  redPlayerHuman->setChecked(false);  tempRedPlayerType = COMPUTER;  }  newGame->hide();  replayGame->hide();  recordGame->setEnabled(false);  if (recordGame->isChecked() == true) {  recordGameBoardSizeMode(game.getRows(), game.getGameMode(), tempBluePlayerType, tempRedPlayerType);  }  updateStatus();  repaint();  } | |
| Any class with poor abstraction? | sosGUI is all private methods to itself except taking a game board  sosGame only returns information with getters or can call one of three methods to start a new game, restart a game or make a move.  Example: sosGame class  enum GameMode{NONE, SIMPLE, GENERAL};  enum GameState{NOTPLAYING, PLAYING, DRAW, PLAYERONE\_WON, PLAYERTWO\_WON};  enum Cell{EMPTY, LETTERS, LETTERO};  enum Turn{P1, P2};  int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void newGame(int boardSize, int gameMode);  void restartGame();  void makeMove(int row, int col, int letter); | |
| Is the visibility of any variable, method, and class inappropriate? | Every Variable and method of my classes are private except the minimum getters for sosGame to return like players turn, board size, game mode, etc. and passing data like making a move to the game.  Example:  Public:  sosGame();  ~sosGame();  enum GameMode{NONE, SIMPLE, GENERAL};  enum GameState{NOTPLAYING, PLAYING, DRAW, PLAYERONE\_WON, PLAYERTWO\_WON};  enum Cell{EMPTY, LETTERS, LETTERO};  enum Turn{P1, P2};  int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void newGame(int boardSize, int gameMode);  void restartGame();  void makeMove(int row, int col, int letter);  private:  int maxCellSize = 25;  int rows;  int cols;  int letter;  int cell[25][25];  int turn;  vector<vector<int>> connectedSOS;  GameState currentGameState;  GameMode currentGameMode;  void initGameBoard();  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 checkGeneralGameOver(); | |
| Is design by contract (pre/post-condition) violated? | No preconditions/postconditions  Example: getCell(row, col) returns cell as empty, letter s or letter O as long as the parament for columns and rows is met  //Return game board cell status  int sosGame::getCell(int row, int col)  {  if (row >= 0 && row < rows && col >= 0 && col < cols)  {  return cell[row][col];  }  else  {  return 0;  }  } | |
| Is the Open-Closed Principle violated? | I don’t use any extensions of inheritance in this project. The sosGUI class does use the sosGame class, so there could be an interface between the two, or added inheritance.  Example of my use:  class sosGUI {  sosGame game;  …  } | |
| Is the Single Responsibility Principle violated? | Must return the SOS connected number within a vector in order to return the required SOS connected vector of an overall Vector. Not ideal.  Example:  Class sosGame  vector<vector<int>> getSOSConnected();  int getSOSConnectedNum(); | |
| Code Smells | Are there magic numbers? | I solved most magic numbers, but a a few do remain for sizing of my UI components  Examples: sosGUI  newGame->resize(110, 32);  newGame->move(535, 360);  recordGame->resize(120, 32);  recordGame->move(40, 360);  boardSizeSpinBox->setRange(3, 10); | |
| Are there unnecessary global / class variable? | I used constant strings for the passing and matching text terms for recording to a text file. There is probably a better way I could have done this.  Example: sosGUI  const QString outputSimpleGameModeText = "SIMPLE";  const QString outputGeneralGameModeText = "GENERAL";  const QString outputHumanPlayerTypeText = "HUMAN";  const QString outputComputerPlayerTypeText = "COMPUTER";  const QString outputBluePlayerText = "BLUEPLAYER";  const QString outputRedPlayerText = "REDPLAYER";  const QString outputLetterSText = "S";  const QString outputLetterOText = "O";  const QString outputDrawWinnerText = "DRAW";  const QString outputPlayerOneWinnerText = "BLUEPLAYER";  const QString outputPlayerTwoWinnerText = "REDPLAYER"; | |
| Is there duplicate code? | Yes. In sosGame, when checking for a connected sosConnected() repeats on if an sos is connected, it adds and sets a bool to true.  Example: sosGame  if (token == LETTERS) {  if (cell[row-1][col-1] == LETTERO && cell[row-2][col-2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row-1, col-1, row-2, col-2);  connectSOS = true;  }  if (cell[row-1][col] == LETTERO && cell[row-2][col] == LETTERS) {  addSOSConnected(playerTurn, row, col, row-1, col, row-2, col);  connectSOS = true;  }  if (cell[row-1][col+1] == LETTERO && cell[row-2][col+2] == LETTERS)  {  addSOSConnected(playerTurn, row, col, row-1, col+1, row-2, col+2);  connectSOS = true;  }  if (cell[row][col-1] == LETTERO && cell[row][col-2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row, col-1, row, col-2);  connectSOS = true;  }  if (cell[row][col+1] == LETTERO && cell[row][col+2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row, col+1, row, col+2);  connectSOS = true;  }  if(cell[row+1][col-1] == LETTERO && cell[row+2][col-2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row+1, col-1, row+2, col-2);  connectSOS = true;  }  if (cell[row+1][col] == LETTERO && cell[row+2][col] == LETTERS) {  addSOSConnected(playerTurn, row, col, row+1, col, row+2, col);  connectSOS = true;  }  if (cell[row+1][col+1] == LETTERO && cell[row+2][col+2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row+1, col+1, row+2, col+2);  connectSOS = true;  }  } | |
| Are there long methods? | My mouse press events method is very long. I could reduce this as checking for and making a move as user vs computer use some similar pieces of code that could be reduced.  Example sosGUI  if ((event->pos().x() >= rectX && event-> pos().y() >= rectY)  && (event->pos().x() < (rectX + cellSize) && event->pos().y() < (rectY + cellSize)))  {  if (game.getGameState() == game.NOTPLAYING)  {  if (generalGame->isChecked() == true) {  game.newGame(boardSize, game.GENERAL);  }  else if (simpleGame->isChecked() == true)  {  game.newGame(boardSize, game.SIMPLE);  }  newSOSGame(game);  }  if (game.getGameState() == game.PLAYING || game.getGameState() == game.NOTPLAYING)  {  if (!(game.getTurn() == game.P1 && bluePlayerComputer->isChecked() == true)) {  bool moveValidEmptyCell = false;  int currTurnMove = game.getTurn();  int rowSelected = (event->pos().y() - rectY) / (cellSize / game.getRows());  int colSelected = (event->pos().x() - rectX) / (cellSize / game.getCols());  int letterSelected = getPlayerTurnLetter();  if (game.getCell(rowSelected, colSelected) == game.EMPTY) {  moveValidEmptyCell = true;  }  game.makeMove(rowSelected, colSelected, letterSelected);  if (recordGame->isChecked() == true && moveValidEmptyCell == true) {  recordGameMove(currTurnMove, rowSelected, colSelected, letterSelected);  }  updateStatus();  }  }  }  repaint();  //Computer Makes moves while its the computer's turn to make a move  bool computerTurn = true;  while (computerTurn != false) {  computerTurn = false;  if (game.getTurn() == game.P1 && bluePlayerComputer->isChecked() == true && game.getGameState() == game.PLAYING){  Sleep(500);  int currTurnMove = game.getTurn();  vector<int> bluePlayerComputerMove = gameAI.getAIMove(game);  game.makeMove(bluePlayerComputerMove.at(0), bluePlayerComputerMove.at(1), bluePlayerComputerMove.at(2));  if (recordGame->isChecked() == true) {  recordGameMove(currTurnMove, bluePlayerComputerMove.at(0), bluePlayerComputerMove.at(1), bluePlayerComputerMove.at(2));  }  repaint();  updateStatus();  computerTurn = true;  }  else if (game.getTurn() == game.P2 && redPlayerComputer->isChecked() == true && game.getGameState() == game.PLAYING) {  Sleep(500);  int currTurnMove = game.getTurn();  vector<int> redPlayerComputerMove = gameAI.getAIMove(game);  game.makeMove(redPlayerComputerMove.at(0), redPlayerComputerMove.at(1), redPlayerComputerMove.at(2));  if (recordGame->isChecked() == true) {  recordGameMove(currTurnMove, redPlayerComputerMove.at(0), redPlayerComputerMove.at(1), redPlayerComputerMove.at(2));  }  repaint();  updateStatus();  computerTurn = true;  }  } | |
| Is there any long parameter list? | Yes. I assume using UI components with all stretched out in the class declatations makes it excessively long and takes up a lot of lines.  Example: sosGUI  const QString outputSimpleGameModeText = "SIMPLE";  const QString outputGeneralGameModeText = "GENERAL";  const QString outputHumanPlayerTypeText = "HUMAN";  const QString outputComputerPlayerTypeText = "COMPUTER";  const QString outputBluePlayerText = "BLUEPLAYER";  const QString outputRedPlayerText = "REDPLAYER";  const QString outputLetterSText = "S";  const QString outputLetterOText = "O";  const QString outputDrawWinnerText = "DRAW";  const QString outputPlayerOneWinnerText = "BLUEPLAYER";  const QString outputPlayerTwoWinnerText = "REDPLAYER";  vector<vector<int>> vecRecordGameData;  QVBoxLayout \*layoutBluePlayer;  QVBoxLayout \*layoutRedPlayer;  QVBoxLayout \*layoutBlueHumanComputer;  QVBoxLayout \*layoutRedHumanComputer;  QHBoxLayout \*layoutGameChoice;  QHBoxLayout \*layoutBoardSize;  QWidget \*holdBlueSO;  QWidget \*holdRedSO;  QWidget \*holdBlueHumanComputer;  QWidget \*holdRedHumanComputer;  QWidget \*holdGameType;  QWidget \*holdBoardSize;  QRadioButton \*bluePlayerS;  QRadioButton \*bluePlayerO;  QRadioButton \*bluePlayerHuman;  QRadioButton \*bluePlayerComputer;  QRadioButton \*redPlayerS;  QRadioButton \*redPlayerO;  QRadioButton \*redPlayerHuman;  QRadioButton \*redPlayerComputer;  QRadioButton \*simpleGame;  QRadioButton \*generalGame;  QLabel \*bluePlayer;  QLabel \*redPlayer;  QLabel \*currentPlayerTurn;  QLabel \*title;  QLabel \*boardSizeLabel;  QSpinBox \*boardSizeSpinBox;  QPushButton \*newGame;  QPushButton \*replayGame;  QCheckBox \*recordGame; | |
| Is there over-complex expression? | Yes. When moving some of my UI components to a location, there exact movement location is over complex to calculate and could be inconsistent.  Examples: sosGUI  holdRedSO->move((((fixedWidthWindow+((fixedWidthWindow/2)+(cellSize/2)))/2)-(holdRedSO->size().width()/2)),  ((fixedHeightWindow/2)-(holdRedSO->size().height()/2)));  holdGameType->move(((fixedWidthWindow/2)-(holdGameType->size().width()/2)),  ((rectY/2)-(holdGameType->size().height()/2)));  holdRedHumanComputer->move((((fixedWidthWindow+((fixedWidthWindow/2)+(cellSize/2)))/2)-(holdRedSO->size().width()/2)),  ((fixedHeightWindow/2)+(holdRedSO->size().height()/2))); | |
| Is there switch or if-then-else that needs to be replaced with polymorphism | Yes. When checking for connected SOSs within sosGame class, I list out all possible if-else statements if letter s or letter O. Could be condensed and simplified to check each without listing every single scenario out.  Example: sosGUI  if (token == LETTERS) {  if (cell[row-1][col-1] == LETTERO && cell[row-2][col-2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row-1, col-1, row-2, col-2);  connectSOS = true;  }  if (cell[row-1][col] == LETTERO && cell[row-2][col] == LETTERS) {  addSOSConnected(playerTurn, row, col, row-1, col, row-2, col);  connectSOS = true;  }  if (cell[row-1][col+1] == LETTERO && cell[row-2][col+2] == LETTERS)  {  addSOSConnected(playerTurn, row, col, row-1, col+1, row-2, col+2);  connectSOS = true;  }  if (cell[row][col-1] == LETTERO && cell[row][col-2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row, col-1, row, col-2);  connectSOS = true;  }  if (cell[row][col+1] == LETTERO && cell[row][col+2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row, col+1, row, col+2);  connectSOS = true;  }  if(cell[row+1][col-1] == LETTERO && cell[row+2][col-2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row+1, col-1, row+2, col-2);  connectSOS = true;  }  if (cell[row+1][col] == LETTERO && cell[row+2][col] == LETTERS) {  addSOSConnected(playerTurn, row, col, row+1, col, row+2, col);  connectSOS = true;  }  if (cell[row+1][col+1] == LETTERO && cell[row+2][col+2] == LETTERS) {  addSOSConnected(playerTurn, row, col, row+1, col+1, row+2, col+2);  connectSOS = true;  }  }  else if(token == LETTERO) {  if (cell[row-1][col-1] == LETTERS && cell[row+1][col+1] == LETTERS) {  addSOSConnected(playerTurn, row-1, col-1, row, col, row+1, col+1);  connectSOS = true;  }  if (cell[row-1][col] == LETTERS && cell[row+1][col] == LETTERS) {  addSOSConnected(playerTurn, row-1, col, row, col, row+1, col);  connectSOS = true;  }  if (cell[row-1][col+1] == LETTERS && cell[row+1][col-1] == LETTERS)  {  addSOSConnected(playerTurn, row-1, col+1, row, col, row+1, col-1);  connectSOS = true;  }  if (cell[row][col-1] == LETTERS && cell[row][col+1] == LETTERS) {  addSOSConnected(playerTurn, row, col-1, row, col, row, col+1);  connectSOS = true;  }  } | |
| Any variable or method name whose intent is unclear? | Yes. My sosGUI Constant string names could be interpreted as different outputs.  Examples:  const QString outputSimpleGameModeText = "SIMPLE";  const QString outputGeneralGameModeText = "GENERAL";  const QString outputHumanPlayerTypeText = "HUMAN";  const QString outputComputerPlayerTypeText = "COMPUTER";  const QString outputBluePlayerText = "BLUEPLAYER";  const QString outputRedPlayerText = "REDPLAYER";  const QString outputLetterSText = "S";  const QString outputLetterOText = "O";  const QString outputDrawWinnerText = "DRAW";  const QString outputPlayerOneWinnerText = "BLUEPLAYER";  const QString outputPlayerTwoWinnerText = "REDPLAYER"; | |
| Any similar methods in different classes? | My sosGUI and my sosGame classes are very different. One handles the GUI components and the other is game logic and control. No similar methods.  Example:  sosGUI  void createSOSGameBoard();  void setContentSOS();  void radioButtons();  void createInputWidgets();  void createLabels();  void pushButtons();  void checkButtons();  void updateStatus();  void newGameStart();  void setGameStart();  void drawGameGrid();  void drawGameBoard();  int getPlayerTurnLetter();  void recordGameBoardSizeMode(int inBoardSize, int inGameMode, int inPlayerTypeP1, int inPlayerTypeP2);  void recordGameMove(int inPlayerTurn, int inRow, int inCol, int inLetter);  void outputRecordGameDataTextFile(int inPlayerWinner);  void replayGameFromDataTextFile();  void paintEvent(QPaintEvent \*);  void mousePressEvent(QMouseEvent \*event);  sosGame  int getRows();  int getCols();  int getGameState();  int getGameMode();  int getTurn();  int getCell(int row, int col);  vector<vector<int>> getSOSConnected();  int getSOSConnectedNum();  void newGame(int boardSize, int gameMode);  void restartGame();  void makeMove(int row, int col, int letter);  void initGameBoard();  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 checkGeneralGameOver(); | |
| **Bugs** | **Buggy code snippet** | **What is the bug?** | **Why is it a bug?** |
| testGame.newGame(10, testGame.SIMPLE);  testGUI.newSOSGame(testGame);  testGUI.show(); | When I run Test Code to change radio button between simple and General Game, it highlights the right one, but doesn’t mark the circle for the correct one | The radio button should be filled and marked when selected and it is not. |

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 | 105 |
| main.cpp | Production code | 12 |
| sosGame.cpp | Production code | 268 |
| sosGUI.cpp | Production code | 628 |
| 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 | | 1955 |

**You will receive no credit for this assignment unless your complete source code is submitted.**

1. Summarize the lessons learned from the entire project by answering the following questions from the perspectives of development processes, coding, design, refactoring, and testing **(6 points)**:

* What did you personally gain from the project?
* What does your project do well, and what could your project do better?
* How could you improve your development process if you develop a similar game from scratch?

Minimum requirement for (5): One full page single spaced, font size no bigger than 12 points.

I personally did learn some things from this project. First off, this is the first full GUI project that I have worked on from start to finish with more complexity than just completing a single task. This is the most lines of code I have put into a project which allowed me to apply the entire design process such as the testing, refactoring, designing, and coding of the UI. As for the testing, it’s the first time I have created and applied automated tests. This gave me a better understanding of how to write tests in a way that inputs and compares what is being output by the program resulting in whether the test is passing or failing based on what I want the output to be. It also showed me how to create tests that covers as much of the program as possible that way I can test as many scenarios as possible for how the UI and the functionality of the program work together to not output the desired results, but also to fix bugs that could possibly associate with how a user could interact with it. When it came to designing the project, this project proved to me why how you design a project from the start is very important. Some things I did in the beginning made testing and refactoring more difficult later and costed me more time when adding and making changes. For example, when it came to coupling and cohesion, some things performed improperly depending on when and what was required and caused repeated pieces of code that weren’t needed. As for refactoring, I’ve only refactored on smaller and simpler programs. When refactoring on this project it showed how applying design principles and heuristics early in the program can make refactoring later either much easier or much more difficult. Also, when it came to the UI toolkit, this allowed me to use a C++ UI that I hadn’t used before in QT which showed me how to adapt to other UI kits easier and apply some of those things to future UI kits I use going forward.

There are a few things I feel that I was able to apply and adapt well to my project. One thing I felt I applied well was encapsulation and information hiding. For encapsulation, I bundled the data for the game within a game class, the UI items within a UI class, and the computer opponent within its own class. This allows for all the necessary data items to be relevant and together for their purpose intended. This prevents more and more classes from being all required and dependent on each other with different pieces of information scattered out making it easier to use, access, and change that data when needed going forward. This also leads to why I believe my project utilizes information hiding properly. When it came the UI class I made sure everything was private and only accessible to that class because the methods and items within don’t need to be now or shown to anything else preventing unwanted changed or data manipulation to the UI. For the game class, I made sure all the variables and data structures were private and only accessible and can be changed within the class. There are the minimum getters to return information about the game that is necessary to know, but none of the information can be changed or manipulated outside of it so that it may function with its intended purpose. I also designed my UI in a way that you could pass a game class into the UI when needed. This allowed for easier testing because I could test, create, and compare logic of game with ease. Then when needed, I could pass that game into the UI class, and see the test results within the displayed UI. I believe my project also has good test coverage when it comes to the game logic. I created many tests with many different variations of moves, game types, game outcomes, board size, etc. That way I can prevent as many possible errors or bugs as possible that could disrupt the game for a user.

Although I feel that I did fulfill all the necessary requirements of this project and applied good design principles programming, I feel there are still improvements that could be made. One thing I could particularly improve upon is the refactoring of the entire project. There are a lot of lines of code that I could condense. This is in part due to a few pieces of code that are duplicated with slight variation or unnecessary use of variables. I could create some abstract methods that could take care of this, turning some of those large blocks of code into a method call. There are also some extra nested conditionals that could be reduced and made more precise. For my UI class there are a lot of lines that just make single changes to a UI item (i.e., size, shape, font size, text, etc.). There is a way I can probably perform multiple of those changes at once or in a more condensed fashion. An improvement I could make to my automatic testing is to create more tests that apply to functions of UI items. I have tests that cover game logic and show the UI with the game, but not the actual UI items. Most of my direct UI testing was just manual testing that took longer and was probably less coverage than what could have been done. I could make some tests that perform direct UI interactions such as mouse clicks and button clicks and compare those actions on the UI as if a user was interacting with it. Although I applied techniques to follow high cohesion and loose coupling, there could still be more improvements made. There are some pieces of code that perform tasks fine as they are but could be made safer and changed to be incorporated for maintenance if changes were made to the code. Some improvements could be changing mixed abstractions into single and independent abstractions for higher cohesion and passing more data into parameters instead of modifying global data to improve looser and safer coupling.

If I were to create a similar game from scratch, there are some things I would change in my development process to improve the quality of the product. One thing I would change would be how I apply the design principles. I would break up and some of my bigger methods and into smaller sub-methods. This would make testing easier because I could test these bigger pieces more individually and in smaller scope to make sure they are operating properly. Also, this would make changes and adding on much easier because these would be broken down into smaller problems. When I went back to refactor, the way I had some of these things designed required me to make changes and it took me longer to refactor. I could change one piece and it would apply much easier than happening to change a much larger pieces of code and methods. Another thing I would do would be to pass more data as parameters as to modules and method calls. That way it would have the direct information now when needed instead of relying on certain pieces of global to be changed by the time that module is needed. This makes it easier to account for what and when is happening in the code structure and can prevent more bugs. When it comes to data structures, I would change some of the data structures that I used for certain things. I used a lot vectors and passing of vectors which requires more memory and can be slower. I would instead use more arrays and pointers to improve the storing and passing of data to improve the speed and required amount of memory of the project.