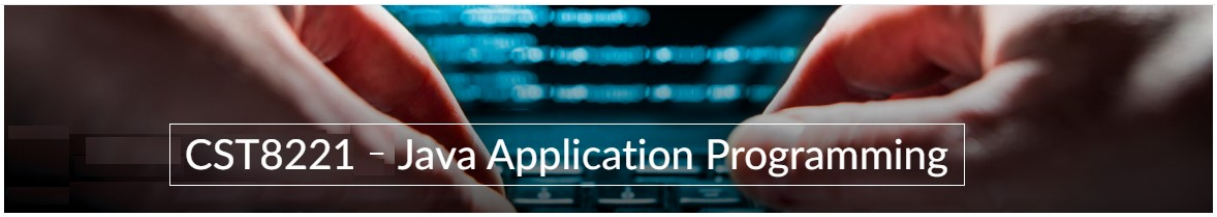
ALGONQUIN COLLEGE



ASSIGNMENT 3.1 – C/S Model (Picross[1])

General View

Due Date: prior or on **Apr 2nd 2023 (midnight)**

- **2nd Due date** (until Nov 9th 2023) - 50% off.

Earnings: 3% of your course grade (4% with bonus).

Purpose: Define the Client-Server model for the Picross Game.

- This is the 5th task in JAP. The purpose is to define the C/S architecture for the game.
- **PART I:** Considering the GUI defined in the A21 (MVC model) and implemented in the A22, define the C/S architecture to be used:
  - Your **Server** is responsible for storing a puzzle (game configuration) to all clients, as well as basic information about each client.
  - Your **Client** can send/receive puzzle and also send data to the server and, obviously, invoke the MVC solution to **design** and to **play** the game.
- **PART II:** Considering your application:
  - Detail the differences between your new project (C/S) application and the previous configuration (standalone MVC).
  - Define what are the eventual problems to be solved during C/S communication.
  - BONUS (additional 1%): Evaluate strategies to use 3-tier architecture using database.

Part I – C/S Architecture to the Game (2pt)

In this first activity, define the layout and main actions to be developed in the C/S solution.

[1] *Check the A11 and A21 specifications for more details.*

1.1. Basic C/S Model

The new version of **Picross** must use **C/S** (Client/Server). The idea is that the MVC Game implemented in the A22 can be played using puzzles coming from one Server, by accessing the method to create **random** configurations.

Note 1: The C/S implementation
We need to maintain the previous MVC solution, but two different applications must be created: The Server and the Client. Only the client is supposed to communicate with the standalone application.

1.2. Basic Components (Server)

The **Server** has very basic functionalities (Fig.1):

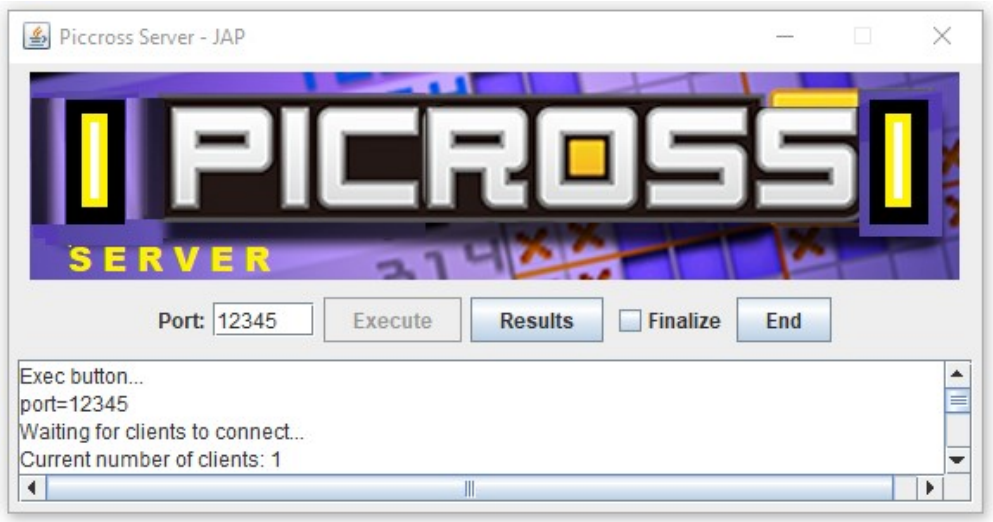


***Fig. 1** – Example of Server interface*

Basically, you need to be able to accept multiple connections between clients. Each client must have one specific ***thread*** to work by using one proper protocol.

- **TODO**:
  - About interface:
    - Define the list of all components (**Swing / JavaFX**) to be used in your server interface.
  - About the controller:
    - **Basic components:** Define the methods (basic actions to be done) when your server:
      - Start: To initialize the server in one specific **port**;

- End: To finish the connection (closing all threads), disabling the EXIT_ON_CLOSE in the JFrame.

▪ **Additional features:** Used to improve elements:

- Results: To show the ranking list of participants (you can sort by alphabetic list or decrescent number of movements, etc.).
- Finalize: When all clients have disconnected, the server will shut down. **Example** (considering the interface in the Figure 1)

```
INTERFACE:
Class: GameServer
→ Components: JLabel: labPort, JTextField: txtPort, etc.
// CONTINUE…

CONTROLLER:
Class: GameServer – Object: "server"
→ Method: Start:
try (
   GameServer server = new GameServer(portNumber);
   GameClient client = server.accept();
}
// CONTINUE…
```

1.3. Basic Components (Client)

The **Client** has very basic functionalities (Fig.2) and it is the only component that can interact with the MVC Solution (see assignments A21 and A22) and must identify at least:

[1] About **networking connection**: the user, the server, the port (for server connection), as well as methods to stablish connections and to finish the client execution.

▪ **Basic components:** Basic communication:

- Connect: To initialize the connection between client and server (receiving one proper Thread for each client);
- End: To finish the connection, sending a message for the server (that must close the thread).

[2] About **game definition and communication**: the new game definition, the communication with server (sending / receiving game configuration), data communication (passing to the server basic game results: points and time) and the game play itself.

▪ **Additional features:** Used to improve elements:

- Send/Receive Game: To transmit and receive the game configuration created (remember the **Design** mode in the MVC).
- Play / Send Data: To be able to invoke the Game (using a previous configuration) and transmit details when the game is executed (**Play** mode)

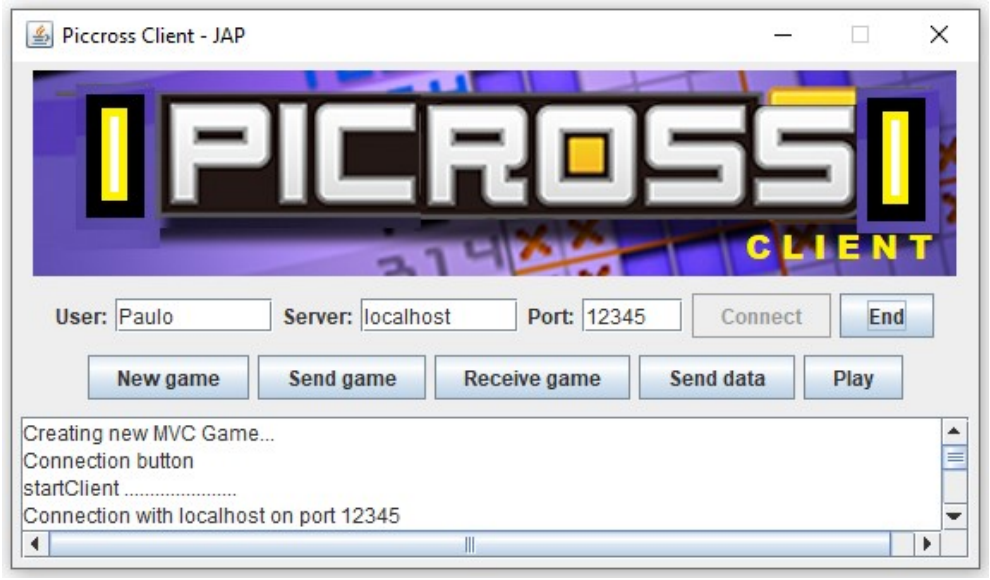


Fig. 2 – Example of Client interface

❖ **TODO**:

➢ As done for the server, define the visual components and the methods to be used in the client controller.

Example (using the interface in the Figure 2)

INTERFACE:

Class: GameClient

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

// CONTINUE…

CONTROLLER:

Class: GameClient – Object: "**client**"

→ Method: Start:

```
try {
    GameClient client = new Socket(hostName, portNumber);
} …
// CONTINUE…
```

1.4. Basic Protocol

Our networking solution is very simple. However, we need to be able to define a minimum protocol defining rules. In fact, to send each message (from client to server or vice-versa) it is necessary to propose a protocol and to obey a proper format[2]. For example:

- **Client communication format (request):**

  <Client_Id><Separator><Protocol_Id>[<Separator><Data>].

- **Server communication format (response):**

  <Client_Id><Separator>[<Data>].

❖ **TODO**:

➢ For each kind of request/response, **propose** one format:

- Protocol 0 (**P0**): When client is connecting with the server;
- Protocol 1 (**P1**): When client is sending a game configuration to server;
- Protocol 2 (**P2**): When server is replying P1.
- Protocol 3 (**P3**): When client is sending game data (user name, points and time) to the server.

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

CONFIGURATION STRING:

Class: GameModel

→ Property: String: gameConfig:

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

→ <dim> = integer (from 2, 3, etc.)

→ <dataSeparator> = comma (,)

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

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

PROTOCOL P1:

→ protocolSeparator: hashtag (#)

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

→ Example: 1#3; 00001,10111,00110,11111,00011

[2] *The protocol present is not the unique way to define the communication between clients and server. However, if they are defined differently, it is impractical to test using the same server.*

It is important to emphasize that during almost all messages between client and server, the identification must be provided and then, correct updates can be done, especially considering the scenario for multiple clients.

Note 2: About Protocols
Is it necessary to define **protocols** for different actions? Basically, it depends on the complexity of the service. If you just need to provide one kind of information (ex: a binary String), you can use only one single kind of message. However, in order to assist multiple clients, the identification is required to be included.

Part II – Updating the Project by C/S Model (1pt)

Once you have proposed the basic elements from the C/S architecture, it is time to think about the new challenges.

2.1. Modifications and Problems

Since your MVC game in the "Play" mode must use configurations set by the server, describe some details.

❖ **TODO**:

- What are the modifications to be done in the MVC?
  - **TIP**: Remember that you need also to be able to send data about results (points, time) to the server.
- Since your C/S uses network, consider all possible error scenarios that can happen.
  - **TIP**: Use the traditional way to create C/S communication and define problems that can happen in each component. Include also problems about wrong configuration used in the protocol.
- Imagine alternatives to solve (or minimize impacts) in each kind of error.
  - **TIP**: Remember problems about shared data, threads, and what are the techniques used to help us (ex: synchronized modifier, try-catch, etc).

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…

2.2. Thinking about 3rd Layer (DB) - Bonus

The use of C/S architecture can be upgraded to include Databases (DB).

- **TODO**:
  - What is supposed to be persisted?
    - **TIP**: Think about data from your Model class.
  - What are the datatypes to be used in the DB?
  - In which moments this data can be INSERT, UPDATED?

Example (About Data)
GAME CONFIG data: To be used in the Model, the game configuration (ex: dataConfig), must be defined as varchar[100], to be updated when a new game is created (manually or randomly). // CONTINUE…

Part III – Documentation and Submission

Since you are in another document proposal, focus on answer the "**TODO**" activities mentioned in this specification. Remember: right now, you do not need to implement nothing.

- **SUMMARY:** In short, create a document, where you specify the details of your C/S solution.

Note 2: About Teams
Only teams already defined are allowed (and just one member should submit). It means that, if you decided to work alone, you will continue until the end of the course.

Evaluation

- Please read the Assignment **Submission Standard and Marking Guide** (at "Assignments > Standards" section).
- **About Plagiarism**: Your code must observe the configuration required (remember, for instance, the "splash screen" using your name. Similarly, we need to observe the policy against ethic conduct, avoiding problems with the 3-strike policy.

Marking Rubric

Maximum Deduction (%)	Deduction Event
-	**Plagiarism:**
Check	3-strike policy[3] (**AA32**, **SA07** and **IT01**)
-	**Severe Errors:**
1.5 pt	Late submission (after 1 week due date)
3.0 pt	Missing demo (zero[4])
-	**Assignment Elements:**
1 pts	Missing architectural elements (Client)
1 pts	Missing details in the Protocol
Up to 1 pt	Other errors
-	**Part II – Game Evolution**
Up to 1 pt	Problems with MVC integration
Up to 1 pt	Missing scenarios in the protocol communication
Up to 1 pt	
1 pt	Other errors (ex: missing components)
1 pt	Github utilization
-	**Bonuses**
Up to +1 pt	Bonus: original ideas developed by language.
Final Mark	**Formula:** 3*((100- ∑ penalties + bonus)/100), max score 5%**.**

Submission Details

- **Digital Submission: Compress** into a **zip** file with all files (including document).
- Upload the zip file on Brightspace. The file must be submitted prior or on the due date as indicated in the assignment.

File update: Mar 12th 2023.

Good luck with A31!

[3] The plagiarism detection will imply in the "3-strike" policy: starting with ZERO, then moving to course failure or program cancelation (see the Algonquin College documents: https://www.algonquincollege.com/policies/).
[4] If a course requires demos, they are not optional. If a student does not demo their work, they should receive a grade of 0 on that assessment, not a grade reduction.