PROJECT

NEDAP UNIVERSITY
YEAR 8
MODULE 1

Author (2023): Jeroen Brinkman (Healthcare)
Author (2022): Stijn Wopereis (Livestock Management)
Author (2021): Dennis Wonnink (Retail) and Bart Hoeben (Healthcare)
Author (2020): Stephan Roolvink (Healthcare) and Dorien Meijer Cluwen (Security Management)
Author (2019): Joel Witteveen (Healthcare) and Djurre de Jong (Healthcare)
Author (2018): Maarten van Megen (Retail) and Robert Krikke (Security Management)

This document is handed out as project description to the students of Nedap University 9.0 (year 9 2023-2025).

Version 5.0

Introduction

Welcome and congratulations! You're still here. We are glad you are part of the Nedap family and we look forward to exciting times ahead. We hope you have learned a lot over the past few months and, above all, had a lot of fun.

This document describes the final assignment for Module 1. This differs from the final assignment provided by the University of Twente. We thought it would be fun to give our own assignment in Nedap style.

Without further ado; turn the page and let's go!

Have fun,

Jeroen, on behalf of all coaches.

Summary

The purpose of this assignment is to demonstrate the design and programming skills you acquired during Module 1. You demonstrate this by developing a client-server application to play a board game named 'Go'.

In short, you will provide the following features:

- Two applications
  - A server application that can host games
  - A client application that connects with the server and allows a player to participate in a game
- Two interface types
  - Textual user interfaces (TUIs) for both server and client
  - A graphical user interface (GUI) for the client for displaying the current state (the base will be provided by us)
- Enforcement of the Go game rules
- Support for playing the game over the network
- Support for 2 players per game
- Support for computer players (AI).

You will also provide a written report (maximum of 5 pages) in which you describe your experiences and findings during the development of the application.

The coaches will review your work. On the last day of the project, we organize a tournament where you can show off your results (client, server and computer AI). There will also be a question session (before the tournament) where we will ask you all different questions about your code. Both to get insights in your understanding of what you made and to learn how the other students did it.

The "Go" board game

If you want to play the actual board game, you can: there is a board available at Healthcare to play around with. There are, of course, many online alternatives as well.

Objective

Go https://en.wikipedia.org/wiki/Go_(game) (圍棋)* is an abstract strategy board game for two players in which the aim is to surround more territory than the opponent.

Rules

The rules of Go https://en.wikipedia.org/wiki/Rules_of_go have seen some variation over time and from place to place. Take a look at Wikipedia articles to get an understanding of the game and its rules.

However, in order to keep things feasible for this assignment, we will use the basic ruleset https://en.wikipedia.org/wiki/Rules_of_Go#Basic_rules. These are the rules you should adhere to in your application:

1. The board of configurable square size is empty at the start of the game.
2. Black makes the first move, after which white and black alternate.
3. A move consists of placing one stone of a player their own color on an empty intersection on the board.
4. A player may pass their turn at any time.
5. A stone or solidly connected group of stones of one color is captured and removed from the board when all the intersections directly orthogonally adjacent to it are occupied by the opponent.
6. Self-capture/suicide is allowed (this is different from the commonly used rules).
7. When a suicide move results in capturing a group, the group is removed from the board first and the suiciding stone stays alive.
8. No stone may be played so as to recreate any previous board position (ko rule https://en.wikipedia.org/wiki/Rules_of_go#Repetition).
9. Two consecutive passes will end the game.
10. A player's area consists of all the intersections the player has either occupied or surrounded.
11. The player with the biggest area wins. This way of scoring is called Area Scoring https://en.wikipedia.org/wiki/Rules_of_go#Area_scoring.
    In case of an equal score, there is a draw.

For a more detailed reference see: https://en.wikipedia.org/wiki/Rules_of_go#Reference_statement on Wikipedia.

Global Description of the Game Application

When the client application is started, the player enters the IP address and port number of the server to connect to, as well as their name.

After entering this information, the client logs on to the server. The client waits for the server to signal that another client logged on.

When the second client is present the game can start.

When the game is ready to be started, the server can assign colors to the players, or the players can choose their own color in order of arrival.

After a game has started, the server should still be ready for new incoming requests from other clients that would like to play the game. So, it should be possible to play several games simultaneously on the same server.

The game itself probably proceeds as follows: the player (human or AI) whose current turn it is, enters a move, considering the rules of the game. The client application checks the move and when it is legal, sends it to the server.

The server should also check the legality of the move. If the move is valid, the server sends the information to all participating clients. The clients can then update their internal game state.

The turn then moves to the next player, who repeats the process. This procedure proceeds until a situation occurs that finishes the game.

Communication Protocol

Your client application should be able to communicate with server applications from other Nedap University students and vice versa. Consequently, all students should use the same protocol for client/server communication. The protocol describes which data will be exchanged between the client and the server, and in which order. This data will consist of the moves in the game, and any extensions that you should wish to implement (keep in mind that the protocol should be extendable).

Deliverables

Application

During development, you should work with version control in Git. Keep your code in a public personal repository via GitHub. We encourage you to frequently make small commits, with decent commit messages.

For submission, link to a version tagged *'final'* on GitHub. Email the link to the tag to your coach.

The submission deadline is Thursday 1st of February at 23:59.

The project should have the following content and structure:

- There is a `README` file with instructions on how to install and start the game. For example: think about which directories and files are necessary, and any pre-conditions that should be met for a successful installation. After reading this file, someone unfamiliar with the project should be able to install and execute the game without any problem. The `README` should be located in the root folder of the project.
- Any non-standard predefined classes and libraries should be included in `.jar`-files or as a maven dependency.

Typical causes that make the installation and compilation procedure fail are names and paths or hardcoded URLs. Test this before submitting your project.

Hint: to verify your `README` contains all the required steps, check out your project into a new directory on your computer, and follow the instructions yourself. Or ask someone else to do this and follow your written instructions.

Warning: You can probably find an implementation of the game on the Internet. Copying (parts of) such an implementation directly is considered to be plagiarism and is not tolerated. At any moment in time, you should be able to explain your work to the Nedap coaches.

Report

The purpose of the report is to reflect on the application you built and the development process. We want you to think critically about your own work. To keep things concise, we ask to keep the number of pages to a maximum of 5 pages (excluding frontpage, attachments, etc). E-mail your report to your coach as a PDF.

Some guidelines for the report:

- Write your report in English or Dutch.
- Give a review of your own code, just like you would peer review the code of someone else
    - What parts of the application went well and why?
    - What didn't go so well and why?
    - What would you do differently the next time?
    - If libraries are used, why are they used?

- Development process
    - How did you decide as part of the group on the communication protocol?
    - Did you test during or after the implementation, why?
    - Etc.
- Discussion of design:
    - Some comments on the overall structure (Classes, packages, etc.)
    - Explain the communication protocol
    - What patterns did you use and why? (MVC, observer pattern, etc.)
    - What did you test, why?

Hints & Tips

- Make sure the protocol is agreed upon soon (plan a meeting to setup your protocol).
- If an upgrade to the protocol is needed, do this together. Make sure everyone is on the same page.
- We value clean, well-documented, and tested code more than tournament winning AI.
- Writing tests will help you understand what is going wrong when you are stuck (which will happen inevitably).
- Test your implementation with other students' clients and servers.
- Start writing your reflection report during your development process and not last minute.

Requirements

Functional Requirements of the Application Server

The server application has the following requirements implemented:

1. When the server is started, a port number should be entered that the server will listen on.
2. A server can support multiple instances of games that are played simultaneously by different clients.
3. The server has a TUI that ensures that all communication messages are written to `System.out`.
4. The server implements the protocol to support the minimal assignment requirements. The server should be able to communicate with all other clients written by other students.
5. If a player quits the game before it has finished, or a client crashes, the other player(s) should be informed, and the game should end cleanly.

Functional Requirements of the Client

The client application has the following requirements implemented:

1. The client provides a user-friendly TUI, which provides options to the user (e.g., the possibility to enter a port number and IP address) to request a game at the server.
2. The client makes use of the provided GUI, which displays the current state of the board to the user.
3. The client supports both human players and computer players.
4. After a game is finished, the player can start a new game.
5. If a player leaves the game before its over, or if the server disconnects, the client must react to this gracefully and not crash.
6. The client should respect the protocol as defined during the protocol session, i.e., the client should be able to communicate with all other servers written by other students.

You don't have to make the GUI yourself from scratch. Find it here: https://github.com/nedap/university-goGame

Minimal Requirements for the Implementation

This paragraph provides a checklist with minimal requirements for your implementation. The implementation should satisfy at least the following requirements:

- The application implements all the functional requirements discussed in the previous paragraphs.
- The program compiles without any problems.

- All self-defined classes are in self—defined packages. The classes are organized in at least three different packages.
- Result values of methods are not used to encode error states; instead, exceptions and exception handling should be used, and all exceptions explicitly thrown in your own code should be self-defined.
- The server is multi-threaded to support multiple concurrent games being played.
- There are test classes and test runs. The tests try different situations described in the functional requirements and game rules (e.g., the specified port for the server is free or in use; playing an illegal move)

Checkstyle Warnings

A checkstyle configuration file is provided with the GUI repository. Your project should not contain any such warnings produced by checkstyle.

Tip:
We recommend that your code follows a good coding style, including layout and variable naming. See for example the Google Java Styleguide https://google.github.io/styleguide/javaguide.html.

Possible extensions

If you have sufficient time, you can extend your game, following one of the suggestions below. Extensions are in no way mandatory!

Note that some of the extensions below also require extending the server and the communication protocol. You should make sure that trying to use the extended functionality with a server from a different student does not lead to a crash.

Server port fallback

If the port number is already in use, a corresponding error message will be displayed. The server should ask to enter a new port number.

Game hints

The client provides hint functionality, which can show a possible move. The move may only be proposed, the human player can decide whether to play this move or make a different one.

Chat Box

When the game application is developed as described earlier, it can only be used to play the game. It can be fun to have the possibility to communicate with your opponents during the game. Therefore, a possible extension is to add an option to the client UI allowing the user to enter texts. After pressing the return button, the message is sent to the server, who then sends it to the other clients participating in the game, after which the message is shown on all clients UIs.

Challenge

Up to now, clients are connected in order of logging on to the server. It would be nicer if a client could choose from the registered clients against who to play. To achieve this, the following changes will be necessary:

- A client should know which other clients are registered
- A client should be able to choose its opponent
- A client should be able to refuse a game

It could also be nice if a player can challenge its opponent to a rematch after the game is over.

Leaderboard

The server could maintain a leaderboard, providing scoring information about all the games played on the server. It should be possible to retrieve this information in various ways (e.g., overall high score, best score of the day, best score of a particular player).

The leaderboard could provide the following functionality:

- Methods to add new scores to the leaderboard database
- For each score, it should be possible to see at which date and time it is achieved, and by which player

- Methods to inspect the scores, I.e. the top scores, all scores above a certain value, all scores below a certain value, average score, average score of the day etc

The overall performance should be as good as possible, meaning that you have to choose a representation that suits your implementations.

Replay

Another possible extension might be to show a replay of a game that has been played. This could be done at the end of the game but also after the game has been played. To achieve this the client (or server) should be able to play back the history of the game. And in case of playing back finished games, the history should be stored

Evaluation

The minimal requirements as outlined in this report are required to achieve a positive evaluation that will let you pass the project. If your implementation exceeds the minimum requirements the evaluation will be even more positive ;)

The coaches will review your application and the report together.

Criteria

The list of criteria below will be used to evaluate your project. Use it to keep track of and judge your own progress.

Code

1. The application has all the required components.
2. There is a `README` file with installation and execution instructions.
3. The application compiles and executes without errors.
4. The application has been sufficiently documented with Javadoc.
5. The implementation of large and/or complex methods has been documented internally.
6. The application layout is understandable and accessible.
7. Packages and accessibility are used sensibly.
8. Non-standard Java classes are included as JARs or maven dependencies.
9. All used test classes and test executions are submitted with the code.

Design

1. The overall design is logical.
2. The implementation corresponds with the design in the report.
3. The program is divided logically into classes.

Programming Style

1. Names of classes, variables, and methods are well-chosen and understandable.
2. Code is efficiently and neatly implemented.
3. The program is easily maintainable (use of constants, variable names, etc.).
4. The group's communication protocol has been properly implemented.
5. The exception mechanism is used appropriately.
6. Concurrency constructs are used properly.
7. Artificial Intelligence for a computer player has been implemented.
8. Use Git in a structured way.

Testing

1. Appropriate unit tests are provided.
2. Appropriate system tests are provided.
3. Sufficient test coverage is reached.
4. All classes in the system have been tested by unit testing.

Organizational overview

Calendar

Week	Day	Time	Activity
Week 3	Thu 18 January	09:00	Kickoff
Week 6	Thu 1 Feb	12:00	Deadline Deliverables
Week 6	Thu 1 Feb	(own time)	Tournament preparation
Week 6	Fri 2 Feb	9:00	Individual evaluation
Week 6	Fri 2 Feb	12:00	Lunch
Week 6	Fri 2 Feb	15:00	Official Tournament

Tournament preparation

This day you will be able to prepare for the tournament later the day. This is to make sure that your server and client will work during the tournament.

Tournament

Friday the 2 of February 2024 in Module Week 10 we will host a tournament! You and your applications will compete against each other. The winner will get a prize!

Also, during the day your work will be reviewed and discussed by all coaches. Before the tournament starts you get your results.

HAVE FUN!