SENG201 – Software Engineering I
2018
Assignment: Heroes and Villains

For assignment administration queries:

Lecturers: Miguel Morales (course coordinator), Matthias Galster
miguel.morales@canterbury.ac.nz, matthias.galster@canterbury.ac.nz

For assignment help, hints and questions:

Tutors: Matthew Ruffell, Megan Chu
msr50@uclive.ac.nz, mch230@uclive.ac.nz

1 Introduction

1.1 Administration

This assignment is a part of the SENG201 assessment process and requires you to **design**, **implement**, **test** and **document** a software product. Note the following:

- The assignment is worth **25%** of the final marks.
- This assignment will **be done in pairs**:
  - **Submissions from individuals will not be accepted.**
  - **You must find your own partners.** Teaching staff will not allocate pairs.
  - Register you pair on Learn before two weeks after the release of the assignment.
  - Pairs are **not** allowed to collaborate with other pairs.
- Plagiarism:
  - You may **not** use material from other sources without appropriate attribution.
  - Plagiarism detection systems will be used on your report and code.
  - Plagiarism will be reported to the university proctor and results in failing the course.
- Due dates:
  - Submit your deliverables on Learn no later than **25 May 2018, 5:00pm**.
  - The drop-dead date is 1 June 2018, 5:00pm (**standard 15% penalty**).
- Demos:
  - Pairs will **demo their assignment in lecture week 12 (week of 28 May 2018)**.
  - A **30% penalty** of your marks for the assignment applies on failure to show up.
  - Demos will be scheduled during lab and lecture times.
  - Students must come as pair. Demo slots can be booked on Learn.
  - Each team member must be prepared to talk about any aspect of the assignment.

1.2 Outline

This assignment is to give you a brief idea of how a software engineer would go about creating a complete application from scratch that has a nice graphical interface. In this assignment, you will be creating a game where a team of super heroes explore, buy items, battle villains, and finally attempt to beat the main super villain. **The idea and scope are slightly open (just like in a real software engineering project in industry)** to allow some room for creativity. **This makes this software engineering assignment different to a programming assignment with clearly defined requirements and instructions**. Therefore, you first need to understand what the requirements are and define an initial design before attempting to code your solution. Ensure you implement the main requirements and assignment tasks as that is what will be graded.

1.3 Getting Help

This assignment 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, Matthew and Megan. You can email them or ask them questions in labs. Also, note that not all classes from the Java API required to implement this assignment may have been covered in detail in the lectures and labs. Therefore, this assignment will require you to study and understand the Java API or to consult other sources, such as websites, forums or text books (again, just like a software engineer in industry would do). 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. This 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 (GUI programming will be covered in the lectures and labs after the term break).

2.1 Setting Up the Game

When your game first starts it should:

1. Ask what the super hero team name will be. The team name needs to be between two and ten characters long.
2. Ask how many "cities" the heroes need to explore to find the super villain. The super villain is always in the final city. You can choose between three and six cities.
3. Ask how many heroes will be on the team. A team should be able to hold 1, 2 or 3 heroes.

2.2 Creating Team of Heroes

Now, you can add the above amount of heroes to the team. For each hero on the team the game should ask:

1. What the name of the hero is. No two heroes can share the same name.
2. What the type of hero you want to add. Each type of hero has different special abilities, and the team should be able to contain different types of heroes. There can be more than one hero of the same type in a team.

3. During the creation of a team and when selecting types of heroes, there should be some way of viewing attributes of each type of hero, so that heroes can be compared to see what their strengths and weaknesses are.

2.3 Playing the Main Game

Once the super hero team has been created, the main game can begin. The super hero team will find themselves in their home base, with the ability to go North, East, South or West. Note that if you go one direction, say North, if you then go South, it must take you back to where you came from.

The layout of the city is the shape of a "cross", with the hero home base in the middle, and North being the upward arm, East being the right arm and so on. You cannot reach another destination from another destination, you must return to the home base and then select another destination (imagine a city surrounded by mountains, rivers and the sea).

The destinations of the map will lead to one of 5 places (home base, shop, power up den, hospital, villains lair) The directions for destinations should be randomised per city.

1. The super hero home base. This is the default starting place of each city. Here you can view the status and attributes of each hero, and use a map if the team has one. Maps reveal the direction of all destinations (e.g., if a shop is North or South).
2. A shop, with an innkeeper that you can talk to which:
   (a) Enables the team to purchase power ups, maps and healing items.
   (b) Shows what objects the team currently owns, their amounts, and the amount of money the team has.
   (c) Shows the prices of each object.
   (d) Shows the attributes of the object (attributes will be better explained in the "Design and Architecture" section of this handout).
   (e) Allows to purchase multiple objects at a time without leaving the shop.
3. A power up den:
   (a) Heroes must visit a power up den to use power up items.
   (b) Here you can select a power up and a hero to apply it to.
   (c) Power ups are consumed and removed from the inventory.
4. A hospital:
   (a) Heroes must visit a hospital to use healing items, which recover a portion of a hereos health.
   (b) Here you can select a healing item and a hero to apply it to.
   (c) Healing items take time to apply, and a clock can be seen showing the time remaining until the hero is at full health.
   (d) The team can exit and re-enter the hospital at any time to check the remaining time on the healing item.
   (e) Healing items are consumed and removed from the inventory on initial application.
5. The villains lair:
   (a) Each city has a villains lair, and there is only one villain per city.
   (b) The heroes can decide if they want to enter the lair, or go back to where they came from.
   (c) If the heroes wishes to battle the villain, the battle screen opens. Defeating villains is the only way to progress to the next city.
   (d) The battle screen will display:

i. The villains name.
ii. A taunt, cried out by the villain.
iii. A mechanism to select a hero to battle the villain. Any hero can battle as long as they are alive.
iv. Controls for the game the villain has decided to play. If the villain wants to play paper scissors rock, then the appropriate options are available to select.
v. A villain is beaten once if they lose a game. If the team member loses a game, then the team member takes damage.
vi. If a hero loses all their health from taking damage, they are dead, and cannot battle further. Dead heroes are removed from the team, and a hero cannot recover from being dead. If all hereos die, the game is over, and the game exits.
vii. Villains can decide to change the game played each battle.
viii. Different heroes can be selected for each battle.
ix. Villains must be beaten three times to be defeated. The team is rewarded with additional money after defeating a villain.
x. Once a villain has been defeated, the team progresses to the next city. The status of the heroes (e.g., their itinerary, health) remains the same.

There will also be some random events which can happen at any time the team enters the hero home base.

1. The team might be robbed:
   (a) The team needs to be alerted to this fact.
   (b) The team looses a random object from their inventory.
2. The team might be gifted an item:
   (a) The team needs to be alerted to this fact.
   (b) A random object is added to the teams inventory.

2.4 Finishing the Game

After all cities have been cleared, and the final villain defeated, a screen will be shown that contains the team name and the time taken to complete the game.

2.5 Extra Credit Tasks

If you have completed the system as described above and have a good looking application, then you will most likely be in for a high mark. If you want some more things to do, then please discuss this with the tutors in the lab first, and make sure that you do not break any of the essential functionality described above. Any additional features must be accompanied by unit test and any other relevant artifacts. If it is not obvious how to use them then please include brief instructions or examples. Here are some ideas, and you do **not** need to implement them all to get full marks:

- A player may want to save the results of a game. This player or other players should be able to view the results later. Players may even want to keep a list of high scores.
- A player may want to save the current state of the game, and be able to load it up and continue the game at a later time.
- A player might want the ability to create their own heroes and villains, by using their own custom names and pictures.
- 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 implement different kinds of games to play that are more complex than the suggested ones. Be careful here and talk to tutors to make sure it is not too hard.

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 Teams

All teams have a name, and contain a list of heroes, a list of power ups the team has, a list of healing items the team has, a list of maps the team has and the amount of money the team has remaining. When a game begins, a team has no items, and all lists should be initialised empty. The team should start with an initial amount of money.

3.2 Heroes

Heroes have a name and a type. Six different heroes will be enough. The type determines what their special ability is, and also influences their health and recovery rates. Heroes have a health level, which defaults to 100%. Special abilities may include, and are not limited to: “getting cheaper prices at the store”, “having a higher chance at winning a game”, “being stronger and dealing more damage”, “always know the direction, aka being a map”, “being more defensive and taking less damage”, or “speeding up recovery rates”. When a hero’s health drops to 0, the hero dies and is removed from the team and special ability benefits are lost.

3.3 Villains

All villains have a name and a taunt phrase. Six different villains will be enough. Some villains may have a favourite game to play and may only stick to that game, and some villains may like to have different games each battle. Some villains may be stronger than other villains and do more damage to heroes on winning a battle. One villian is a "Super Villain" and will always be present in the final city.

3.4 Games

There should be three games that the heroes battle the villain with. “Paper, scissors, rock” is one game. “Guess the number between 1 - 10” is another, where the villain will say higher and lower, and the hero has two chances to get the number right. "Dice rolls" is another game, with the highest roll winning the game. The outcome of games can be influenced by the villian, special abilities of the selected hero or powerups applied to the selected hero.

3.5 Power Ups

There should be different types of power ups, three will be enough. All power ups have a price, and prices will depend on how good the power up is. Power ups can influence the outcomes of battles, like “being able to see the future” where a hero may have an increased chance of knowing what the villain will play, or “being more lucky” and skewing the results of a game towards the hero. You can choose to make some power ups apply to only one type of game, or for all games.

3.6 Healing Items

There should be different types of healing items, three will be enough. All healing items have a price, and prices will depend on how much health is restored. Health is restored on 25% increments, and each healing item has an “application time” which is how long it takes for all 25% increments to be applied to the hero. Healing items cannot be applied to dead heroes.

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 that option happen. The game environment keeps track of a team, a list of villains, and the current city layout. The game environment instantiates heroes, villains, power ups and healing items, and places the objects where they belong. All of the game logic will be placed in the game environment, such as the `heal()` method may call `hero.heal(healingItem);` or similar. This class will get large, please try and keep it modular.

4 Assignment Tasks

4.1 Initial Modeling in 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). Creating an initial UML class diagram will most likely require you to read through Section 2 and Section 3 several times to properly understand the scope of your product, class candidates, their attributes behavior (methods) and interactions and relationships between classes. At this point you can also start thinking about generalization, specialization, interfaces, etc.

4.2 Implementing a Command Line Application

Begin implementing classes, starting with Teams, Heroes and Villains, moving onto Power Ups, Healing Packs, Games and the Game Environment. Make the Game Environment a simple command line application which works in a simple runtime loop:

1. Print out a list of options the player may choose, with numbers next to the options.
2. Prompt the player to enter a number to select an option.
3. Read the number, parse it and select the correct option.
4. Call the method relating to the selected option and if necessary:

(a) Print out any information for that option, such as “Use a power up”.

(b) Read in a number for a selction, such as "select the second powerup in the list".

(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 on to implementing more features. Once you are feature complete and have a working game, you may implement 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.

4.3 Implementing a Graphical Application

You will be implementing a graphical application for your game using Swing, which will be explained in lectures and 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 on-screen, 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 on-screen 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 underestimate 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. We expect that Javadoc is written as you write methods and classes, rather than only at the end.

4.5 Writing Unit tests

You should design JUnit tests for your smaller, basic classes, such as Hero, Villain, Power Ups and Healing Items and Games 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. We expect that unit tests are written as you write methods and classes, rather than only at the end.

4.6 Preparing 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, patterns, the choice of collection types and how inheritance was used. You might want to reference your UML class diagram.

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

Include on the second page:

- A brief retrospective of what went well, what did not go well, and what improvements you could make for your next project.
- A statement of agreed contributions (in %) from both partners, including a brief paragraph per partner that describes key contributions.
- Your thoughts and feedback on the assignment.

4.7 How to Get Started

We suggest that you carefully read this document, the description of requirements, initial design considerations, and then work through the following steps.

- Find a partner and register on Learn.
- Agree with partner on a work schedule and availability. Create a plan with your partner for how to tackle this project. Discuss expectations (e.g., do you just want to pass the course or are you aiming for a high grade).
- Set up an Eclipse project.
- Create a high-level class diagram based on the information provided in Section 3.
- Implement basic model classes (e.g., Team, Hero). Test each class (using unit tests) to make sure they work properly.
- Enhance you initial class diagram based on the information provided in Section 2.
- Implement a command line application as outlined in Section 4.2.
- Implement a graphical application as outlined in Section 4.3.
- Write Javadoc and unit tests as you move along.

5 Deliverables

5.1 Submission

Please create a ZIP archive with the following:

- Your source code and unit tests (exported from your Eclipse project using File - Export - General - File System); make sure the exported project contains all relevant files.
- Javadoc (already compiled and ready to view).
- UML class diagrams (as a PDF or PNG; do not submit Umbrello or Dia files).
- Your report (as a PDF; 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 along the lines of: `java -jar usercode1_usercode2_hereos_villians.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. Before you submit your assignment, please check that your Eclipse project

- compiles,

- can be imported into Eclipse,
- can be run from inside Eclipse,
- can be run from outside Eclipse (command line JAR file).

5.2 Lab Demos

During the last week of term, you will be asked to demo your program 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 time slot.**You must ensure you are both available, as you must come as a pair.** Also, please note that **there are no demos after the drop dead date, but all demons happen in lecture week 12.**

6 Marking Scheme

Marking will take into account the professionalism of your approach, the quality of the work you have done, the degree of success you have achieved and the extent to which you have recorded your results and communicated them to the marker.

6.1 Overall Assignment [100 Marks]

6.1.1 Functionality [40 Marks]

We will be testing the extent to which your code meets the requirements using the graphical interface. 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 [20 Marks]

We will be examining technical quality (e.g., use of properties, methods, algorithms), your naming conventions, layout and architecture, and use of object oriented features. Quality of your in-line comments is also very important. You may wish to run `checkstyle` over your code before you hand it in.

6.1.3 Javadoc [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 code.

6.1.4 Unit Tests [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. 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. If you do not turn up to demo in your time slot, you will be penalised 30% of your marks for the assignment. Each demo will last around 15 minutes. During the demos, you may be asked to:

- Construct a new game, and create a hero team.
- Go to the map, showing directions.
- Find the store and buy items.
- Show that random events occur, such as obtaining random items or sub villain strikes.
- Show how to battle villains.
- Show that the game can be completed.
- Show that the application runs without errors, obvious bugs or crashes.
- Show that the application fulfills 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.

You and your partner should be able to answer the examiner's questions in the demos time slot, answers outside that time will not be considered.