Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A31

Game C/S Model – Collaboration Diagram

Team:

[Hetvi Patel] - Id: [041077850] / [Dhara Patel] - Id: [041081736]

Battleship C/S Proposal

***This template is suggested (not mandatory) to answer A31 Specification.***

|  |  |
| --- | --- |
| **Part**  **1** | **C/S Architecture** |

* 1. **UC Model (1pt)**

🛠 Update the UC diagram and the UC table (actors and functionalities).

**Fig. 1** – C/S Model for Chat[[1]](#footnote-1)

**UC Diagram** (**update** your UC diagram to reflect the networking changes):

A black and white image of a pyramid

Description automatically generated

**Actors table** (example):

|  |  |
| --- | --- |
| **Actors** |  |
| Host | |  | | --- | | The player who initiates and hosts the game session. | |

**UC table** (example):

|  |  |
| --- | --- |
| **Use Cases** |  |
| Start Game | The Host initiates a new game session or the Client connects to an existing game session. |
| Place Fleet | Both players place their fleets on the game board in their respective grids. |
| Make Move | A player makes a move by selecting a position on the opponent’s grid to attack |
| Send Chat Message | A player sends a chat message to the opponent during the game session. |
| Restart Game | Either player can request to restart the game session after it has ended. |
| Disconnect | Either player can disconnect from the game session, ending the connection between Host and Client. |

* 1. **Class Diagram (2pt)**

🛠 To draw the diagram, you can use tools (ex: <https://app.diagrams.net/>) or desktop applications (ex: Visio / Powerpoint) or simply take photos from drawings.

**Include the updated Class Diagram** (change this diagram to accommodate the actors and functionalities to be used):

A close-up of a document

Description automatically generated

**Fig. 3** – ClassD for a chat

**Class table** (example):

|  |  |
| --- | --- |
| **Class name** | **View** |
| Inner Fields | model: Model  controller: Controller |
| Relationships | **Model:** Uses methods from Model to get game state information and update the display.  **Controller:** Receives instructions from the Controller to update the display and handle user inputs. |
| Methods | View(model: Model, controller: Controller): Constructor to initialize the view with the model and controller.  printMsg(msg: String): Prints a message to the view.  clearMessages(): Clears any displayed messages.  redrawBoard(): Redraws the game board.  updateScores(): Updates the displayed scores.  enterShipPlacementMode(): Enables ship placement mode.  draw(): Draws the game elements.  showGameOver(message: String): Displays a game-over message.  resetView(size: int): Resets the view to a specified size.  enableLifeline(enable: boolean): Enables or disables lifeline functionality.  placeShipIcons(x: int, y: int, length: int, horizontal: boolean): Places ship icons on the board.  setModel(model: Model): Sets the model component.  displayConnectToWindow(): Displays the connection window.  displayAcceptConnectionWindow(): Displays the accept connection window |

|  |  |
| --- | --- |
| **Class name** | **Model** |
| Inner Fields | playerName: String  size: int  currentPlayer: int  board: char[][]  player1Score: int  player2Score: int  lifelineActivated: boolean  lifelineUsed: boolean  player1RedoUsed: boolean  player2RedoUsed: boolean  port: int  ip: String |
| Relationships | **View:** Provides game state information to the View.  **Controller:** Receives updates and game state changes from the Controller.. |
| Methods | * Model(size: int): Constructor to initialize the model with a board size. * makeMove(x: int, y: int): int: Makes a move at the specified coordinates and returns the result. * undoLastMove(): Undoes the last move. * getPlayer(): int: Gets the current player. * nextPlayer(): Switches to the next player. * setCurrentPlayer(player: int): Sets the current player. * tryPlaceShip(player: int, length: int, x: int, y: int, horizontal: boolean): boolean: Attempts to place a ship for the specified player. * get(x: int, y: int): char: Gets the character at the specified coordinates. * getSize(): int: Gets the size of the board. * isGameOver(): boolean: Checks if the game is over. * getBoard(): char[][]: Gets the game board. * isLifelineActivated(): boolean: Checks if the lifeline is activated. * useLifeline(): Uses a lifeline. * isLifelineUsed(): boolean: Checks if a lifeline is used. * swapBoards(): Swaps the boards of the players. * getPlayer1Score(): int: Gets the score of player 1. * getPlayer2Score(): int: Gets the score of player 2. * activateLifeline(): Activates a lifeline. * deactivateLifeline(): Deactivates a lifeline. * isPlayer1RedoUsed(): boolean: Checks if player 1 has used a redo. * setPlayer1RedoUsed(used: boolean): Sets the redo status for player 1. * isPlayer2RedoUsed(): boolean: Checks if player 2 has used a redo. * setPlayer2RedoUsed(used: boolean): Sets the redo status for player 2. * getPlayerName(): String: Gets the player's name. * setPlayerName(name: String): Sets the player's name. * setPort(port: int): Sets the port. * getPort(): int: Gets the port. * setIP(ip: String): Sets the IP address. * getIP(): String: Gets the IP address. |

|  |  |
| --- | --- |
| **Class name** | **Controller** |
| Inner Fields | size: int  model: Model  view: View |
| Relationships | **View:** Sends instructions to update the display and receive user input.  **Model:** Updates the game state based on user input and game rules. |
| Methods | Controller(size: int): Constructor to initialize the controller with a board size.  start(): Starts the game.  main(args: String[]): Main method to run the game.  placePlayerShips(): Places the player's ships.  placeComputerShips(): Places the computer's ships.  mouseClicked(e: MouseEvent): Handles mouse click events.  mousePressed(e: MouseEvent): Handles mouse press events.  mouseReleased(e: MouseEvent): Handles mouse release events.  mouseEntered(e: MouseEvent): Handles mouse enter events.  mouseExited(e: MouseEvent): Handles mouse exit events.  computerMove(): Handles the computer's move.  toggleDirection(): Toggles the ship placement direction.  newGame(): Starts a new game.  revealOpponentShip(): Reveals the opponent's ship.  saveGame(): Saves the game state.  loadGame(): Loads a saved game state.  swapBoards(): Swaps the boards.  redoLastMove(): Redoes the last move.  listenForConnection(port: int): Listens for connections on the specified port. connect(ip: String, port: int): Connects to another player.  sendPlaceShip(x: int, y: int): Sends ship placement coordinates. receivePlaceShip(x: int, y: int): Receives ship placement coordinates.  makeMove(x: int, y: int): Makes a move.  sendMove(x: int, y: int): Sends a move.  receiveMove(x: int, y: int): Receives a move.  sendMoveResult(x: int, y: int, result: int): Sends the result of a move.  receiveMoveResult(x: int, y: int, result: int): Receives the result of a move.  resetGame(): Resets the game.  sendResetRequest(): Sends a reset request.  receiveResetRequest(): Receives a reset request.  sendChatMessageToServer(chatMessage: String): Sends a chat message to the server.  sendChatMessageToClient(chatMessage: String): Sends a chat message to the client.  receiveChatMessageFromClient(chatMessage: String): Receives a chat message from the client.  receiveChatMessageFromServer(chatMessage: String): Receives a chat message from the server.  disconnectServer(): Disconnects from the server.  disconnectClient(): Disconnects from the client.  sendDisconnectToClient(): Sends a disconnect message to the client.  sendDisconnectToServer(): Sends a disconnect message to the server.  receiveDisconnectFromClient(): Receives a disconnect message from the client.  receiveDisconnectFromServer(): Receives a disconnect message from the server.  calculateWinLossOrTie(): Calculates the win/loss/tie status.  sendWinLossOrTieMessageToClient(): Sends the win/loss/tie message to the client.  receiveWinLossOrTieMessageToServer(): Receives the win/loss/tie message from the server. |

*Update tables for all classes.*

* 1. **Updated Translation Tables**

*Include the A21 tables, but include the new translations required as well.*

Translation table :

|  |  |  |
| --- | --- | --- |
| Location | English Title | French Translation |
| File Menu | File | Fichier |
| File Menu | Save | Enregistrer |
| Game Menu | Game | Jeux |
| Game Menu | Reset | Recommencer |
| File Menu | Network | Réseau |
| File Menu | Language | Langue |
| File Menu | Help | Aide |
| Game Menu | Lifeline | Lifeline |
| Game Menu | Redo move | Rétablir le déplacement |
| Game Menu | Send Button | Bouton d’envoi |
| Game Menu | Swap Button | Bouton d’échange |

|  |  |
| --- | --- |
| English Title | French Translation |
| Connect | Connecter |
| Disconnect | Déconnecter |
| Host | Hôte |
| Connection refused | Connexion Refusée |
| Invalid Host | Hôte Invalide |
| Waiting for player | En attente du joueur |
| Opponent's Turn | Tour de l'adversaire |
| Game Over | Jeu terminé |
| Send Message | Envoyer un message |
| Start Game | Commencer le jeu |
| Restart Game | Redémarrer le jeu |
| Place Fleet | Placer la flotte |
| Enter Player Name | Entrez le nom du joueur |
| Enter Host IP Address | Entrez l'adresse IP de l'hôte |
| Enter Port Number | Entrez le numéro de port |
| Player Connected | Joueur connecté |
| Player Disconnected | Joueur déconnecté |

* 1. **Networking Messages**

*What messages are required? What format are your network messages?*

|  |  |
| --- | --- |
| **Message Format** | **Description** |
| Message Type | (How will a message type be indicated? Digits, letters, other  The first word of the message will be the message type |
| Separator | (What separator character, if any?)  Colon |

*(Duplicate this table for each type of message.)*

|  |  |
| --- | --- |
| **Message** |  |
| Content format | place:<playername>:<x coordinate>:<y coordinate> |
| Description/purpose | Sends the player ship placement to the server |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | isReady:<playername> |
| Description/purpose | Informs the server that player is ready |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | click:<playername>:<x coordinate>:<y coordinate> |
| Description/purpose | Sends the player move to the server |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | move:<playername>:<x coordinate>:<y coordinate>:<result> |
| Description/purpose | Sends the player move from the server to other player and the result of the move,0 for miss, 1 for hit |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | ready:<playername> |
| Description/purpose | Server Informs the other player that player is ready |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | result:<0|1|-1|2|-2|3> |
| Description/purpose | Informs the player of the result of his move  0 for miss, 1 for hit, -1 invalid move,2 for win, -2 for loss, 3 for tie |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | reset:<playername> |
| Description/purpose | Player request a reset. |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | sendMessage:<playername>:<message> |
| Description/purpose | Player sends a chat message to the server |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | message:<playername>:<message> |
| Description/purpose | Server informs a player that another player has sent a message |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | wantDisconnect:<playername> |
| Description/purpose | Player informs the server that he wants to disconnect |

|  |  |
| --- | --- |
| **Message** |  |
| Content format | disconnect:<playername> |
| Description/purpose | Server informs the player that other player wants to disconnect |

**References**

*Deep AI to learn more about the methods in Networking Section how to connect the server and client using codes.*

Algonquin College

Summer, 2024

1. See <https://www.researchgate.net/figure/System-Architecture-Use-Case-Diagram-Client-side-Functionality_fig2_318502492>. [↑](#footnote-ref-1)