Interaction Diagrams for TwitterNethack
Assignment in the course PA1415 Programvarudesign <2017-04-26>
Author: Morgan Lexander, Henrik Nilsson, Magnus Nyqvist och Victor Olsson.

Author Name	Social Security Number	Thinking	Writing
Morgan L.	971107-xxxx	25%	25%
Henrik N.	970416-xxxx	25%	25%
Magnus N.	940429-xxxx	25%	25%
Victor O.	970612-xxxx	25%	25%

System description

The system is going to be based around the classic game NetHack but in 2D graphics. NetHack is a sort of adventure and roleplaying game where you create a character and goes through random generated dungeons. In the dungeons you can meet various different monsters and collect items. The final goal is to find the amulet of Yendor and escape with it alive.

Nethack is a very advanced game and because of this and the limited time we are given, our version of NetHack is not going to include all of the features that the original have. The functionalities we are aiming to implement:

- An opening screen where you can start a new game or join an existing one.
- A screen where you can setup the character.
- Monsters and creatures that freely move around the dungeons.
- Different items to pick up and drop.
- Multiplayer support.
- Generate dungeons with help of twitter.

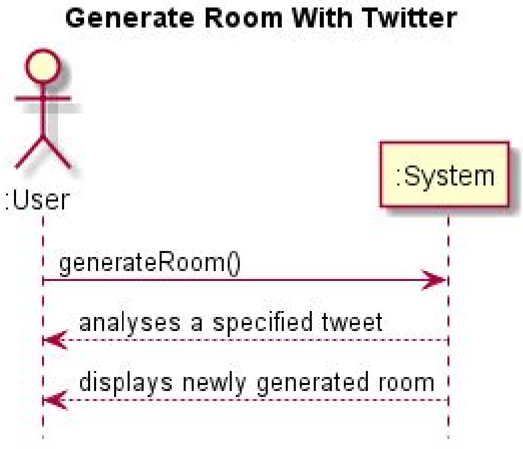
The dungeons in our NetHack version is going to be generated by the Twitter API. Its job is to read certain keywords in Twitter posts and use these to create specific types of monsters and items for every room in the game. When you enter a new cave, your system shall briefly display information about the “owner” of the cave.

Prioritised List of Use Cases

1. Generate Room with Twitter
2. Select a Twitter feed
3. Start Singleplayer
4. Spawn an Enemy
5. Enemy Attack Player
6. Attacks Enemy With Melee Weapon
7. Gain Experience Points
8. Opens Door
9. Attacks Obstacle
10. Destroy Obstacle
11. Open Inventory
12. Picks Up Money
13. Help Injured NPC
14. Accept a quest
15. Step into trap
16. Equip armor piece
17. Player Uses Potion
18. Ranged Attack
19. Player Chooses Race
20. Display Server list
21. Create player group
22. Attack other player with melee weapon
23. Killing another player
24. Trading items
25. chooses gender

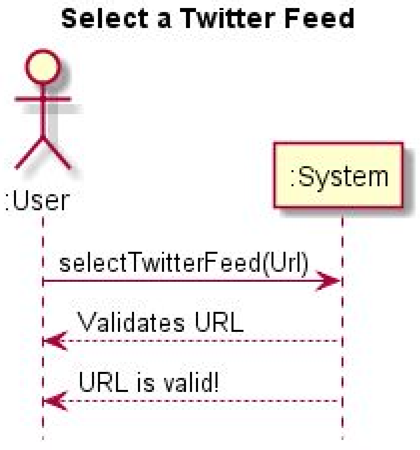
System Events

1. **System Events for Use Case <*Generate Room with Twitter*>**


Generate Room With Twitter
:User
:System
generateRoom()
analyses a specified tweet
displays newly generated room

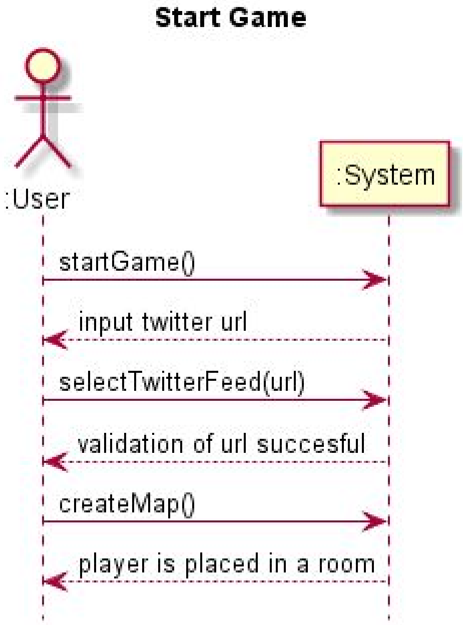
   a. generateRoom()

2. **System Events for Use Case <*Select a Twitter Feed*>**



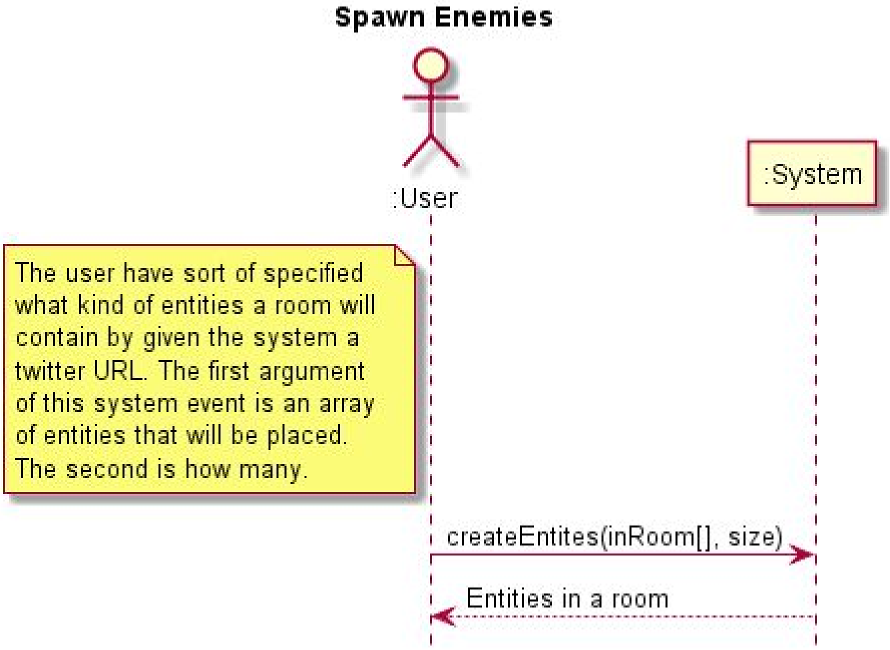
   a. selectTwitterFeed(Url)

3. **System Events for Use Case <*Start Singleplayer*>**



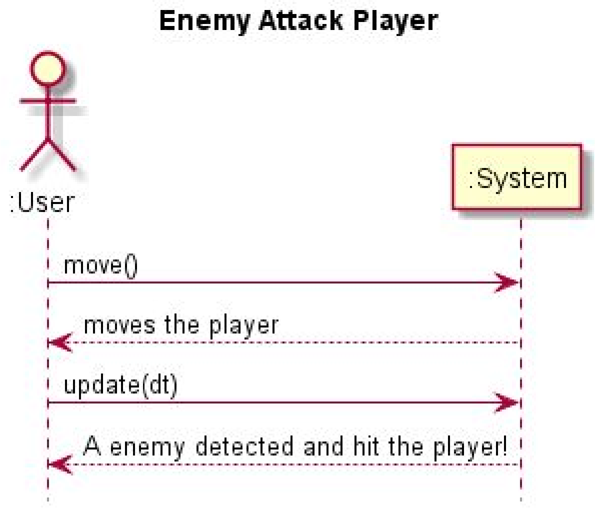
   a. startGame()
   b. selectTwitterFeed(url)’
   c. createMap()

4. **System Events for Use Case <*Spawn an Enemy*>**


Spawn Enemies
:User
:System
The user have sort of specified what kind of entities a room will contain by given the system a twitter URL. The first argument of this system event is an array of entities that will be placed. The second is how many.
createEntites(inRoom[], size)
Entities in a room

   a. createEntities(inRoom[], size)

5. System Events for Use Case<Enemy Attack Player>

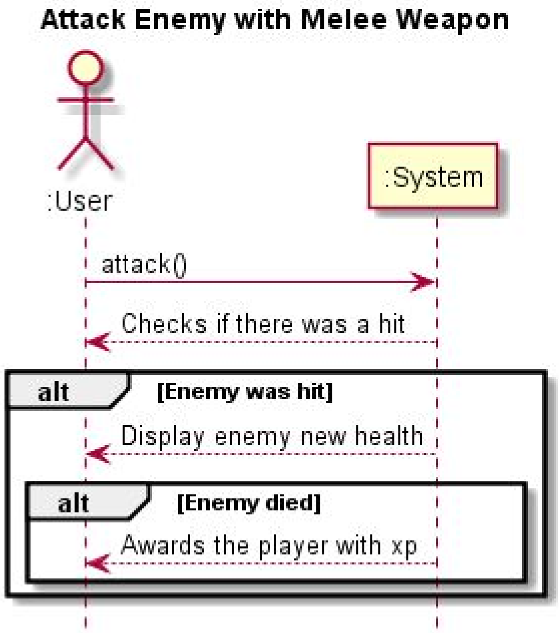


a. move()

b. update(dt)

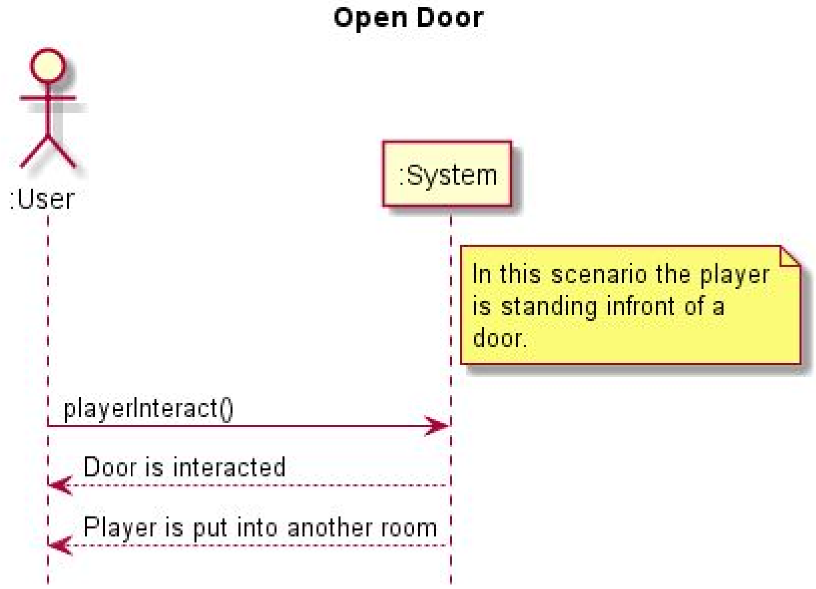
6. System Events for Use Case <Attacks Enemy with Melee Weapon>

In this system event we also feature the use case < Gain Experience Points>


Attack Enemy with Melee Weapon
:User
:System
attack()
Checks if there was a hit
alt
[Enemy was hit]
Display enemy new health
alt
[Enemy died]
Awards the player with xp

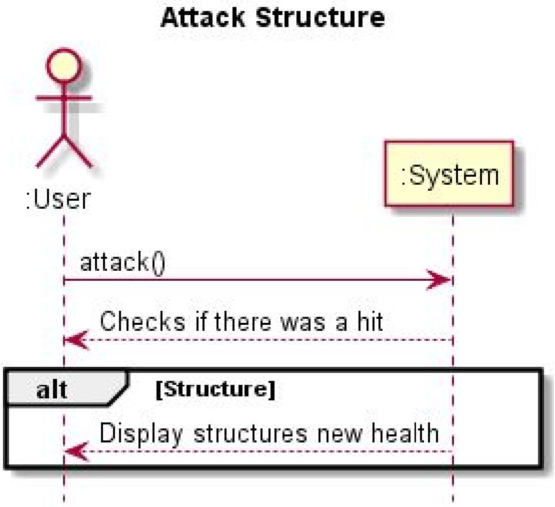
a. attack()

7. System Events for Use Case <*Open Door*>


Open Door
:User
:System
In this scenario the player is standing infront of a door.
playerInteract()
Door is interacted
Player is put into another room

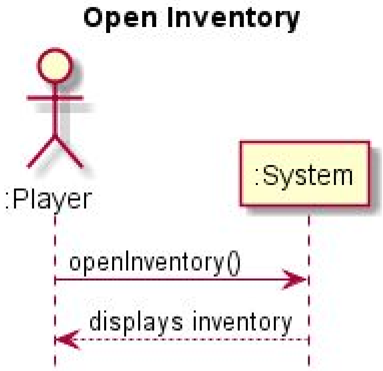
a. playerInteact()

8. System Events for Use Case <Attack Obstacle> and <Destroy Obstacle>



a. attack() (structure)

9. System Events for Use Case <Open Inventory>

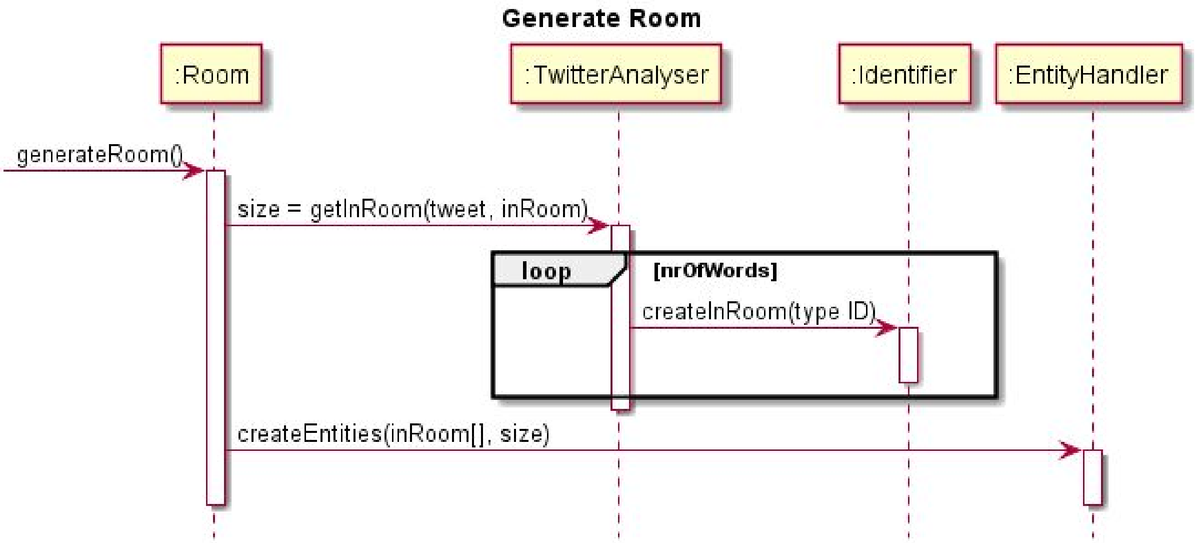


a. openInventory()

Interaction Diagrams

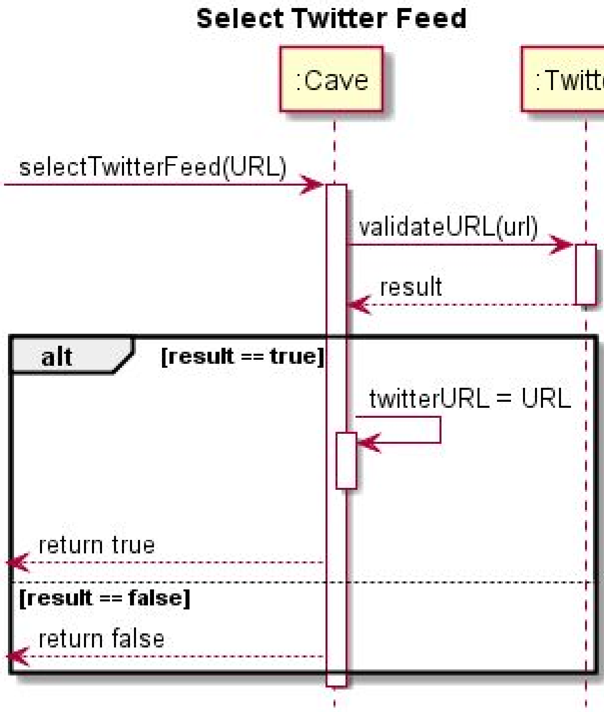
1. Interaction Diagrams for Use Case <*Generate Room with Twitter*>

(1)



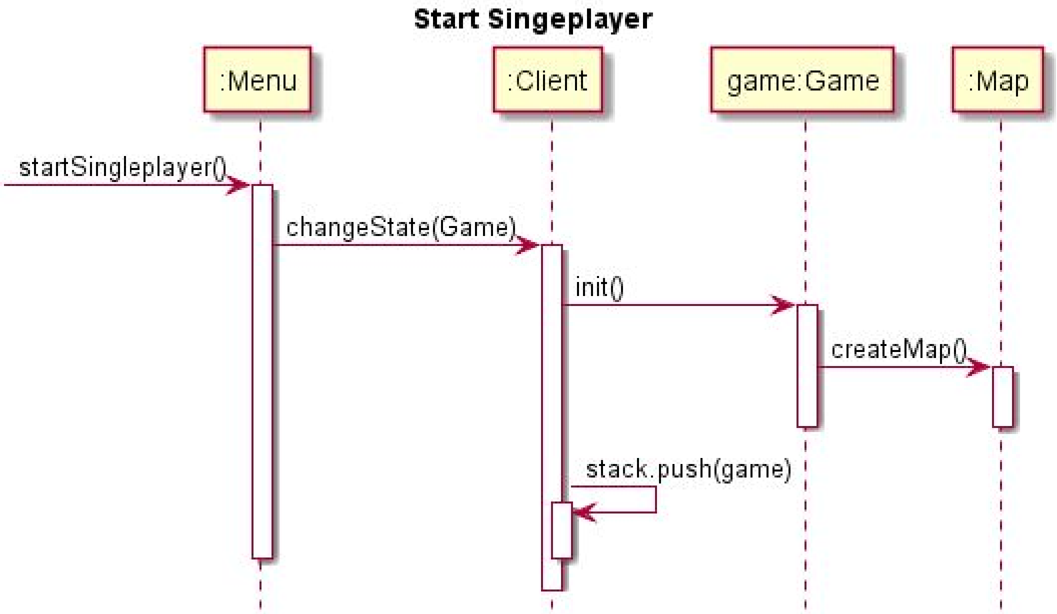
2. **Interaction Diagrams for Use Case <*Select a Twitter Feed*>**

(2)



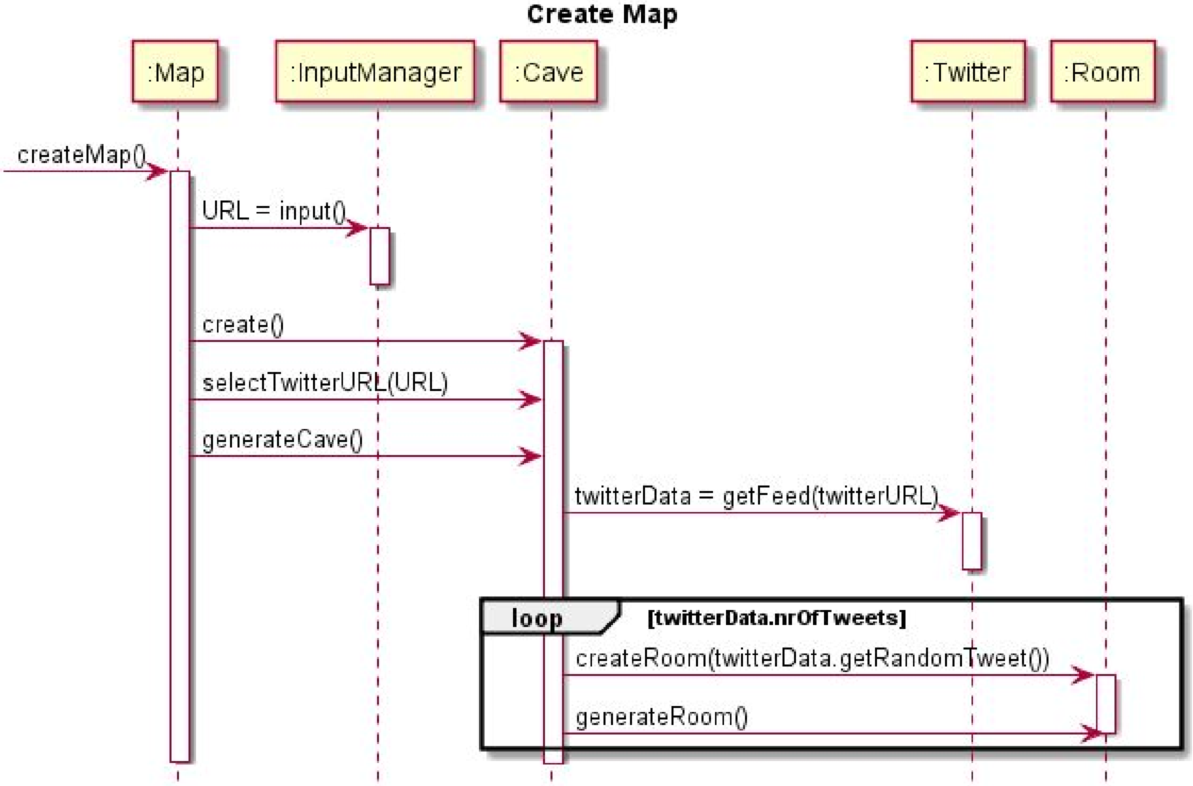
3. **Interaction Diagrams for Use Case <*Start Single Player*>**

(3)



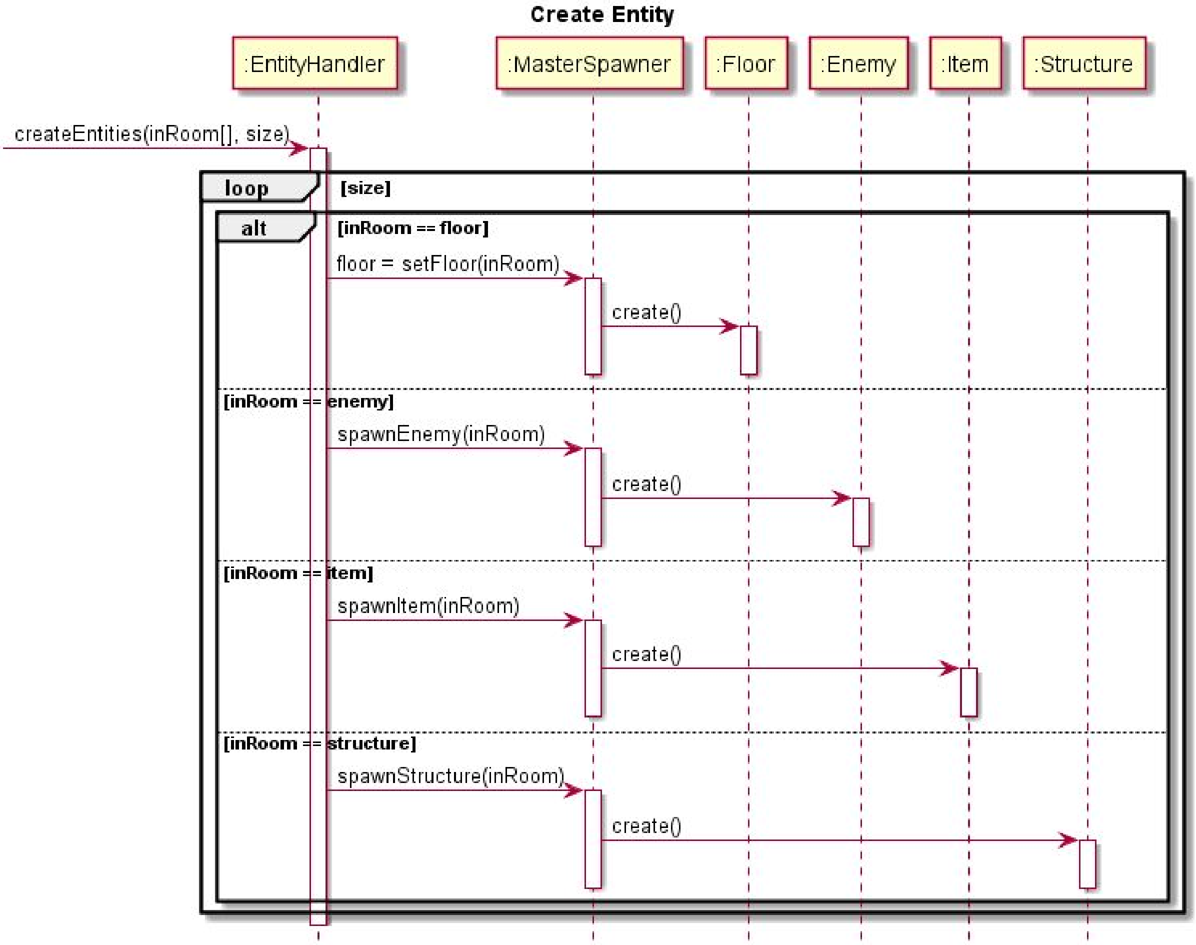
(2) *Select twitter feed (se under use case <Select a Twitter feed>).*

(4)


Create Map
:Map
:InputManager
:Cave
:Twitter
:Room
createMap()
URL = input()
create()
selectTwitterURL(URL)
generateCave()
twitterData = getFeed(twitterURL)
loop
[twitterData.nrOfTweets]
createRoom(twitterData.getRandomTweet())
generateRoom()

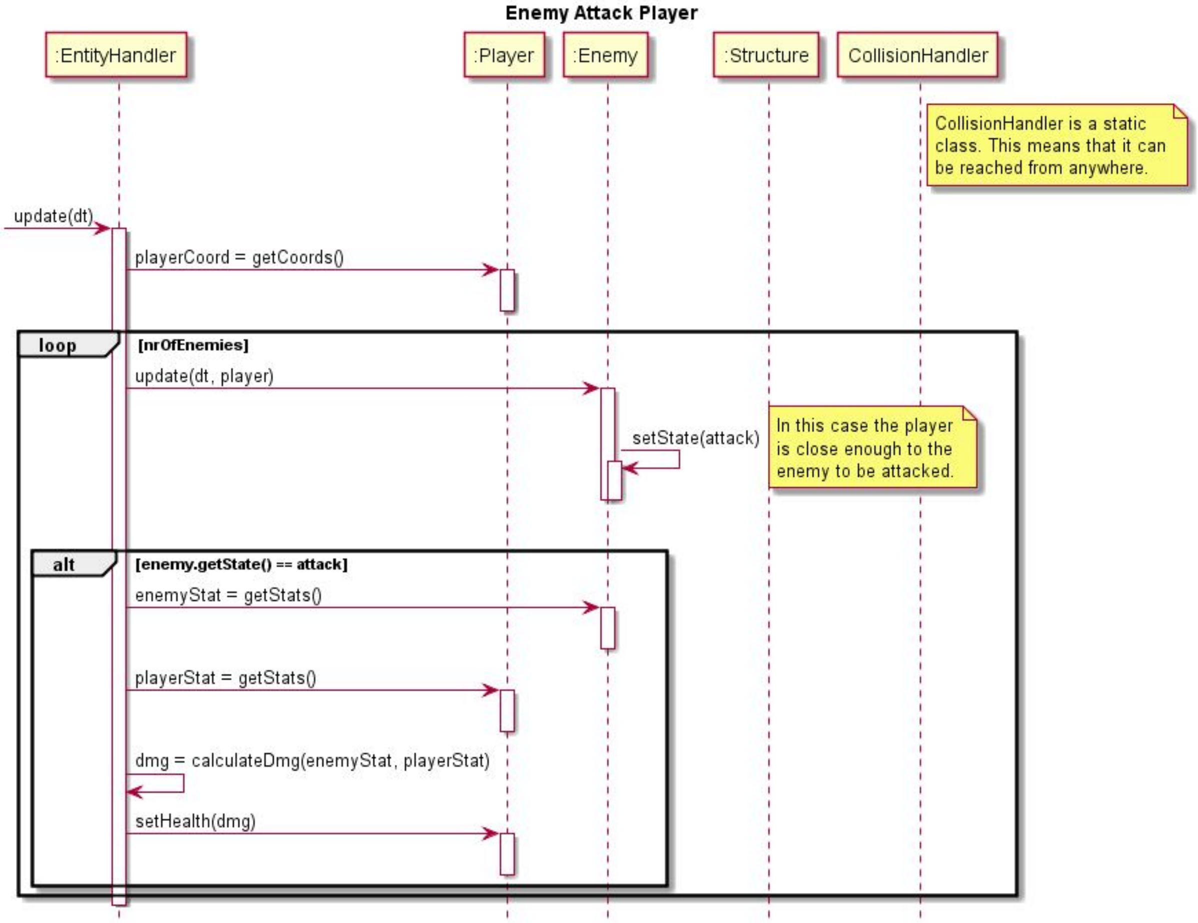
4. **Interaction Diagrams for Use Case <*Spawn an Enemy*>**

(5)

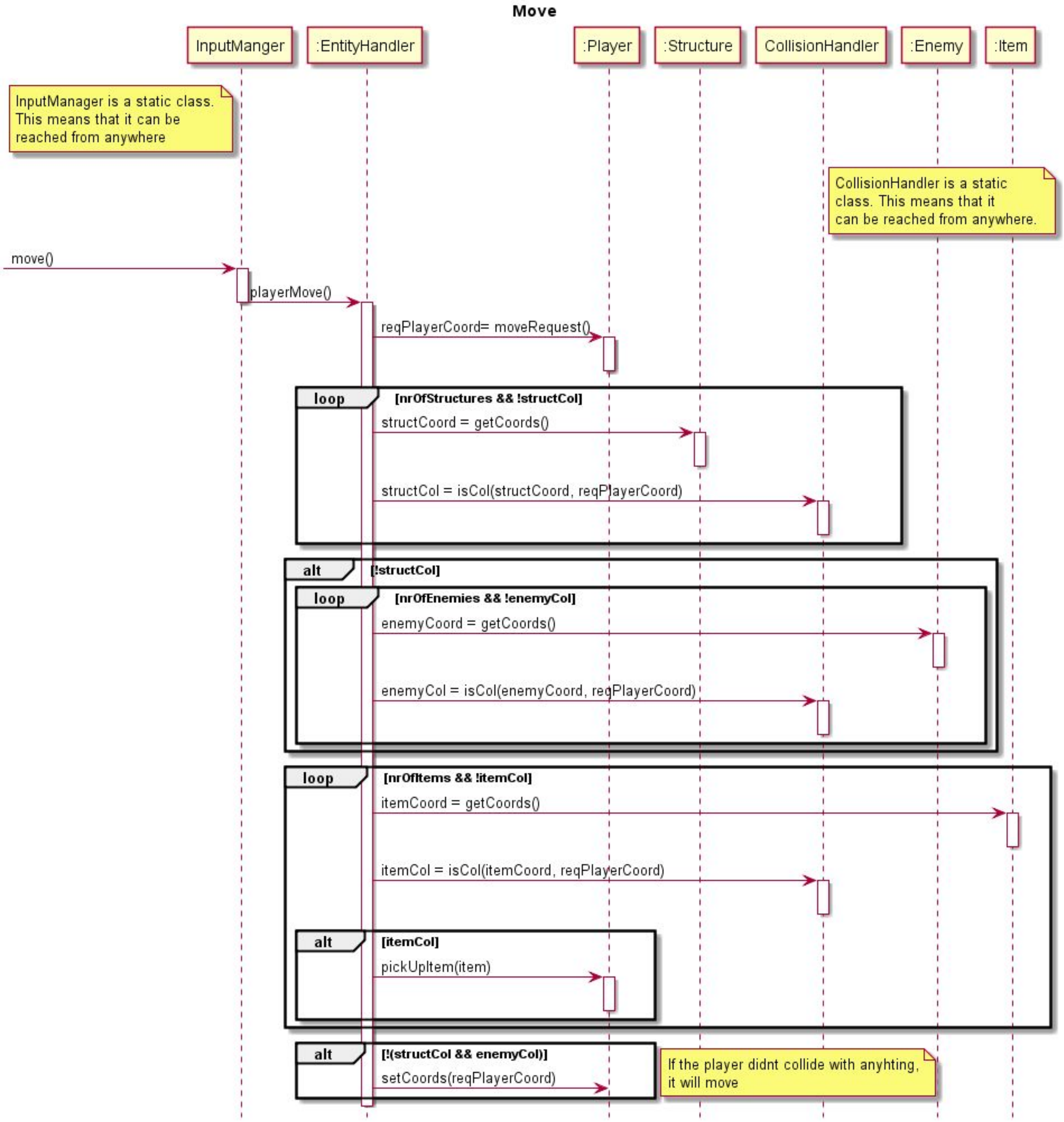


5. Interaction Diagrams for Use Case <*Enemy Attack Player*>

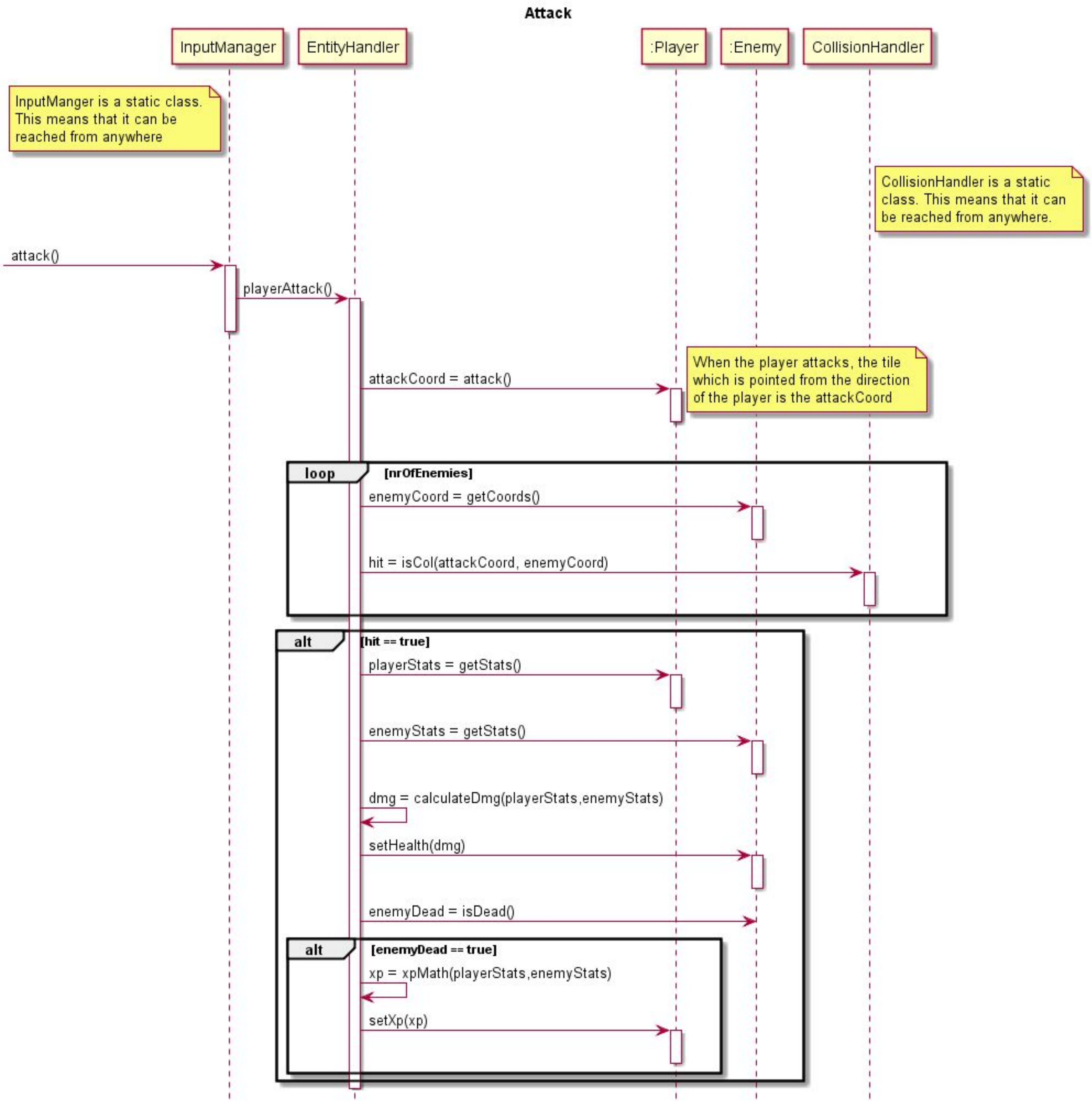
(6)


Enemy Attack Player
:EntityHandler
:Player
:Enemy
:Structure
CollisionHandler
CollisionHandler is a static class. This means that it can be reached from anywhere.
update(dt)
playerCoord = getCoords()
loop
[nrOfEnemies]
update(dt, player)
setState(attack)
In this case the player is close enough to the enemy to be attacked.
alt
[enemy.getState() == attack]
enemyStat = getStats()
playerStat = getStats()
dmg = calculateDmg(enemyStat, playerStat)
setHealth(dmg)

(7)

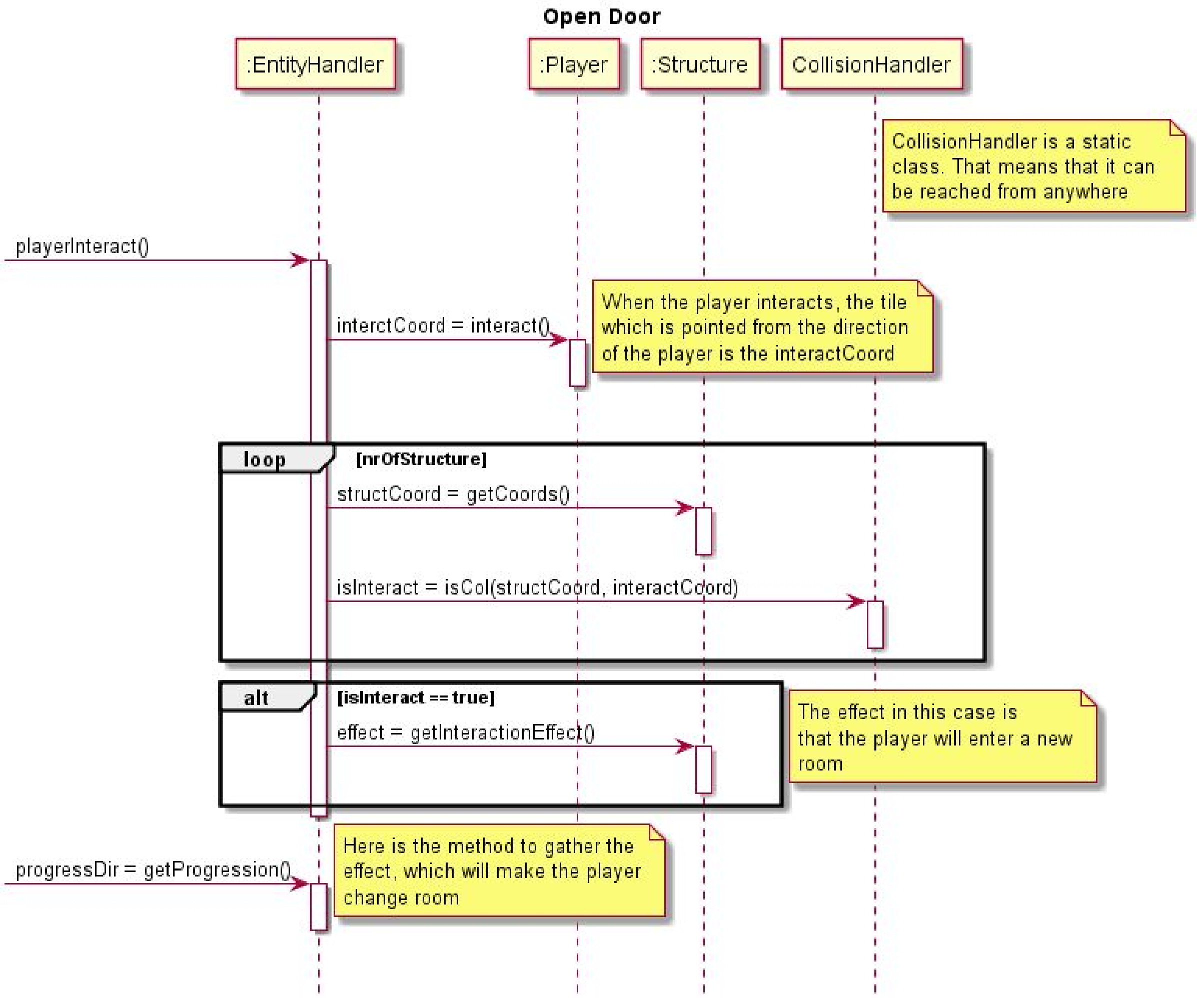

Move
InputManger
:EntityHandler
:Player
:Structure
CollisionHandler
:Enemy
:Item
InputManager is a static class. This means that it can be reached from anywhere
CollisionHandler is a static class. This means that it can be reached from anywhere.
move()
playerMove()
reqPlayerCoord= moveRequest()
loop
[nrOfStructures && !structCol]
structCoord = getCoords()
structCol = isCol(structCoord, reqPlayerCoord)
alt
[!structCol]
loop
[nrOfEnemies && !enemyCol]
enemyCoord = getCoords()
enemyCol = isCol(enemyCoord, reqPlayerCoord)
loop
[nrOfItems && !itemCol]
itemCoord = getCoords()
itemCol = isCol(itemCoord, reqPlayerCoord)
alt
[itemCol]
pickUpItem(item)
alt
[!(structCol && enemyCol)]
setCoords(reqPlayerCoord)
If the player didnt collide with anyhting, it will move

6. Interaction Diagrams for Use Case <*Attacks Enemy with Melee Weapon*> (8)

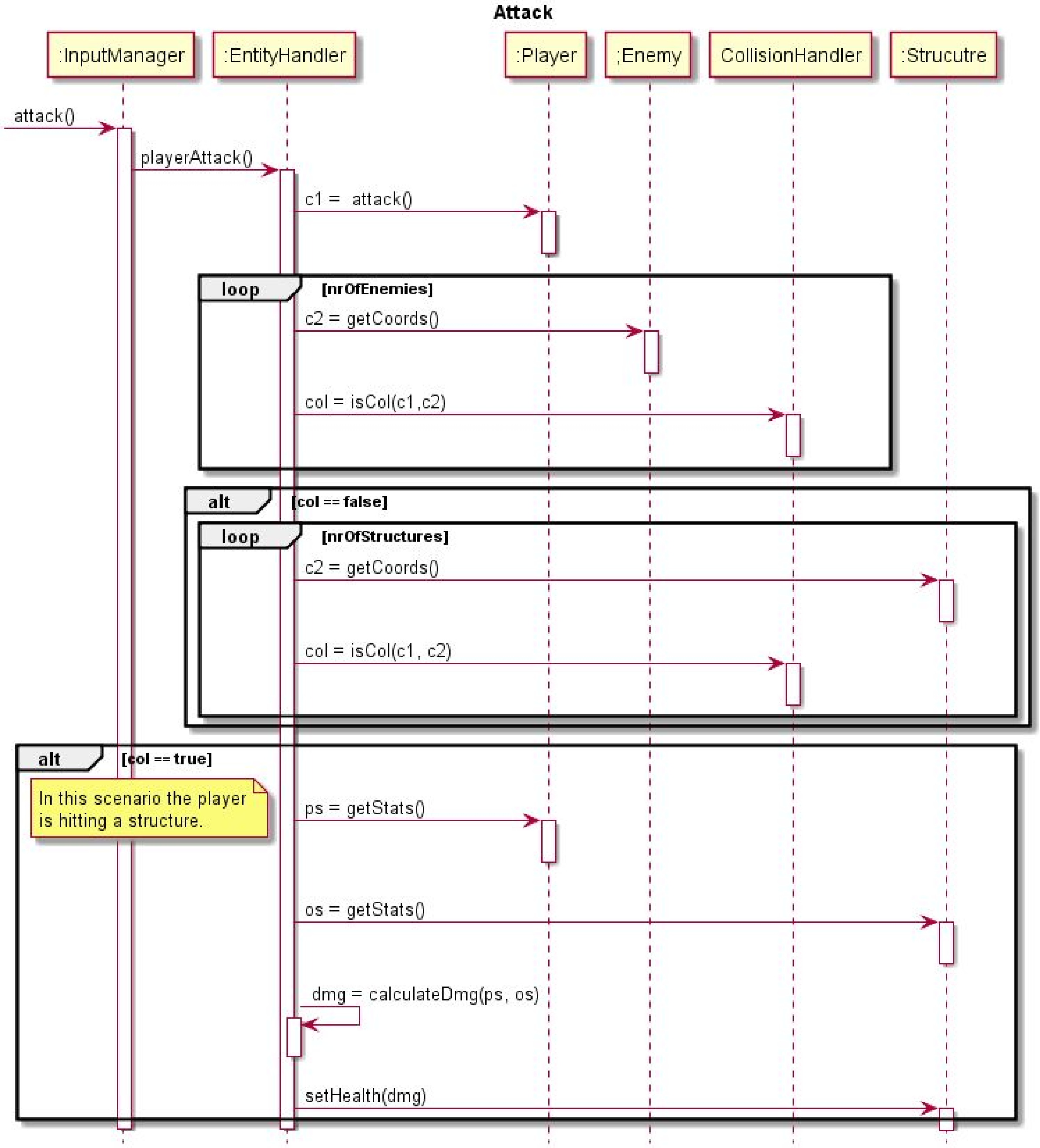

Attack
InputManager
EntityHandler
:Player
:Enemy
CollisionHandler
InputManger is a static class. This means that it can be reached from anywhere
CollisionHandler is a static class. This means that it can be reached from anywhere.
attack()
playerAttack()
attackCoord = attack()
When the player attacks, the tile which is pointed from the direction of the player is the attackCoord
loop
[nrOfEnemies]
enemyCoord = getCoords()
hit = isCol(attackCoord, enemyCoord)
alt
[hit == true]
playerStats = getStats()
enemyStats = getStats()
dmg = calculateDmg(playerStats,enemyStats)
setHealth(dmg)
enemyDead = isDead()
alt
[enemyDead == true]
xp = xpMath(playerStats,enemyStats)
setXp(xp)

7. Interaction Diagrams for Use Case <*Open Door*>

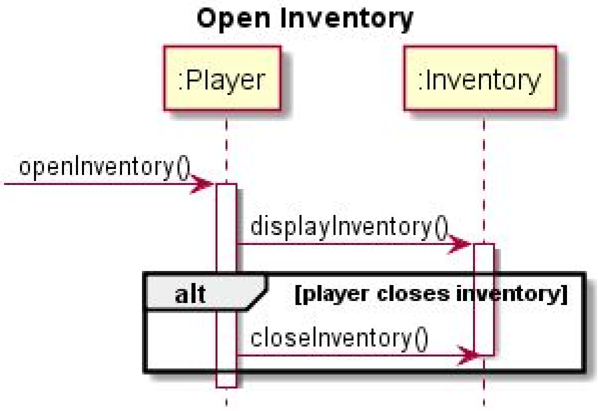
(9)


Open Door
:EntityHandler
:Player
:Structure
CollisionHandler
CollisionHandler is a static class. That means that it can be reached from anywhere
playerInteract()
interctCoord = interact()
When the player interacts, the tile which is pointed from the direction of the player is the interactCoord
loop
[nrOfStructure]
structCoord = getCoords()
isInteract = isCol(structCoord, interactCoord)
alt
[isInteract == true]
effect = getInteractionEffect()
The effect in this case is that the player will enter a new room
progressDir = getProgression()
Here is the method to gather the effect, which will make the player change room

8. Interaction Diagram for Use Case <*Attacks Obstacle*> and <*Destroy Obstacle*> (10)



9. **Interaction Diagram for Use Case "Open Inventory"**
(11)


Open Inventory
:Player
:Inventory
openInventory()
displayInventory()
alt
[player closes inventory]
closeInventory()