Assignment 3 Design Rationale (REQ 1, 2)

By Ruilin Lu

This assignment I worked on req 1 and 2 and some req in assignment 2. Req 1:Cliff was implemented by executing deathaction to actor on top of the cliff ground. Req1: Golden Fog Door was implemented by letting the class to have a location that it would teleport actor who stands on top of it to, and only player can enter and use it. Each door is created and added to the map in application.
Req2 was easy to implement. Create two new environments inherited from the ground and set spawnchance and enemy that they spawn. And new enemy inherited to a new abstract class(enemy type) then add their attribute.

Changes Made from Last Assignment

AttackAction class is separated into AttackAction class and AttackAOEAction class so they only focus on their part, enhancing SRP and improving modularity and maintainability. The same goes for AttackBehaviour.
Added reset to the code so when player die reset would trigger and the game would reset based on the req. All that require reset is implemented with interface reset and at construction it would store the Resettable instance to the resetManager class. resetManager would run all reset of Resettable instance when run. Player reset() would move the player and heal full and drop runes to the location before it dies as required. Enemy would all despawn.

Downcasting and instanceOf are all fixed using capability. Ability enum is used as capability to determine whether it is aoe attack or not. And respawnable from status enum is used to dtermine whether an actor can become a pile of bone or not in deathaction.

Behaviours determine whether an action should be performed, and action determines how it is supposed to run. Enforcing SRP.

Evaluation:
All above is designed with good OOD and didn't break SRP. Each class has one respawnsibilty. ResetManager runs reset to stored resetable and reset reset it's class. Ability enum only store capability that affect attack. Status only store actors current status. Behaviours determine whether an action should be performed, and action determines how it is supposed to run.

For OCP: new class can implement reset and store in reset manager as resettable with out modifying reset manager or other class, new enemies can use behaviour to determine their action without editing the behaviour or the action. New behaviour and action can be add without modifying other class.

For LSP: reset can be implemented to actor class in the engine or subclass and subclass of subclass of enemies without breaking the application.

For ISP: Resettable interface seperate the responsibility for the class that doesn’t need resetting(grounds). And resetManager will make will reset class need reset. Same with behaviour.

For DIP: Resettable is implemented to Enemy class which means its subclass has to rely on the implementation of reset in Enemy class. Making it easier to change subclass with out affecting parent class. Improve flexibility and modularity of the code.

- Pros:
  - Abstraction - reduce dependency and less repetition, more flexible. Improve Code Reusability
  - Modularity - easier to understand and maintain
  - Efficiency - ISP making code more efficient by only letting required class participate.
  - Responsibility Segregation
  - Consistency
  - Polymorphism

- Cons:
  - Complexity
  - Code Duplication in AttackAOEAction and AttackAction, but they separate responsibilities

Cliff Class

Cliff class inherited from the ground class. In tick function it runs deathaction and display the result.

SRP: class only deals with cliff related behaviours

OCP: Can add new properties to cliff without modifying the ground class

LSP: Cliff should be usable where ground is expected not break-in the program

ISP: cliff rely on capable and printable interface in ground

DIP: Cliff rely on method in ground class

Pros:

allowing for extensibility and substitution

Golden Fog Door Class

This class inherited from the ground class. New class variable are created to store the the location of where the door lead to. allowableActions is overridden to only show actor.hasCapability(Status.HOSTILE_TO_ENEMY) to go to the stored location. And this location is set in creation which is in application.

SRP: the class solely focus on golden fog door properties and behaviour

OCP: Can add new properties to cliff without modifying the ground class

LSP: should be usable where the ground is expected not break-in the program

ISP: rely on capable and printable interface in ground

DIP: rely on method in ground class

Pros:

allowing for extensibility and substitution

Allows customization to this class such as how actor interacts with the class

Cage and **Barrack class**

A ground that spawns enemies by a spawn chance. Override the tick method to spawn.

SRP: cage class only focus on cage properties, same goes for barrack class

OCP: Can add new properties to cliff without modifying the ground class

LSP: should be usable where the ground is expected not break-in the program

ISP: rely on capable and printable interface in ground

DIP:rely on method in ground class

Pros:

allowing for extensibility and substitution due to following OCP and LSP principles (random number generator to determine spawn)

Cons:

Design is coupled with specific enemy child child class, changing the child class would affect the game.

Stormveil class

Abstract class inherited from enemy. Used in attackaction to determine whether two enemy is the same type.

SRP: solely focus on storm-veil type enemy class

OCP: can be extended and add new functionality without modifying the it self and it can add new functionality without editing its parent class.

LSP: any method expected from enemy or actor can substitute stormveil in without affecting the correctness of the code.

DIP:rely on method in it’s parent class

Pros:

allowing for extensibility and substitution due to following OCP and LSP principles

Dog and GodrickSoldier class

Inherited from the Stormveil class and add their own attribute.

SRP: solely focus on Dog or GodrickSoldier class type enemy class

OCP: can be extended and add new functionality without modifying the it self and it can add new functionality without editing its parent class.

LSP:any method expected from enemy or actor can substitute them in without affecting the correctness of the code.

DIP: rely on method in it’s parent class

Pros:

allowing for extensibility and substitution due to following OCP and LSP principles

Cons:

As weapon is added to GodrickSoldier class in the code could mean for future enemy with weapon, their will be a lot of repetition on adding weapon for each enemy class.