H15A_EAGLES Assignment II: DungeonManiaBlog

Task 1) Code Analysis and Refactoring 🛠️

a) From DRY to Design Patterns

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/2

> i. Look inside `src/main/java/dungeonmania/entities/enemies`. Where can you notice an instance of repeated code? Note down the particular offending lines/methods/fields.

Mercenary.java and ZombieToast.java both contain repeated code within their move functions.

- In `if (map.getPlayer().getEffectivePotion() instanceof InvincibilityPotion)`, the logic contained within the if statement is the same.
- Also random movement contains repeated code → the difference being, mercenaries create a new random generator each time random movement is needed, and zombies have a private gen. To merge these in to a single strategy, we must make the behave the same → ideally both create random gens when needed.

> ii. What Design Pattern could be used to improve the quality of the code and avoid repetition? Justify your choice by relating the scenario to the key characteristics of your chosen Design Pattern.

Using the Strategy Pattern:

- Can represent different types of movement strategies like invincibilityPotionMovement, invisibility movement, random movement, no movement, spider movement, default enemy movement.
- Create movement strategy interface
- public classes for each movement strategy.
- Each enemy will keep track of its current strategy, saved as a private variable.
- A function will be used to switch between strategies when needed, specific to each enemy.

Folder with movement strategies: `dungeonmania/entities/movement`

> iii. Using your chosen Design Pattern, refactor the code to remove the repetition.

To implement the strategy pattern, we isolated all the repeated code in the enemies' 'move' functions. Some parts were specific to each enemy and were kept inside the respective functions (such as logic for checking spiders on boulders). The code for calculating the next position is what we moved into strategies. Each strategy implements an interface which outlines a `getNewPositionMethod()`. After moving the code to these strategies, we created a private variable in the Enemy class to save what the current movement method is. At construction, this is set to null, and when move is called, we call `determineMovementStrategy`, which uses the previous logic to change the movement strategy to the required one.

b) Observer Pattern

> Identify **one place** where the **Observer Pattern** is present in the codebase, and outline how the implementation relates to the **key characteristics** of the Observer Pattern.

One instance of where the Observer Pattern is used is in the `Bomb` class.

- **Subject:** `dungeonmania/entities/collectables/Bomb.java`

- The `Bomb` class allows `Switch` entities to subscribe as observers to be notified when the bomb explodes
- It maintains a list of the observers via `List<Switch> subs`

- **Observer:** `dungeonmania/entities/Switch.java`
  - The `Switch` class represents the observer and maintains a list of which `Bomb` entities it is subscribed to and reacts to changes in their state (by exploding)
- **Attach/Notify Mechanism:**
  - In the `onPutDown(GameMap map, Position p)` of the `Bomb Class`
    - When a `Bomb` entity is cardinally adjacent to a `Switch`, the `Switch` entity subscribes to the `Bomb` entity by adding it to its list, `List<Bomb> bombs` through the `Switch Class`
  - When a `Boulder` entity overlaps with a switch, the `Switch` entity is activated and triggers the `notify` method in the `Switch` class for all its subscribed `Bomb` entities. It will iterate through each `Bomb` entity and call the `explode()` method.

c) Inheritance Design

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/5

> i. Name the code smell present in the above code. Identify all subclasses of `Entity` which have similar code smells that point towards the same root cause.

Code smell

The Entity class currently uses abstract methods with `onOverlap()`, `onMovedAway()`, and `onDestroy()`. There are multiple subclasses of Entity which are forced to implement these methods even though they are redundant. This violates the DRY Principle and produces the 'Empty Override/Method' smell.

Subclasses containing the smell:

In `src/main/java/dungeonmania/entities:`

- Boulder.java
- Door.java
- Exit.java
- Player.java
- Portal.java
- Switch.java
- Wall.java

In `src/main/java/dungeonmania/entities/buildables:`

- Buildable.java

In `src/main/java/dungeonmania/entities/collectables:`

- Arrow.java
- Bomb.java
- Key.java
- Sword.java
- Treasure.java
- Wood.java

In `src/main/java/dungeonmania/entities/collectables/potions:`

- Potion.java

In `src/main/java/dungeonmania/entities/enemies:`

- ZombieToastSpawner.java
- Enemy.java

> ii. Redesign the inheritance structure to solve the problem, in doing so remove the smells.

To fix this, we created interfaces for `onOverlap()`, `onMovedAway()` and `onDestroy()`

Each entity would then implement the required interface, removing the need to implement all three original methods for each subclass.

d) More Code Smells

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/8

> i. What design smell is present in the above description?

'Shotgun Surgery' smell

- A single change to system required the team to make further changes in the codebase. This lead to more code smells like:
- **Violation of the DRY Principle:**
  - In onOverlap() the logic contained within this method is prevalent in every single collectable entity:

```java
@Override
   public void onOverlap(GameMap map, Entity entity) {
       if (entity instanceof Player) {
           if (!((Player) entity).pickUp(this))
               return;
           map.destroyEntity(this);
       }
   }
```

> ii. Refactor the code to resolve the smell and underlying problem causing it.

- We created a `CollectableEntity Class` which extends `Entity` and implements the interfaces `CanOverlap` and `CanMoveOnto`
- Each collectable entity then became a subclass of `CollectableEntity`

e) Open-Closed Goals

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/3

> i. Do you think the design is of good quality here? Do you think it complies with the open-closed principle? Do you think the design should be changed?

It's not the best. The heavy use of switch cases and hard coded goal values ("exit", "boulders", "treasure", "AND", "OR") are violations of the open-closed principle. This code becomes hard to expand as it requires modification of every function that makes use of these values. It effects both the Goal.java and GoalFactory.java. We do think the design should be changed.

> ii. If you think the design is sufficient as it is, justify your decision. If you think the answer is no, pick a suitable Design Pattern that would improve the quality of the code and refactor the code accordingly.

These code smells point to the solution being a Composite pattern which will be used in combination with the goal factory to build composite goals. We will need separate classes for AND goals, OR goals, and Leaf Node goals. The factory will build these composite goals instead of single class goals. Goal.java will become an abstract class so that these composite goals can be used in the code base without further change.

f) Open Refactoring

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/4

We chose to refactor the poorly implemented state pattern for player potion effects. The State classes previously did nothing besides describe the current state of the player. Work such as applying buffs, has been removed from Player.java and forwarded to respective state class. State Transitioning is also handled by the abstract class PlayerState instead of an internal method in player.

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/6

Simple refactoring to remove the use of deprecated entity.translate functions throughout the code base. Errors were only present in two files: GameMap.java, and Bomb.java.

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/9

Buildable entities, previously had no connection to their recipe, which was hard coded into inventory methods. These functions counted the players inventory items and compared them to magic numbers that associated with the recipe of the desired entity. This is an issue because these magic numbers were repeated throughout the code, making errors regarding inconsistencies more likely. It also limited expansion, as adding a new buildable entity would require adding it to each individual method.
Our solution was to create a class for recipes that stored an entity recipe as a map.

The map links the class of the items needed and the amount needed to build the entity.

Each subclass of buildable was given a static variable that stores a List of its recipes. The Recipes are constructed by an internal function that creates the hash map. The method used to implement this has some repeated code between classes. This is due to the need of static methods that cant be moved to the superclass. Some of the repeated code could be removed with more refactoring, but to be fair, the system is already over-engineered enough.

Inside the Inventory class, buildable entities are made by comparing the inventory items against the recipe. If the player has enough materials, they are granted the item. There is also now a generic method for creating a buildable object from a string → it is located inside the entity factory.

Whenever a new buildable entity is created it must be added the the factory function, and the inventory.getBuildable() function. Other than this, creating the class and its appropriate recipes is enough to integrate it into the code base → which is an improvement over before.

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/10

BattleItems: Bow, Shield, Sword has repeated code in `use()` method. This was resolved by moving this code to a default method in the BattleItem interface. Implementing `setDurability()` in each of the subclasses allowed for this method to work for potion class as well.

suggestions;

- idk cud possibly jus like collectable entity make battleitemEntity? future us probrem!!
- Battle Facade has a very big method that definitely breaks single responsibility principle.

Add all other changes you made in the same format

Task 2) Evolution of Requirements 👽

a) Microevolution - Enemy Goal

In this section you will need to make a series of small-scale changes and additions to the code based on the following new requirements.

The following new goal has been introduced:

- Destroying a certain number of enemies (or more) AND all spawners;

Other goal rules, including rules of conjunction/disjunction and exits must be completed last, still apply.

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/11

Notes:

We need to make an enemyGoal.java which requires two conditions for this goal to be achieved:

1. Player has killed enough enemies
   - keep in mind of bombs that can be placed down and kill non-allied entities
   - add `private int slainEnemiesCount = 0;` to Player.java
   - add public method: `public int getSlainEnemiesCount()` as well
     - How do we know when an enemy gets killed?
     - In the `battle()` of `Game.java` maybe we can add method to Player.java, slainEnemy() to increment the count to `int slainEnemiesCount`

```java
if (enemy.getBattleStatistics().getHealth() <= 0) {
    map.destroyEntity(enemy);
    player.slainEnemy();
}
```

2. All ZombieToastSpawners have been destroyed

Problems: code currently does not destroy a spawner (can use this in task 3)

- Ed Discussion *"How then are we supposed to check for 'destroyed spawners' if there is no indicated that they've been destroyed (apart from the fact that it was successfully interacted with)."*
- Ed Discussion *we are required to implement destroying spawners upon interaction (i.e. walking into a spawner)? And this would also require modifying the regression tests involving interacting with spawners.*
- We'll need to:
  - modify test 10-7 in `ZombieTest.java`
  - also write new tests?
  - keep track of destroying spawners:
    - The code uses a Hashmap: `Map<Position, GraphNode> nodes = new HashMap<>()`
    - we can use GameMap's `destroyEntity()`
  - Keep track of destroyed enemies
    - we can do this by giving the player a battlesWon count.
  - Create an EnemyGoal class and potentially associated json files
    - add this goal to the goal factory
    - Goal success can count spawners and check battlesWon
-

Assumptions

Assume that enemies destroyed by a bomb count toward the players kill count.

Design

<Design>

- What fields/methods you will need to add/change in a class
- What new classes/packages you will need to create

Test list

- **Testing Scenarios:**
  - single enemy destroy goal → no spawners **C c_task2Test_oneEnemy.json**
  - single enemy destroy goal → one spawner **E c_task2Test_oneEnemyAndSpawner.json**
    - Tag "`Test achieving an enemy goal with single zombie and spawner`"
  - multiple enemy destroy goal → multiple spawners **C c_task2Test_multipleEnemyAndSpawner.json**
    - check goal not achieved before all criteria met as well
    - Tag: `Test achieving an ememy with multiple enemies and multiple spawners`
  - explosion destroys spawner and zombie **E c_task2Test_bombDestroyEnemy.json**
    - Tag: `Test achieving enemy goal with bomb`
  - add enemy_goal to complex goals test as well. **C c_task2Test_enemyComplexGoal.json**
    - Tag: `Test achieving enemy goal within a complex goal`
  - killing an unallied mercenary should increase but unallied should not
- Add this to config file `"enemy_goal": 1,`
- And this to dungeon `"goal-condition": {"goal": "enemies"}`

Other notes

<Any other notes>

---

d) More Buildables - Sunstone, Sceptre and Midnight Armour

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/15

Assumptions

<Any assumptions made>

Design

Composite pattern is needed for buildables. The current design uses a list of maps to store all the combinations of possible recipes for that entity. This is a good implementation if an entity has multiple unrelated recipes. I.e. (1 Wood + 1 Arrow + 3 Treasure) OR (2 Keys). This, however, is not how the task is designed. While this implementation does work for the required task, a composite pattern will be more concise.

- interface for durability
  - will need to probably rework
- Sun stone class collectable
- Sceptre and midnight armour class
  - extend buildable
  - implement their recipes using the composite pattern.
  - Sceptre recipe will need an extra check for two sunstones that is furthest on the right so its executed last. In this recipe, a sunstone can only replace a key / treasure if there is two of them.

- Add condition to sceptre canBuild() method. Override superclass → implement condition → call super method again.

Changes after review

<Design review/ Changes made>

Test list

- **Sunstone**
  - Test works as key and does not get destroyed upon use.
  - Test replaceable as key in buildable recipe and is not destroyed. (already did in sceptre)
  - Test replaceable as treasure in buildable recipe and is not destroyed. (already did in sceptre)
  - *"Since it is classed as treasure it counts towards the treasure goal".*
    - the treasureCount wont decrease after item is built bc treasure goal is cumulative total of treasure thats been collected
- **Sceptre**
  - Test crafting recipe → 1 wood, 1 treasure, 1 sunstone
  - Test crafting recipe → 1 wood, 1 key, 1 sunstone
  - Test crafting recipe → 2 arrows, 1 treasure, 1 sunstone
  - Test crafting recipe → 2 arrows, 1 key, 1 sunstone
  - Test crafting recipe → 2 arrows, 2 sunstones
  - Test crafting recipe → 1 wood, 2 sunstones
  - **This will not b tested** Ed Discussion Test crafting recipe → Does not work with one sunstone, 1 wood

  - Test sceptre mind controls
- **Midnight Armour**

  -

- Can be crafted with (1 sword + 1 sun stone) if there are no zombies currently in the dungeon. Midnight armour provides extra attack damage as well as protection, and it lasts forever.
- Build
- battle enemy → check amount of damage taken/output → Craft armour → check no zombies in dungeon → battle enemy, check more damage has been outputted and less damage
  - In same test → does not work with zombies in dungeon
  - In same test → check player has damage and resistance buff

Other notes

- This sucked.

---

d) Dungeon Generation

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/19

Assumptions

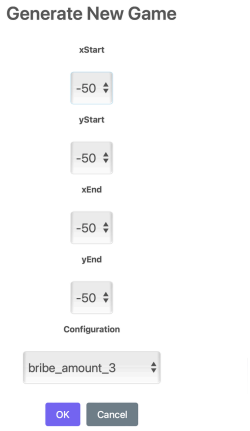
<Any assumptions made>

- they won't test where dungeon start is same as end

Design

<u>todo</u>

- forums related:
  - *"Inside* `utils/Api.ts` *you will need to add a new asynchronous Promise which calls your dungeon generation endpoint."*
    - Ed Discussion In `utils/Api.ts` include a new asynchronous post request with all the parameteres the dungeon generator needs
    - Ed Discussion how to add drop-down list in frontend
  - possible issues we may also have:
    - Ed Discussion *Route has not been mapped in spark*
- frontend:
  - in `client/src/scenes/MenuScene.ts`: *you will need to add a button which allows the user to generate a new dungeon. When the user clicks the button a popup along the lines of the following should allow for the generation of a new dungeon:*



  -
- backend:

Changes after review

<Design review/ Changes made>

Test list

- Using DFS/BFS to test
  - Ed Discussion *"DFS/BFS would be useful to test if there is a solution to your maze. During this test, you can also check that there are no cycles (so that it can't be empty)."*
  - Ed Discussion *"start from start from TestUtils.getPlayerPos and use TestUtils.getCardinallyAdjacentPositions. Then you could use TestUtils.entityAtPos to see whats at each position the BFS/DFS is on. (I didn't do this task but hopefully this guess helps)"*

Other notes

<Any other notes>

If you did more tasks add them here too

Task 3) Investigation Task !?

Ed Discussion

"The correct answer is finding the places where the MVP fails to meet the spec, fixing those areas of code/tests, and documenting your change.

We award marks based on number of issues satisfactorily identified and resolved. Bonus marks would require substantial investigation into the the provided codebase."

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H15A_EAGLES/assignment-ii/-/merge_requests/11

While completing Task2 a) we noticed that the ZombieToastSpawner class didn't destroy itself upon interaction with a player.

changes made:

- within `ZombieToastSpawner class` 's `interact()` method, we also added the `destroyEntity()` method to destroy the spawner after it's interacted by a player with a weapon.
- fixed a minor bug in the `ZombieTest` where it expected the spawner to still be existing on the map after it was destroyed.

Merge request 2

- The spec says that a player is only meant to possess one key at a time. However the given implementation allows you to carry two keys at once.
- We fixed this by editing Player.java in the pickup() method to not pick up when the inventory

Add all other changes you made in the same format