Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A31

Game C/S Model

Team:

Joshua Fearnall - Id: [041019251]

NumPuz 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: NumPuzServer

→ Components: JLabel: labPort, JTextField: txtPort, etc.

// CONTINUE…

CONTROLLER:

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

→ Method: Start:

try (

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

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

}

// CONTINUE…

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

INTERFACE:

Class: GameServer

Components: JLabel: serverImage, JLabel: portLabel, JTextField: portTextField, JButton: startButton,

JButton: resultsButton, JCheckBox: finalizeCheckBox, JButton: endButton, JTextArea: informationTextArea

CONTROLLER:

class GameServer **{**

ArrayList**<**Socket**>** clientList **=** **new** ArrayList**<**Socket**>(**5**);**

int port**;**

ServerSocket serverSocket **=** **null;**

Socket clientSocket **=** **null;**

DataOutputStream output **=** **null;**

DataInputStream input **=** **null;**

// Starting server

public void startServer**()** **{**

**try** **{**

serverSocket **=** **new** ServerSocket**(**port**);**

**}** **catch** **(**IOException ioe**)** **{**

System**.**out**.**println**(**"IO Exception: " **+** ioe**);**

**}**

waitForConnection**();**

**}**

// Shutting down server

public void stopServer**()** **{**

**try** **{**

**for** **(**int i **=** 0**;** i **<** clientList**.**size**();** i**++)** **{**

clientList**[**i**].**close**();** // interrupt threads?

**}**

output**.**close**();**

serverSocket**.**close**();**

**}** **catch** **(**IOException ioe**)** **{**

System**.**out**.**println**(**"IO Exception: " **+** ioe**);**

**}**

**}**

// Wait for client connections

public void waitForConnection**()** **{**

**while** **(true)** **{**

**try** **{**

clientSocket **=** serverSocket**.**accept**();**

**}** **catch** **(**IOException ioe**)** **{**

System**.**out**.**println**(**"IO Exception: " **+** ioe**);**

**}**

**new** GameClientThread**(**clientSocket**).**start**();**

clientList**.**add**(**clientSocket**);**

**}**

**}**

**}**

// Create new thread for client

class GameClientThread **extends** Thread **{**

Socket threadSocket**;**

public GameServerThread**(**Socket clientSocket**)** **{**

**this.**threadSocket **=** clientSocket**;**

**}**

public void run**()** **{**

// listen for client input?

**}**

**}**

* 1. **Client Model**

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

**Example** (see A31 specification)

INTERFACE:

Class: NumPuzClient

→ Components: JLabel: labUser, JTextField: txtUser, etc.

// CONTINUE…

CONTROLLER:

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

→ Method: Start:

try {

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

} …

// CONTINUE…

INTERFACE:

Class: GameClient

Components: JLabel userLabel, JTextField userTextField, JLabel serverLabel, JTextField serverTextField,

JLabel portLabel, JTextField portTextField, JButton connectButton, JButton endButton,

JButton newGameButton, JButton sendGameButton, JButton receiveButton,

JButton sendDataButton, JButton playButton, JLabel clientImage,

JTextArea informationTextArea

CONTROLLER:

class GameClient **{**

Socket clientSocket **=** **null;**

int port**;**

String host**;**

DataInputStream input **=** **null;**

DataOutputStream output **=** **null;**

// Connect to server

public void connectToServer**()** **{**

**try** **{**

clientSocket **=** **new** Socket**(**host**,** port**);**

output **=** **new** DataOutputStream**(**client**.**getOutputStream**());**

input **=** **new** DataInputStream**(**client**.**getInputStream**());**

**}** **catch** **(**UnknownHostException uhe**)** **{**

System**.**out**.**println**(**"Unknown Host Exception: " **+** uhe**);**

**}** **catch** **(**IOException ioe**)** **{**

System**.**out**.**println**(**"IO Exception: " **+** ioe**);**

**}**

**}**

// Disconnect from server

public void disconnectFromServer**()** **{**

**try** **{**

output**.**close**();**

client**.**close**();**

**}** **catch** **(**IOException ioe**)** **{**

System**.**out**.**println**(**"IO Exception: " **+** ioe**);**

**}**

**}**

// Request game config from the server

public void requestGameConfig**()** **{**

// send a message requesting the game config

// wait to receive game config

**}**

// Send game config to the server

public void sendGameConfig**()** **{**

// send a message containing the game config

**}**

// Send game data to the server

public void sendGameData**()** **{**

// send a message containing the game data

**}**

**}**

* 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: NumPuzModel

→ 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:

numerical;1,2,3,4,5,6,7,8,0

text;M,y, ,g,a,m,e,!,•.

PROTOCOL P1:

→ protocolSeparator: hashtag (#)

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

→ Example: 1#3,numerical;1,2,3,4,5,6,7,8,0

**PROTOCOL 0 P0**: (CLIENT CONNECTING TO SERVER)

clientId = integer (0, because the client has not yet connected)

protocolID = integer (0, ID for connection)

protocolSeparator = #

Format <clientId><protocolSeparator><protocolId>

Example: 0#0

**PROTOCOL 1 P1**: (CLIENT SENDING GAME CONFIG)

protocolID = integer (1, ID for sending game config)

protocolSeparator = #

clientId = integer (1 to N)

data = string in the format (dimension,type,data)

Format: <clientId><protocolSeparator><protocolId><protocolSeparator><data>

Example: 1#1#3,Text,h,e,l,l,o, ,w,o,r,

**PROTOCOL 2 P2**: (SERVER REPLY P1)

protocolSeparator = #

clientId = integer (1 to N)

data = string message ( connected to server )

Format: <clientId><protocolSeparator><data>

Example: 1#Connected to server,

**PROTOCOL 3 P3**: (CLIENT SENDING GAME DATA)

protocolID = integer (2, ID for sending game data)

protocolSeparator = #

clientId = integer (1 to N)

data = string in the format (username, points, time)

Format: <clientId><protocolSeparator><protocolId><protocolSeparator><data>

Example: 1#2#Paulo,8,17,

|  |  |
| --- | --- |
| **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.*

**MODEL**:

We already have access to the data needed from the model in the controller, so we don’t need to make any access changes here. We might need to verify the data sent from the server however.

**VIEW**:

The initialization methods are called from the controller, so we don’t need to worry about changing anything in there.

**CONTROLLER**:

The controller needs to be able to connect to the client application, it should be able to send data from the model to the client application, and receive data from the client application, send it to the model, and start a new game with the new data.

We also need a way to send the results of a game to the client application, to send it to the server.

**Example** (About MVC modifications)

MODEL component:

Public methods to change private data (ex: dataConfig), that can receive inputs, but evaluate if they are valid.

// CONTINUE…

* 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

Spring / Summer, 2022