SENG201 (Software Engineering I) Project
Space Explorer

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

For project help, hints and questions:
Jack Steel — Maree Palmer
{jes143, mpa588}@uclive.ac.nz

March 25, 2019

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, and you must find your own partners. Please register your partnership on LEARN before two weeks from the project being handed out. 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.

Please submit your deliverables on LEARN no later than **5pm, Friday 24th of May**. Students will be asked to **demo their project during labs and lectures** in the week of the **27th of May** (week 12). The drop dead date is the **31st of May**, with a standard 15% late penalty.

1.2 Outline

This project is to give you a brief idea of how a software engineer would go about creating a complete application from scratch that has a graphical interface. **In this project, you will build a game in which you and your crew are lost in space with your spaceship. Your spaceships has been broken and its pieces are scattered throughout the surrounding planets. You will need to find the missing pieces of your spaceship so that you can repair it and get home.** 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.

1.3 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, **Jack** and **Maree**. 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. Try thinking of each requirement as a separate ticket that needs to be closed **before** others started, and it will help you have code which works and can be built upon, instead of a lot of broken spaghetti code.

Hint: Functionality in each subsection can be placed in its own module 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 how many days they would like the game to last (between 3 and 10 days). The number of pieces that the player needs to find is derived from the number of days. There should be about 2/3 the amount of pieces to find as there are days, rounding down if it doesn't divide evenly. For example, a game lasting three or four days should have two pieces to find. A game lasting five days should have three pieces to be found, and so on.

2.2 Creating Crew Members

1. Select the number of crew members you would like.
2. Choose your crew:
    (a) Choose 2 - 4 crew members from 6 different types of potential crew members. The same type of crew member may be selected more than once.
    (b) Different types of members should have different strengths (more health, more skilled at repairing spaceship or searching for parts).

   (c) The player should be able to name each crew member.

3. Name your ship.

4. Start adventure.

2.3 Playing the Main Game

Once the adventure has been started, the main game can begin. The crew find themselves on a far away planet with their spaceship. There will be a series of options displayed to the player. Some of these options constitute an "action", and each crew member may only perform a maximum of two actions per day. The player should be able to perform the following activities (note that these do not count as crew member actions):

1. View the status of the crew member. This includes viewing their hunger, tiredness, and health levels. Crew attributes will be better explained in section 3.

2. View the status of the spaceship. The player should be able to view the shield health of the spaceship.

3. Visit the nearest space outpost and:

   (a) View objects, such as food and medical supplies that are for sale.

   (b) Show what objects the player currently owns, their amounts, and the amount of money the player has.

   (c) See the prices of each object.

   (d) See the attributes of the object, attributes will be better explained in section 3.

   (e) Enable the player to purchase objects such as food and medical supplies.

   (f) Be able to purchase multiple objects at a time without leaving the outpost.

4. Move on to the next day.

   (a) The player should be able to move to the next day at any time, even when there are still actions remaining for any of the crew members.

Crew members should be able to perform the following actions, unless stated otherwise, each counts as one action for the crew member performing it.

1. Eat food or use medical supplies from the ship's inventory.

   (a) The player should be able to choose what food or medical item the crew member eats or uses.

   (b) This should consume the food or medical item, and decrease the crew member's hunger or increase health level by some amount.
   (c) Food will be explained more in depth in section 3.

2. Sleep.
   (a) Sleeping should make the crew member less tired.

3. Repair the shields of the ship.
   (a) This should increase the shield level of the ship by some amount.

4. Search the planet for spaceship parts.
   (a) When searching a planet, a crew member may find a transporter part, a food item, a medical item, money, or nothing. There should be a random chance of finding any of these.
   (b) Only one transporter part may be found per planet.

5. Pilot the ship to a new planet.
   (a) Two crew members are required to pilot the ship to a new planet, so the player must have two crew members with at least one action left each to perform this action.
   (b) The player should be able to choose which crew members will pilot the ship.

There will also be some random events which you will need to implement. These should only happen at the start of a new day (or when moving to a new planet for an asteroid belt). The player should be alerted when any of these random events occur:

1. Alien pirates.
   (a) Alien pirates board the ship and steal an item.
   (b) A random item is removed from the ship inventory.

2. Space plague.
   (a) One or more of your crew members comes down with space plague.
   (b) This should decrease the health of those affected.
   (c) The crew member remains sick and loses more health each day until they take medicine for the space plague (which is one of the medical items that may be purchased from the space outposts).

3. Asteroid belt.
   (a) The ship goes through an asteroid belt.
   (b) This should cause damage to the ship, decreasing the shield level.
   (c) The effect of the damage should be scaled by the shield level of the ship at the time of encountering the asteroid belt.

2.4 Finishing the Game

After all pieces of the spaceship have been found or all days are completed, the game should end. A screen should display the ships name, the number of days taken to complete the game, and whether all pieces of the spaceship were found.

A final score should be displayed. How you score is up to you, but we recommend computing a daily score of status attributes.

2.5 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:

- 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.
- The player might want the ability to create their own crew and ship, by using their own custom names and pictures.

3 Design and Architecture

This section provides some **ideas** with 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 what roles inheritance should play.

3.1 Crew

All crews have a name and a ship, and contains a list of crew members, a list of medical items the team has, a list of food items the crew has and the amount of money the crew has remaining.

3.2 Crew Members

Crew members have a name and a type. Six types will be enough. The type influences the rate at which their health, hunger, and tiredness degrade over time. Crew members have a health level, which defaults to 100%. Some crew members may have a special ability, such as being more effective at repairing the shields of the ship. When a crew member's health drops to 0, the crew member dies and is removed from the crew.

3.3 Medical Items

There should be different types of healing items, three will be enough. One of the items must be a cure to the space plague. Other medical items should increase the health of the crew member by some amount. All medical items have a price, and prices will depend on how much health is restored.

3.4 Food Items

There should be different types of food items, six will be enough. All food has a price, and the prices will vary depending on the food. Food will also differ in nutrition. By feeding food to a crew member, their hunger level should be affected.

3.5 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 that option happen. The game environment keeps track of a crew. The game environment instantiates crew members, items, and transporter pieces, and places the objects where they belong.

All of the game logic will be placed in the game environment, such as the `feed()` method may call `crewMember.feed(foodItem);` or similar. This class will get large, please try and keep it modular.

4 Assignment Tasks

4.1 Writing UML

Before you start writing code, sketch out 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 crew, crew members, items, ship and the game environment. Make the game environment a simple command line application which 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 use a food item.
   (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 Crew, Crew member, Medical Items, Food Items 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.
- Where collections have been used and in what form.
- 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).
- 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).
- Your report as a PDF file. (Do not submit MS Word or LibreOffice documents).
- 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_SpaceExplorer.jar`.

Submit your ZIP archive to LEARN before the due date mentioned on the first page. 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, please ensure you are both available, as you must come 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 [-30% penalty on failure to show up]

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 create a crew.
- Perform actions with crew members.
- View the status of the space ship.
- Move on the next day.
- Find a space outpost and 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 30% of your marks for the assignment.