Assignment 1: Planning and Design

Designosaurs

Due: Friday 23th April 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**.

Getting Started

The initial code base will available in a repository that will be created for you on git.infotech.monash.edu at the end of Week 6. In the meantime, design documents for the system are available for you on Moodle. Download these and familiarize yourself with the design.

You'll need a partner for your project. There's a group self-select module under Week 6 on Moodle that you can use – please choose a partner from your own lab class. We won't create groups with students from different lab classes as this makes it less clear who's supposed to mark you. If you don't have a partner by Friday evening, we'll assign one to you at random from the other unpaired students in your lab class.

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/. 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.

In this game, you are running a dinosaur park. Players must care for the dinosaurs and maintain an ecological balance so that they have enough to eat.

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.

Dirt, trees and bushes

Your dinosaurs are going to need a lot of food. Currently, they are in a very bare park with a limited number of trees. You should implement a system that will allow for bushes to grow from the dirt. These produce fruits, similar to the trees. Dinosaurs can move around on dirt and bush areas.

Here are the rules for growing plants:

- At the beginning of the game (and at the beginning of each turn), each square of dirt has a 1% chance to grow a bush.
- On any turn, any square of dirt that is next to at least two squares of bush has a larger (10%) chance to grow a bush
- In any square of dirt that is next to a tree there is no chance for a bush to grow.

- On any turn, any tree has a 50% chance to produce one ripe fruit and a bush 10%.
- On any turn, any ripe fruit in a tree has a small (say, 5%) chance to fall. Dropped fruit will sit on the same square as the tree. Fruit left on the ground will rot away in 15 turns.

The player can interact with plants in the following ways:

- The player can pick up fruit that is lying on the ground or from a bush. Fruit in the player's inventory does not rot.
- The player can try to pick fruit from a tree or bush in the same square. This has a chance of failing (say, 60%) with a message such as "You search the tree or bush for fruit, but you can't find any ripe ones."

You will need to experiment with the probabilities and timings in order to provide a balanced, challenging game, so bear that in mind as you design. This also applies for the requirements described below.

Hungry dinosaurs

You have been provided with a small herd of stegosaurs, but at the moment they are pretty boring. Your first task is to implement hunger and feeding.

A stegosaur should start out with a "food level" of 50 out of a maximum of 100. This should decrease by 1 on every turn. If the food level gets to zero, the stegosaur becomes unconscious and cannot move or act unless it is fed. After 20 turns of unconsciousness, the stegosaur dies.

A stegosaur that is hungry (i.e., if its food level is below 90) should move towards a food source and eat it. Stegosaurs are herbivores and can eat fruits only from bushes or a fruit laying on the ground under a tree. They can't eat from trees because their necks are short and their jaws are too weak to bite through branches.

When a stegosaur eats a fruit from a bush or a fruit laying on the ground under a tree, that fruit should disappear and should increase the stegosaur's food level by 10.

If the player is standing next to a stegosaur and holding fruit, they should be able to feed it to the stegosaur. Fruit given directly by the player should increase the stegosaur's food level by 20.

When a stegosaur becomes hungry, a suitable message should be displayed (e.g. Stegosaur at (19, 6) is getting hungry!)

Again, you might need to experiment to find the optimal food capacity for stegosaurs, and food values for different kinds of stegosaur feed.

Brachiosaur

You will also create a small heard of brachiosaur which are also herbivores (2 males and 2 females). In these game, these long neck dinosaurs can only eat fruit from trees. Indeed, if they step on a bush, there is a 50% chance they will kill the bush.

A brachiosaur should start out with a "food level" of 100 out of a maximum of 160. A brachiosaur is hungry if its food level is below 140. Their food level should also decrease by 1 on every turn. If the food level gets to zero, the brachiosaur becomes unconscious and cannot move or act unless it is fed. After 15 turns of unconsciousness, the brachiosaur dies.

A brachiosaur can eat as many fruits it finds in a tree in a single turn, but each fruit only increases their food level by 5 since they digest fruits poorly. They can be fed by the player. If this happens, each fruit increases the dinosaurs' food level by 20.

Breeding

If a stegosaur or a brachiosaur is sufficiently well-fed, i.e. has a food level over 50 for stegosaurus or 70 for brachiosaur, it has a chance to breed. A dinosaur that wants to breed will try to move towards another dinosaur of the opposite sex, if there is one nearby. Once in an adjacent square, the dinosaurs will mate (only with those of their same specie). Ten turns (for the stegosaur) and thirty turns later (for the brachiosaur), the female of the pair will lay an egg.

Eggs will hatch after a while (experiment to find a length of time that works well), into a baby dinosaur. Baby dinosaurs are hungry: its starting food level should only be 10. Baby dinosaurs cannot breed. The brachiosaur may have higher chances to become extinct.

After 30 turns for the stegosaur and 50 turns for the brachiosaur, the baby dinosaur should grow into an adult.

Eco points and purchasing

This game uses "eco points" for currency. Eco points are gained whenever any of the following happens:

- a ripe fruit is produced by a tree (1 point).
- a ripe fruit is harvested from a bush or a tree (10 points).
- fruit is fed to a dinosaur (10 points)
- a stegosaur hatches (100 points)
- a brachiosaur hatches (1000 points)
- an allosaur hatches (1000 points)

You must place a vending machine on the map. This vending should sell:

- fruit (30 eco points)
- vegetarian meal kit (100 points)
- carnivore meal kit (500 points)
- stegosaur eggs (200 points)
- brachiosaur eggs (500 points)
- allosaur eggs (1000 points)
- laser gun (500 points)

Vegetarian and carnivore meal kits are items that the player can feed to a vegetarian or carnivorous dinosaur. The meal kit will fill the target dinosaur up to it maximum food level and then disappear.

The laser gun is a weapon that does enough damage to kill a stegosaur in one or two hits. The player can use this to ensure that the stegosaurs do not grow too quickly for the available food supply, and to create food for allosaurs.

Allosaurs

Finally, you must implement allosaurs.

Like stegosaurs, allosaurs must be able to feed, breed, and grow. But unlike stegosaurs, Allosaurs are carnivores – they eat meat. If they go near a stegosaur, they will attack it reducing by 20 the food level of the stegosaurus (the allosaurus increases their food levels by 20, this means *health* and *food* levels are the same). If the stegosaurus doesn't die in the attack, the allosaurus cannot attack the same stegosaurus in the next 20 turns. If it dies, it can keep feeding from the corpse (see below). If they go near a dead stegosaur or brachiosaur, they will move toward it and eat it. Allosaurus cannot attack living brachiosaur.

Allosaurs do not appear on the map at the start of the game. Their eggs can be purchased from the vending machine for 1000 points each. Adult Allosaurs have a maximum food level of 100. They can eat dead Allosaurs or Stegosaurus for an increase of 50 in their food level. They fill their maximum food level (100) if they feed from a brachiosaur corpse. They can eat eggs for an increase of 10.

Similar to the other dinosaurs, allosaurus can also breed (if their food levels are above 50) and lay eggs (20 turns after mating). Eggs will hatch after 50 turns, into a baby allosaurus. Baby allosaurus are hungry: its starting food level should only be 20. Baby allosaurus cannot breed but they can attack stegosaurus (with only an increase of 10 food points while they are babies). After 50 turns, the baby allosaurus should grow into an adult.

Death
Make sure that when a dinosaur dies, the corpse remains in the game for a set period of time (e.g. 20 turns for an allosaurus or stegosaurus and 40 turns for a brachiosaur).

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 **Project Requirements** section.

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.

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 at the top of the first page. 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.

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy,[1] 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. If you want to continue to make changes to the repository for some reason, create another branch.

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
- 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?

[1] https://www.monash.edu/exams/changes/special-consideration

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 requirementsand 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.