Assignment 3 specifications

Assignment 3

Design O' Souls (Extended Edition)

Due: ~~Friday October 15th~~ (extended to Sunday October 17th), 23:55

Melbourne time (if enrolled at Clayton campus) or KL time (if enrolled in Malaysia campus)

In this assignment, you will design and implement some new game functionality.

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
- use git to manage your team's files and documents

The marking scheme for this assignment reflects these expectations.

Project requirements

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

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

In this assignment, you will design and implement three (3) fixed requirements and either one (for Group of 2) or two flexible requirements **(Group of 3)**. In total, you will develop four/five new requirements to complete the Design O' Souls game.
Similar to the previous assignment, some specific features that are tagged as **(OPTIONAL)** are provided to complete the playable game for portfolio purposes (it adds a bit of the FUN element). Unfortunately, it is no longer part of the bonus mark as we had allocated the bonus mark to the "creativity" aspect.
Please note that FLEXIBLE requirements **ARE NOT OPTIONAL**, so you **MUST** include them.

You may watch this video to help you understand the requirements below better.

Fixed Requirements

Requirement 1: New Map & Fog Door

Create a new map to allow the Player to progress the adventure. The size of the new map can be similar or smaller to the existing map, but the content must be different. You can design the content of the second map with more Skeletons and Cemeteries (for Undead). This second map can only be accessed through a **fog door (display character "=")**. The name of the first map will be "Profane Capital" and the second map will be "Anor Londo".

You must place this fog door at the most southern part of the first game map at any horizontal axis. The Player that steps on this fog door can interact with it. Interacting with it will bring causes the Player to appear at the most northern part of the second map with a similar horizontal point/position. Enemies cannot interact or pass through with this fog door.

Note: Here, a new map doesn't mean just making the size of your map bigger. You must create a new map instance. The engine will help you in rendering the map that the Player steps on. All activities in the other maps should run even though the Player is on a different map (again, the engine has done this for you if you do the implementation correctly).

Requirement 2: Updated Bonfire

Place a new Bonfire somewhere on the second map. You'll name it "Anor Londo's Bonfire". This new Bonfire is not activated yet. When the Player stands on top of it or next to it for the first time, there is only one available action: "Player lights the Bonfire". Activating it will print a message in the console, such as "Bonfire Lit". Additionally, when this new Bonfire is activated, the Firelink Shrine's Bonfire is ready to be used. Additionally, there is a new action from Bonfire that allows the Player to teleport to other bonfires.

For example, the following actions from Anor Londo's Bonfire will be available:

- **"Rest at [current]'s Bonfire"** // replace current with Bonfire's name
- **"Unkindled moves to [destination]"**
- **"Unkindled moves to [another-destination]"** // not exist yet. You can create more than 2 bonfires in this game.
- ...

The squared-bracket "destination" of Anor Londo's Bonfire will be "Firelink Shrine". In the scenario above, using this action moves the Player from Anor Londo's Bonfire to Firelink Shrine's Bonfire.

The Player will be bound to the last Bonfire that the Player interacts with (activating/lights a bonfire and rest action). This feature relates to the previous feature about dying in the game. For example, if the Player is killed in the first map and the last B
that Player interacted before was Anor Londo's Bonfire, the Player will spawn at Anor Londo's Bonfire instead of Fire Link Shri

that Player interacted before was Anor Londo's Bonfire, the Player will spawn at Anor Londo's Bonfire instead of Fire Link Shrine.

(OPTIONAL) On the other hand, teleportation is slightly different. The target bonfire will be considered as the last Bonfire that the player interacted with. For example, a Player teleports from Bonfire A to Bonfire B. The Player walks around and accidentally falls into the Valley. The Player will spawn at the second Bonfire (target bonfire) instead of the first Bonfire. (**Note** that even though Bonfire A is the last Bonfire that Player interacted with, the Player should respawn at Bonfire B since the Player teleported to B).

Requirement 3: Aldrich the Devourer (Lord of Cinder)

You will have another Lord of Cinder, a unique and difficult enemy in this game. We will name it **"Aldrich the Devourer"**. It starts with 350 hit points and equips a Darkmoon Longbow (see table below). By holding this weapon, Aldrich the Devourer can attack a Player that is within 3 squares away. It provides 5000 souls and drops a "cinder of the lord" (corpse) if defeated.

You must place Aldrich the Devourer somewhere in the second-map, and create a chamber/room with walls, dirt, floors, and a fog door (so it cannot escape). You can decide the chamber's size. Aldrich the Devourer will do nothing in the middle of its chamber unless it got engaged by the Player in a similar manner towards other enemies (when it is standing in adjacent squares). Then, it will follow and attack the Player whenever possible.

(OPTIONAL) When it has less than 50% health, it enters the **second phase (**Ember Form**)** where it will heal Aldrich by 20% of its remaining hit points and activate the "Fire Arrows" skill from its weapon. If it is activated, whenever it successfully lands an attack on the Player, it will burn the target's ground. See the table below for detailed information.

(OPTIONAL) Similar to Yhorm the Gaint: a reset will heal Aldrich the Devourer to maximum health, move back to its original position, and deactivate its weapon's skill.

Weapon Name	**Price**	**Damage**	**Success Hit Rate**
Darkmoon Longbow (Bow)	N/A	70	80%
	Ranged weapon (passive): It can attack an enemy that is far-away or in other words, beyond hand-to-hand distance/adjacent squares. The maximum range is 3-squares away. If there is a wall within its path, the attack will automatically 'miss' the target. **Critical hit (passive):** It has a **15%** success rate to deal double damage with a normal attack; *Displaying its effect is optional.* **Fire Arrows (Active) - (OPTIONAL):** This is a skill to allow the holder toggling the weapon status (activate/deactivate) of the fire arrow attack. When it is activated, every successful attack will burn the target's ground. The fire/burned ground last for the next three (3) turns and hurt anyone (except the holder) that stands on it by 25 hit points.		

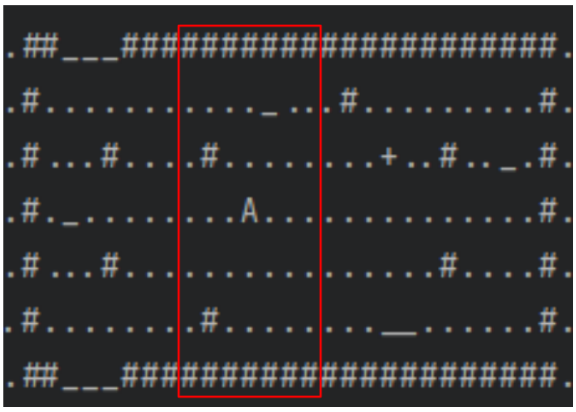

Image 1. Range attack radius (3-squares away). Any opponents that enter the red area can be attacked.

Flexible Requirements

We hope that you enjoy working on the assignment so far. We also hope that you are comfortable working with the engine code by now. Since this unit is about design, we give you the opportunity to be creative. In this part, you have two choices as follows:

1. **Structured Mode:** we decide the features that you must include in your work.
2. **Creative Mode:** you can go wild with your imagination. In other words, you can be creative in designing and implem

your own requirements. However, you must strictly follow rules and standards that we set so that we can grade your work fairly.

You must clearly declare in the README.md at the root of your repository whether you choose **Structured Mode** or **Creative Mode**.

A **Group of 3** must design and implement **two (2) flexible features**, while a team of two must design and implement only **one (1) flexible feature**.

Structured Mode

You must NOT follow these two features if you decide to do **Creative Mode** (it is after this section).

Requirement 4: Mimic / Chest

This is an ambiguous new enemy to give you a faster way to get more souls. You need to place it randomly on the game map before the game starts. It has **"?"** display character. It will do nothing until the Player interacts with it. The first available action will be "Open the Chest". When the Player tries to open the chest, it has two scenarios:

1. **50% chance becomes an enemy:** it becomes **Mimic** and replaces its display character from "?" to be "M". The Player can attack it. It has 100 hit points, doesn't equip any weapon, but it will attack the Player with a kick for 55 damage. When the Player manages to defeat it, Mimic will drop 1, 2, or 3 tokens of souls that are worth 100 souls each. It also gives Player additional 200 souls instantly. Reset will heal Mimic to maximum hit points, put Mimic back to its original position, and transforms it back to "?" chest.
2. **50% chance to drop the token of souls(s):** It either drops one, two, or three **tokens of souls** on the ground**.** One token of souls contains 100 souls.

NOTE: If you are a group of 2, we suggest adding a Vendor to make the game much enjoyable. Souls are useless unless you can trade them with rewards. At least and if you have time, design & implement the "increasing the Player's maximum number of hit points" feature.

Requirement 5: Trade Cinder of Lord

In the previous and current requirements, all Lords of Cinders will drop **Cinders of Lord** when they are defeated. The Player can pick it up and trade it to a Vendor to get the corresponding enemy's weapon. The Player can also use the ability of the weapon, whenever possible. The previous requirement that force only the Lord of Cinder can use that weapon becomes obsolete (e.g., "only Yhorm the Giant can hold Yhorm's Giant Machete" feature is not needed anymore).

NOTE: For the sake of simplicity, the Player that holds those Lord of Cinder's weapons can still get the damage from fire/burned ground.

Creative Mode

If you prefer to create your own features, we will compensate for your creativity and such extra work with a bonus mark. The total original group score becomes **105%.** In other words, you will get an extra 5% mark. For instance, you will get 82.95% if the original mark is 79%. However, the following rules apply:

- You must use the **MINIMUM REQUIREMENTS** table to meet our standards. The description must be specific with the appropriate design reasoning.
- You must attach this information (description and tables) in the **README.md document**, at the root of your repository. If you are a group of three, you will have two separate tables (2 complex features).
- You must ensure new features do not affect the existing system (old features).
- Your feature(s) must be new and original (you cannot use the major ideas from the Structured Mode features).
- You must explain why each feature illustrates the application of SOLID principles.

- For the sake of simplicity, you cannot combine features from Structured Mode with this Creative Mode. For example, you cannot pick one feature from Structured Mode and then create a new unique feature.

If your requirements violate any of the rules above, you will **NOT** get a bonus mark and we will mark you normally. Don't worry, no penalties apply.

> You must discuss your requirements' proposal with corresponding TAs to get approval **at least a week** before the deadline. Following that, you must present the proposal of features with the provided **MINIMUM REQUIREMENTS template table** in your discussion.

MINIMUM REQUIREMENTS table for ONE feature that we can consider complex (Copy & Paste this):

Requirement 4: *Write down the title of your requirement. (Replace 4 with 5 for the other requirement.)*
Description: *Write down the summary of your requirements here.*

Requirements	**Features (HOW) / Your Approach / Your Answer**
Must use at least two (2) classes from the engine package.	
Must use/re-use at least one (1) existing feature (either from A2 and/or A3 - fixed requirements)	
Must use existing or create new abstraction/interfaces (apart from the engine code).	
Must use existing or create new capabilities.	
Must explain why it adheres to the SOLID principles.	

Here are several general examples to give you some ideas. You can pick **one, or combine two or more** of the following ideas to be considered as one COMPLEX feature.

1. If you have played the real game, you can complete the rituals by combining Lord of Cinder's cinders/corpses (ideally there are 3 cinders, but any numbers should be okay). By doing so, the Player will reach the end game OR fight another final boss.
2. Create a new boss that has unique abilities and new weapons (see Dark Souls 3 bosses for reference). Place it somewhere in the second map or third map (create a new map).
3. Fog of war feature to increase the tension/difficulty of the game, where the enemies are invisible from the map until the Player is within a certain radius.
4. A system to generate random locations to place cemeteries, skeletons, and other enemies every time the game starts.
5. Create a new ally actor that can accompany your adventure.
6. Upgrade Estus Flask charges by consuming a new item or buy the service from the Vendor to upgrade it.
7. A patrol behaviour (moves at the set path, back and forth).
8. Blink to a location that is several steps away within one turn.
9. A new weapon to attack the enemy from far away (range attack) such as a bow or spear with a unique ability.
10. A magic system that can be cast from far away to deal heavy damage, enhance a current weapon, or becomes invisible for several turns (enemies will ignore you).
11. Additional buff effect (burned, poisoned, frostbite) where it can inflict or deal damage per turn or reduce the Player's maximum hit points.
12. More unique services and items from the Vendor such as increase strength, intelligence, stamina, agility, and more.
13. Remove all restrictions (i.e., "for the sake of simplicity" features) from the whole requirements to complete a fun and interesting game.

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 TS 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 you

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.

Submission instructions

The due date for this assignment is at the top of this document, 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. Additionally, please, make sure at least one team member uploads a copy of the repository to Moodle. This step is compulsory.

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 your final Assignment 2 commit before starting on Assignment 3, so you can find that version easily). *https://git-scm.com/book/en/v2/Git-Basics-Tagging*

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. 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

– 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

– clarity of writing

Marks may also be **deducted** for:

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

Last modified: Friday, 29 October 2021, 6:54 AM

◄ Assignment 2 protocol and rubric

Jump to...

Assignment 3 protocol and rubric ►