The University of Melbourne
Department of Computer Science and Software Engineering
SWEN20003 Object Oriented Software Development

Project 2, Semester 1, 2019

Released: Monday 29th of April
Project 2A due: Friday 10th of May, 7:00pm
Project 2B due: Friday 31st of May, 7:00pm
Updated Saturday 11th of May

Overview

In this project, you will create a graphical real-time strategy game in the Java programming language, continuing from your work in Project 1. We will provide a full working solution for Project 1; you are welcome to use all or part of it, provided you add a comment explaining where you found the code.

This is an **individual** project. You can discuss it with other students, but all submitted code must be your own work. You can use any platform and tools you like to develop the game, but we recommend using the Eclipse IDE, since that is what we are supporting in class.

You will not be required to design any aspect of the game itself; this document should provide all necessary information about how the game works. You will, however, be required to design the classes for your software solution before you implement it.

There are two parts to this project, with different submission dates.

The first task, Project 2A, requires that you produce a class design demonstrating how you plan to implement the game. This should be submitted in the form of a UML diagram showing all the classes you plan to implement, the relationships (e.g. inheritance and associations) between them, and their attributes, as well as their primary public methods. (Methods such as getters and setters need not be explicitly included.) If you so choose, you may show the relationships separately to the class members, but you must use correct UML notation. **Please submit as PDF only.**

The second task, Project 2B, is to complete the implementation of the game as described in the rest of this specification. You do not have to follow your class design; it is there to encourage you to think about object-oriented principles before you start programming. If you do end up changing your design, we ask that you produce an updated UML diagram and submit it with your project.

Shadow Build

Game overview

Shadow Build is a strategy game where the player must build structures on an alien planet and train units to mine the planet for the precious resource *unobtainium*. The game takes place on a large map containing different kinds of terrain and resources. Broadly speaking, the game is divided into *units* that can be controlled to travel to different parts of the map, *buildings* that can be controlled to create new units, and *resources* that certain types of units can collect.

The two resources are *metal* and *unobtainium*. The goal of the game is to collect unobtainium as efficiently as possible; it has no other purpose. Creating buildings and units costs a certain amount of metal.

The game map

The map contains background tiles, some which are *solid* and cannot be moved through by units, and others which are not solid. The map should be loaded in the same manner as in Project 1. Some tiles are now *occupied*, which means that buildings may not be created such that the centre of the building lies on them. This is a property implemented in the same way as the *solid* property from Project 1.

For Project 2, you will additionally have a file `objects.csv`; this file contains a list of units, buildings, and resources that should be created at the start of the game. It is in comma-separated value (CSV) format, and contains lines of the following form:

```
commandcentre,812,748
```

The first entry is the name of the object to be created, followed by the `x` coordinate and the `y` coordinate it should be created at.

Units

There are four types of units in this game. All units can be *selected* by **left**-clicking within 32 pixels of them; only one unit can be selected at a time, and if you click away from any unit, the selected unit (if there is one) should be *deselected*. If there are multiple units close to a click, you may choose any one to be selected. When a unit is selected, **right**-clicking should make the unit move towards the location that was clicked until the unit reaches that location.

A unit that is selected should have a highlight drawn under it; this can be found in the file `highlight.png`. **Units can move through buildings and resources.**

- **The scout**

  The scout moves the fastest out of any unit, but has no other special attributes. The scout should move at a rate of 0.3 pixels per millisecond.

- **The builder**

The builder can create factories to train trucks. When the builder is selected and the 1 key is pressed, the builder starts building at a cost of 100 metal. After 10 seconds, a factory should be created at the builder's current location. The builder should not move during this time. If there is not enough metal available, pressing the 1 key should do nothing.

The builder should move at a rate of 0.1 pixels per millisecond.

- **The engineer**

The engineer can mine metal and unobtainium. When the engineer has spent 5 seconds near a resource, the engineer starts carrying some of the resource, beginning at 2 units of the resource. The engineer cannot carry more than this maximum, unless pylons are activated (see below).

When this happens, the engineer should start moving directly (in a straight line)towards the nearest command centre. When the engineer reaches the command centre, they should drop the resource off, and the amount should be added to the player's total. The engineer should then start moving directly back towards the resource.

If the engineer is interrupted by being ordered to move to another location, the engineer stops their planned motion.

The engineer should move at a rate of 0.1 pixels per millisecond.

- **The truck**

The truck can create command centres. It should behave similarly to the builder, except that it should take 15 seconds, cost no metal, and when the command centre is complete the truck should be destroyed.

The truck should move at a rate of 0.25 pixels per millisecond.

Resources

Resources are located at different parts of the map. They begin with a certain amount of the relevant resource which is depleted when engineers mine them. When the resource has none of the

resource left, it should be destroyed.

- **The metal mine**

  The metal mine begins with 500 metal.

- **The unobtainium mine**

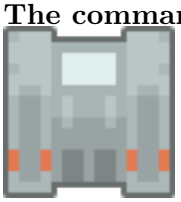
  The unobtainium mine begins with 50 unobtainium.

Buildings

Buildings are either already on the map, or can be constructed by units. They can be selected in the same way as units; a building and a unit cannot both be selected at the same time. If a unit and a building are close together, the unit should be selected first.

When a building is selected, it should have a highlight drawn under it, given by the file `highlight_large.png`.

- **The command centre**

  The command centre can train (create) most units when selected. When the 1 key is pressed, a scout should begin training at a cost of 5 metal. When the 2 key is pressed, a builder unit should begin training at a cost of 10 metal. When the 3 key is pressed, an engineer should begin training at a cost of 20 metal.

  A unit takes 5 seconds to train, and only one unit can be trained at a time. When the time is up, the unit should be created at the building's location.

- **The factory**

  The factory can train trucks in the same manner as the command centre. When the 1 key is pressed, a truck should begin training at a cost of 150 metal.

- **The pylon**

The pylon cannot be built. Instead, they are already on the map. When a unit comes within 32 pixels of the pylon, the pylon becomes activated and should change to the active image (permanently). Each active pylon should allow **all** engineers to carry 1 additional resource.

The camera

The camera should behave similarly to Project 1, except that it should follow the currently selected unit or building. If the player presses the W, A, S, and D keys, the camera should stop following the selected unit, and move 0.4 pixels per millisecond up, left, down, or right respectively. If a unit is selected after the player presses one of these keys, the camera should start following that unit until the player presses one of the keys again (moving instantly to follow the unit).

The text display

The player needs to know some information about the game. At the coordinate (32,32), the text `Metal: <amount>\nUnobtainium: <amount>` should be drawn, with the appropriate amounts listed. When a unit or building that can create other units or buildings is selected, the actions associated with the 1, 2, and 3 keys should be drawn at the coordinate ~~(100,32)~~(32,100). For example, the command centre should draw `1- Create Scout\n2- Create Builder\n3- Create Engineer\n`.

Implementation checklist

This project may seem daunting. As there are a lot of things you need to implement, we have provided a feature checklist, ordered roughly in the order we think you should implement them in, together with the marks each feature is worth:

1. The initial objects are loaded and visible on screen (1 mark)
2. All units can be selected and deselected, and move correctly (1 mark)
3. Engineer mines the resources correctly (1 mark)
4. The text display is correct (1 mark)
5. Command centre can create units (1 mark)
6. Builder can create factories (1 mark)
7. Factory can create trucks (0.5 marks)
8. Truck can create command centres (0.5 marks)
9. Pylons activate when a unit is near them (0.5 marks)

10. Pylons allow engineers to mine faster (0.5 marks)

Customisation

Optional: we want to encourage creativity with this project. We have tried to outline every aspect of the game design here, but if you wish, you may customise any part of the game, including the graphics, types of units, buildings, resources, game mechanics, etc. You can also add entirely new features. However, to be eligible for full marks, you must implement all of the features in the above implementation checklist.

For those of you with far too much time on your hands, we will hold a competition for the best game extension or modification, judged by the lecturer and tutors. The winning three will be demonstrated at the final lecture, and there will be a prize for our favourite. Past modifications have included drastically increasing the scope of the game, implementing jokes and creative game design, adding polish to the game, and even introducing networked gameplay. If you would like to enter the competition, please email the head tutor, Eleanor McMurtry, at `mcmurtrye@unimelb.edu.au` with your username and a short description of the modifications you came up with. I can't wait to see what you've done!

The supplied package

You will be given a package, `oosd-project2-package.zip`, which contains all of the graphics and other files you need to build the game. You can use these in any way you like.

Submission and marking

Technical requirements

- The program must be written in the Java programming language.
- The program must not depend upon any libraries other than the Java standard library and the Slick library.
- The program must compile fully without errors.
- Every **public** method, attribute, and class must have Javadoc comments as explained in later lectures.

Submission will take place through the LMS. Please zip your project folder in its entirety, and submit this `.zip` file. **Do not submit a .rar, .7z, .tar.gz, or any other type of compressed folder.** Especially do not submit one of these files that has simply been renamed to have a `.zip` extension, as then we will be unable to open your file.[1]

Ensure all your code is contained in this folder.

[1] This may sound ridiculous, but it has happened several times in the past!

Good Coding Style

Good coding style is a contentious issue; however, we will be marking your code based on some of the following criteria:

- You should not go back and comment your code after the fact. You should be commenting as you go.
- You should be taking care to ensure proper use of visibility modifiers. Unless you have a very good reason for it, all instance variables should be private.
- Any constant should be defined as a static final variable. Don't use magic numbers!
- Make sure each class makes sense as a cohesive whole. A class should contain all of the data and methods relevant to its purpose.
- Make sure no single class is too large. Responsibilities should be carefully delegated to keep the code comprehensible.

Extensions and late submissions

If you need an extension for the project, please email Eleanor at `mcmurtrye@unimelb.edu.au` explaining your situation with some supporting documentation (medical certificate, academic adjustment plan, wedding invitation). If an extension has been granted, you may submit via the LMS as usual; please do however email Eleanor once you have submitted your project.

The project is due at **7:00pm sharp**. Any submissions received past this time (from 7:01pm onwards) will be considered late unless an extension has been granted. There will be no exceptions. There is a penalty of 1 mark for a late project, plus an additional 1 mark per 24 hours. If you submit late, you **must** email Eleanor with your student ID so that we can ensure your late submission is marked correctly.

Marks

Project 2 is worth **22** marks out of the total 100 for the subject.

- Project 2A is worth 6 marks.
  - Correct UML notation for methods (1 mark)
  - Correct UML notation for attributes (1 mark)
  - Correct UML notation for associations (1 mark)
  - Good breakdown into classes (1 mark)
  - Sensible delegation of methods and attributes (1 mark)
  - Appropriate use of inheritance and interfaces (1 mark)
- Project 2B is worth 16 marks.
  - Features implemented correctly: 8 marks (see Implementation checklist for details)

- Coding style, documentation, and good object-oriented principles: 8 marks
  - Delegation: breaking the code down into appropriate classes (2 marks)
  - Use of methods: avoiding repeated code and overly complex methods (1 mark)
  - Cohesion: classes are complete units that contain all their data (1 mark)
  - Coupling: interactions between classes are not overly complex (1 mark)
  - General code style: visibility modifiers, magic numbers, commenting etc. (1 mark)
  - Use of documentation (javadocs) (1 mark)