SENG201 – Software Engineering I
Project Specification
Island Trader

For project admin queries:
Matthias Galster — Miguel Morales
{matthias.galster, miguel.morales}@canterbury.ac.nz

For technical support, project help, hints and questions:
Danita Sun — Frankie Oprenario —
Jack van Heugten Breurkes — Sam Shankland
danita.sun@canterbury.ac.nz, {fop12, jsv22, sjs227}@uclive.ac.nz

March 29, 2021

1 Introduction

1.1 Administration

This project is a part of the **SENG201** assessment process. It is worth **25%** of the final grade. This project will be done in pairs (i.e., teams of two students). You must find your own partner and register the team on LEARN. Submissions from individuals will not be accepted. Pairs are not allowed to collaborate with other pairs, and you may not use material from other sources without appropriate attribution.

Plagiarism detection systems will be used on your report and over your code, so do not copy the work of others. Plagiarised projects will be referred to the University proctor for disciplinary actions. Your deliverables must be submitted on LEARN. Students will be asked to demo their project during lab and lecture times.

Actual dates and times are detailed in Section 6.3. The drop dead policy and other penalties are detailed in Section 6.4.

1.2 Outline

This project will give you an idea of how a software engineer would go about creating a complete application (which features a graphical user interface) from scratch. **In this project you will build a game in which you are a trader**

who travels by ship to trade goods on multiple islands. You will buy goods on one island then travel to another to sell them. You will keep traveling between islands and trading goods with the goal to earn as much profit as possible before the game ends.

The idea of the game is slightly open to allow some room for creativity, but please ensure you implement the main project requirements as that is what will be graded. We encourage you to think of the assignment as a *project* rather than a *programming assignment*. Programming assignments often have a clearly defined specification that you can follow step-by-step. However, in practice, software engineering projects require a lot of thinking from the engineers to find out what exactly to build, why, for whom, and how. Therefore, specifications and requirements are often vague and need to be clarified during development.

1.3 Project Management

Adequate planning will help you minimize risks which can negatively impact your project (e.g., falling behind, poor testing). Defining time estimates, clear goals and realistic milestones will give you a much higher chance to succeed. To help you with this, you must record the following data weekly:

- Hours you spent working on the project during the week.
- Activities you carried out during the week.
- If you achieved the goals planned for the week or not.
- The risks you identified or faced during the week.
- How satisfied you are with your and your partner's contribution to the project during the week.

A link to record these data is available on LEARN.

You will earn marks by entering the data on time. Similarly, you will lose marks if you fail to regularly record the data, see Section 6.4 for more details.

1.4 Help

This project can get confusing and frustrating at times when your code does not work. This will likely be the largest program you have written thus far, so it is **very important** to break larger tasks into small achievable parts, and then implement small parts at a time, **testing as you go**.

Having a nice tight modular application will help with debugging, so having appropriate classes is a must. If you are having problems, try not to get too much help from your classmates, and instead ask for help from your tutors. You can email them or ask them questions in labs. Always save your work and have backups, do not assume that the CSSE department will be able to recover any lost data.

2 Requirements

This section describes what your game must do and is written as a set of requirements. At the beginning of your project and to get started, try thinking of each requirement as a separate ticket that needs to be closed **before** others are started, it will help you have code which works and can be built upon, instead of a lot of broken spaghetti code.

Hint: Functionality can be placed in its own package or class. Modularisation is the key, especially when you begin GUI programming.

2.1 Setting Up the Game

When your game first starts it should ask the player to:

1. Choose a trader name. The length must be between 3 and 15 characters and must not include numbers or special characters.
2. Select how many days they would like the game to last (between 20 and 50 days).
3. Choose a ship to captain:
   (a) Different ships should have different characteristics. These shall include sail speed, cargo capacity, number of crew and endurance (amount of damage the ship can take). Get creative!
   (b) Each ship shall have a number of crew members based on some characteristic such as ship size.
   (c) The player shall have four ships to choose from.
4. Start your trading adventure.

During setup the set of islands, their stores and the routes between them should be constructed. These can be the same each time the game is played. One island should be selected as the trader's home island. This could be preset or assigned randomly.

2.2 Playing the Main Game

Once the adventure has been started, the main game can begin. The trader starts out on their home island with a default amount of money, ready to buy goods to sell. The amount of money is up to you. The following options will be displayed to the player:

1. View the amount of money you have and the number of days remaining.
2. View properties of your ship. This shall include:
   (a) The name of the ship.

(b) The total number of crew and the wage cost per day to sail.

(c) The remaining capacity of the cargo hold. Each purchased good shall take some amount of space in the cargo hold until it is sold.

(d) Any upgrades you have purchased for your ship.

(e) Any ship damage and the cost to repair.

3. View the goods you have purchased. For each show:

   (a) The amount paid for the item.

   (b) The amount the item was sold for and the island where it was sold if the item has been sold.

4. View properties of each island. This shall include:

   (a) The name of the island

   (b) For each route to the island from the current island where your ship is docked:

      i. The distance you must sail to reach the island.

      ii. Details about the route. E.g. this could include a description and/or probability of encountering specific random events (described below).

   (c) The items that the island's store sells and the prices of each item.

   (d) The items that the island's store buys and the prices they will pay for each item.

5. Visit the store on the island where your ship is docked and:

   (a) View items that are for sale and the prices of each item. Each island should have its own characteristics that determine the type of items that its store sells and buys. Items could include goods as well as upgrades like cannons for your ship.

   (b) View items that the store will buy and the prices they will pay for each item.

   (c) View the amount of money you have and the goods previously purchased as described in point 3 above.

   (d) Be able to purchase items that are for sale.

   (e) Be able to sell items that the store buys.

   (f) Be able to purchase and/or sell multiple items at a time without leaving the store.

6. Set sail to another island

   (a) There shall be five islands.

   (b) The player can to travel to another island at any time.

(c) The player can sail between any two islands but the origin must be the island where your ship is docked.

(d) The sailing duration from one island to another shall be measured in days and calculated from the route distance and ship speed. Upon arrival on an island the number of days played must be increased by the sailing duration.

(e) There could be more than one route to travel from one island to another. Some routes could be more dangerous than others, but may have benefits such as requiring fewer days travel and greater profitability.

(f) You must pay your crew wages before setting sail to an island. Wages must reflect the sailing duration and be based on the per day wage cost for your ship's crew.

(g) You must repair any damage to your ship before you can sail.

There will also be some random events you will need to implement. These will only happen while at sea. The chances of these events occurring should depend on the route. The player should be alerted when any of these random events occur:

1. Pirates!
   (a) You encounter pirates. They will try to board your ship and steal your goods.
   (b) To fend off the pirates you must play and win a number game based on rolling a die. The chances of winning this game should be based on the properties of your ship (E.g. a ship equipped with cannons could increase your chances of winning).
   (c) If the player loses the game the pirates take all the player's goods and:
      i. If the value of the goods do not satisfy the pirates then they take your ship and your money and make you and your crew walk the plank. The game is over.
      ii. If the pirates are satisfied with the goods they let you go and you continue your journey minus your goods.
2. Unfortunate weather
   (a) You meet unfortunate weather and take damage to your ship.
   (b) The amount of damage is random.
   (c) Upon arrival at the destined island you must pay to repair your ship before you can sail again. Repair cost should be based on the amount of damage incurred.
3. Rescued sailors
   (a) You and your crew rescue some sailors!
   (b) The saved sailors give you a monetary reward.

2.3 Finishing the Game

The game ends when one of the following occurs:

1. All days have passed or the number of remaining days is not enough to sail anywhere.
2. You lose all of your money or do not have enough to sail anywhere.
3. You are involved in an unfortunate event with pirates and are made to walk the plank.

Upon ending a screen will display the trader's name, the selected game duration in days, the actual duration in days (excluding any remaining days) and the profit made in this time. A final score shall also be displayed. How you score is up to you, but we recommend looking at the selected/actual game duration and the profit made.

2.4 Extra Credit Tasks

If you have finished early, and have a good looking application, then you will be in for a very high mark. If you want some more things to do, then please discuss it with the tutors in the lab, but you are free to add any features you wish, just be careful to not break anything. Here are some ideas, and you do NOT need to implement them all to get full marks:

- You may want to make sailing take real time and show progress as you sail. E.g. 1 day sailing could take 1 second.
- A player may want to save the current state of the game, and be able to load it up at a later time.
- You might want to put a story line into your game, so have consistent characters, and a plot which gets told through pop ups or dialogue.
- You might want to add some artwork.
- You might want to add some music or sound effects.

3 Design and Architecture

This section provides some **ideas** on how your program should be structured. The following are some class candidates, and should be represented in your program. Important things to think about here are interactions between classes, what classes should be instantiated, and how you could use interfaces and/or inheritance.

3.1 Ship

There shall be different types of ship. Four shall be enough. Each ship has at least a name, a crew, a maximum cargo capacity and damage status.

3.2 Island

There shall be five different islands. Each island has a name, store and a collection of routes to the other islands. There shall be a minimum of four routes (one to each island) but you may decide to have more than one route to an island.

3.3 Store

There shall be one store for each island. A store should have a collection of items that it sells and a collection of items that it buys.

3.4 Route

A route defines a trip between two islands. It will include the number of days sailing required and the two islands at either end of the route. It may also include a description or event probability to help the player decide how safe the route is.

3.5 Item

Items are goods that the player can buy and sell to make profit or could be upgrades for your ship. Each item has a name and description. Different items may have different sizes and shall take up space in the ship's cargo hold. The item price may depend on the island where the item is bought/sold. E.g. Different islands may be willing to pay more for the same item or sell the same item for less.

3.6 Random Event

A random event represents a circumstance that can happen during sailing. It should include behaviour to handle the event.

3.7 Game Environment

The game environment contains your game, and will implement functions to provide the options mentioned above, and will call methods of the above classes to make those options happen. The game environment keeps track of the game and should handle requests from the user interface and report back updates.

Try to keep your code modular and avoid putting user interface code in your game environment class. Keeping the game logic separate from the UI code will ensure that you can create both a command line interface and a GUI interface without having to alter your game environment.

4 Assignment Tasks

4.1 Sketching UML

Before you start writing code, sketch out a UML use cases diagram detailing the program's users (actors) and their interactions with the system (use cases). This diagram will help you to identify the main functionalities of the program and to visualize its scope.

In addition to this diagram, create a UML class diagram of how you think your program will look like. It will help you get an idea of what classes there are, what class attributes are required, and will get you thinking of what classes communicate with other classes (call methods of another class).

4.2 Implementing a Command Line Application

Begin implementing classes, starting with ship, island, store, items, route and the game environment. Make a command line UI class that drives a simple command line application that works in a simple runtime loop which:

1. Prints out a list of options the player may choose, with numbers next to the options.
2. Prompt the player to enter a number to complete an option.
3. Read the number, parse it and select the correct option.
4. Call the method relating to the option and if necessary:
   (a) Print out any information for that option, such as select a route.
   (b) Read in the number for the information offered.
   (c) Parse the number and complete the action.
5. Go back to step 1.

This will enable you to slowly build up features, and we recommend to only implement one feature at a time. Make sure you test your feature before moving onto implementing more features. Once you are feature complete and have a working game, you may move onto implementing a graphical application.

The command line application will only be assessed if there is no graphical application, or if there are fatal bugs in the graphical application. In this case, partial marks will be awarded for correct command line functionality. **Hint:** For a very basic solution to read input from the command line you may want to explore class Scanner in the Java API.

4.3 Implementing a Graphical Application

You will be implementing a graphical application for your game using **Swing**, which will be explained in labs. For the purposes of this assignment, we do not recommend writing the Swing code by hand, and instead using the interface builder offered by the Eclipse IDE. This lets you build graphical interfaces in Swing by dragging and dropping components onto a canvas onscreen, and it will automatically generate the code to create the graphical application you built.

Please note, you are required to ensure that any automatically generated code complies with the rest of your code style, so you will need to change variable/method names and code layout.

Once you have built your interface, the next task is to wire up the graphical components to the methods your command line application supplies, and to update the onscreen text fields with the new values of your class attributes/member variables. Most of these functions are triggered on `onClick()` methods from buttons. Start small, and complete Section 2.1 "Setting up the Game" first to get used to GUI programming. You might need to slightly adjust your methods to achieve this. Then move onto the main game.

Note that this is the largest task to complete and many students under estimate how much time it will take. Try to be at this stage one week after the term break if possible.

4.4 Writing Javadoc

Throughout your application, you need to be documenting what you implement. Each attribute of a class should have Javadoc explaining what its purpose is. Each method needs to explain what it does, what variables it takes as parameters, and what types those variables are. You should be building your Javadoc regularly, as it integrates into the IDE very nicely, and will aid you in writing good code.

4.5 Writing Unit Tests

You should design JUnit tests for your smaller, basic classes, such as Ship, Island, Store, and their descendants if you think necessary. Try and design useful tests, not just ones that mindlessly verify that getters and setters are working as intended.

4.6 Report

Write a short two page report describing your work. Include on the first page:

- Student names and ID numbers.
- The structure of your application and any design choices you had to make. We are particularity interested in communication between classes and how inheritance was used. You might want to reference your UML class diagram.

- An explanation of unit test coverage, and why you managed to get a high/low percentage coverage.

Include on the second page:

- Your thoughts and feedback on the project.
- A brief retrospective of what went well, what did not go well, and what improvements you could make for your next project.
- The effort spent (in hours) in the project per student.
- A statement of agreed % contribution from both partners.

4.7 A note on effort distribution

The "typical" or "common" distribution of the overall effort spent on these activities is: around 5% creating the UML diagrams; 20% developing the command line application; development of the graphical application is the most time consuming task taking around 50% of your time; documenting your code would take 5%; creating the unit tests around 15% and writing the report would take the last 5%.

However, note that these numbers vary between students and these percentages are not supposed to be the exact amount of effort invested in each task. They are only a rough guideline (e.g. we would not expect you to spend 50% of your time on creating UML diagrams or writing the report; on the other hand, we would expect that implementing the graphical user interface takes quite a substantial portion of the effort).

5 Deliverables

5.1 Submission

Please create a ZIP archive with the following:

- Your source code (including unit tests). We want your exported project as well so that we can easily import it into Eclipse.
- Javadoc (already compiled and ready to view).
- UML use case and class diagrams as a PDF or PNG (do not submit Umbrello or Dia files; these will not be marked).
- Your report as a PDF file (do not submit MS Word or LibreOffice documents; these will not be marked).
- A README.txt file describing how to build your source code and run your program.

- A packaged version of your program as a JAR. We must be able to run your program **in one of the lab machines** along the lines of: `java -jar usercode1_usercode2_IslandTrader.jar`.

Submit your ZIP archive to LEARN before the due date mentioned in Section 6.3. Only one partner of the pair is required to submit the ZIP archive.

5.2 Lab Demos

During the last week of term, you will be asked to demo your project during lab and lecture time. Each team member must be prepared to talk about any aspect of your application, we will be asking questions about any and all functionality. There will be a form on LEARN in which you can book a timeslot, ensure you are both available, as you must attend the demo as a pair.

6 Marking scheme

6.1 Overall Assignment [100 marks]

6.1.1 Functionality and Usability [45 marks]

We will be testing the extent to which your code meets the requirements using the graphical interface. This includes running the program and executing its main functionalities.

If your graphical application is broken or faulty, partial marks will be awarded for your command line application.

6.1.2 Code quality and Design [15 marks]

We will be examining the code quality and design of your program. This aspect include: your naming conventions, layout and architecture, and use of object oriented features, among others.

Quality of your comments is also very important. You may wish to run `checkstyle` over your code before you hand it in.

6.1.3 Documentation [15 marks]

We will be looking at your use of Javadoc, and how well it describes the attribute or method you are commenting on, and how well it covers your codebase.

6.1.4 Testability [15 marks]

We will run your unit tests to see how well they cover your code, and we will examine the quality of those tests. Try to make your tests do something other than verifying that getters and setters work.

6.1.5 Report [10 marks]

Your report will be marked based on how well written it is and the information it conveys about your program. The report must include what is specified in section 4.6

Employers place a lot of value on written communication skills, and your ability to reflect on a project, especially in agile programming environments.

6.2 Lab Demos

During the lab demos, you and your partner will be showing the examiners how your program works and you may be asked to:

- Construct a new game, and choose a ship.
- Perform actions as the captain (view your goods).
- View the status of the ship.
- Sail to another island.
- Visit a store and sell/buy items.
- Show that random events occur.
- Show that the game can be completed.
- Show that the game runs without errors, obvious bugs or crashes.
- Fulfils any or all of the requirements set.
- You may be asked to explain how your graphical interface is written, and point to specific code.
- You may be asked about how inheritance works in your program, again pointing to specific code.
- Anything else that the examiner wishes to ask about.

If you do not turn up to demo in your timeslot, you will be penalised (see Section 6.4).

6.3 Important Dates

- **Project launch:** March 29, 2021.
- **Weekly progress deadlines:**
  - **Week 6** (29 March-04 April, 2021) progress must be recorded by April 06, 2021 at 11:59pm.
  - **Week Break-1** (05-11 April, 2021) progress (if any) should be recorded by April 13, 2021 at 11:59pm.

  - **Week Break-2** (12-18 April, 2021) progress (if any) should be recorded by April 20, 2021 at 11:59pm.
  - **Week Break-3** (19-25 April, 2021) progress (if any) should be recorded by April 27, 2021 at 11:59pm.
  - **Week 7** (26 April-02 May, 2021) progress must be recorded by May 04, 2021 at 11:59pm.
  - **Week 8** (03-09 May, 2021) progress must be recorded by May 11, 2021 at 11:59pm.
  - **Week 9** (10-16 May, 2021) progress must be recorded by May 18, 2021 at 11:59pm.
  - **Week 10** (17-23 May, 2021) progress must be recorded by May 25, 2021 at 11:59pm.
  - **Week 11** (24-27 May, 2021) progress must be recorded by May 29, 2021 at 11:59pm.

- **Last day for registering teams:** April 30, 2021 at 5pm. After this time, the teaching team will randomly allocate those students without a pair. Allocations will be final.
- **Last day for booking a time slot for the demo:** May 27, 2021 at 5pm. After this time, the teaching team will allocate time slots to those teams without a booked slot. Allocations will be final.
- **Project submission due date:** May 27, 2021 at 5pm.
- **Lab demos:** during lab and lecture time of May 31-June 04, 2021.

6.4 Penalties

- **-5 marks** for failure to register your team by the given deadline.
- **-3 marks** for failure to record your weekly progress by the weekly deadline, see Section 6.3 for details. This will be applied only to the student in the pair who failed to record their weekly progress.

  The penalty applies as many times as you fail to record progress. For example, if you missed it twice, a penalty of -6 marks is applied (2 x (-3) = -6).

  It is not allowed to record the weekly progress in one go at the end of semester.

  Students who record their weekly progress in a timely manner for all weeks (excluding the break weeks) will be granted a one-time bonus of +5 marks. This will only apply to the student in the pair who recorded their weekly progress not later than two days after the reported week ended.

  You are not expected to work on the project during the term break. However, if you decide to do this, it would be beneficial for you to keep recording your weekly progress.

- **-5 marks** for failing to book a time slot for the demo by the given deadline, see Section 6.3 for details. It applies to both members of the team.
- **-15 marks** for submitting after the due date PLUS **-1 mark** per each hour of delay. It applies to both members of the team and it is capped up to -100 marks, i.e., the total marks of the assignment.

  For example, if you submit 16 hours after the due date, a penalty of -31 marks will be applied [(-15) + (-16) = -31].
- **-30 marks** for failing to show up to the demo. It applies only to the student who missed the demo.
- **-100 marks** if plagiarism is detected.