Project Sprint #4

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/jsb58p/CS449---Software-Engineering-Project

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 must 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.

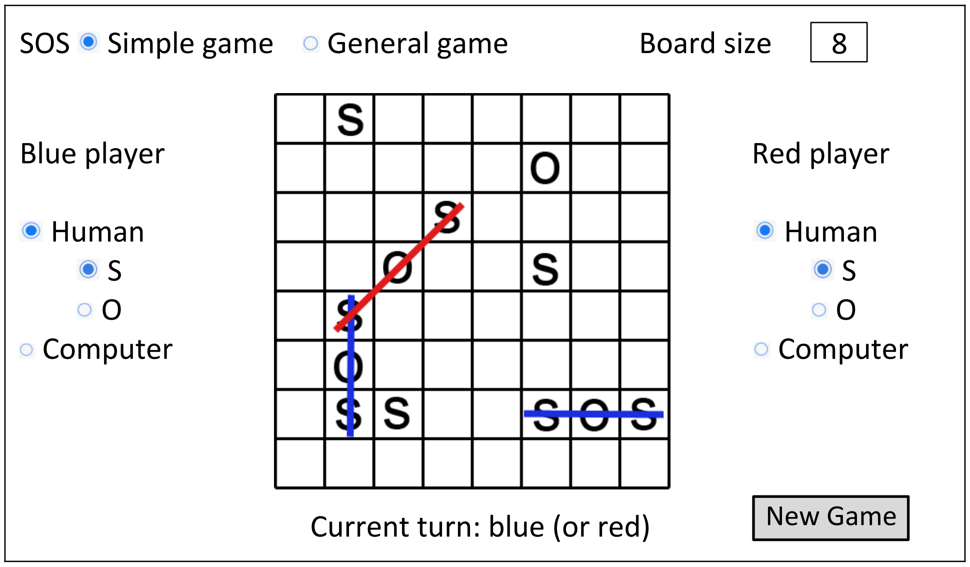


Figure 1. Sample GUI layout of the working program for Sprint 4

Total points: 24

1. **Demonstration (8 points)**

   Submit a link to a video of no more than five minutes, clearly demonstrating that you have implemented the computer opponent features and written some automated unit tests. No points will be given without a video link.
   YouTube/Panopto link: https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=e26aa4ce-cb6f-40cd-9b88-b39a0015b164
   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.

2. 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 opponent assigned letter.	As a player, I want to assign a computer opponent 'S' or 'O', so that I can control the computer's strategy.	High	6
9	Computer opponent selects move	As a player, I want the computer opponent to select high-scoring moves, so that the game is challenging and engaging.	Medium	6

3. 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)
8 story eight	8.1	AC 8.1 Computer opponent assigned letter. Given an opponent (Red or Blue) is assigned to the computer AND 'S' or 'O' is selected for that opponent, When a new game is initialized Then the opponent(s) assigned to the computer will play their given role automatically	Completed
9 story nine	9.1	AC 9.1 Computer opponent selects move Given that an opponent is assigned to the computer When it is the given opponent's turn Then the opponent will automatically select one of the highest-scoring moves based on their letter, 'S' or 'O'.	Completed

4. Summary of All Source Code (1 points)

Source code file name	Production code or test code?	# lines of code
RadioButtonGroup.java	Production code	91
BoundaryCheck.java	Production code	64
ComputerOpponent.java	Production code	28
CpuOpponentO.java	Production code	53
CpuOpponentS.java	Production code	53
GameModeLogic.java	Production code	89
SimpleMode.java	Production code	66
GeneralMode.java	Production code	62
SOSGameGUI.java	Production code	370
SOSGame.java	Production code	428
TestSOSGame.java	Test code	174
	Total	1,478

5. 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 and Name	AC ID	Class Name(s)	Method Name(s)	Status (complete or not)	Notes (optional)
8 Computer opponent assigned letter	8.1	CpuOpponentS, CpuOpponentO	computerChooseStrategy()	Complete	CpuOpponentS places 'S', CpuOpponentO places 'O' by calling makeMove() with corresponding letter
		SOSGame	cpuMoveCheck()	Complete	Uses instanceof checks to determine if cpu object matches selected letter; recreates cpu object if letter selection changes
		SOSGameGUI	getBluePlayerOpponent(), getRedPlayerOpponent(),	Complete	Checks opponent selection for each player
9 Computer Opponent selects move	9.1	CpuOpponentS, CpuOpponentO	computerChooseStrategy()	Complete	CpuOpponentS places 'S', CpuOpponentO places 'O' by calling makeMove() with corresponding letter.
		SOSGame	cpuMoveCheck()	Complete	Uses switch statements for Blue and Red opponent;

					instanceof checks to determine if cpu object matches selected letter; recreates cpu object if letter selection changes
		SOSGameGUI	getBluePlayerOpponent(), getRedPlayerOpponent	Complete	Checks opponent selection for each player

6. 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)**
8	8.1	TestSOSGame	testCpuOpponentSPlacesLetterS()	Checks that the computer opponent ‘S’ selects a valid board row and column, then checks board to check that the row and column is filled.
		TestSOSGame	testCpuOpponentOPlacesLetterO()	Checks that the computer opponent ‘O’ selects a valid board row and column, then checks board to check that the row and column is filled.
9	9.1	TestSOSGame	testCpuOpponentOSelectsHighestScoringMove( )	Places two ‘S’ with a column between them, then tests that the computer opponent ‘O’ makes a move at the row/column between them to score.
		TestSOSGame	testCpuOpponentSSelectsHighestScoringMove()	Places an ‘S’ with an ‘O’ directly beneath it in the same column, then tests that the computer opponent ‘S’ makes a move in the row beneath ‘O’ to score.

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**
8	8.1	Start 3x3 simple game with blue as	‘S’ is placed in a “random” cell on the	The ‘S’ is placed by the computer opponent as expected.

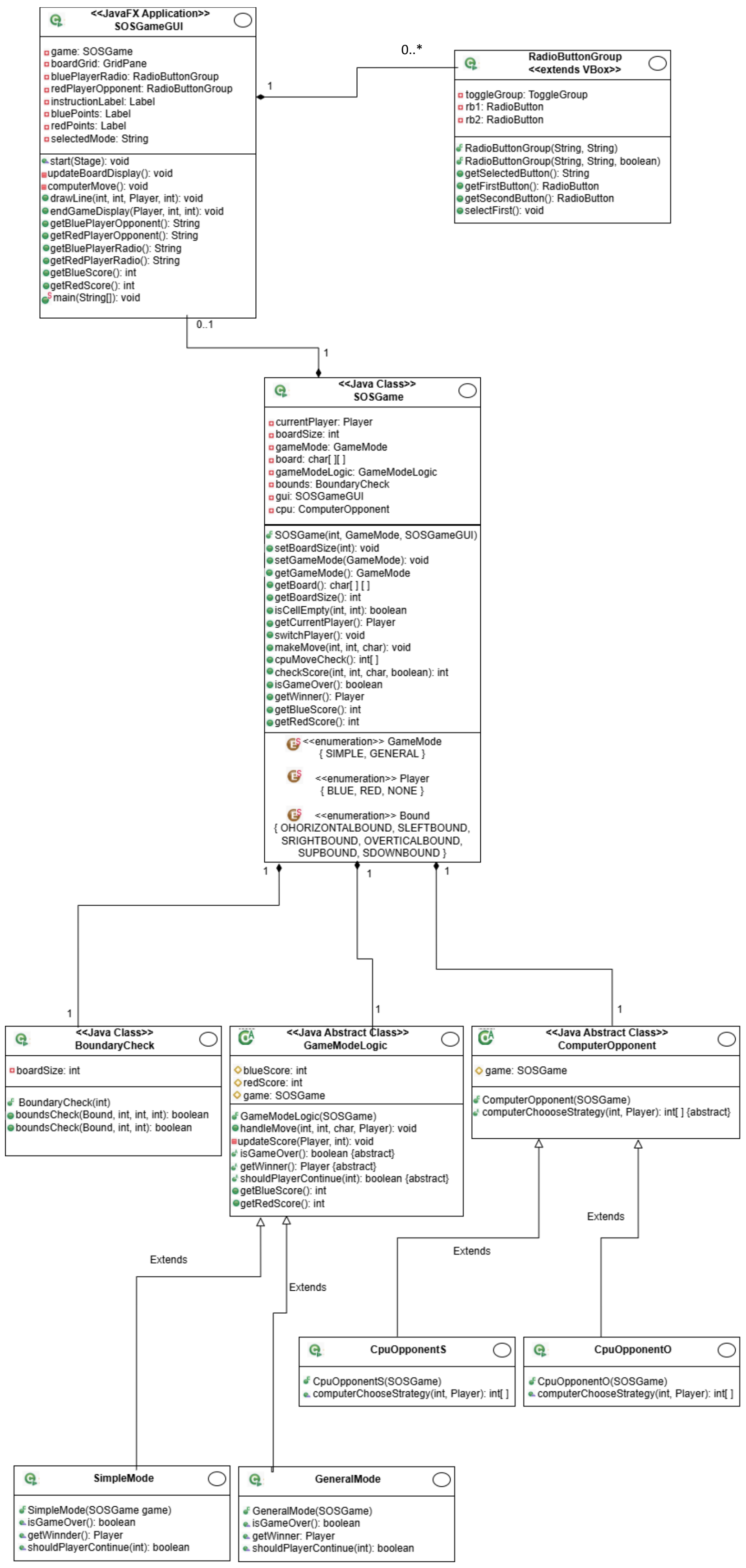
		human ‘O’ and red as computer ‘S’. Human places letter ‘O’ in row 0, column 0.	board.	
		Start 3x3 simple game with blue as computer ‘S’ and red as human ‘O’.	‘S’ is placed in a “random” cell upon selection of New Game.	The ‘S’ is placed in a random cell as expected.
9	9.1	Start 8x8 general game with blue as human ‘S’ and red as computer ‘O’. Human places at 0,0 then computer opponent ‘O’ places randomly, then human places S at 0,2.	The computer selects the cell between the two ‘S’ to form the SOS.	The computer opponent selects the cell between the two ‘S’ as expected.
		Start 8x8 general game with blue as computer ‘O’ and red as human ‘S’. After computer random placement, human places at 0,0 then computer opponent ‘O’ places randomly, then human places S at 0,2.	The computer opponent selects the cell between the two ‘S’ to form the SOS.	The computer opponent selects the cell between the two ‘S’ as expected.

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**

7. 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)?

DIAGRAM AND DESCRIPTION ON NEXT PAGES



<<JavaFX Application>>
SOSGameGUI
game: SOSGame
boardGrid: GridPane
bluePlayerRadio: RadioButtonGroup
redPlayerOpponent: RadioButtonGroup
instructionLabel: Label
bluePoints: Label
redPoints: Label
selectedMode: String
start(Stage): void
updateBoardDisplay(): void
computerMove(): void
drawLine(int, int, Player, int): void
endGameDisplay(Player, int, int): void
getBluePlayerOpponent(): String
getRedPlayerOpponent(): String
getBluePlayerRadio(): String
getRedPlayerRadio(): String
getBlueScore(): int
getRedScore(): int
main(String[]): void
1
0..*
RadioButtonGroup
<<extends VBox>>
toggleGroup: ToggleGroup
rb1: RadioButton
rb2: RadioButton
RadioButtonGroup(String, String)
RadioButtonGroup(String, String, boolean)
getSelectedButton(): String
getFirstButton(): RadioButton
getSecondButton(): RadioButton
selectFirst(): void
0..1
1
<<Java Class>>
SOSGame
currentPlayer: Player
boardSize: int
gameMode: GameMode
board: char[ ][ ]
gameModeLogic: GameModeLogic
bounds: BoundaryCheck
gui: SOSGameGUI
cpu: ComputerOpponent
SOSGame(int, GameMode, SOSGameGUI)
setBoardSize(int): void
setGameMode(GameMode): void
getGameMode(): GameMode
getBoard(): char[ ] [ ]
getBoardSize(): int
isCellEmpty(int, int): boolean
getCurrentPlayer(): Player
switchPlayer(): void
makeMove(int, int, char): void
cpuMoveCheck(): int[ ]
checkScore(int, int, char, boolean): int
isGameOver(): boolean
getWinner(): Player
getBlueScore(): int
getRedScore(): int
<<enumeration>> GameMode
{ SIMPLE, GENERAL }
<<enumeration>> Player
{ BLUE, RED, NONE }
<<enumeration>> Bound
{ OHORIZONTALBOUND, SLEFTBOUND, SRIGHTBOUND, OVERTICALBOUND, SUPBOUND, SDOWNBOUND }
1
1
1
1
1
1
<<Java Class>>
BoundaryCheck
boardSize: int
BoundaryCheck(int)
boundsCheck(Bound, int, int, int): boolean
boundsCheck(Bound, int, int): boolean
<<Java Abstract Class>>
GameModeLogic
blueScore: int
redScore: int
game: SOSGame
GameModeLogic(SOSGame)
handleMove(int, int, char, Player): void
updateScore(Player, int): void
isGameOver(): boolean {abstract}
getWinner(): Player {abstract}
shouldPlayerContinue(int): boolean {abstract}
getBlueScore(): int
getRedScore(): int
<<Java Abstract Class>>
ComputerOpponent
game: SOSGame
ComputerOpponent(SOSGame)
computerChoooseStrategy(int, Player): int[ ] {abstract}
Extends
Extends
Extends
Extends
CpuOpponentS
CpuOpponentS(SOSGame)
computerChooseStrategy(int, Player): int[ ]
CpuOpponentO
CpuOpponentO(SOSGame)
computerChooseStrategy(int, Player): int[ ]
SimpleMode
SimpleMode(SOSGame game)
isGameOver(): boolean
getWinnder(): Player
shouldPlayerContinue(int): boolean
GeneralMode
GeneralMode(SOSGame)
isGameOver(): boolean
getWinner: Player
shouldPlayerContinue(int): boolean

The class hierarchy for the computer opponent requirements uses and abstract base class ComputerOpponent with two subclasses: CpuOpponentS and CpuOpponentO. This is similar to how the game modes were established with GameModeLogic and its subclasses.

ComputerOpponent contains a protected reference to the SOSGame instance, allowing the subclasses to access the game state information such as the board configuration and scoring methods. CpuOpponentS and CpuOpponentO implement nearly identical algorithms, but for their particular letter. This can be expanded upon later to add more complex strategies to the computer opponent. The strategies for blocking moves or making moves to set up better future moves is different depending on the letter assigned, and these strategies could be implemented as functions within each subclass.

Currently, both CpuOpponentS and CpuOpponentO will select only one of the highest scoring moves, with an element of randomness in its selection with the use of the probabilistic condition 0.01 * Math.*pow*(10.0 / boardSize, Math.*log*(5) / Math.*log*(2)). This includes if the highest scoring move is 0. The purpose of the probabilistic condition is to add a level of scaling depending on the board size, to try to create a feeling of random selection that scales with the board size.

The SOSGame class integrates the computer opponent system through its cpuMoveCheck() method. This method uses a switch statement to check if the current player (Blue or Red) is controlled by the computer based on GUI settings retrieved through gui.getBluePlayerOpponent() and gui.getRedPlayerOpponent(). If a computer opponent is active, it checks which letter that player has selected using gui.getBluePlayerRadio() or gui.getRedPlayerRadio().

The method uses instanceof checks to determine if the current cpu object matches the required strategy. If the player has changed their letter selection, the code instantiates the correct subclass.