Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A31

Game C/S Model

Team:

[Skylar Phanenhour] - Id: [041024402] / [Ahnaf Kamal] - Id: [041034479]

Picross Proposal

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

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

* 1. **Server Model**

*Describe how your server interface should be organized and the main methods to be defined*

* + - ***Example****:*

**Example** (see A31 specification)

INTERFACE:

Class: GameServer

→ Components: JFrame: serverWindow, JPanel: titlePanel,

ImageIcon picrossLogo, JLabel picrossLabel,

JPanel: serverPanel, JPanel: serverButtonPanel,

JLabel: serverPortLabel, JTextField: serverPortText,

JButton leaderboardButton, JButton disconnectButton, JButton endConnections,

JPanel: serverLogPanel, JTextArea logTextArea, JScrollPane logScroll

CONTROLLER:

Class: GameServer – Object: “**server**”

→ Method: Start:

try (

GameServer **server** = new GameServer (portNumber);

GameClient client = **server**.accept();

}

→ Method: End:

GameServer.close();

GameClient.close();

→ Method: Leaderboard:

model.getBestTime();

model.getBestScore();

model.getUsername();

→ Method: Disconnect:

GameServer.close();

MODEL:

→ Method: getBestTime:

→ Method: getBestScore:

→ Method: getUsername:

* ***Note****: The professor interface continues being a proposal. Focus on your ideas using the best user experience.*
  1. **Client Model**

*Describe aspects of your client (interface and methods) considering the proposed idea.*

**Example** (see A31 specification)

INTERFACE:

Class: GameClient

→ Components: JFrame: clientWindow, JPanel: titlePanel,

ImageIcon picrossLogo, JLabel picrossLabel,

JPanel: clientPanel, JPanel: clientButtonPanel,

JLabel: clientUserLabel, JTextField: clientUserText,

JLabel: clientServerLabel, JTextField: clientServerText,

JLabel: clientPortLabel, JTextField: clientPortText,

JButton clientConnect, JButton clientEnd, JButton clientPlay, JButton clientLoad,

JButton clientSendGame, JButton clientSendData, JButton clientNewGame (design mode),

JPanel: serverLogPanel, JTextArea logTextArea, JScrollPane logScroll

CONTROLLER:

Class: GameClient – Object: “**client**”

→ Method: Connect:

try {

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

} …

→ Method: End:

GameClient.close();

* 1. **Protocol Proposal**

*Finally, what is your idea to define the protocol to be used.*

**Example** (using the string definition mentioned in the A21 specification)

CONFIGURATION STRING:

Class: GameModel

→ Property: String: clientConnect:

→ Format: <hostName><dataSeperator><portNumber>, where:

→ <hostName> = string of the IP that the client is connecting

→ <dataSeparator> = comma (,)

→ <portNumber> = sequence of integers

→ Example: localhost,12345

PROTOCOL P0: // when client is connecting

→ protocolSeparator: hashtag (#)

→ Format: <clientId><protocolSeparator><data>

→ Example: 1#locahost,12345

------------------------------------------------------------------------------------------------------------------------------------------------

→ Property: String: gameConfig:

→ Format: <dim><dataSeparator><dataConfig>, where:

→ <dim> = integer (from 2, 3, etc.)

→ <dataSeparator> = comma (,)

→ <dataConfig> = chars (example: 1-9), obeying the formula (dim2)2.

→ Example: 5,00001,10111,00110,11111,00011

PROTOCOL P1:

→ protocolSeparator: hashtag (#)

→ Format: <clientId><protocolSeparator><data>

→ Example: 1#5,00001,10111,00110,11111,00011

-------------------------------------------------------------------------------------------------------------------------------------------------

→ Property: String: serverReply:

→ Format: <clientId><dataSeparator><reply>, where:

→ <clientId> = integer of the protocol that the client is sending

→ <dataSeparator> = comma (,)

→ <reply> = string message that is sent back to the client

→ Example: 1,“The user has successfully connected to the server”

PROTOCOL P2:

→ protocolSeparator: hashtag (#)

→ Format: <clientId><protocolSeparator><data>

→ Example: 1#1,“The user has successfully connected to the server”

-------------------------------------------------------------------------------------------------------------------------------------------------

→ Property: String: clientGameData:

→ Format: <username><dataSeparator><bestTime><dataSeparator><bestScore>, where:

→ <username> = string of client’s entered name

→ <dataSeparator> = comma (,)

→ <bestTime> = integer of the client’s best time, converted to seconds

→ <bestScore> = integer of the client’s best score

→ Example: testUser,61,10

PROTOCOL P3:

→ protocolSeparator: hashtag (#)

→ Format: <clientId><protocolSeparator><data>

→ Example: 1#testUser,61,10

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

* 1. **Notes about upgrading the game**
  + *Describe the main modifications to be proposed in the C/S version of the game.*
    - *What are the differences between the original proposal (A11 / A21) and the current project to be developed (A31).*
    - *If so, explain why you need to do some adjustments.*

**Example** (About MVC modifications)

MODEL component:

portNumber, hostname, GameServer, GameClient

VIEW component:

We need to change our launcher to be able to include a way for the user to connect a client to the server or start up the server. When either the client or server buttons are clicked, separate JFrames will open that will have their own visual components.

We also have to add a chat input text field in our play game, should be placed under our history. This was planned in A11/A21 but had to be removed during implementation.

CONTROLLER component:

Actions listeners for all of the buttons/components. Allow the connections between the clients and server.

Input validator for the text fields, to make sure that valid inputs are being accepted.

General Notes:

When developing A31 and A32, we need to make two new classes, one for the GameServer, and one for the GameClient.

We will be implementing a database for our game leaderboards that stores the username, user’s best time and best score.

The GameClient will have a GUI that will allow the user to enter their name, server and port they want to connect to. There will be various buttons that will allow them to load, save, send/receive data with the server, play the game, design mode, and end the connection with the server (if connected). A text area will be included to communication with the server.

The GameServer will have a GUI that will establish the port number, buttons to start the server connection, leaderboard button that will display all users data (sorted by best score), a disconnect server, and end client connection threads.

The GameServer will also keep track of the player names and their scores, to display to all clients when the game is over. The clients will stored in an arraylist.

* 1. **GitHub / Database Integration (Bonus)**
  + *The use of GitHub is also a bonus to be considered:*
    - *Be sure that you can inform the updated repository and branch.*
    - *TIP: To avoid problems, also include the document (template answer) in the BrightSpace.*
  + *Considering this proposal for 3-tier architecture using Databases, define:*
    - *What to persist.*
    - *What is the DB datatype to be used.*
    - *How frequently to update.*

**References**

*[Include eventual references used here]*

Algonquin College

Winter, 2022