Assignment 1: Planning and Design

Braaaains

Due: Friday of Week 7 at 11:55pm, your local time

For the rest of the semester, you will be working in teams on a relatively large software project. You will design and implement new functionality to add to an existing system that we will provide to you.

A document explaining the FIT2099 Assignment Rules has been uploaded to the Assessments section in Moodle. Please read it and make sure you understand it before you begin the project – **you are expected to follow what it says, and will almost certainly lose marks if you do not**.

Background

You will be working on a text-based "rogue-like" game. Rogue-like games are named after the first such program: a fantasy game named `rogue`. They were very popular in the days before home computers were capable of elaborate graphics, and still have a small but dedicated following. If you would like more information about roguelike games, a good site is `http://www.roguebasin.com/`.

In this game, you are a survivor of the Zombie Apocalypse. The dead have risen from their graves and destroyed society as we know it. You and a small group of other people have retreated to a compound in the woods to defend it as best you can.

If you're not familiar with zombies in popular culture, don't worry. You don't have to know about any particular game, movie, or TV show to complete this assignment. If you like, you can get an overview of the sort of situation we're talking about at `https://tvtropes.org/pmwiki/pmwiki.php/Main/ZombieApocalypse`.

As it stands, the game functions, but is missing a lot of desired functionality. Over the course of this project, you will incrementally add new features to the game.

Assignment 1 and 2 Requirements

Here are the features we would like you to add to the game in the first round. In Assignment 1, you will design the code that you will use to implement these features; in Assignment 2, you will implement your design, refactoring it if required. Assignment 3 will add more features for you to design and implement.

You are not required to write any code for Assignment 1. You may do so if you like, but your marker will ignore it.

Zombie attacks

At the moment, the Zombies in the game are boring. They chase the nearest human and attack them. You need to make Zombies and their attacks more interesting.

- Zombies should be able to bite. Give the Zombie a bite attack as well, with a 50% probability of using this instead of their normal attack. The bite attack should have a lower chance of hitting than the punch attack, but do more damage – experiment with combinations of hit probability and damage that make the game fun and challenging. (You can experiment with the bite probability too, if you like.)
- A successful bite attack restores 5 health points to the Zombie
- If there is a weapon at the Zombie's location when its turn starts, the Zombie should pick it up. This means that the Zombie will use that weapon instead of its intrinsic punch attack (e.g. it might "slash" or "hit" depending on the weapon)

- Every turn, each Zombie should have a 10% chance of saying "Braaaaains" (or something similarly Zombie-like)

Beating up the Zombies

Zombies are well-known to fall to pieces quite readily due to decay, but keep on going regardless.

- Any attack on a Zombie that causes damage has a chance to knock at least one of its limbs off (I suggest 25% but feel free to experiment with the probabilities to make it more fun)
- On creation, a Zombie has two arms and two legs. It cannot lose more than these.
- If a Zombie loses one arm, its probability of punching (rather than biting) is halved and it has a 50% chance of dropping any weapon it is holding. If it loses both arms, it definitely drops any weapon it was holding.
- If it loses one leg, its movement speed is halved – that is, it can only move every second turn, although it can still perform other actions such as biting and punching (assuming it's still got at least one arm)
- If it loses both legs, it cannot move at all, although it can still bite and punch
- Lost limbs drop to the ground, either at the Zombie's location or at an adjacent location (whichever you feel is more fun and interesting)
- Cast-off Zombie limbs can be wielded as simple clubs – you decide on the amount of damage they can do

Crafting weapons

Now that you have zombies dropping bits of themselves everywhere, you can make the game more interesting by letting the player craft better weapons out of them.

- If the player is holding a Zombie arm, they can craft it into a Zombie club, which does significantly more damage.
- If the player is holding a Zombie leg, they can craft it into a Zombie mace, which does even more damage

Rising from the dead

As everybody knows, if you're killed by a Zombie, you become a Zombie yourself. After a Human is killed, and its corpse should rise from the dead as a Zombie 5-10 turns later.[1]

Farmers and food

You must create a new kind of Human: Farmers. A Farmer shares the same characteristics and abilities as a Human, but:

- when standing next to a patch of dirt, a Farmer has a 33% probability of sowing a crop on it
- left alone, a crop will ripen in 20 turns
- when standing on an unripe crop, a Farmer can fertilize it, decreasing the time left to ripen by 10 turns
- when standing on or next to a ripe crop, a Farmer (or the player) can harvest it for food. If a Farmer harvests the food, it is dropped to the ground. If the player harvests the food, it is placed in the player's inventory
- Food can be eaten by the player, or by damaged humans, to recover some health points

[1] You do not have to literally turn the corpse object into a Zombie object; objects in Java cannot change their types. Just make it look to the player as though the corpse is rising from the dead.

Bear in mind as you design that you **may not change any code the engine package**. You can do whatever you like to code in other packages, and you may call methods in the engine package if it's public (or if it's protected and you're calling it from a subclass). You may also inherit from classes in the engine package, if they are public.

A note about requirements

You will see that these requirements are imprecise. In many cases, this is because we want to give you the opportunity to experiment with different values – probabilities, damage levels, and so on – to find one that makes the gameplay more fun. This *will not be marked*. Changing these values should not affect the structure of your system, so feel free to play with them, especially if you are studying Game Development or are otherwise interested in game design.

If you're wondering why we have only told you what the game needs to do and not told you how to do it, that's because we are marking you on your ability to make good design choices – that is, to make designs that fit in with the design principles covered in lectures, in the compulsory readings, and in the Object-Oriented Fundamentals in Java notes.

Getting Started

The initial code base is available in a repository that has been created for you on `git.infotech.monash.edu`. It includes a folder containing design documents for the system. Download it and familiarize yourself with the code and its documentation.

Your repository is ready for you to clone and use, so you don't need to create a new local one and push it. Instead, follow the instructions on the last page of the Supplementary Notes on git that tell you how to check out a fresh copy of your repo. A video walkthrough will be posted under Assessments in Moodle when your repositories are created to show you how this can be done.

What to submit for Assignment 1

You are expected to produce the following three design artefacts in Assignment 1:

- Class diagrams
- Interaction diagrams
- A design rationale

You will be implementing your design later, but for Assignment 1 you are not required to write any code. Instead, you must produce preliminary *design documentation* to explain how you are going to add the specified new functionality to the system. The new functionality is specified in the **Assignment 1 and 2 Requirements** section, above.

We expect that you will produce *UML class diagrams* and *UML interaction diagrams* (i.e. sequence diagrams or communication diagrams) in accordance with the FIT2099 Assignment Rules. These Rules are available on Moodle.

Your class diagrams do not have to show the entire system. You only need to show the new parts, the parts that you expect to change, and enough information to let readers know where your new classes fit into the existing system. As it is likely that the precise names and signatures of methods will be refactored during development, you do not have to put them in this class diagram. However, the overall responsibilities of the class need to be documented *somewhere*, as you will need this information to be able to begin implementation. This can be done in a separate text document if you prefer.

To help us understand how your system will work, you must also write a *design rationale* to explain the choices you made. You must explain both how your proposed system will work and *why* you chose to do it that way.

The design (which includes *all* the diagrams and text that you create) must clearly show:

- what classes will exist in your extended system

- what the roles and responsibilities of any new or significantly modified classes are
- how these classes relate to and interact with the existing system
- how the (existing and new) classes will interact to deliver the required functionality

You are not required to create new documentation for components of the existing system that you *do not* plan to change.

Work Breakdown Agreement

We require you to create a simple Work Breakdown Agreement (WBA) to let us know how you plan to divide the work between members of your team. There is more information on WBAs in the FIT2099 Assignment Rules. These are available on Moodle in the Assessment section.

Submission instructions

You must put your design documents and work breakdown agreement (in one of the acceptable file formats listed earlier) in the `design-docs` folder of your Monash GitLab repository.

The due date for this assignment is **Friday of Week 7 (8th May) at 11:55pm, your local time**. We will mark a snapshot of your repository as it was at the due time. This means that you will need to notify your marker if you finished late and let them know which version they should check out.

Our git server has limited bandwidth and in previous years some students have experienced timeouts when pushing to the server. If this happens to you, don't panic: try again ten minutes later. Your submission time is the timestamp on the commit (i.e. to your local repo), not the timestamp on the push to server, so as long as you finish and commit on time, it's okay if it takes a while to get it uploaded.

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy,[2] late submissions will be penalized at 10% per working day late.

It is both team members' responsibility to ensure that the correct versions of the documentation are present in the repository by the due date and time. Once both teammates have agreed on a final Assignment 1 submission, do not make further commits to the master branch of the repository until the due date has passed unless your teammate agrees.

Marking Criteria

This assignment will be marked on:

Design completeness Does your design support the functionality we have specified?

Design quality Does your design take into account the programming and design principles we have discussed in lectures? Look in lecture slides, and check the Object-Oriented Fundamentals documents for principles like

Do Not Repeat Yourself

Practicality Can your design be implemented as it is described in your submission?

Following the brief Does your design comply with the constraints we have placed upon it — for instance, does your design leave the engine untouched, as required?

Documentation quality Does your design documentation clearly communicate your proposed changes to the system? This can include:

- UML notation consistency and appropriateness
- consistency between artefacts
- clarity of writing

[2] `http://www.monash.edu/exams/changes/special-consideration`

- level of detail (this should be sufficient but not overwhelming)

Explanation Can you adequately explain your design and the reasoning behind it, both in your rationale and in response to interview questions from your marker?

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;

2. Evaluate the quality of object-oriented software designs, both in terms of meeting user requirements and in terms of good design principles, using appropriate domain vocabulary to do so;

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:

- read and understand UML design documentation for an existing Java system
- propose a design for additional functionality for this system
- create UML class diagrams and interaction diagrams to document your design, using a UML drawing tool such as UMLet or Visual Paradigm – you are free to choose which one
- write a design rationale evaluating your proposed design and outlining some alternatives
- use git to manage your team's files and documents

The marking scheme for this assignment reflects these expectations.