Designing the Sprint 1 version

From specification to UML

Requirements

The player should be able to walk around between the rooms in the “cave” and if there are any things in a room, the player should be able to pick them up (move them from the room to the player’s inventory). The player can also drop down a thing to the current room. Doing so should add the thing to the room’s list of things and remove the thing from the player’s inventory. The rooms in the cave are connected according to the data in the database (you also have some PDFs with maps - ask the teachers).

Requirements - Nouns - candidate classes

The ***player*** should be able to walk around between the ***room***s in the "***cave***" and if there are any ***thing***s in a ***room***, the ***player*** should be able to pick them up (move them from the ***room*** to the ***player***'s ***inventory***). The ***player*** can also drop down a ***thing*** to the current ***room***. Doing so should add the ***thing*** to the ***room***'s list of ***thing***s and remove the ***thing*** from the ***player***'s ***inventory***. The ***room***s in the ***cave*** are connected according to the data in the database (you also have some PDFs with maps - ask the teachers).

- Player
- Room
- Cave
- Thing
- Inventory

Candidate classes - Player

- Player keep/scratch?

Seems like a keeper - a central object in the game, since it is the Player which goes around in the Room:s and picks Thing:s up.

How many Player:s do we need? One (Singleton - could be discussed)

What does a Player instance have or know?

- List of Thing:s (inventory)
- Current Room - Where the Player is in the “Cave”

Candidate classes - Room

- Room keep/scratch?

Seems like a keeper! It is mentioned many times together with the Player.

What's in a Room?

From the specification, we learn that a Room has a list of Thing:s.

Room:s seem to be connected to each other via directions in the Cave "map"

```
      R1

R2    R3    R4        From Room R3 you can go North to Room R1 etc since they are connected

      R5
```

Candidate classes - Cave

- Cave keep/scratch?

What is “the cave”? It seems to be a bunch of connected Room:s.

So this is not a class, but rather a collection of Room:s.

But how are Room:s connected? Via some index? No, via eachother. More on this later.

Decision: Scratch it! (as a class)

Candidate classes - Thing

- Thing keep/scratch?

What is a Thing?

Thing:s are items like the Bird, Cage, Skeleton Key etc which can be picked up by the Player. It seems rather central to the whole game functionality in this version.

We say: Keep it!

List of Things will be part of the Room and Player instances.

Candidate classes - Inventory

- Inventory keep/scratch?

What is in the inventory, really? The Player’s list of….

What’s in it? … Thing:s, right? So do we need it?

We say: Yes, but it is not a class, it’s a collection of Thing:s

Who has this list of things? The Player!

Player is aggregated with a list of Thing:s (has a list of Things)

Scratch the Inventory class, since it is simply a List<Thing> of the Player

Design - Classes - properties

```
Player
-inventory : Thing [*]
-currentRoom : Room

Room
-description: String
-things : Thing [*]
-north : Room  ← Can you do that? Yes, why the heck not?
-south : Room
-east  : Room
-west  : Room

Thing
-name : String
```

Requirements - Verbs - behavior

The player should be able to ***walk*** around between the rooms in the “cave” and if there are any things in a room, the player should be able to ***pick them up*** (move them from the room to the player’s inventory). The player can also ***drop down*** a thing to the current room. Doing so should add the thing to the room’s list of things and remove the thing from the player’s inventory. The rooms in the cave are connected according to the data in the database (you also have some PDFs with maps - ask the teachers).

- go(*where?*) - who goes? What is the result?
- takeThing(*what thing?*) Who takes? What is the result?
- dropThing(*whatThing*) Who drops? What is the result?

Requirements - Verbs - behavior - go

- go(*where?*) - who goes? What is the result?

Who goes? - Player

Where? A Direction (aha! noun, so why not an enum Room.Direction?) like NORTH.

What is the result? Nothing returned, so void. Result: currentRoom changes *if there is a Room from the current Room in the given direction*

```
+go(direction : Room.Direction) : void
```

Requirements - Verbs - behavior - takeThing

- takeThing(what thing?) Who takes? What is the result?

Who? - Player “picks up/takes a Thing”

What? A Thing (from the Room’s list of Thing:s).

What is the result? Nothing returned, so void. Result: The Thing is removed from the Room’s list of Thing:s and added to the Player’s list of Things.

```
+takeThing(thing : Thing) : void
```

From the results, we find that the Room needs a behavior for removing a Thing!

Requirements - Verbs - behavior - removeThing

- removeThing(what thing?) Remove from where? What is the result?

Where? - When Player “picks up/takes a Thing”, the Thing is removed from the Room’s list of things.

What? A Thing (from the Room’s list of Thing:s).

What is the result? Nothing returned, so void. Result: The Thing is removed from the Room’s list of Thing:s.

```
+removeThing(thing : Thing) : void
```

Requirements - Verbs - behavior - dropThing

- dropThing(what thing?) Who drops? What is the result?

Who? - Player “drops down a Thing down”

What? A Thing (from the Player’s list of Thing:s/inventory).

What is the result? Nothing returned, so void. Result: The Thing is removed from the Player’s list of Thing:s and added to the Room’s list of Thing:s.

```
+dropThing(thing : Thing) : void
```

From this, we learn that we need a behavior for the Room - addThing!

Requirements - Verbs - behavior - addThing

- addThing(what thing?) Add to what? What is the result?

To what? - Player “drops down a Thing down” which is added to the Room’s list of Thing:s.

What? A Thing (from the Player’s list of Thing:s/inventory).

What is the result? Nothing returned, so void. Result: The Thing is removed from the Player’s list of Thing:s and added to the Room’s list of Thing:s.

```
+addThing(thing : Thing) : void
```

Requirements - Verbs - describing the Room

- description()

We need to describe the Room somehow - “You are in a dark boring classroom. The projector is not working. There’s a higher seminar going on in the background.”

```
+description() : String
```

Requirements - Verbs - behavior - Thing's name()

What about the Thing class? No behavior there? Boo-hoo!

OK, some consolation prizes (suggestions):

```
+name() : String

+equals(thing : Object) : boolean

// Perhaps even:

+hashCode() : int
```

Properties and behavior - Player

```
Player
-----------------------------------------
-inventory : Thing [*]
-currentRoom : Room
-----------------------------------------
+go(direction : Room.Direction) : void

+takeThing(thing : Thing) : void

+dropThing(thing : Thing) : void

-----------------------------------------
```

Properties and behavior - Room

```
Room
-----------------------------------------
-description: String
-things : Thing [*]
-north : Room  ← Can you do that? Yes, why the heck not?
-south : Room
-east  : Room
-west  : Room
-----------------------------------------
+removeThing(thing : Thing) : void

+addThing(thing : Thing) : void

+description(): String
-----------------------------------------
```

Properties and behavior - Thing

```
Thing
-----------------------------------------
-name : String
-----------------------------------------
+name() : String

+equals(thing : Object) : boolean

-----------------------------------------
```