ESIR SR-S9 – Projet Monitoré

François Taiani (`ftaiani.ouvaton.org`)*

1 Objective - Overview

Implementing a distributed game that is both efficient and robust is difficult, in particular on wide-area networks (WAN). To learn more about current technological solutions, you can read the following two on-line articles:

- "Le netcode des jeux de combat" (in French)
  `https://basgrospoing.fr/fr/articles/le-netcode-des-jeux-de-combat`
- "Peeking into VALORANT's Netcode"
  `https://technology.riotgames.com/news/peeking-valorants-netcode`

In this project, you are asked to design and implement a simple distributed game. By default, you do not need to implement any of the delaying or rollback strategies mentioned in the above articles (as your game will in most cases only be deployed in a LAN setting), but you can explore these issues to gain bonus points if you wish to (see just below).

To solve this problem, you can decide to use a standard client-server design code in Java RMI (standard design). You can also decide to explore more adventurous designs (e.g. peer-to-peer), languages / technologies (Scala with Akka agents, Erlang, Node.js, a PaaS platform, GWT, Kotlin, Apache Thrift, etc.), or mechanisms (delays, rollbacks). Choosing a more adventurous design will earn you extra marks. Precise marking details are provided at the end of this document.

2 Objective: Gaming System

Your system should implement the following game: several players share a field strewn with sweets (Figure 1). The player who collects the most sweets wins. A game ends when there are no sweets left in the game. Note that two players cannot be at the same position, and that one sweet can only be collected by at most one player.

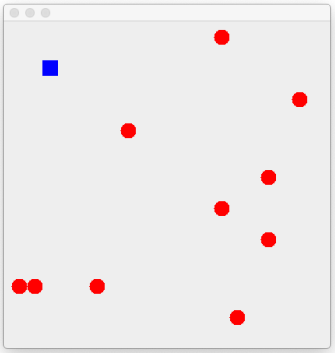

Figure 1: A game field with 10 sweets and one player

If you choose a standard client-server architecture, this gaming system should consist of a central gaming server and one client program:

*francois.taiani@irisa.fr

- The client program should enable a player to connect to a gaming server, and start collecting sweets, while displaying the position of sweets and other players. The client should display the player's current score, and whether the player is currently winning (i.e whether she has the top current score).
- The gaming server should provide the game logic, and handle client connections and disconnections. When a game ends, the gaming server should notify all clients of who the winner is, and start a new game.

(All machines will play the two roles in case of a peer-to-peer design.)

In addition to the above, you should also build an automated suite of *tests* to assess the robustness of your game engine. Testing a distributed application can be challenging, and you might need to include dedicated methods, interfaces, and stubs (bouchons) to improve the testability and controllability of your code.

3 Development Process and Deployment

- You are asked to create and use a repository throughout your project to manage your code on the ESIR/ISTIC GitLab platform `https://gitlab.istic.univ-rennes1.fr`. All members of your pair should commit and contribute to the repository in the course of the project. You should also set your repository as private, and (ii) invite your lab tutor to your repository.
- You are strongly encouraged to adopt an agile philosophy for your development, by adopting two best practices: (i) draw up a road-map of your development in which each step should deliver a deployable, and executable artefact with some minimal user-facing functionality; (ii) be sure to traverse **all technological layers** of your project in each step of your roadmap (including the first one), to uncover any potential blocking factors or incompatibilities. (Sometimes called a tracer bullet strategy.)
- To deploy your system in a real distributed infrastructure, you are encouraged to use the ESIR/ISTIC VM creation service at `https://vm.istic.univ-rennes1.fr`. Be sure to respect any security advice, and to harden your ssh access (see for instance `http://acmeextension.com/secur-ssh-server/`) when managing your VM(s) to avoid any security breach.

4 Notes

- **Graphical interface:** You are **not** expected to provide a sophisticated user interface for the client program (simple geometric shapes are ok). A basic straw man example demonstrating how to display a geometric shape that can be moved with the keyboard is provided as a starting point.
- **Concurrency:** Your game engine should ensure that concurrent actions by the clients always result in a consistent state of the server.
- **Peer-to-peer:** If you opt for a decentralized (aka peer-to-peer) architecture, you will greatly benefit from a group communication (or broadcasting) library. JGroup (`http://www.jgroups.org/`) is one such mature library for Java (a quick tutorial about JGroups is available on the course's site). Spread (`http://www.spread.org/`) is another C/C++ group communication library with Python and Java APIs.
- **Tests:** Your tests should particularly focus on **end-to-end tests** that exercise all the technological layers of your game.
  - You may for instance adopt a **chaos engineering** strategy and develop an automated player client that randomly moves its player across the gaming field (monkey test). Launching many such clients during a predefined period may help you discover (simple) concurrency or performance bugs.

- Going one step further, you may script specific scenarios, that start from a predefined game state, to verify that your game behaves as expected. You may need to extend the interfaces of your distributed entities (clients, servers, or peers depending on your architecture) to improve their testability (by making them remotely controllable for instance) or their observability (by exposing parts of their internal state to verify invariants).
- If you are short of time, unit tests will earn you some points, but not the maximum for this part of the marking scheme.

5 Deliverables and Marking Scheme

You are asked to **work in pairs** for this coursework. Each pair of students will be asked to **demonstrate** their solution during the **final marking session** (see introductory session and on-line course site). In addition to your functional code, you should implement a series of automated tests to exercise your game engine (see notes above). Specific marks are allocated to the **quality of your tests**, and the use of the an **on-line repository** (typically using `git`).

In addition you are asked to write an **individual report** of roughly 900 words ($\sim$ two pages) that you should submit electronically in addition to the source code of your solution. Your report should describe the overall design of your solution, and its motivation (why have you chosen to do things the way you did them). It should also highlight any technical challenge or limitation you encountered. Finally you should discuss how your work could be further improved and extended.

Finally, some **bonus marks** are reserved to those students who choose to explore an innovative design or language (or both, e.g. a peer-to-peer design, an actor framework or language, the use of advanced development frameworks such as Spring Boot or Spring Roo, the use of a PaaS Cloud, including the delaying or rollback strategies mentioned earlier to mask network delays on WAN deployment, etc.).

Your final coursework grade (on 20 points) will be calculated as follows:

- Completion of client and server programs as specified (demo, in pair) 30% (6 points)
- Appropriate suite of automated testing of the game logic 20% (4 points)
- Use of an appropriate versioning repository 10% (2 points)
- Bonus for innovative design 20% (4 points)
- Short report ($\sim$ 900 words, individual) 20% (4 points)

6 Deadline, marking, and submission:

6.1 IMPORTANT

- The marking session for the coursework will be announced on the on-line course web site. Make sure you know how to access it, otherwise ask.
- Your solution (archive containing your source code or a link to your repository, + individual report) should be **submitted electronically** through the course web site by **midnight the day before your marking session** (this might depend on your lab group).
- If you include a link to your repository (e.g. on `https://gitlab.istic.univ-rennes1.fr` or equivalent) rather than your source code, be sure to (i) set your repository as **private**, and (ii) **invite** your lab tutor to your repository.
- Please use the naming convention: "FAMILYNAME1_Firstname1_ FAMILYNAME2_Firstname2.zip" for your archive, and "FAMILYNAME_Firstname.xxx" for your report.
- **Late submissions** will receive a grade of zero.
- You will be asked to **demonstrate** your work from your submission. This demonstration is compulsory: submissions that are not demonstrated will receive a grade of zero.

6.2 Other points

- You need to come to the marking session with your exercise completed. We will not be able to provide support during the marking session.
- You should be able to explain what you have done clearly, to show that you understand the concepts introduced.
- Checks for plagiarism and collusion between pairs of students will be carried out.

7 Collusion, plagiarism and fraud

7.1 English version (French version below)

The code that you submit must be the result of personal work by your Lab pair. Helping each other on graded lab work is not forbidden, but it must be limited to occasional advice and/or high-level advice.

It is however **strictly forbidden** to submit all or part of another pair's solution (plagiarism/copying), or to submit a solution common to several pairs, whether the common parts are partial or total (collusion). Having understood the code you submit without being the author is not acceptable, as it is your ability to produce a solution by yourself that is being assessed.

Plagiarism and collusion checks will be carried out on submitted code. Any student suspected of fraud may be called before the university's **disciplinary committee**, and may be subject to penalties which may include **temporary or permanent exclusion** from the university.

7.2 Version française

Le code soumis doit être le résultat du travail personnel de votre binôme. L'entraide entre binômes lors d'un travail noté n'est pas interdite, mais elle doit se limiter à des conseils ponctuels et/ou de haut niveau.

Il est en revanche **strictement interdit** de soumettre tout ou partie de la solution d'un autre binôme (plagiat/copie), ou de soumettre une solution commune à plusieurs binômes, que les parties communes soient partielles ou totales (collusion). Le fait d'avoir compris le code que vous soumettez sans en être l'auteur n'est pas acceptable, car c'est ici votre capacité à produire une solution par vous-même qui est évaluée.

Des contrôles de plagiat et collusion seront réalisés sur les codes soumis. Toute étudiant ou étudiante suspecté(e) de fraude s'expose à être convoqué(e) devant la **commission disciplinaire** de l'université, et encourt des peines qui peuvent entre autres aller jusqu'à **l'exclusion temporaire ou définitive** de l'établissement.

— End of the lab brief —