Assignment 3

Boom, headshot![1]

Due: Friday 12th June, 23:55

In this assignment you will design and implement some new game functionality, and write a review and set of recommendations for changes to the game engine.

Project requirements

All the requirements stated in the Assignment 1 and 2 specifications still apply.

You must document your designs as you did for Assignment 1 and 2.

Going to town

You must implement a town level.

Place a vehicle somewhere on your existing map. The vehicle should provide the player with the option to move to a town map. The design of the town is up to you, as is the placement of any Humans, Zombies or extra items on that map.

Somewhere on the town map, place a sniper rifle and a shotgun.

Note that this must be *another map* – don't just make the current map bigger!

When you leave a map, any creatures in the old map should continue to move and act.

New weapons: shotgun and sniper rifle

You must implement two new kinds of ranged weapon: a shotgun and a sniper rifle.

Ranged weapons use ammunition. Place boxes of shotgun and rifle ammunition on the town and compound maps in the locations of your choice.

The **shotgun** has a short range, but sends a 90° cone of pellets out that can hit more than one target – that is, it does area effect damage. Rather than being fired at a target, the shotgun is fired in a direction. Its range is three squares. So, if the shotgun is fired north, it can hit anything in the three squares north of the shooter, northeast of the shooter, northwest of the shooter, or anything in between.

It has a 75% chance of hitting any Actor within its area of effect.

Tune your shotgun to determine a good level of damage for it to do – it shouldn't be so powerful that it knocks everything out in one hit.

The **sniper rifle** is a weapon that rewards patience. It is fired at a target, and does more damage and is more likely to hit if you spend time aiming it.

When the player fires your sniper rifle, they should be presented with a submenu allowing them to choose a target. Once they have selected a target, they have the option of shooting straight away or spending a round aiming. Spending time aiming improves the result as follows:

- No aim: 75% chance to hit, standard damage
- One round aiming: 90% chance to hit, double damage
- Two rounds aiming: 100% chance to hit, instakill

If the shooter takes any action other than aiming or firing during this process, their concentration is broken and they lose their target. They will also lose their target if they take any damage.

[1] `https://tvtropes.org/pmwiki/pmwiki.php/Main/BoomHeadshot`

Mambo Marie

Mambo Marie is a Voodoo priestess and the source of the local zombie epidemic. If she is not currently on the map, she has a 5% chance per turn of appearing. She starts at the edge of the map and wanders randomly. Every 10 turns, she will stop and spend a turn chanting. This will cause five new zombies to appear in random locations on the map. If she is not killed, she will vanish after 30 turns.

Mambo Marie will keep coming back until she is killed.

Ending the game

At the moment, there is no way to end the game.

You must add the following endings:

- A "quit game" option in the menu
- A "player loses" ending for when the player is killed, or all the other humans in the compound are killed
- A "player wins" ending for when the zombies and Mambo Marie have been wiped out and the compound is safe

Recommendations for change to the game engine

Over the course of your three assignments, you have had to become familiar with the game engine (located in the package `edu.monash.fit2099.engine`). You may have found aspects of the game engine design and implementation difficult to understand, or frustrating to use. On the other hand, maybe you enjoyed the experience of working with this engine.

Write a short document (e.g. two or three A4 pages of text) reviewing describing changes you would recommend for the game engine. For each change you recommend, you must explain:

- what problem you perceive
- the design change you propose to address the problem
- the advantages, and any disadvantages, of the proposed change

If you use UML diagrams, don't count the space they take up in the "two or three A4 pages of text".

If you can't think of anything bad about the engine, then write a justification of your positive opinion with reference to the design principles you've learned in this unit.

Note that any recommendations you make must be suitable for any application of the game engine. They must not be specific to the current scenario – we change that every year. (Over the last few years, we've used Star Wars, Harry Potter, Spider-Man, Jurassic Park, and Lord of the Rings.)

Design is important

One of the primary aims of this unit is for you to learn the fundamentals of object-oriented design. In order to get a high mark in this assignment, it will not be sufficient for your code to "work". It must also adhere to the design principles covered in lectures, and in the required readings on Moodle.

If you would like informal feedback on your design at any time, please consult your lab demonstrator in a lab or attend any consultation session.

Updating your design

You might find that some aspects of your existing design need to change to support the new functionality. You might also think of a better approach to some of the requirements you have already implemented — your understanding of the requirements and codebase will have improved as you have progressed. You might also spot places where your existing code (and thus design) can benefit from refactoring.

If you want to update your design, you may do so; if you decide to do this, be sure to update your design documents so that they match the code, and write a brief explanation of your changes and the reasons behind them. his will help your marker understand the thinking behind your code.

Coding and commenting standards

You must adhere to the Google Java coding standards that were posted on Moodle earlier in the semester.

Write javadoc comments for *at least* all public methods and attributes in your classes.

You will be marked on your adherence to the standards, javadoc, and general commenting guidelines that were posted to Moodle earlier in the semester.

Bonus marks

As stated in the Assignment 2 specification, there are bonus marks available for groups that come up with ideas for new features for the game, and then design and implement them.

A feature must be quite complex to qualify for a bonus mark. You must discuss your plans for bonus mark features with your tutor and gain approval before beginning work on them. Different tutors may have different standards so to ensure that bonuses are treated fairly, they will be determined centrally by an independent assessor. Bonus marks will be awarded after the submission of Assignment 3.

To apply for bonus marks, write a brief proposal and email it to your campus lecturer by Friday Week 10 (11th October) – `robyn.mcnamara@monash.edu` if you're at Clayton, `jasbirkaur.dhaliwal@monash.edu` if you're at Sunway. Include enough information about what you plan to do that we will be able to estimate how much effort it will take you, and how hard it will be to design well. This will be about a page for most teams.

A maximum of three bonus marks are available (these are whole marks for the unit). No one has to attempt bonus marks, and it is possible to get full marks for the unit without any bonus marks. Note that you will only get all of the marks you're eligible for if your code is well-designed, properly documented, and works properly.

Submission instructions

The due date for this assignment is *Friday 12th June at 23:55*, your local time. We will mark your Assignment 3 on the state of the "master" branch of your Monash GitLab repository at that time. If you have done any work in other branches, make sure you merge it into master before the due time.

Do not create a new copy of your work for Assignment 3. Continue working on the same files, in the same directory structure (you might like to add a tag to the your final Assignment 2 commit before starting on Assignment 3, so you can find that version easily).[2]

As we said above, you may update your design and implementation if you find that your Assignment 2 solution is not suitable for extension, or if you think of a better approach during Assignment 3.

You must update your Work Breakdown Agreement to include the work necessary for the Assignment 3 requirements, and any bonus mark work you attempt. We will take your Work Breakdown Agreement into account when marking if there seems to be a major discrepancy in the quality of different parts of the submission, or if the code is missing major sections. If you choose to reallocate tasks, make sure you keep your WBA up to date.

Marking Criteria

This assignment will be marked on:

- Functional completeness
- Design quality
  - adherence to design principles discussed in lectures

[2] `https://git-scm.com/book/en/v2/Git-Basics-Tagging`

  - UML notation consistency and appropriateness
  - consistency between design artefacts
  - clarity of writing.
  - level of detail (this should be sufficient but not overwhelming)
- Code quality
  - readability
  - adherence to Java coding standards
  - quality of comments
  - maintainability (application of the principles discussed in lectures)
- Correct use of GitLab
- Quality of recommendations for changes to the codebase
  - understanding of codebase displayed
  - usefulness of recommendations
  - understandability of recommendations
  - generality of recommendations (i.e. must not be specific to the current scenario)
  - practicality of recommendations
  - clarity of writing

Marks may also be **deducted** for:

- late submission
- inadequate individual contribution to the project
- academic integrity breaches

Note: Learning outcomes for this assignment

This assignment is intended to develop and assess the following unit learning outcomes:

1. Iteratively construct object-oriented designs for small to medium-size software systems, and describe these designs using standard software engineering notations including UML class diagrams (in conceptual and concrete forms), UML interaction diagrams and, if applicable, UML state diagrams;

3. Implement object-oriented designs in an object-oriented programming language such as Java, using object-oriented programming constructs such as classes, inheritance, abstract classes, and generics as appropriate;

5. Use software engineering tools including UML drawing tools, integrated development environments, and revision control to create, edit, and manage artifacts created during the development process.

To demonstrate your ability, you will be expected to:

- design and implement further extensions to the system
- use an integrated development environment to do so
- update your UML class diagrams and interaction diagrams as required, to ensure that they match your implementation
- comment on the design of the existing game engine
- use git to manage your team's files and documents

The marking scheme for this assignment reflects these expectations.