Developing Maintainable Software
Coursework Specification Sheet

Boon-Giin, Lee; Heng, Yu
boon-giin.lee@nottingham.edu.cn; heng.yu@nottingham.edu.cn

University of Nottingham Ningbo China, School of Computer Science

1 Synopsis

This coursework is about maintaining and extending a re-implementation of classic retro game (Sokoban). This version has never been completed, but at least it runs, once it is set up properly. More information about the original Sokoban game and its history is available here. To get started, download the original re-implementation from the Moodle "Coursework - Source Code".

Set up a project (e.g. JavaFX or Maven) in IDE (prefer IntelliJ) and embed the file that just downloaded. Note that the Gitlab repository mentioned above only provides source code and resources but no project files, as it is good practice to ignore IDE-specific generated files for source control. Add a *.gitignore* file to the project, ensuring follow this good practice as well. Set up a remote git repository at the school's GitLab server. The remote repository need to be named "AE2DMS-CW-StudentID" (e.g. "**AE2DMS-CW-20121023**") and needs to be set to **PRIVATE**. Then follow the setup instructions provided in GitLab to "push an existing folder" (i.e. do an initial push to upload files from your local repository to your remote repository). Now, you are ready for coding with version control.

NOTE This coursework is about maintaining and extending existing code. So, for the maintenance part, you must use the existing code as a basis, and not write your own Sokoban game from scratch. The school's GitLab server ONLY can be ACCESSED when CONNECTED to eduroam. Follow the instructions "Add Users to Group" to add two GitLab members "z2019017" (Bryan) and "z2019078" (Heng Yu) with role permission as "Maintainer" for your coursework assessment.

IMPORTANT Make sure you understand what you are writing in your Javadocs. We reserve the right to briefly interview you if we think that you do not understand what you write about. So, when you write your Javadocs, do not simply copy/paste large portions of text from existing articles or other resources - as this does not demonstrate your understanding of the topic. You might also run into issues with plagiarism.

You should roughly spend 50 hours on this coursework.

2 Assessment

The marks will be split as follows (total 100%):

- **[15%]** Git Use (e.g. push, branch, merge, providing *.gitignore*).
- **[30%]** Refactoring.
- **[30%]** Additions.
- **[15%]** Documentation (e.g. README.md, Javadocs and class diagrams).
- **[10%]** Video (explaining improved activities and additions).

IMPORTANT It is always a good practice to refer to the assessment rubric to understand how to secure high mark in the coursework. The rubric is provided in last page for your reference.

Table 1: Overview

Weight	75%
Issue Date	October 23rd, 2020 on Friday by 10 a.m.
Submission Date	December 7th, 2020 on Monday by 5 p.m.
Late Policy	Standard Policy
Feedback Date	January 11th, 2020 (Expected)
Feedback Method	Individual GitLab repository

3 Deliveries

- A "*README.md*" file (with maximum 500 words), documenting the work you conducted (highlighting the key changes you made for maintenance and extension, where you made them, and why you made them). You should clearly stated all the applied OO design patterns, GUI design patterns, refactoring methodologies with ***detail text explanations and proper references*** to the code in the *README.md* file. Otherwise, they will be treated as non-implemented. In addition, convert the *README.md* to PDF (online converter), name it as "StudentID-README.pdf", e.g. "**20121023-README.pdf**" and submit to Moodle.
- High level class diagram in diagram folder that shows the structure of the final version of your game which consider only classes (exclude fields and methods, unless they are relevant for understanding design principles/patterns), interfaces, relationships and multiplicity. The final class diagram must be in PDF. Name your PDF as follow: "StudentID-ClassDiagram.pdf", e.g. "**20121023-ClassDiagram.pdf**". If you separated class diagram into several PDF, do supply a markdown file in this folder to indicate the respective information. Feel free to use any UML modeling tool or Visual Paradigm (computing lab session) or refer to "Draw Diagrams with Markdown" (see Fig. 1).
- The source code documentation (Javadocs) should be delivered in form of docs folder inside your project. Besides reading your *README.md* file, we will look at the Javadocs to find out how you maintained and extended the game. If it is not obvious from there, we might miss it. Please make sure to provide informative but concise Javadocs.
- It needs to be possible to IMPORT (or OPEN) and RUN your project in IntelliJ. To avoid disappointment later, download the source code from your GitLab repository (in zip file) and test your final version on different computer. This should help to uncover hardcoded path dependencies, which was a major issue in previous few years.
- Finally, you have to make a video (very briefly) up to 2 minutes in video folder inside your project, showing your software running. Then (for the main part), explaining your refactoring activities and additions. You should also highlight two achievements you are most proud of. Please name your video as follows: "StudentID-Video.EXT", where EXT represents the extension related to your video format (prefer mp4 and avi), e.g. "**20121023-Video.mp4**".

For moderate marks (40%/100%):

- Set up a **PRIVATE** Git repository on the school's GitLab server and use it actively for version control activities.
- Do some basic maintenance of the delivered code base (e.g. adding meaningful Javadocs, organising files in a meaningful way into packages, breaking up large classes in a meaningful way to support the idea of single responsibility, improving encapsulation etc.).
- Extend the delivered code base by adding:
  - A **START** screen, displaying a picture related to the game and a button that provides access to a **INFO** screen explaining the game operation.
  - A **HIGH SCORE** pop-up, appearing at the end of each round, showing the scores from each round, highest at the top.

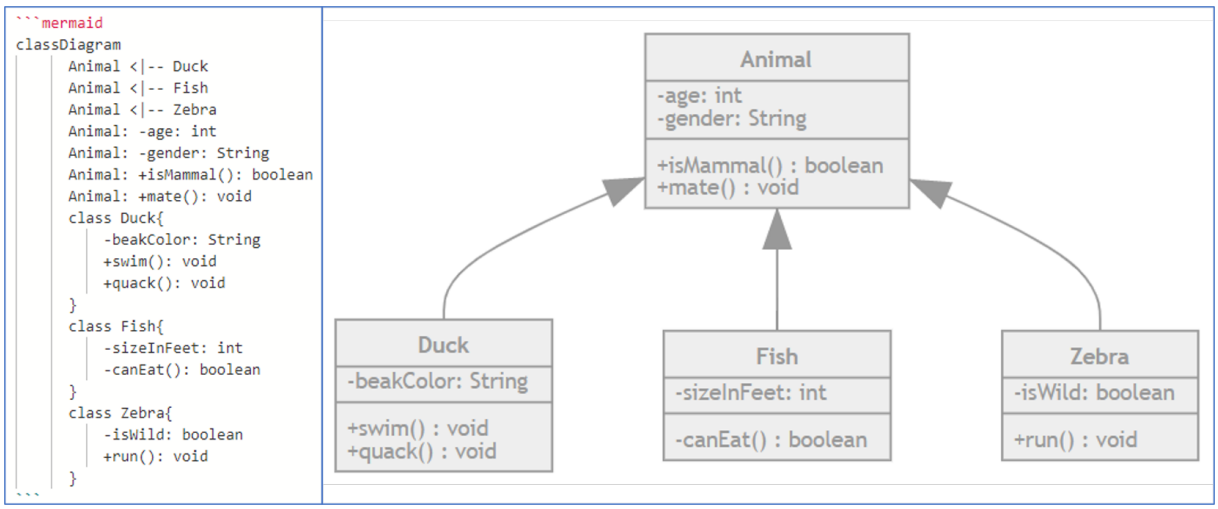

Figure 1: Sample code and output of class diagram with Markdown.

For higher marks: In addition to the previous, do some of the following ...

- Refactor the code by adding some design patterns to enhance maintainability.
- Organise the code to adhere to the MVC pattern.
- Create a permanent high score list (using a file to store scores).
- Add interesting levels to the game (either follow the original Sokoban game levels or come up with your own ideas).
- Add meaningful JUnit tests.
- Use build files (Ant or Maven or Gradle).
- **Other additions of game feature or other implementations that innovative and beyond creativity - surprise us!**

NOTE See Fig. 2 for GitLab repository template. Good programming practice will gain higher marks. Furthermore, nicely presented and easiness of using interfaces will be rewarded. A proportion of the marks will depend on you supplying a working version of your game, and submitting a video of it in use. The mark and feedback of the coursework will be uploaded to the Moodle coursework submission page. **High quality additions of game features are expected to gain higher mark, e.g. implementation of 3 basic and simple features (i.e. START screen, HIGH SCORE screen, sound effects etc.) will not gain you the "Good" performance, however, implementation of 1 or 2 creative features (at least half of the total (3) implemented features) allow you to gain the "Good" performance (see Marking Rubric).**

IMPORTANT You need to use Java 15 and JavaFX 15 for the implementation. The project files you are submitting need to be compatible with IntelliJ. **Fail to comply with any naming convention or structure organisation will result in penalty of -1 each.**

4 Plagiarism

You are gently reminded that we are at liberty to use plagiarism detection software on your submission. Plagiarism will not be tolerated, and academic offenses will be dealt with in accordance with UNNC policy and as detailed in the student handbook. This means you may informally discuss the coursework with other students but your submission must be your own. Please also note that it is not permitted for you to copy and paste text from another source without correct referencing. If you are unclear about this process, please discuss with the module convenors during the computing lab sessions or at the end of lecture sessions.

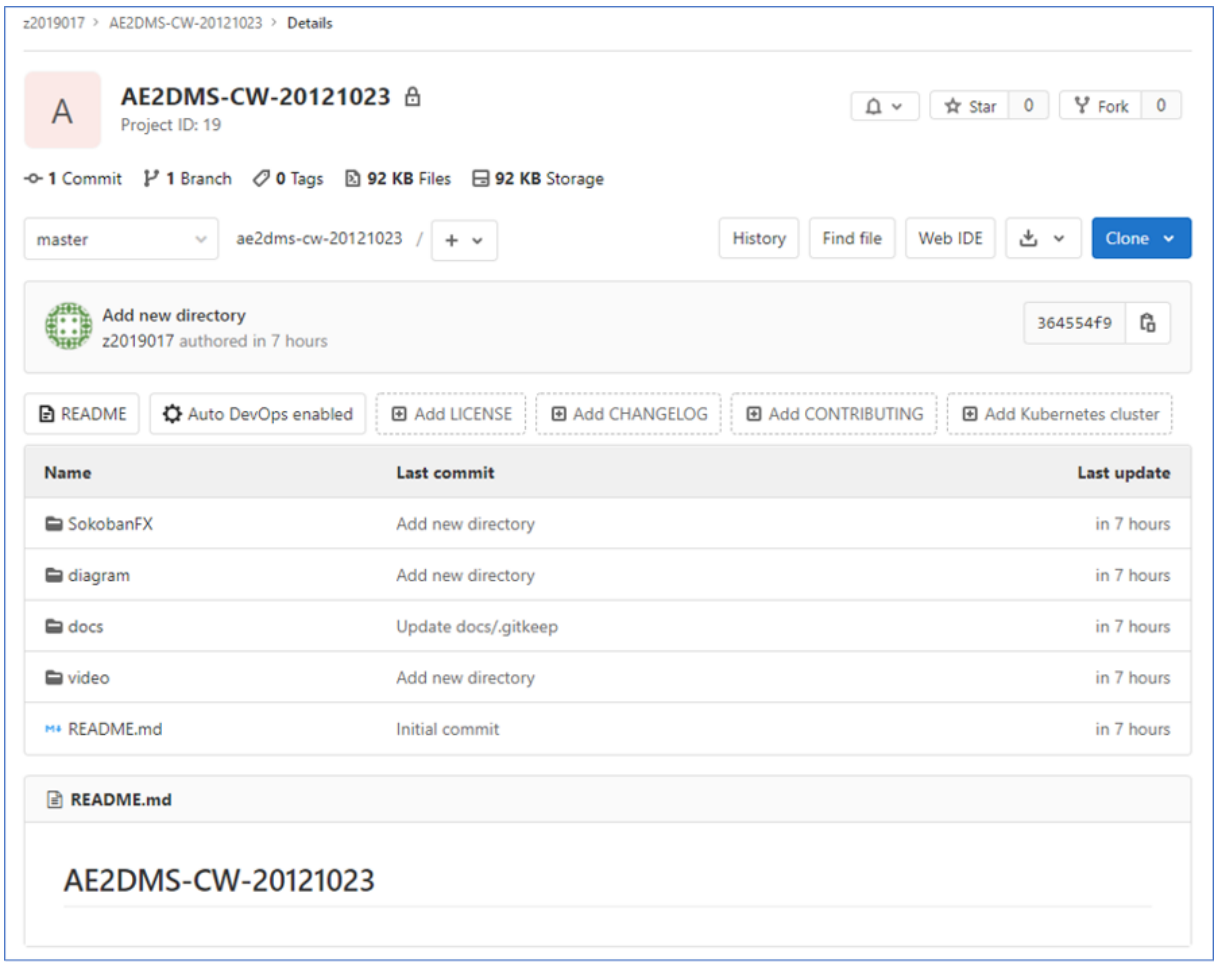

Figure 2: Template View of GitLab repository.

5 References

If you use any text or information from any resources such as books, journals, articles, proceedings etc., you have to provide those references. A guide for references and citation can be found here.

6 Q & A

Finally, if you have any questions, you are always welcome to post your questions in the forum under the topic Coursework-Q&A. Dr. Bryan and Dr. Heng Yu will of course be happy to answer questions and give high level interim formative feedback on your coursework during the computing lab sessions. However, we are refused to answer very detailed technical questions or very general questions such as "What do you think about my project so far?".

REMINDER Make sure you follow strictly the instructions/guideline provided for posting the questions in the forum.

Good Luck, Have Fun, Code Well.

DMS: "It's working. I'm Safe. Coursework is secure!"

Criteria	Fail	Very Poor	Poor	Good	Very Good	Excellent	Exceptional
Git Use **[15%]**	[0%] No evidence of git repository set up on this project.	[1% - 2%] Remote Git repository is set up but low usage activities recorded. Evidence of local Git repository is found but little-to-no knowledge of git use evidence presented.	[3% - 5%] Remote Git repository is set up but low usage activities recorded. Evidence of local git repository is found but little-to-no evidence of commitments and purpose use of Git are presented.	[6% - 8%] Remote Git repository is set up with basic usage evidenced. Local repository is found with basic commitment and poor explanation. The purpose use of git is not sufficiently presented.	[9% - 11%] Remote Git repository is set up with basic usage evidenced. Local repository operation is found with moderate commitment and with proper and detailed explanations. Branching feature is incorporated.	[12% - 13%] Remote Git repository is set up with extensive usage evidenced. Local repository operation is found with high commitment and with informative and clear explanations. Principal use of branching is greatly illustrated.	[14% - 15%] Remote Git repository is set up with rich set of usage evidenced. Local repository operation is found with high commitment and stunning explanation, branching is used extensively. Git work flow is clearly explained, demonstrated professional way of exercising git tool for software development and maintenance, satisfying the aim of this module.
Refactoring **[30%]**	[0%] No evidence of code refactoring presented in this project.	[1% - 5%] There is little evidence that code refactoring is presented with no improvement to this project.	[6% - 10%] There is some evidence that relevant code refactoring principles have been incorporated into this project. At best, there are loose principles.	[11% - 15%] Your code refactoring refers to couple of refactoring principles with at least one OO design patterns documented, but largely lacks precision and citations.	[16% - 20%] Your code refactoring shows evidence that aligned with MVC GUI design pattern and at least two OO design patterns, but your terminology may be imprecise. You may be missing some key concepts or some of your refactoring may be incorrect or your references may be incorrect.	[21% - 25%] Your code refactoring shows significant evidence that were supported by the MVC GUI design pattern and at least two OO design patterns. Your terminology may is precise and correct. Appropriate references have been used.	[26% - 30%] The majority code refactoring principles are justified by appropriate citations with MVC GUI design pattern and three or more OO design patterns. Those patterns are clearly articulated, use precise terminology, and cite appropriate source.
Additions **[30%]**	[0%] None additions of game features documented or presented in this project.	[1% - 5%] Some detailing that hint towards additional game features, but not explained.	[6% - 10%] Some detail and explanation of very small additional game features. Poorly explained.	[11% - 15%] 3+ additional game features are nicely implemented. Briefly explained. The additions are evaluated based on implementation difficulty and complexity.	[16% - 20%] 4+ additional game features are nicely implemented. Clearly explained but poorly reasoned/justified. The additions are evaluated based on implementation difficulty and complexity.	[21% - 25%] 5+ additional game features implemented and well justified. The additions are evaluated based on implementation difficulty and complexity.	[26% - 30%] 6+ additional game features are implemented which demonstrates creativity beyond expectations. All are well explained and justified. The additions are evaluated based on implementation difficulty and complexity.
Documentation **[15%]**	[0%] No documentation provided in this project.	[1% - 2%] Little-to-no high level diagrams provided and package information is missing. README.md and Javadoc folder created, but with low commitment, quality, lack of justified explanations.	[3% - 5%] High level class diagrams provided, but insufficient package information. README.md and Javadoc folder created with moderate commitment but a number of quality issues exists, includes poor explanations of parameters and methods.	[6% - 8%] High level class diagrams provided with package information, but with some errors. README.md and Javadoc folder created with high commitment, explanations of parameters and methods, but with some errors.	[9% - 11%] High level class diagrams provided with organized and justified package information, but with some minor errors. README.md and Javadoc folder created with high commitment, well explained and justified, but with minor errors.	[12% - 13%] High level class diagrams provided with clear explanations of package information and presented in appropriate layout. README.md and Javadoc folder created with high commitment, all codes are with clear explanations, justified and show consistency in the project.	[14% - 15%] The high level class diagrams provided in professional, clear and considered manner with high quality package information. Javadoc presented with stunning explanations, correct format and identical quality with Java online libraries documentation. High quality and professional README.md with clear illustrations including utilisation of Markdown styling.
Video **[10%]**	[0%] No video submitted on this project.	[1%] Video showed with no explanations.	[2% - 3%] Insufficient and lack of informative contents presented in the video.	[4% - 5%] Meaningful information and small references of improvements presented but no text and/or no audio explanations in the video.	[6% - 7%] Informative contents with relevant references of improvements presented, but with no text and/or no audio explanations.	[8% - 9%] A moderate quality of game demonstration and relevant references to improvements, supported with text and audio explanations in the video.	[10%] A wealth of high quality, impactful of game demonstration and relevant references to improvements, supported with text and audio explanations. Video contents not lengthy, straight to the points with clear and appropriate pronunciation.