Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A31

Game C/S Model

Team:

Isaac Proulx - Id: 041007853

Picross Proposal

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

* 1. **Server Model**

Server View and Controller

INTERFACE:

Class: ServerView

→ Components:

VBox: mainPane:

ImageView: logo,

HBox: controlPane:

Label: portLabel,

TextField: portText,

Button: startButton,

Button: resultsButton,

CheckBox: finalizeBox,

Button: endButton

TextArea: logArea

→ Method: showError: // should display the provided error message

→ Method: showResults: // should display the game results

→ Method: getPortText:

Return portText.getTextProperty();

→ Method: enableStart:

startButton.setDisable(false);

→ Method: disableStart:

startButton.setDisable(true);

→ Method: log:

logArea.appendText(logText);

CONTROLLER:

Class: ServerController:

→ GameServer: server

→ ServerView: view

→ ArrayList<GameClient>: clients

→ boolean: finalize

→ Method: Start: // triggered by startButton

**server** = new GameServer (Integer.parseInt(view.getPortText())); //also need to validate port number

**server**.start();

view.disableStart();

→ Method: end: // triggered by endButton

server.close();

view.enableStart();

→ Method: toggleFinalize: // triggered by finalizeBox

finalize = !finalize;

→ Method: showResults: // triggered by resultsButton

* 1. **Client Model**

Client View and Controller

INTERFACE:

Class: ClientView

→ Components:

VBox: mainPane:

ImageView: logo,

HBox: controlPane:

Label: userLabel, serverLabel, portLabel,

TextField: userText, serverText, portText,

Button: connectButton, endButton, newGameButton,

Button: sendGameButton, sendDataButton, playButton,

TextArea: logArea

→ Method: getUserText:

return userText.getTextProperty();

→ Method: getServerText:

return serverText.getTextProperty();

→ Method: getPortText:

return portText.getTextProperty();

→ Method: disableConnect:

connectButton.setDisable(true);

→ Method: enableConnect:

connectButton.setDisable(false);

→ Method: log:

logArea.appendText(logText);

→ Method: showError: // should display the provided error message

CONTROLLER:

Class: ClientController

→ Socket: client

→ Method: Start:

try {

**client** = new Socket(hostName, portNumber);

}

→ Method: end:

client.close();

* 1. **Protocol Proposal**

CONFIGURATION STRING:

Class: GameModel

→ Property: String: gameConfig:

→ Format: <dim><dataConfig>, where:

→ <dim> 1 byte

→ <dataConfig> ceil((<dim>\*<dim>)/8) bytes

→ Example: \x05\x00\x4C\xE0\xEC

PROTOCOL P1:

→ idLength: 1 byte

→ clientId: idLength bytes

→ opCode: 1 byte (the operation to perform)

→ dataLength: 4 bytes

→ data: dataLength bytes (data to be used by the operation)

→ Format: <idLength><clientId><opCode><dataLength><data>

→ Example: \x04AAAA\x01\x00\x00\x00\x0CDataGoesHere

|  |  |
| --- | --- |
| **Part**  **2** | **Game Evolution** |

* 1. **Notes about upgrading the game**

For the multiplayer to work, some new methods will need to be added to the game so that the client app can interface with it. Data will need to be sent to and from the server, so the model will likely need more getters and setters to allow this. New methods will need to be added to the view to allow visual components to be toggled remotely, for example: showing game results.

Algonquin College

Winter, 2023