MODULE 1 NEDAP PROGRAMMING PROJECT

TABLE OF CONTENTS

SUMMARY

Purpose of this project is to demonstrate the design and programming skills you have acquired during this module. You will demonstrate this by developing a client/server application to play a board game.

The application should at least provide the following functionality:

- a game server which offers the possibility to host games;
- a client that can connect with the server and allow a player to participate;
- textual user interfaces (TUIs) for the server and client;
- make use of the provided graphical user interface (GUI) for client for displaying the current state.
- enforcement of the game rules;
- support for playing the game over the network;
- support for 2 players per game;
- support for computer players.

In addition to the application you will provide a report (max. 5 pages), which describes your findings during the development of the application.

On the last day of the project we will host a tournament in which you can show off your result (client & server & computer AI). Also during the day we, the coaches, will review your work together.

RULES OF THE GO BOARD GAME

Game objective

Go (圍棋) is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

Game Rules

The rules of Go have seen some variation over time and from place to place. In order to keep things simple we've defined our own basic ruleset:

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 one's 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 adjacent to it are occupied by the enemy.
6. *Self-capture/suicide* is allowed (but pretty useless)
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*).
9. Two consecutive passes end the game. However, since black begins, white must end the game.
10. A player's *territory* consists of all the points the player has either occupied or surrounded.
11. The player with more territory wins.

For a more detailed reference see this article on wikipedia. We'll follow the basic rules described there, with 2 additions to make things simpler:

- We'll use the *area scoring* method, which allows for easier an simpler end-of-game rule (see rule 9)
- In case of an equal score there is a draw

GLOBAL DESCRIPTION OF THE GAME APPLICATION

After starting a client, the user can enter the IP address and port number of the server, and his own name. After entering this information, the client will log on to the server. The client will wait for the server to signal that there is another client logged on. When a second client has logged on, the game can start, or (in case of a multi-player game) the clients can decide to wait for more players. When the game is ready to be started, the server can assign the colours to the players, or the players can choose their colour in order of arrival.

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

The game itself could proceed as follows. The player whose turn it is enters a move, taking into account the rules of the game. The client checks the move, and when it is legal, it sends it to the server.

The server also checks legality of the move, and if it is legal, it will send this information to all participating clients, who can then update their internal game state. The turn then moves to the next player, who should again enter a legal move. This procedure proceeds until the rules of the game indicate the game has finished.

Communication Protocol

Your client application should be able to communicate with server applications from other Nedap University students, and vice versa. As a consequence, all students within the same tutorial group 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 inspection requests for the leader board.

DELIVERABLES

Application

During development you should work with version control in Git. You should keep your code in a public repository via GitHub.

For submission you will link to a version tagged 'final' on GitHub. Email the link to the repo to **nu2coaches@nedap.com**.

The project should have the following content and structure:

- There should be a README file with information about installation and starting the game, indicating for example which directories and files are necessary, and conditions for the installation. After reading this file, a user should be able to install and execute the game without any problem. The README file should be located in the root folder of the project.
- Any non-standard predefined classes and libraries should be included in jar-files.

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

Warning: It might be possible to find an implementation of the game on the Internet. Copying such an implementation directly will be considered as fraud, and will not be tolerated. At any moment in time. you should be able to explain your own solution to the Nedap coaches.

Report

The purpose of the report is to reflect on the realised application and 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 max +/- 5 pages (excluding frontpage, attachments etc.). You should email the report to **nu2coaches@nedap.com** as Word or PDF.

Some guidelines for the report:

- The report may be written in English or Dutch.
- Give a review of your own code, just like you would peer review code of someone else.
  - What parts of the application went well and why?
  - What didn't went so well and why?
  - What would you do differently the next time?
  - etc.
- Development process
  - How did you decide as part of the group on the communication protocol?
  - Did you do testing during or after the implementation, why?
  - etc.
- Discussion of design:
  - Some comments on the overall structure (Classes, packages, etc.)
  - What patterns did you use and why? (MVC, Observer, etc.)
  - What did you test, why?

REQUIREMENTS

Functional Requirements of the Application

Your application should implement a number of requirements.

For the server, the following requirements should be implemented

1. When the server is started, a port number should be entered that the server will listen to.
2. If the port number already is in use, an appropriate error message is returned, and a new port number can be entered.
3. A server should be able to support multiple instances of the game that are played simultaneously by different clients.
4. The server has a TUI that ensures that all communication messages are written to System. out.
5. The server should respect the protocol as defined for the tutorial group during the project session in Week 7, i.e., the server should be able to communicate with all other clients from the tutorial group.

For the client, the following requirements should be implemented

1. The client should have a user-friendly TUI, which provides options to the user (e.g., possibility to a enter port number and IP address) to request a game at the server.
2. The client should make use of the provided GUI, which displays current state of the board to the user.
3. The client should support human players and computer players.
4. The move time of the computer player should be a parameter that can be changed via the client TUI.
5. The client should provide a hint functionality, which can show a possible move. The move may only be proposed, the human player should have the possibility to decide whether to play this move, or make a different one.
6. After a game is finished, the player should be able to start a new game.
7. If a player quits the game before it has finished, or the client crashes, the other player(s) should be informed, and the game should end cleanly. In this case, the other player(s) should be allowed to register again with the server for a new game.

8. A server might at all times disconnect. The clients should react to this in a clean way, closing all open connections etc.
9. The client should respect the protocol as defined for the tutorial group during the project session in Week 7, i.e., the client should be able to communicate with all other servers from the tutorial group.

Finally, as a global requirement, the client and the server should always be in the same game state.

Minimal Requirements for the Implementation

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

- The application should implement all the functional requirements discussed in the previous paragraphs.
- The program should compile without any problems.
- All self-defined classes must be in self—defined packages. the classes should be organised in at least three different packages.
- The code must follow a good coding style, including layout and variable naming. See for example the Google Java Styleguide. Specifically, it must not produce any checkstyle warnings (see below for details).
- Result values of methods should not be used to encode error states; instead, exceptions and exception handling should be used, and all exceptions explicitly thrown in own code should be self-defined.
- ~~For the three most complex classes in the server (which can also be classes that are shared with the client, see below for a definition of "most complex"): the classes and all their methods should be documented in Javadoc, with IML pre- and postconditions and class invariants (IML specifications must type-check with OpenJML).~~
- The server should be multi—threaded to support multiple concurrent games being played.
- There should be test classes and test runs. The tests should try different situations described in the functional requirements and game rules (e.g., the specified port for the server is free or in use; different number of players; or playing an illegal move)

Checkstyle Warnings

You should download the checkstyle configuration from Blackboard (go to COURSE MATERIAL and find the checkstyle configuration file as well as an instruction for importing it in the Tools item). Each violation of a check in the configuration will be reported in the Eclipse PROBLEMS View as a warning. As type of the warning "Checkstyle Problem" will be specified. Your project should not contain any such warnings produced by checkstyle.

POSSIBLE EXTENSIONS OF THE APPLICATION

If you have sufficient time, you can extend your game, following one of the suggestions below. Implementing more extensions makes your project more advanced and gives you a better rating. However, extensions will only be evaluated if the application respects the basic functional requirements.

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 team does not lead to a crash.

Chat Box

When the game application is developed as described above, 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 client 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 his opponents;
- a client should be able to refuse a game.

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 it scores, all scores above a certain value, all scores below a certain value, average score, average score of the day etc; and

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

Security

As soon as a leaderboard is in place, players might want to cheat to get to the highest place. In addition, simply accepting any connection leaves one vulnerable to Denial-of-Service attacks. One way to help prevent all this is to properly authenticate clients to the server. Simple password-based approaches are a possibility (but how would you prevent the password from being sniffed on the network?).

Multiple Game variants (only if you have leftover time)

The standard game allows for other variants like:

- Game with more than 2 players
- Simultaneous moves
- Multiple moves per turn.

Note that these variants might require changes your communication protocol. If you do change your protocol it should be backwards-compatible (possibly by using different versions).

EVALUATION

The minimal requirements as sketched above are necessary to passing rating for the project assignment. If your implementation does better than the minimum requirements, you can get a better rating.

We as the coaches as a group will review your application and report and tournament results during the final tournament day.

Rating Criteria

Finally, we give the list of rating criteria that will be used to rate the project assignment. You can use this list to judge your own progress with the project.

Code

1. The program has all the required components.
2. There is a README file with installation and execution instructions.
3. The program compiles and executes without errors.
4. The program has been sufficiently documented. with Javadoc, ~~and JML pre- and postconditions~~, and class invariants.
5. The implementation of large and/or complex methods has been documented internally, including the use of loop invariants.
6. The program layout is understandable and accessible.
7. Packages and accessibility is used in a sensible way.
8. Bytecode of predefined, external, non-standard Java classes is submitted with the code.
9. All used test classes and test executions are submitted with the code.

Design

1. The overall design is logical.
2. The implementation corresponds to the design in the report.
3. The program is divided in a logical way into classes.
4. All classes have detailed documentation.

Programming Style

1. Names of classes, variables and methods are well-chosen and clear.
2. Code is efficient and neatly implemented.
3. The program is easily maintainable (use of constants, variable names etc.)
4. The tutorial 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.

Testing

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

Extensions

1. Multiple game variants
2. The chat functionality is correctly implemented and documented.
3. The challenge functionality is correctly implemented and documented.
4. The leaderboard functionality is correctly implemented and documented.

ACTIVITIES

Calendar

MODULE WEEK	DAY	TIME	ACTIVITY
Week 8	Mon 16 jan	10:00	Project Kick-off
Week 10	Wed 1 feb	23:55	Deadline Deliverables (Code + Report)
Week 10	Thu 2 feb	all-day	Tournament preparation (students)
Week 10	Fri 3 feb	13:00 - 17:00	Official Tournament + Review

Feedback Sessions

Each week there will be a planned feedback session between the student and the coach.

Tournament preparation

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

Tournament

Friday 3 feb in Week 10, a tournament will be organised where the different players will compete against each other. Bonus points can be gained by winning the tournament.

Also during the day your work will be reviewed and discussed with all the coaches. At the end of the day if everything goes well, you will get the results.