## Changes in Design

***This documentation only shows the major changes that we made and the reasons behind. For the complete updated documentations , see "Documentations".*

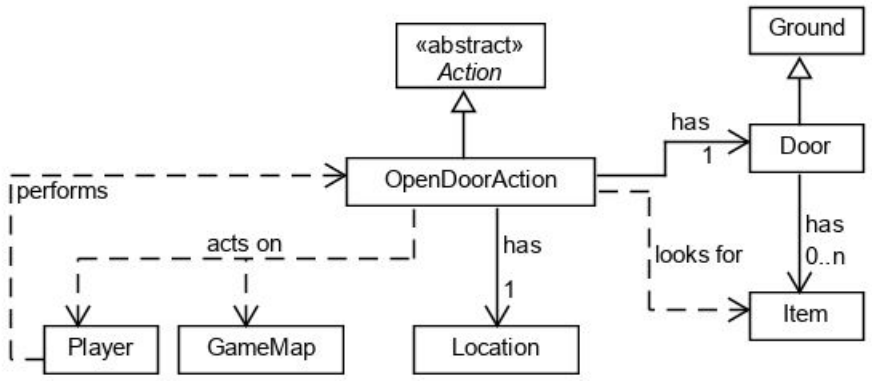


Figure 1: Updated class diagram for requirement – Doors and Keys

When an actor opens the door, we planned to move the actor to the opposite side of the door in our previous design. However, we then realise that if the player is on one side of the door, there is a high chance that Grunt or Goon will be at the other side of the door since they have FollowBehavior and they will follow the player as closely as possible. If that is the case, both actors now cannot pass through the door even if they have the key because both sides of the door are now occupied. Therefore, now we want the actor to move to the location of the door instead.

Besides, in the previous class diagram of the newly added classes for the "Doors and Keys" requirement, there is no "has a" relationship between OpenDoorAction and Location. Now, we modify OpenDoorAction to have a field of type Location. This is because we realized we need the reference to the location of the door to move actor to there but there is no method in Ground class that can return the location of the Ground. Thus, we change our

design so that the constructor of OpenDoorAction will take both the door and location of Door as parameter then initialize the fields.

Another changes in Door class is that, we mentioned that our keys are all the same, but now we decided to create unique keys to make the game more interesting. This means that each key can only open one particular door, but there may be duplicated keys. So, we give Door a field of type List<Item> to store the list of duplicate keys that can open the door.

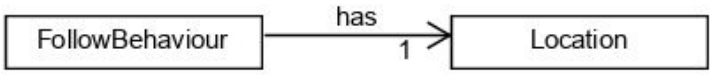


Figure 2: Added association for FollowBehaviour

The following change we made is adding a “has a” relationship between FollowBehaviour and Location. This is because we found out that when the player is stunned by Ninja, since the player is removed from the GameMap and it is replaced by StunnedPlayer, the enemies that have FollowBehaviour(Grunt and Goon) cannot detect the location of the player anymore. Therefore, they are not able to move towards the player when the player is stunned by Ninja. To solve this problem, we decided to create an instance variable “previousTargetLocation” of type Location. Then in FollowBehaviour’s getAction() method, this variable will be set to the location of the player detected by the enemies respectively at each round. By doing this, when the player is stunned and it is removed from the GameMap, previousTargetLocation will now be the location where the player last appeared before being stunned by Ninja, that way, even when the player is stunned and it disappears from the GameMap, Grunt and Goon will move towards previousTargetLocation which is the location where the player last appeared before it is being stunned by Ninja.

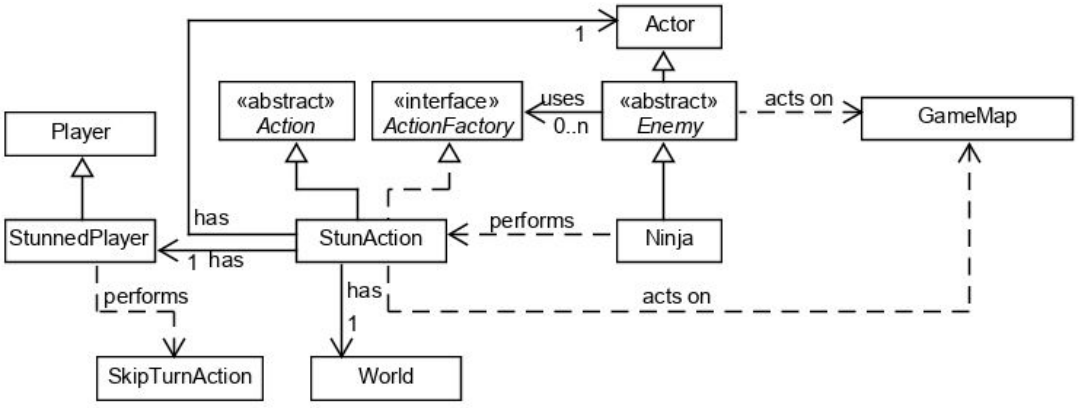


Figure 3: Updated class diagram for requirement – New Types of Enemy: Ninja

Another change is to make StunnedPlayer to inherit Player instead of Actor. This is because we want StunnedPlayer to inherit Player's showMenu() so that when it is the player's turn, although there is only one option which is to skip turn, a menu can be printed out to be selected by user. This is because we are afraid that user may be confused if the program skips their turn without letting the user to select the SkipTurnAction himself.

Next, in the previous interaction diagram of Player, Ninja and StunAction, we mentioned that, when stunStatus == 0 and player is stunned successfully, we will remove Player from map and add StunnedPlayer to map. We realized that this will cause the game to stop running because player will not exist in actorLocations anymore and World's stillRunning() method will return false. Therefore, we need to make a few changes in StunAction class:

- When player is stunned, instead of adding StunnedPlayer to map, we add it to world using World's addPlayer(). By doing this, in World's stillRunning() method, it will check if StunnedPlayer is still on the map instead of Player.

- To do so, we need to have the reference of the World in Application, so we add a field named world of type World which will be initialized during construction of StunAction to the World of the game created in Application. This means that the constructor of StunAction will also take in a parameter of type World.
- Similarly, when the player is freed from stun, we need to add Player back to world using World's addPlayer instead of adding it to map.
- We will also need to add field to store reference of StunnedPlayer so that when player is freed from stun, StunnedPlayer can be removed from map and its damage taken during its lifetime can obtained using this reference.

Before this, we initialise stunnedPlayer's hit points to be the same as the player's. However, we realised that there is no method that can return us the player's hitpoints. So, we now we decided to give the stunnedPlayer a very high hitPoints by using Integer.MAX_VALUE so that when the stunnedPlayer replaces the Player in the game, it will not get defeated due to the damages caused by other enemies even if Player should still be conscious. If the stunnedPlayer dies, the game will end because the World's player is now stunnedPlayer but it is not supposed to end yet. When the player is freed from the stun, we will also heal the stunnedPlayer to its maximum hitPoints so that when the player is stunned again, it can return the value of damage accurately.

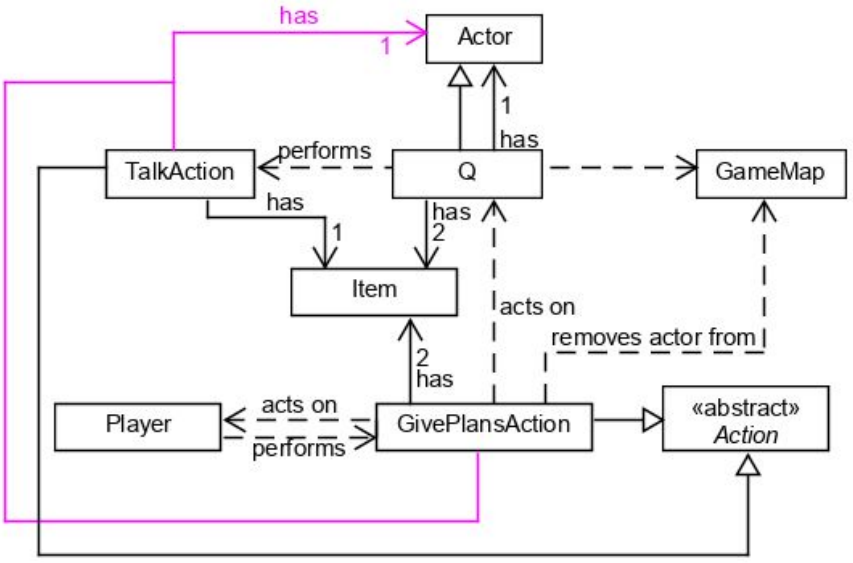


Figure 4: Updated class diagram for requirement – Q

The next change is to remove the WanderBehaviour class. WanderBehaviour is supposed to find out the various exits of Q and randomly returns a MoveActorAction to one of them. We found out that we can code this in its playTurn() method without having to create a new class. Since now Q do not have a behavior, the field named actionFactories of type List<ActionFactory> that we planned to give Q to aid in the processing of behavior can be removed too. In Q's improvised playTurn(), it first checks whether the target player is on an adjacent ground, if it is, TalkAction will be returned, otherwise, it will add all possible MoveActorAction into an Actions object, then call super.playTurn() with that Action objects which randomly selects one of the MoveActorAction. In the case of actions do not contain any MoveActorAction, SkipTurnAction will be returned. By making this change, we can reduce the amount of code that we need to write, thus reducing error.

We have also mentioned before that we planned to hardcode DrMaybe's hitpoints in its constructor. Since hard-coding is not a good programming practice, we call the super's

constructor, then divide its hit points by 2 instead. In Grunt class, getIntrinsicWeapon() will not be override so that it has the same damage as Enemy. Therefore, DrMaybe will have half the hit points of Grunt, which meets the requirement.