Design Rationale REQ3: The Wandering Undead

Introduction

In this section, I will provide a detailed explanation of the design and implementation of the "Wandering Undead" Actor, also why "Wandering Behavior" and "Floor" was refactored.

Design Implementation

1. Creating Wander Undead class

   This class is inherited by Actor class, and is given instances of wander behavior and attack behavior. The methods of the actor class is overwritten to tailor to its needed.

   This is allowed to attack the actors that has a status of "HOSTILE_TO_ENEMY". It uses its intrinsic weapon to attack, with a custom damage, verb and a hit rate tailored to it.

   The use of consistent naming and comments makes it easier for this class to be maintained as well as this carries the single responsibility principle.

2. refactoring the Wander behavior class

   This class is inherited by behavior class. The actors exits are used to determine where this actor moves next. These destinations are added to the actions, where a random action is guess next.

   This class is purely responsible on how the wandering undead moves to block the player.

3. Creating the attack behavior class

   This class is inherited by the behavior class. This determines how the actor attacks the player.
   This overwrites the get action method to attack the player if its nearby. Whether the player is near or not the next method "isplayernearby" is used, where it takes the exits of the actor and checks whether the player is present. When attacking the direction of attack is determine by the method "findplayerinmap".

   This class is purely responsible on how the wandering undead attacks the player.

4. Refactoring the floor class

   This class was needed to be refactored as the actors except the player should be blocked in stepping onto, hence the player is given a status of "CAN_ENTER_FLOOR" and the floor is made to accept actor with this status only.

5. Refactored the void class

   This class was refactored to not kill the actor wanderingundead when stepped on a void.

Summary

In summary, the detailed design rationale encompasses various object-oriented principles, including encapsulation, the SRP, the Open-Closed Principle, the Interface Segregation Principle, and modular and flexible design practices. The implementation

details considered in each design decision ensure code maintainability, code reusability, and overall system flexibility.