SENG201 – Software Engineering I
Project
Farm Simulator

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

For technical support, project help, hints and questions:
Sam Shankland — Bach Vu — Luke Walsh — Danita Sun
{sjs227, bvv10, lwa383 }@uclive.ac.nz, danita.sun@canterbury.ac.nz

April 09, 2020

1 Introduction

1.1 Administration

This project is a part of the **SENG201** assessment process. It is worth **30%** 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.

See Section 6.3 for actual dates and times. 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 starting a new life on your new country farm. You can buy crops and animals to manage on your farm. You will need to care for your crops and**

animals to make the biggest profit for your farm. 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 Project Management

An adequate planning will help you minimize risks which can negatively impact your project (e.g., falling behind). Defining time estimates, clear goals and realistic milestones will give you 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.
- How satisfied you are with 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. Try thinking of each requirement as a separate ticket that needs to be closed **before** others are 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 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 how many days they would like the game to last (between 5 and 10 days).

2.2 Creating The Farm

1. Give your farmer a name. The length of the name must be between 3 and 15 characters and must not include numbers or special characters.
2. Select the type of farm you would like to own.
   (a) Different types of farm should have different characteristics (e.g., strengths and bonuses), such as faster crop growth, higher animal happiness, or a larger cash bonus for starting. Get creative!
   (b) The player should be able to choose between four different types of farm.
3. Name your farm.
4. Start your farming adventure.

2.3 Playing the Main Game

Once the adventure has been started, the main game can begin. The farmer starts out on their brand new farm with a set amount of money, ready to fill with crops and animals. There will be a series of options displayed to the player. Some of these options constitute an "action", and the farmer may only perform a maximum of two actions per day. Furthermore, the player should be able to perform the following activities (which do not count in the farmer's two daily actions):

1. View the status of the farm's crops and animals. This includes viewing a crop's time growing, the time left until a crop's harvest, and an animal's happiness levels. Crop and Animal attributes will be explained in more detail in section 3.
2. View the status of the farm. The player should be able to view the farm's current money available.
3. Visit the county's general store and:
   (a) View items, such as crops, animals, and farming supplies that are for sale.
   (b) Show what items the player currently owns, their amounts, and the amount of money the player has.
   (c) View the prices of each item.

   (d) View the properties of each item, those will be better explained in section 3.
   (e) Enable the player to purchase items such as crops, animals, and farming supplies.
   (f) Be able to purchase multiple items at a time without leaving the store.

4. Move on to the next day.
   (a) The player should be able to move to the next day at any time, even when the farmer still has actions left.
   (b) At the end of each day, the farmer will get bonus money depending on the happiness and healthiness of any animals they have.

A farmer should be able to perform the following actions, unless stated otherwise, with each counting as one action.

1. Tend to crops.
   (a) Tending to crops will speed up their growing process, decreasing the amount of time until they can be harvested.
   (b) A farmer will only be able to tend to one variety of crop in one action - the days pass by quickly with the busy farming life!
   (c) Crops can be tended to by watering them (a free option), or by using an item on them.
2. Feed animals.
   (a) Feeding animals will make their healthiness increase.
   (b) Food items are used to feed animals.
3. Play with animals.
   (a) Playing with the animals will make their happiness increase.
4. Harvest crops.
   (a) This allows crops that have fully grown to be harvested for a money bonus.
5. Tend to the farm land.
   (a) Tending to the farm's land keeps the farm tidy and well maintained.
   (b) Keeping the farm well maintained allows for more crops to be grown, and keeps the animals happier for longer with no obstacles to keep them from living their best life.

There will also be some random events which you will need to implement. These should only happen at the start of a new day. The player should be alerted when any of these random events occur:

1. Drought.
   - (a) The wells have dried up, and the crops are thirsty.
   - (b) The player loses half of their growing crops. The exact crops are determined randomly.
2. Broken fence.
   - (a) One or more of your animals have escaped through a broken fence.
   - (b) The player loses one or more of their animals, the exact number determined randomly.
   - (c) The remaining animals lose a substantial amount of happiness.
3. County fair.
   - (a) Your farm has won the top award at the annual county fair.
   - (b) The player earns a reasonable sum of money.
   - (c) The amount of money earned should be scaled by the number of crops and animals the farm contains.

2.4 Finishing the Game

After all days are completed, the game should end. A screen should display the farm's name, the game's duration in days, and the profit the farm made in this time.

A final score should be displayed. How you score is up to you, but we recommend looking at game duration, number of crops and animals, animal status, and money earned.

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 customize their farm, by using their own custom names for animals, and potentially artwork.
- You might want to make some calming music play while your players tend to their farm.

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 Farm

All farms have a name, a type and a farmer, and contain a list of crops, a list of animals, and the amount of money the farm currently has. The type of the farm determines which bonuses the farmer will have during the game, such as crop growing speed, or animal happiness bonuses. Four types will be enough.

3.2 Farmer

Farmers have a name and age. Extra attributes can be added to the farmers and integrate those to the game, for example, farmer skills or an avatar.

3.3 Crops

There should be different types of crops, six should be enough. Each crop has a purchasing price, and a selling price. Crops should have a set number of days before they are ready for harvest. This amount of days could be altered by tending to the crops.

3.4 Animals

There should be different types of animals, three will be enough. All animals should have a purchase price. Animals give a daily amount of money from tending to them at the end of each day. They should also have health and happiness attributes, which affect their daily bonuses.

3.5 Items

Items can be used in game to tend to crops or to feed animals. Six types of items would be sufficient. Items should have a name, a price, and they should specify the health given (for food items for animals), or the number of days the harvest time for crops has been sped up by (if it is a crop item).

3.6 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 farm. The game environment instantiates farmers, crops, and animals, 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 `animal.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 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 farm, farmer, crops, animals, items, 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 Farmer, Crops, Animals, 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_FarmingSimulator.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 create a farm.
- Perform actions with the farmer.
- View the status of the farm.
- Move on the next day.
- Find a general store 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 marks).

6.3 Important Dates

- **Project launch:** April 21, 2020.
- **Weekly effort deadlines:**
  - Week 6 (21-27 April, 2020) effort must be recorded by April 29, 2020 at 11:59pm.
  - Week 7 (28 April-04 May, 2020) effort must be recorded by May 06, 2020 at 11:59pm.
  - Week 8 (05-11 May, 2020) effort must be recorded by May 13, 2020 at 11:59pm.
  - Week 9 (12-18 May, 2020) effort must be recorded by May 20, 2020 at 11:59pm.
  - Week 10 (19-25 May, 2020) effort must be recorded by May 27, 2020 at 11:59pm.
- **Last day for registering teams:** May 1, 2020 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 18, 2020 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 25, 2020 at 5pm.
- **Lab demos:** during lab and lecture time of May 26–29, 2020.

6.4 Penalties

- **-5 marks** for failure to register your team by the given deadline.
- **-3 marks** for failure to record your weekly effort 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 effort.

  The penalty applies as many times as you fail to record effort. 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 effort in one go at the end of semester.

  Students who record their weekly effort in a timely manner for all 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 effort not later than two days after the reported week ended.
- **-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.