FIT2099
Assignment 2: Planning and UML

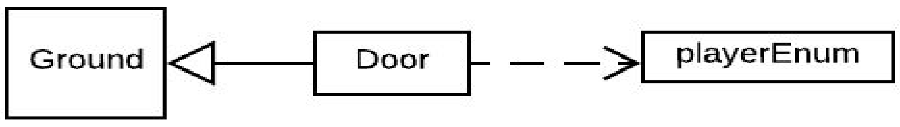
By:Shourya Raj and Tith Sothearith
In Team: Snowden

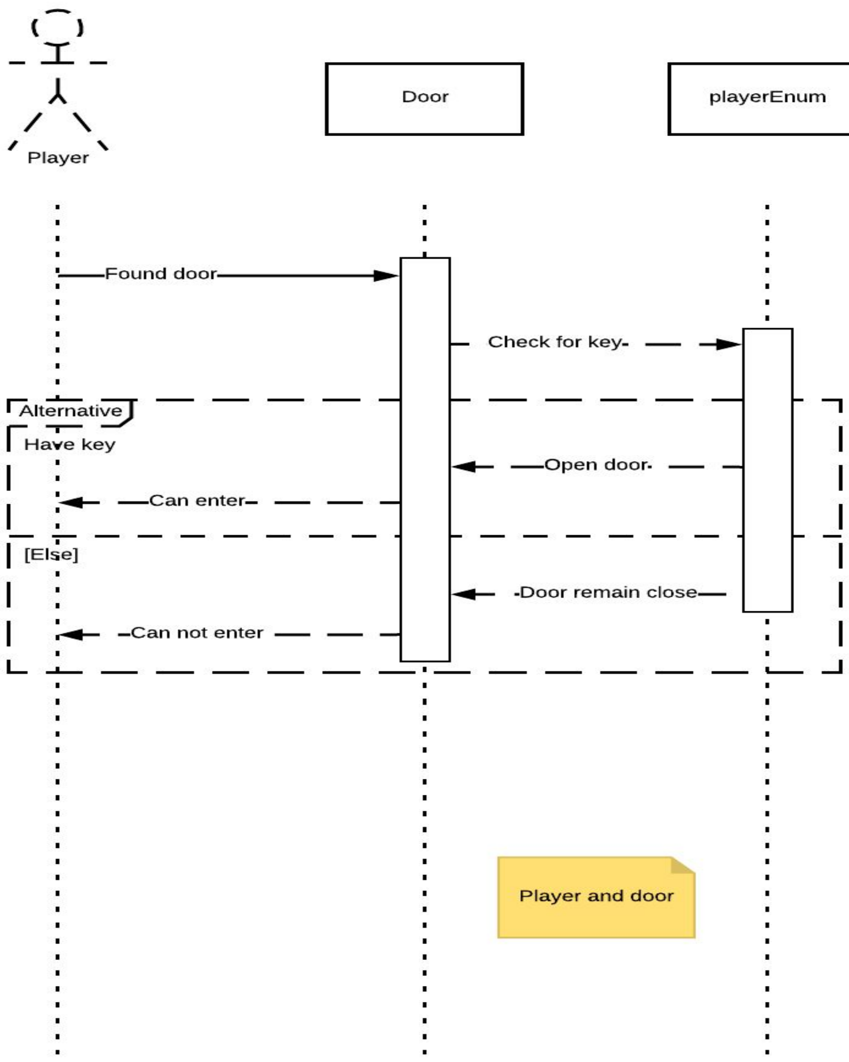
WBA:

We will devide the workload according to the tasks given in the assignment specification. In addition we planned to have a extra section with additional UML diagram
Delivery date: 2 days before the due date
Test and review by alternate team member.

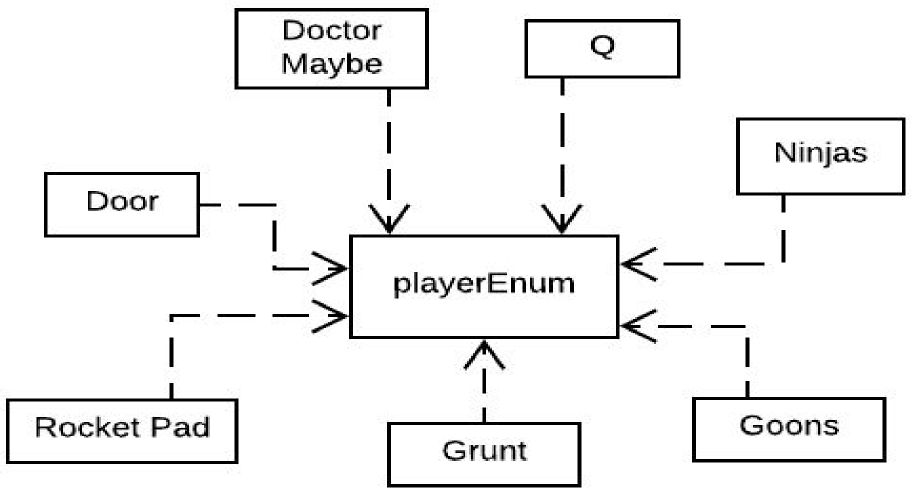
Task1: Door – Shourya raj
Task2: PlayerEnum – Tith Sotheraith
Task3:Goons– Shourya Raj
Task4:Ninja– Tith Sotheraith/ Shourya Raj
Task5:NPC(Q) – Tith Sotheraith
Task6: MiniBoss – Shourya Raj
Task7: Rocket Body - Tith Sotheraith

Door
Door will extend from the Ground helping it to reuse and follow the DRY- Don't repeat yourself- principle. Thus, there will be many reused codes and methods from its superclass, only requiring act, CanActorEnter method to only be true when player has got the key form the playerEnum, this can be achieved by defeating any of the enemy.
The role of this class is to make a sign of a door, player only enters when it has got a key to open it.
Door is depended upon the playerEnum to know about the key item is available, or not so that Method CanActorEnter works.





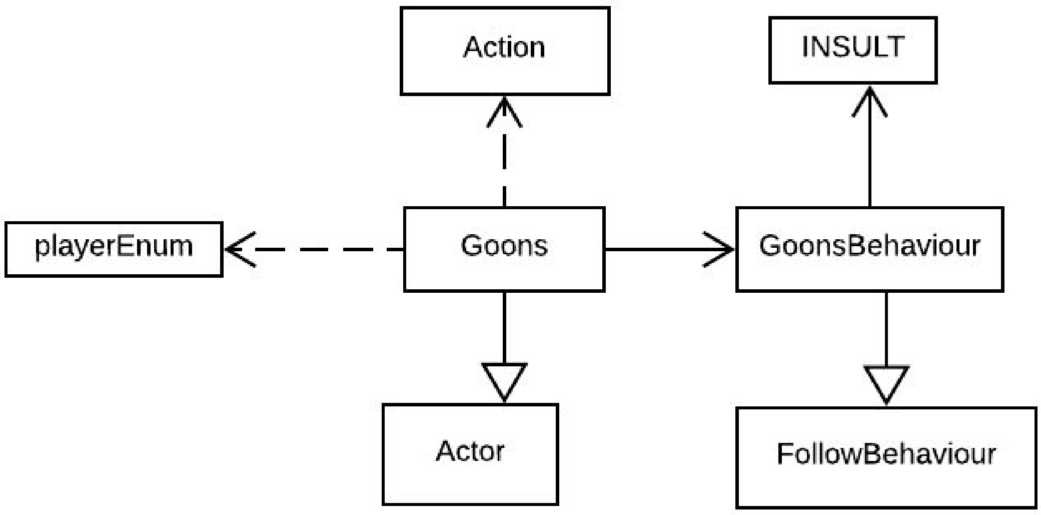
PlayerEnum
It is an Enum of the values Door key, Rocket body, Rocket Plan and Rocket engine. We use enum because Door key, Rocket body, Rocket Plan and Rocket engine are fixed constants. The advantage of using enum is that it can be used in a loop or a case statement which we will be using. Furthermore, enum can have fields, constructors and methods which make the code more flexible.
Those enum will be added as addSkill of the item. The purpose to do this thing is to check later that if that Actor has got that Item. This approach will ensure that the code stick with DRY principle and clean code.
All the actors are depended on the playerEnum to add or remove Door key, Rocket body, Rocket Plan and Rocket engine whereas Door and Rocket Pad depend on playerEnum to check for the values inside playerEnum to make decision.

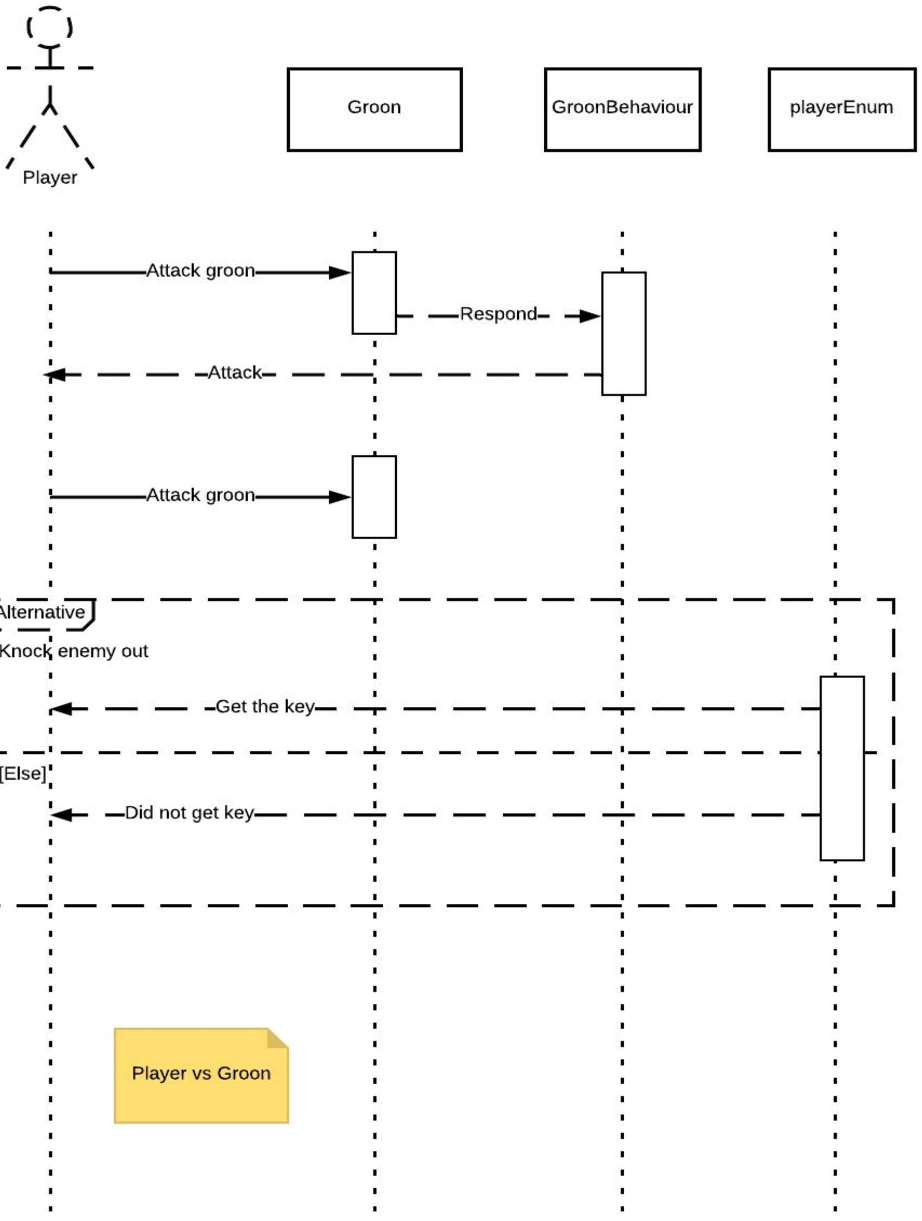


Goons
Goons will extend from the Actors, and follow same dependences and associations as the grunt have. Except the followbehaviour because goons have another extra functionality than grunt. It will use another class, goonBehaviour, which is extended from the followBehaviour class and INSULT class which will give out the insult. This helps to reuse and follow the DRY principle.

Improvement:
Having Methods for the extra behaviours for the goon- insult and twice damage as grunts.

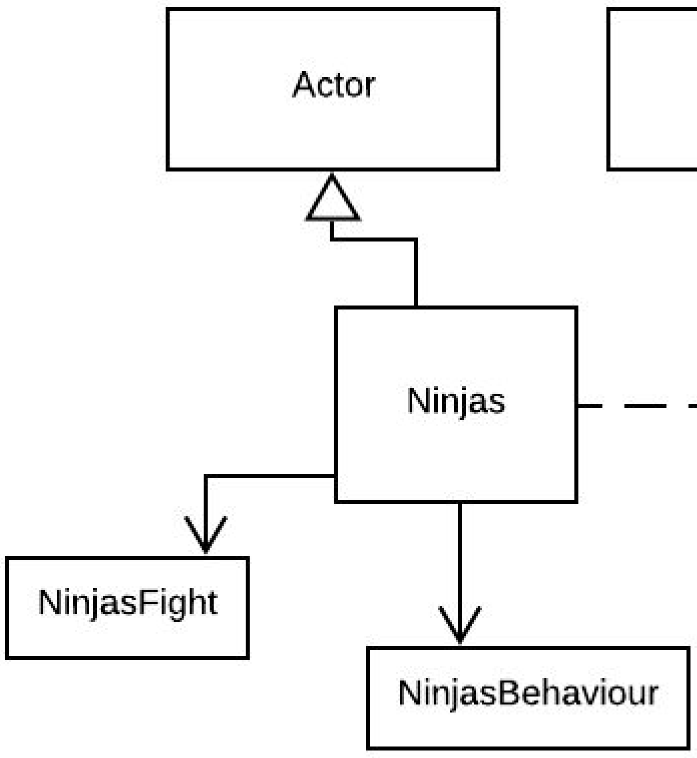
In the goons Behaviours, for the goon-insult method INSULT class will be called and INSULT class will use Math.random for every turn of the player if the value is between 0.1 to 0.2, 10% chance to come this value, then it will insult the player on the Display. Similarly, we need to add additional damage feature in an override function of the Action Class in the goon class.

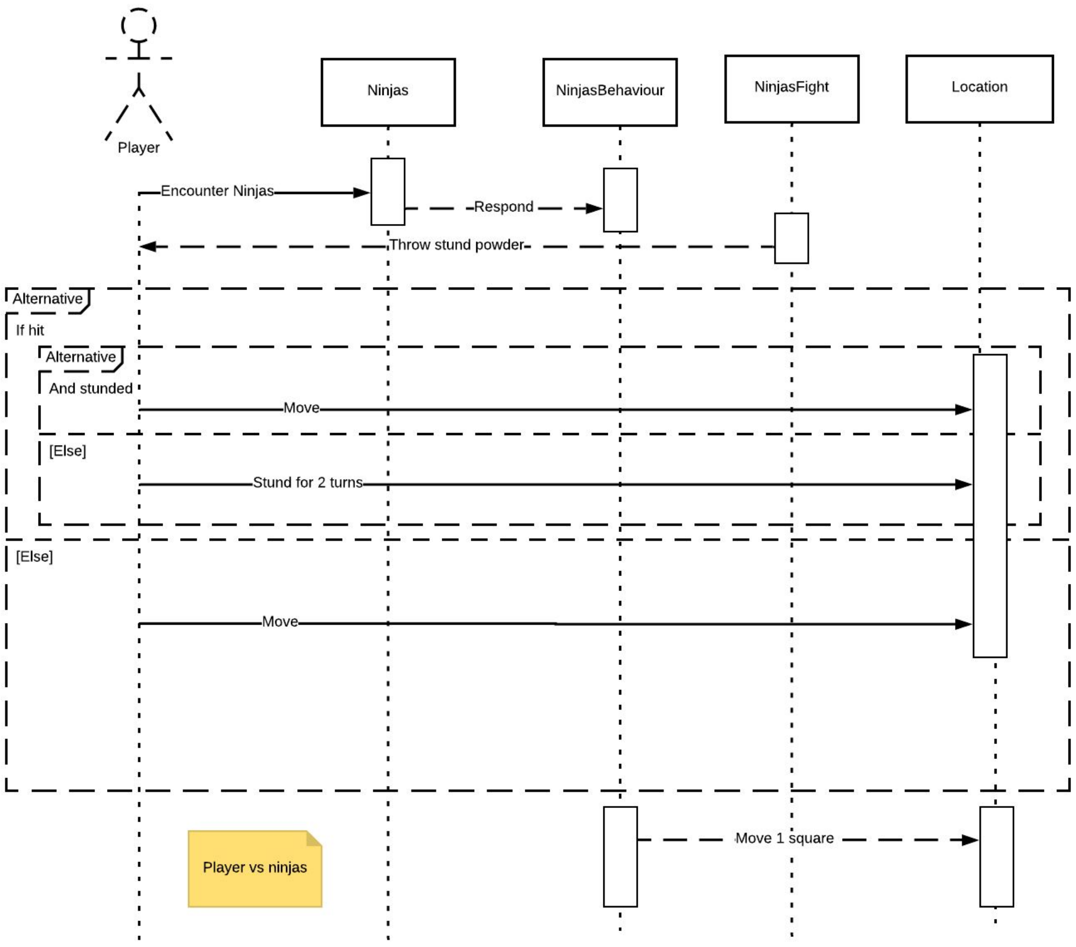

Action
INSULT
playerEnum
Goons
GoonsBehaviour
Actor
FollowBehaviour



Ninjas
Ninjas will extend form the Actors, and follow same dependences and associations as the grunt have. Expect the followBehaviour because Ninjas has got totally different behaviour. Thus, we will make another class for the Ninja behaviour.
Improvment:
Addid ing extra class:
NinjasFight- throw the stun powder, which follow same dependences and associations as the followBehaviour but having different methods.

Methods on the Ninjas have the methods as an override of the Actors Methods with having different behaviours. This will lead to having less dependences on any other class just by extending from the Actors class thus, obeying the DRY principle.
For the behaviour class we will add the two methods, throw the stun powder-its effects and movement of the Ninja on the map. In NinjasBehaviour class, the movement of ninjas will be implemented and in NinjasFight class, ninjas throwing stun powder will be implemented.
In the case of Ninja, Player cannot encounter the Ninja because of its movement behaviour thus, better not to put key in the Ninja Actor. Hence there no dependencies on the PlayerEnum from the Ninjas.





NPC(Non player character) : Q

The one character in the game that moves randomly is Q and for this we will create a class for Q to extend from the Actor, with keeping in mind about the DRY principle. Q class will have different behaviour so there will be different class for this.

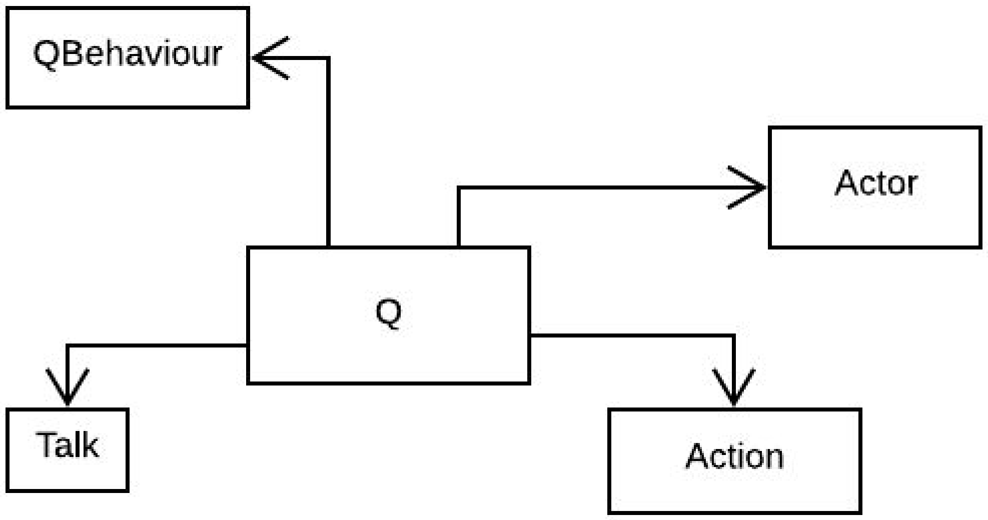
Q behaviour class will have methods for the interaction with Player which includes Talk or exchange the plans and rocket body and the random movement at the Map, which make it very hard for the Player to find Q on the map. To make it less annoying for Player, Q will move randomly with an interval of time.

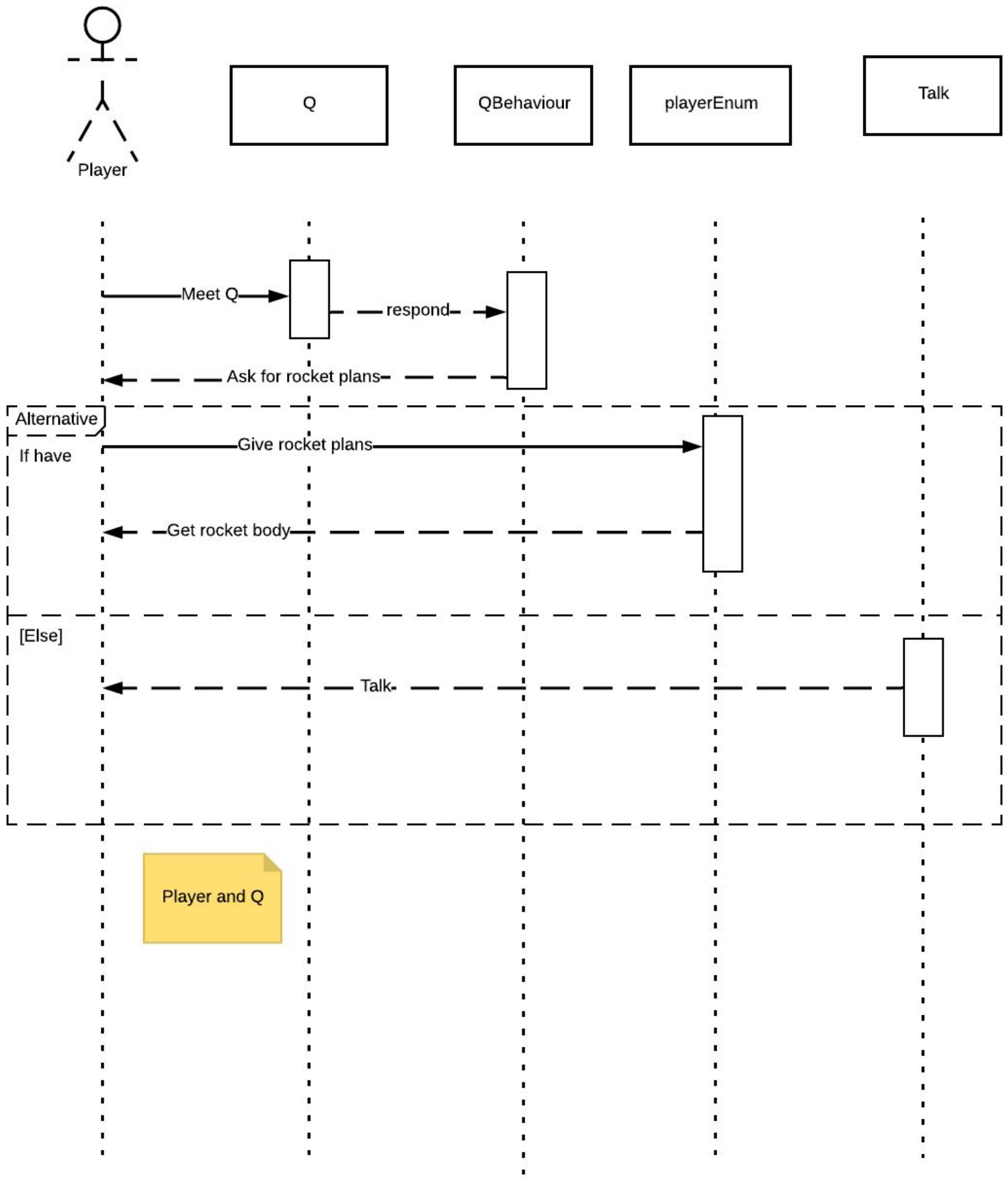
It knows about the playerEnum in order for the exchange of Rocket Plan and Rocket Body to happen.

Improvement:

Added one class to Talk and exchange the plan with the player, an actor can execute one function at a time making a class of a action would be the great choice to re use the method and follow the Dry principle.

.

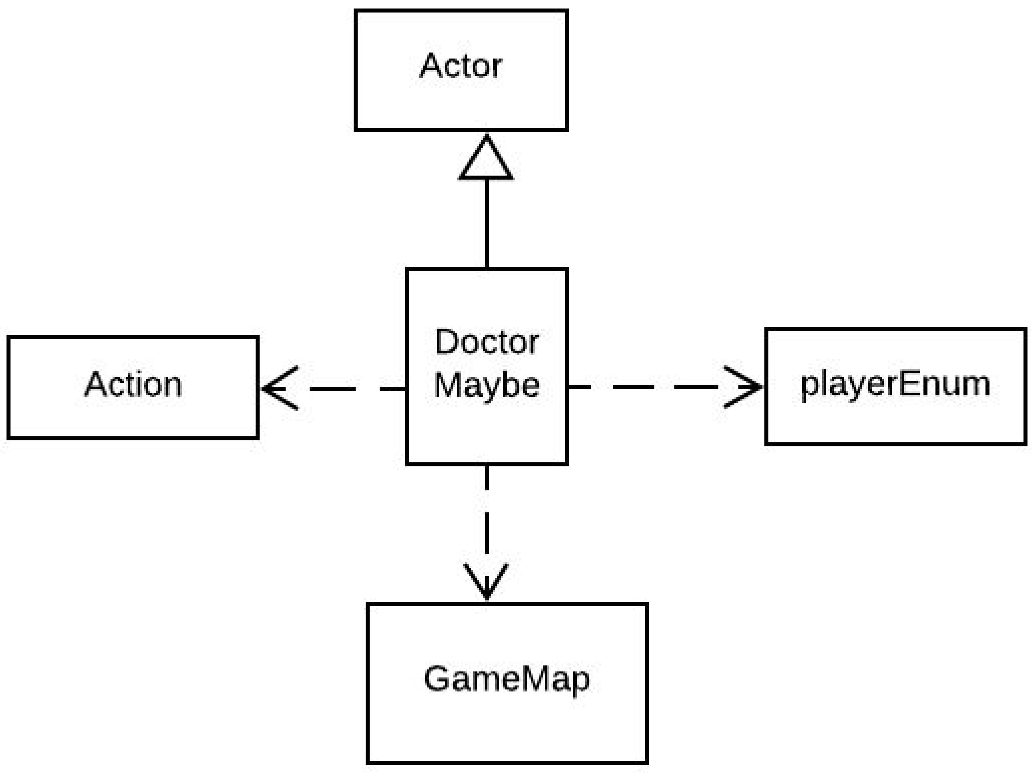

QBehaviour
Actor
Q
Talk
Action

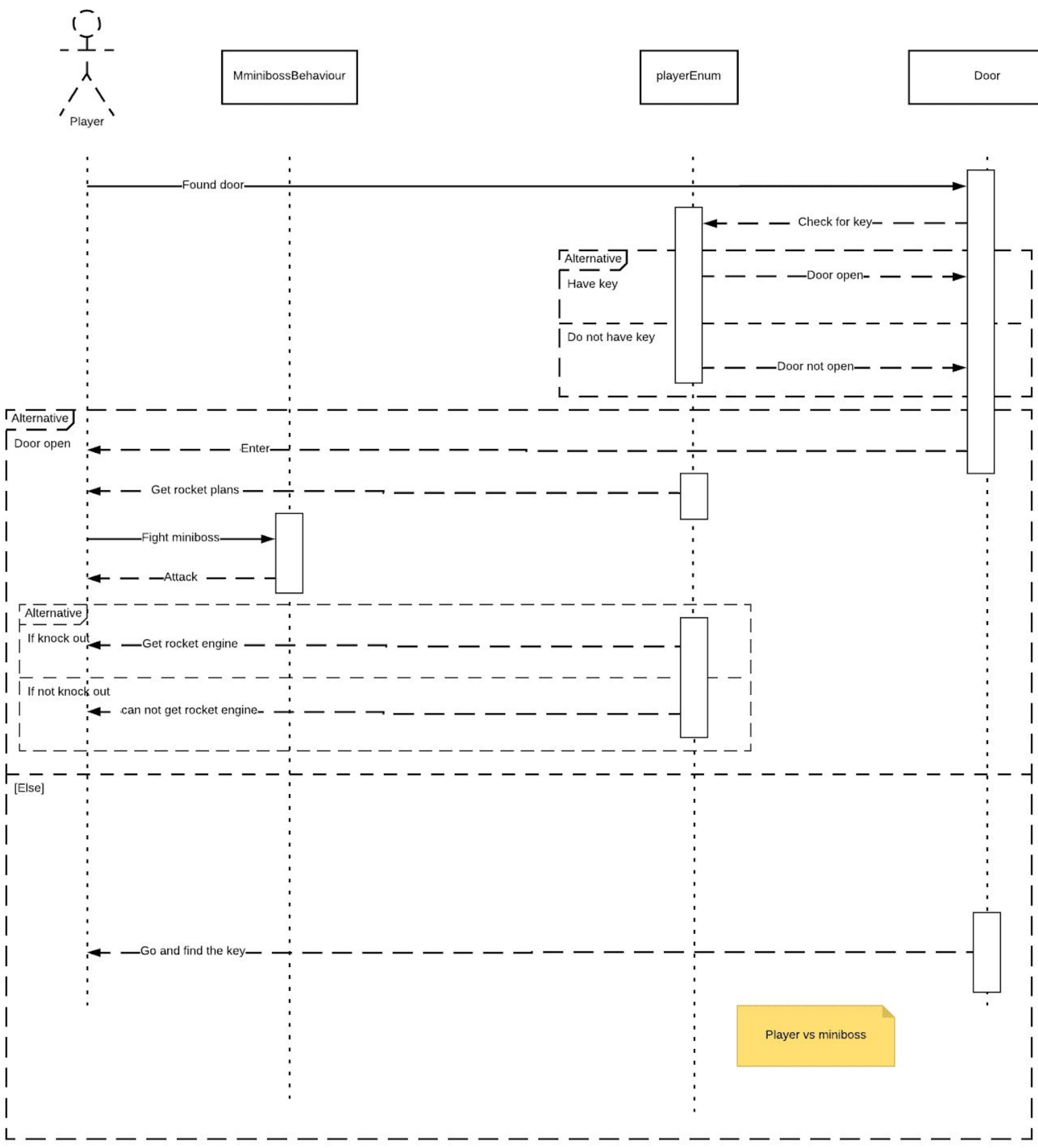


Mini Boss: Doctor Maybe

Mini Boss will extend from the Actor class, and follow same association and dependences like Q Class.

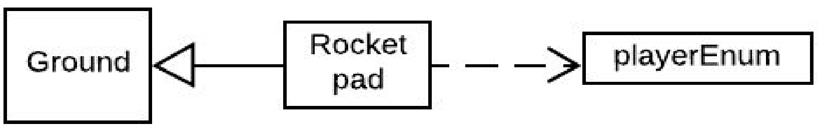
In this class, there is no need of any special behaviour class for the Mini boss because Mini boss stay at the same place. This class will use Method for half the damage of Grunt, and drop the item “Rocket engine” when defeated. Therefore, it knows about the playerEnum for the Rocket Engine.

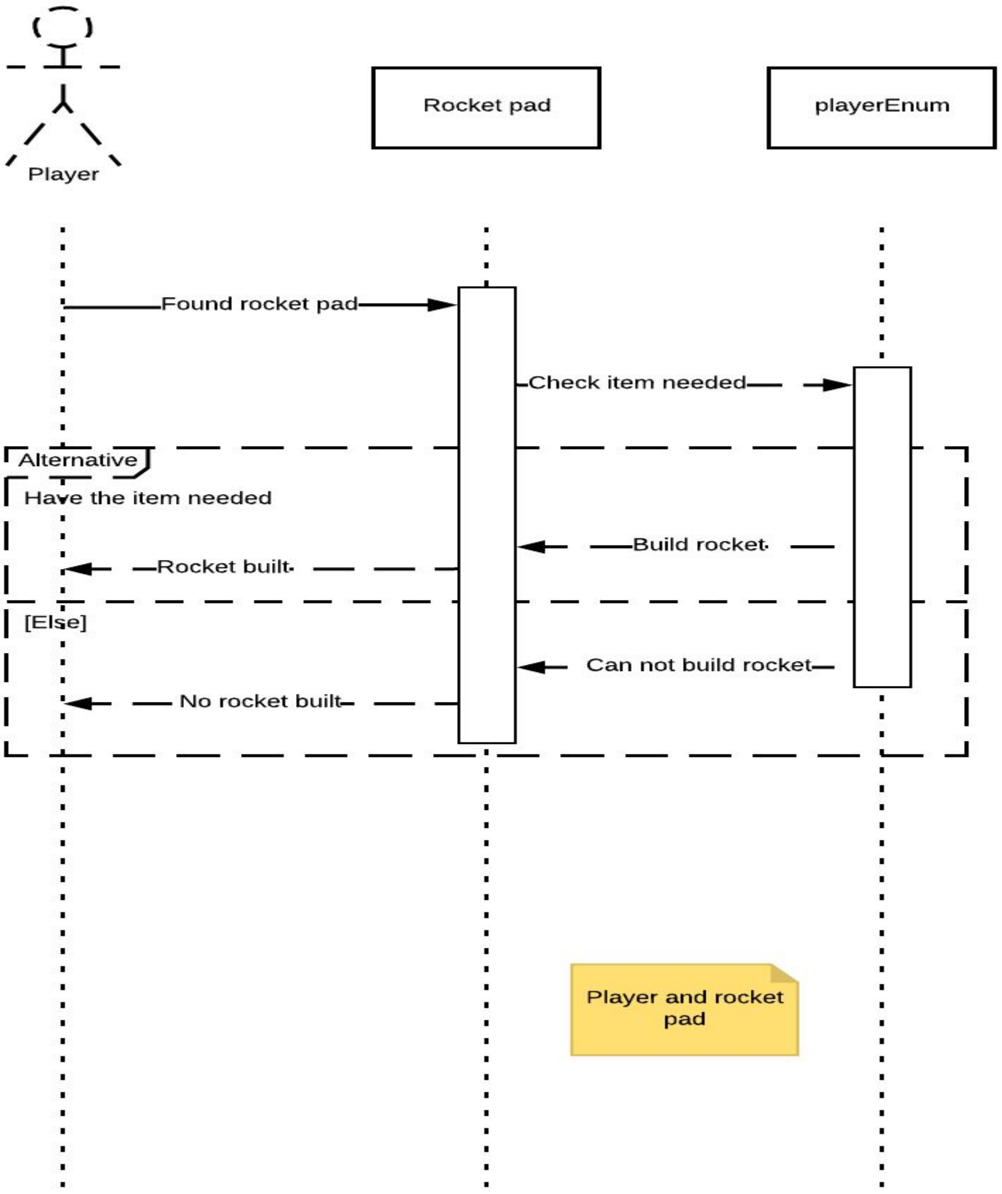

Actor
Doctor Maybe
Action
playerEnum
GameMap



Rocket Pad

Rocket Pad is a particular location on the Game Map. Therefore, it is better to reuse the code from Ground class i.e, rocket Pad will extend from the Ground class. In order to build a rocket, Rocket Pad knows about the Rocket engine, Rocket body. Hence, this can be achieve by checking their respective Enums. So it is depended upon the PlayerEnum.




Player
Rocket pad
playerEnum
Found rocket pad
Check item needed
Alternative
Have the item needed
Build rocket
Rocket built
[Else]
Can not build rocket
No rocket built
Player and rocket pad

Added Classes:

- Player2 class is the extended class from the Player class of the game engine, making of this class to add more additional functionality of the player which can interact with the surrounding.
    1) Stun powder behaviour
    2) Talk with the Q
    3) RocketBody

  Player2 is associated with the Talk class, MovePlayer and RocketBuild.
  Creating class approach provides not to repeat the code.

- NinjasFight class that will throw stun powder at the player if player is within the range.
- Talk class that allow player to interact with Q
- MinibossBehaviour class which restrict miniboss from moving and only attack player if player is close to miniboss
- INSULT class is an extends from GroonBehaviour which throw out insult at player
- RocketBuild - display a string when successfully build a rocket
- Rocket Body class is an extended of Item class to check and remove item from actor inventory
- Rocket Engine class is an extended of Item class and remove item from actor inventory
- Rocket Plan class is an extended of Item class and remove item from actor inventory

Extra feature:
For the extra feature we have created a new map, Thanos World, where player can go after the building Rocket, player have to go Launch pad to travel Thanos World. In the Thanos World there are following things: Thanos, Gemstones, Gauntlet and Snap location. In Thanos world, player have to collect all five Gemstones and gauntlet and use the combination of gemstones and gauntlet to remove half of the map.

Extra classes:

- MovePlayer class extends Action class
- gemStone class extends Item
- Gauntlet class extends Item
- Snap class extends action
- Snapped class extends GameMap
- RemoveActor class extends action
- Thanos world class extends GameMap
- Thanos extends Actor
- Space extends GameMap