PROJECT

1. PROJECT SPECIFICATION

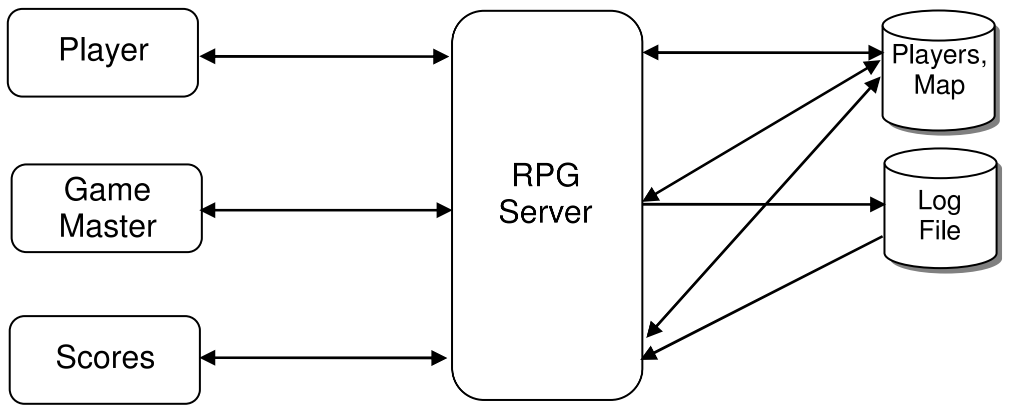
1.1 OBJECTIVES

The objective of the project is to implement an online Role-Play Game system.

Any resemblance between the rules proposed here and any existing game is just a coincidence.

1.2 SPECIFICATION

The Role-Play Game system should be based on a client-server architecture:



Three different clients should exist:

- a client for managing the operations of a player with a character in the game, called **Player**;
- a client for managing interactions between players, called game **Master**;
- a client for getting system statistics, called **Scores**.

Players play in a game map with size 5x5, being able to move around and attack other players, among other functions that will be described in detail next.

A **Player** client can manage a single character in the game. A character has the following properties:

- name: it should be unique in the game at a given instant; if the character dies, a new character can be created with the same name (resurrection of the character), independently of the properties he had before dying;
- attack power: when the character is created, 50 points should be distributed between the attack power and defense power, according to the players' choice; by default he gets 25 attack power points and 25 defense power points;
- defense power;
- energy: initially, the character gets 10 energy points; if energy gets below 0, the character dies; every 30 seconds, 1 energy point is restored, up to the

maximum of 10 energy points; no character can have more than 10 energy points at any time;

- experience: initially, when created the character gets 1 experience point. The experience can increase without bound, and never decreases.

When a player is created, it is placed in a random map position.

The Player client allows its character to move around in the map, going up, down, left or right in the map. The character cannot go outside the map. There is no limit to the number of players that can be at the same map location simultaneously.

The Player client also allows showing the current player character property values.

The Game **Master** client allows managing interactions between different player characters and between characters and objects existing in the game map.

The Game Master client allows showing what exists in the current player character's map location. These can be:

- other players: if other player characters exists in the same map position, the player can choose to attack one of them;
- food: it can be eaten by player characters, increasing its energy by one unit up to the maximum. One unit food existing at that location is spent every time a character eats it.
- traps: if the player reaches a map position with a trap, he loses 1 energy to escape the trap and gains 1 experience point; the trap will exist for the whole game duration;
- training center: if the player reaches a training center, he can train his attack or his defense skills; for each train, he spends 1 energy and gains 1 attack power point or 1 defense power point.

The result of attacks are to be decided according to the following formula depending on the current value of the attacker and attacked characters' properties:

Attacker wins if (attacker.attack + attacker.energy + attacker.experience) / (attacked.defense + attacked.energy + attacked.experience) * Rand(0.5, 1.5) > 1;

Otherwise, the attacker loses and the attacked wins. Assume that Rand() returns a random number uniformly distributed within the provided bounds.

As result of a combat, both characters involved in the combat spend 1 energy point, the winner gains 1 experience point and the loser loses 1 additional energy point.

The Game Master client also allows creating food, traps and training centers in random maps locations. These commands can be repeated as many times as the Game Master wants, to create more items in the map.

The **Scores** client allows showing game statistics and the game map. The statistics to be provided are:

- top players score: showing a ranking for of the 10 best players; the ranking can by shown by attack power, defense power, energy, experience, number of combats, or total player score, corresponding to the sum of the different player properties (attack, defense, energy, experience);

- list players score: similar to the top, but shows all players and the value of the property requested (attack, defense, energy, experience) for each player;
- combat score: for the given player, how may combats have been won and lost;
- event log: for each player, how much he has travelled, how much food he has eaten, how many times he fell in a trap, how many times he trained in a training center;
- map: four types of maps can be drawn: i) showing were each player is; ii) showing how much food is in each location; iii) showing locations with traps; iv) showing locations with training centers.

The server should manage the games and the playing logs. The server should provide application layer confirmations of all messages received from clients. The server should support multiple simultaneous clients of each type without any client blocking other clients.

In order to fully commit each member of the group with this work, the responsibilities of each one of the 3 client types must be assigned to a different member of the group. Group members should agree on this assignment between themselves. Each group member should specify, implement and test the functionality assigned to him.

The server should store all information in text files. This allows the server to be stopped without losing information and also helps interactions between the code of the different students. The format of the files should be specified by the students.

If the group has less than 3 members, students should implement only the functionality corresponding to the number of group members. No additional grade will be given for specifying or implementing functionalities of more clients than group members. If the group has only 2 members, the functionality of the Scores client should not be implemented.

Finally, group members should integrate their work into a single project.

2. Project Activity Plan

2.1 Activity 1: Communication Protocol Specification

The week from 18-22 March will be for introduction to the communication protocol specification. Students should decide which part of the project each of them will implement.

The students should prepare their communication protocol specification and present it to the professor in the class on the week from 25-29 March. There is a 10 minutes time limit per group. The professor will provide some feedback about the specification during the class.

A report with the communication protocol specification should be delivered through the Fenix system by Sunday, 31st March, 23:59h. Score 6/20. A ZIP file should be submitted including the protocol specification of each student's part in a separate PDF format file. Each student will only be evaluated by the component that he/she is responsible for.

The report should include:

- The functionality assignment to each student. The communication protocol specification part for each student should be a separate PDF file.
- Transport protocol to be used and justification.
- Format of the messages to be used.
- Time diagrams identifying the possible sequences of actions, or state machines of the client and server applications.

2.2 Activity 2: Application Implementation – Standalone Test

Each client's functionality should be implemented separately by each group member, submitted to the Fenix system in a ZIP file by 8th April, 23:59h, and shown separately during the classes in the week of 8-12 April. Score 4/20.

During the standalone test, the students should perform a set of tests to show that their individual component of the application is working properly. If the students have already started integration (next activity), they can show an integrated or partially integrated version. There is a 10 minutes time limit per group. Each student will only be evaluated by the component that he/she is responsible for.

Some feedback can be provided to students during the classes.

The code should:

- Be developed according the client-server architecture rules.
- Be easily readable, modular and with appropriate comments if found necessary.

2.3 Activity 3: Application Implementation – Integrated Test

Students should integrate the different functionalities they implemented.

During the integration, the students should perform a set of tests that illustrates the different use cases and enables them to assess the status of integration. Each student will only be evaluated by the component that he/she is responsible for.

- In case of not being able to successfully integrate the different parts, each student may illustrate the complete functionalities by defining and realizing a complete set of tests that illustrate the different use cases independently of other parts.
- In case of partial or fully integration, both test and results will be used for evaluation.

The final implementation of the applications, together with the list of tests (in case of not having an integrated version), should be delivered in a ZIP file through the Fenix system by Monday 22nd April, 23:59h. Score 4/20.

The code should:

- Respect the specification delivered. If, during implementation, the specification was changed, an updated version of the specification can be delivered with the code.

2.4 Activity 4: Application Demonstration

The final version of the implemented project should be demonstrated by the students to the professor during the laboratory class in the week of 22-26th April. Score 6/20. There is a 10 minutes time limit per group.

During the demonstration, the students should perform a set of tests to show their application is working properly:

- Show the functionality implemented by each group member.
- Show the integrated functionality that the group successfully implemented, if it exists.
- You may also show partial, incomplete functionality, or functionality not integrated.
- Show exceptional communication situations.
- Examples of tests to consider: i) normal application operation; ii) communication with a server that is not available; iii) wrong number of parameters provided by the user; iv) invalid parameters provided by the user (e.g. invalid request, statistics for a player that does not exist); v) multiple simultaneous users.

Other situations not shown during the demonstration may be tested afterwards.

2.4 Activity 5: Individual Evaluation

Students will be provided with a client-server programming problem to be solved individually. Each student will have to solve the assigned programming problem using the lab PC. The schedule and details for this individual evaluation will be provided later.