Software Engineering Final Examination

Academic Year 2023-2024

Requirements

Index

1 Introduction

The project consists of developing a software version of the Codex Naturalis board game. The nal project should include:

- high-level UML diagrams of the application, showing, with all and only useful details, the overall design of the application;
- detailed UML diagrams showing all aspects of the application. These diagrams po- may be generated from the project source code using automated tools;
- functioning implementation of the game in accordance with the game rules and speci cations in this document;
- Documentation of the communication protocol between client and server;
- peer review papers (one related to the first peer review and one related to the second peer review);
- source code of the implementation;
- Javadoc documentation of the implementation (generated from the code);
- unit test source code .

Delivery date: Friday, June 28, 2024 13:00:00 CEST
Valuation date: to be announced (as of Monday, July 1, 2024)

Assessment methods, classrooms and times, will be announced later by the teacher in charge and may vary from teacher to teacher.

2 Project Requirements

The project requirements are divided into two groups:

- Game-speci c requirements concerning the rules and mechanics of the game.
- Game-agnostic requirements involving design, technological, or implementation aspects.

2.1 Requirements game-speci c

The rules of the game are described in the two le `CODEX_Rulebook_[ITA-ENG].pdf`, uploaded to the course's WeBeeP site. If there is any discrepancy between these l e refer to the Italian version.
The names of classes, interfaces, methods, variables, and in general all identi cations in the code must be in English. Also in English must be the comments in the code and the technical documentation (JavaDoc). The language of the application user interface may be freely chosen between English and Italian.
Two possible sets of rules are referred to for evaluation (see Table 1): the simplified rules and the complete rules.

- Sempli cate Rules: consider a sempli cated version of the game that does not include personal objective cards (secret objective) and their scores. Common objective cards are included instead.
- Complete Rules: consider all the rules for the conduct of normal games as stated in the game manual.

Each player identi cated by a nickname that is set client-side. This nickname must be unique, meaning that no two users can be logged on at the same time with the same nickname. The uniqueness of the nickname must be guaranteed by the server during player acceptance.

2.2 Game-agnostic requirements

This section presents the technical requirements of the application.
The project consists of implementing a distributed system with client-server technology. For each game, one server instance will be launched that can handle one game at a time and two or more clients (one per player) that can participate in only one game at a time. The use of the Model-View-Controller - MVC pattern is required to design the entire system.

2.2.1 Server

Below is the list of technical requirements for the server side.

Must implement the rules of the game using JavaSE.

It must be instantiated only once at ne to handle a single match (or multiple m a t c h e s i n case the advanced multiple matches feature is implemented).

It allows the various players to take their turns via client instances, connected to the server through an appropriate network protocol (TCP-IP sockets and/or RMI).

Where client-server communication is implemented either via socket or via RMI, it must be able to support matches in which different players use different technologies.

2.2.2 Client

Below is the list of technical requirements for the client side.

It must be implemented with JavaSE and be instantiatable multiple times (one per player), even on the same machine.

The gra ca interface must be implemented using Swing or JavaFX.

In cases where both a text interface (TUI) and a graphical interface (GUI) are implemented, at startup it must allow the player to select the type of interface to be used.

In the case where client-server communication is implemented either via Socket or via RMI, at startup it must allow the player to select the technology to be used.

It is assumed that every player who wants to participate in a game knows the IP address or URL of the server. When a player connects:

If there are no matches being started, a new m a t c h is created, otherwise the user automatically joins the match being started.

The player who creates the game chooses the number of players who will be part of it.

If there is a game being started, the player is automatically added to the game.

The game starts as soon as the expected number of players is reached (based on the choice and etted by the first player when creating the game).

It is necessary to handle both the case in which players leave the game and the case in which the network connection drops. In both cases the game will have to end (even if it is starting) and all players will be noti cated of this event.

2.3 Advanced Features

Advanced features are optional requirements to be implemented in order to increase the score in the evaluation phase. These features are evaluated only if the game-speci c and game-agnostic requirements presented in the previous sections have been su ciently implemented. The advanced features that can be implemented are:

Multiple Matches: Implement the server so that it can handle multiple matches simultaneously. For the purposes of implementing this additional feature, the previously speci cated rules regarding match creation can be modi cated according to implementation or user interface needs. For example, to allow players in the entry phase to choose which open and not yet started game to connect to or to create a new game.

Persistence: Have the server periodically save the status of the game to disk, so that execution can pick up where it left off even after the server crashes. To resume a game, players will have to reconnect to the server using the same nicknames once it is back up and running. The disk of the machine on which the server runs is assumed to constitute fully a dable memory.

Resilience to disconnections: Players disconnected as a result of network or client crash can reconnect and continue the game. While a player is d i s c o n n e c t e d , the game continues by skipping that player's turns. If only one player remains active, the game is suspended
no until at least one other player reconnects or a timeout expires that decrees that the only player left connected wins.

Chat: Clients and servers must o reate the ability for players involved in a game to chat with each other, sending (text) messages addressed to all players in the game or to an individual player.

3 Evaluation

Table 1 shows the maximum scores obtainable according to the implemented requirements.

Requirements Met	Maximum rating
Simpli cate Rules + TUI + RMI or Socket.	18
Complete Rules + TUI + RMI or Socket	20
Complete Rules + TUI + RMI or Socket + 1 FA	22
Complete Rules + TUI + GUI + RMI or Socket + 1 FA	24
Complete Rules + TUI + GUI + RMI + Socket + 1 FA	27
Complete Rules + TUI + GUI + RMI + Socket + 2 FAs	30
Complete Rules + TUI + GUI + RMI + Socket + 3 FAs	30L

Table 1: Evaluation Table (FA=Advanced Functionality)

The following are the aspects that are evaluated and contribute to the de nition of the nal score:

The quality of design, with particular reference to appropriate use of interfaces, heir- tariety, composition between classes, use of design patterns (static, communication, and architectural), and division of responsibilities.

The correctness and stability of implementation in accordance with speci cations.

The readability o f written code, with particular reference to variable/method/class/package names, the inclusion of English comments, and the lack of repeated code and excessively long methods.

The quality of the documentation, with reference to the English JavaDoc comments in the code, additional documentation regarding the communication protocol, and peer review papers.

The e cacy and coverage of test cases. The name and comments of each test should clearly speci cate the functions tested and the components involved.

The use of tools (IntelliJ IDEA, Git, Maven, ...).

Autonomy, commitment and communication (with managers and within the group) during all phases of the project.