Tomb of the Forgotten Memories

Design Document

Ivan Canet & Sarah Chourouq · Apr 5, 2018 (Apr 5, 2018)

This is the design document of the project of Oriented-Object Conception and Oriented-Object programming of the 2nd semester of the DUT of Computer Science in Bordeaux.

TABLE OF CONTENTS

I. DESIGN

1. Representation of a Room

A room is represented by:

- A description
- Items that can be found when searching
- Items in chests
- Notes
- Its neighbors
- (Enemies are found in rooms but are not part of the rooms; see)

Let's detail all of these. (Note: the room does NOT know its own position in the World, because it's unneeded).

Description

The room's description is a String that is printed to the GUI when the user walks into the room. It can not change, and must be internationalized (thanks to the Translator platform).

Searching a room

A player can use one of its skills to search a Room. A room can only be looted once, which means that the Items do not need to be stored in the Room object (they are generated on the fly when the user searches the room); therefore the Room only stores whether it was already looted or not.

Items in the container

A room can contain a container (read, a chest) that has different items in it. Multiple types of containers exists but they are only differentiated by their description and their capacity.

Notes

Notes are story elements, they only consist of a description and, optionally, an author. The user can find them in rooms, but cannot take them with him. Notes are generated from a list of notes loaded from a file (this allows to easily change the story elements without needing to recompile).

Neighboring rooms

The neighbors of a Room are stored as a Map of Direction and Room, that is, any given Room associates directions where it has a neighbor with that neighbor. This way, you can easily:

- Check if a Room has a neighbor in a specific direction (thanks to `.containsKey`)
- Get the neighboring Room in that direction (thanks to `.get`)

However, this is not the only way to access a Room, as they can be found through the World object, that knows the location of every room.

2. User input

The user will input commands through button pressing. The GUI is split into 4 parts;

- The Player (left side), contains information about the player (health points...)
- The Story (top), contains the data from the game (the enemy's stats, the description of the room...)
- The Actions (bottom center), contains the buttons to interact with the game
- The Map (bottom right), contains a 2D-representation of the floor the player is in. In our case, the Map representation is very important to the gameplay, as it is the only way the player can know if enemies are awaiting him. In a first time, we will implement the map in ASCII-art (alike what the game A Dark Room did), and, if possible, we will change it to a Map drawn with squares and circles at the of the project.

3. Commands

Several commands are necessary to play the game. They are implemented into different fields :

- `REST` - The character gets some sleep and gains more energy in order to restore some of his abilities (strength). Accelerate the AIs speeds for a few turns. The character cannot do anything else while resting.
- `STOP_REST` - The character stops resting and can carry on.

Inventory

- `OPEN/CLOSE` - Open the inventory to view each item owned, their type, the description, their ability and their amount of pods.
- `THROW/DROP` - Throw away an item from the inventory resulting in lowering pods and increasing the loot in the room.
- `GRAB` : Pick up an item found in a room and choose whether or not it needs to be kept.
- `EQUIP_ITEM` : Equip an item from the inventory.
- `USE_ITEM` : Use an item. May be done during a fight.
- `CHOOSE-ITEM` : Select an item from the inventory. May be done while fighting.

Room

- `OPEN/CLOSE DOOR` - Close door.
- `MOVE (NORTH/SOUTH/EAST/WEST)` : Move to a direction.

- `SEARCH` : Search room to find items. Some hidden items may be found depending on the player's search luck.
- `READ_NOTE` : Read a note.

Combat

- `FLEE` : Escape a fight.
- `ATTACK` : Attack the enemy.
- `SEARCH_BODY` : Search body of defeated AIs.
- `USE_ITEM` : Use an item. May be done during a fight.

4. Setting of the story

The game is set in a medieval-like fantasy-world. Because there are no graphics, however, this will only be noticeable through the description of rooms, items, and through the story notes - as with the different item types.

There are four item types: Weapons, Armor, Magicals, and Unique Use.

Armors are used to protect yourself from your enemies

Weapons are used to attack enemies using your stamina: the stronger you are, the more damage you inflict upon your enemies – and vice-versa, the weaker you are, the less damage you inflict.

Magical items are mainly weapons: they are used to attack your enemies by consuming mana (mana increases overtime). Unlike physical items, they do not alter your damage based on the quantity of mana left (nor your stamina, for that matter). This makes magical items the counterpart of physical items, they each have their pros and cons, so you should probably carry both kind with you.

Unique Use items are similar as Physical and Magical weapons, because they can consume mana or reduce your stamina - what makes them special is that they cannot be used twice (that is, they are cleared from your inventory on usage). They can go from spells (inflict damage on your opponent) to potions (regenerate your health points, improve your stamina), explosives... In the code though, they only differ by their description and their effect.

5. Character attributes

5.1. Entity attributes

The characters (called "entities", see glossary) have several attributes:

- Health Points (HP): Can be regenerated using potions (for example). When HP reaches 0, the entity dies (read, is removed from the World)
- Stamina: Decreases when the entity moves, fights... Can be replenished through sleeping
- Mana: The mana the entity is able to use to power magical items

An entity also has:

- A position (Location object)
- The enemy: last targeted enemy (none if none was targeted or if it was defeated)

5.2. Player attributes

The player has several attributes that a regular Entity doesn't have:

- An inventory (a container of items, as seen in a Room)
- The search luck: is improved overtime during playing
- The weight he can carry (used to choose whether the player can take a new Item in their inventory)

6. Optional features

Representation of the map in 2D

- Different AIs
- Different enemy types (kamikazes...)
- Loading JSON through reflexivity (see File/Database)
- AIs can cooperate (one AI 'drives' multiple entities)
- Allow AIs to join a fight (1v* fights)

7. Use-Case scenario

Plain-english scenario of the game:

- The user starts the game
- In the general menu, the user chooses to load a save file
- The user is prompted to choose the file
- The file is loaded and the story is displayed in the state it was left in
- The user changes room (dir. North)
- The user chooses to use an item
- The item is used
- The user saves the game
- The user exits the game

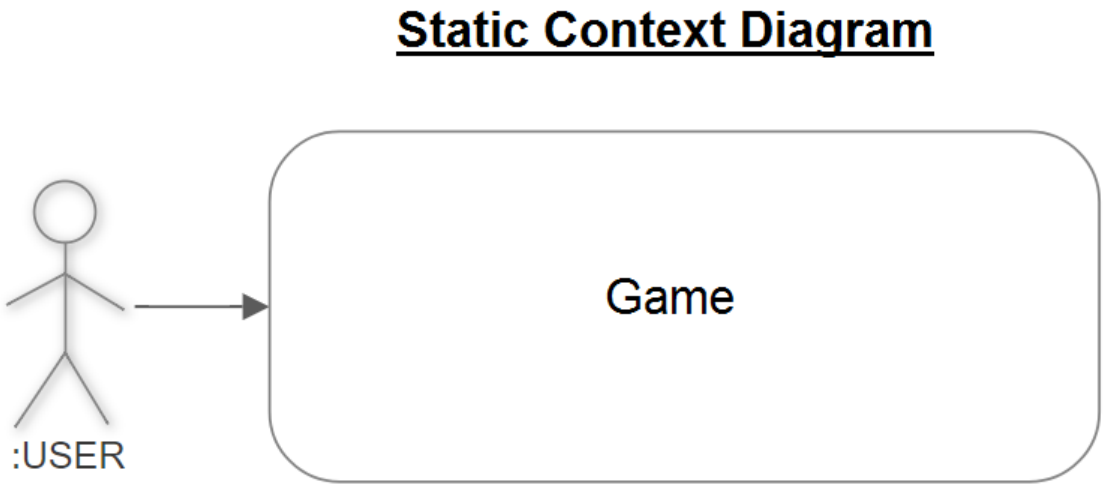
II. FILES & DATABASE

The save files will be stored using the JSON format.

In a first version, the objects will be loaded using a traditional Builder object, and saved with the same object.

In a second version, we'd like to be able to save/load the objects using annotations; the loader would see what the class ‘wants’ and assign it. This is an optional feature, as we do not know if we will be able to do it.

III. STATIC-CONTEXT DIAGRAM



IV. EVENTS

1. External events

Let's order the external events by the context in which they can be sent.

During exploration

- `ARR_Move()`: change room
- `ARR_Rest()`: Accelerates the game speed (the player rests to refill their stamina bar, however this can lead to enemies breaking in)
- `ARR_StopRest()`: (only when resting) Stops resting, resets the game speed
- `ARR_Search()`: (only if the room has not been searched) The user searches the room he's in to find items
- `ARR_OpenChest()`: (only if chest exists) The user opens the chest of the room

During fights

Fighting mode is automatically started when the player is in the same room as any other Entity. It can either end with ARR_Flee() or with the death of either the player or the enemy. On the enemy's death, the player's inventory is opened so the player can take the loot.

- `ARR_Flee()`: Similar to `ARR_Move()`, but consumes more stamina and can fail. On success, stops the fighting mode
- `ARR_Attack()`: Opens the inventory and prompts for an item, then attacks using that item

When looking at the inventory

The player can always open their inventory.

- `ARR_CloseInventory()`
- `ARR_ThrowItem()`: throws an item
- `ARR_TakeItem()`: if comparing inventory with a container, takes an item from the container and puts it into the player's inventory
- `ARR_SelectItem()`: selects an item and closes the inventory

2. Time events

Even though there are actions that happen “overtime” (like regeneration…), there are no time events: whenever the player does an action[1], every object is notified that time has passed; the only exception being “rest mode”, in which the player doesn't act anymore. In “rest mode”, there exists:

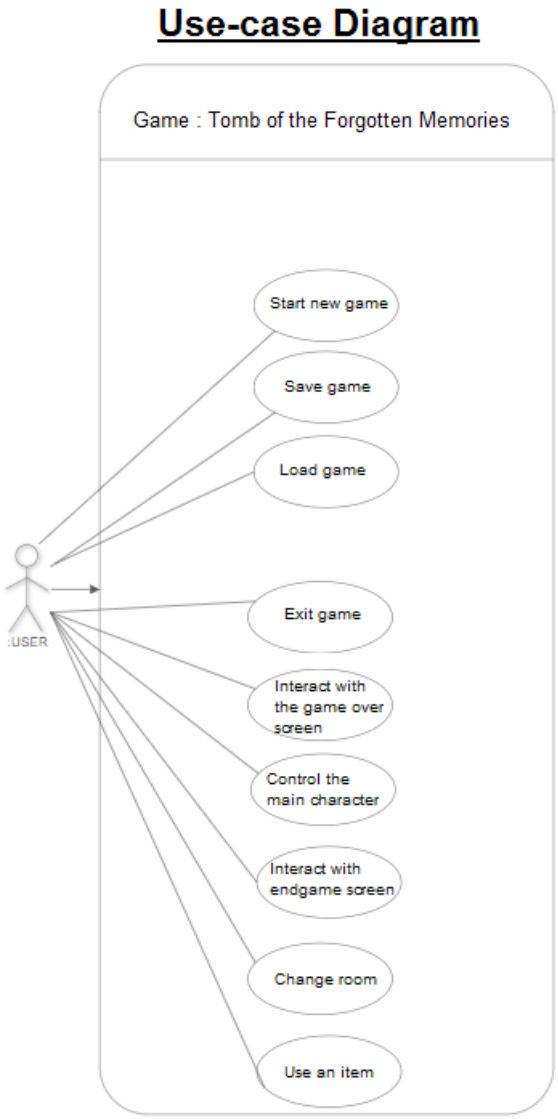
- `TIME_NextTick()`: Updates every objects (entities move, stamina bar replenishes...), the player cannot do anything except `ARR_StopRest()`

3. Result events

- `OUT_Refresh()`: updates the screen
- `OUT_SelectInInventory()`: allows the player to act on its inventory

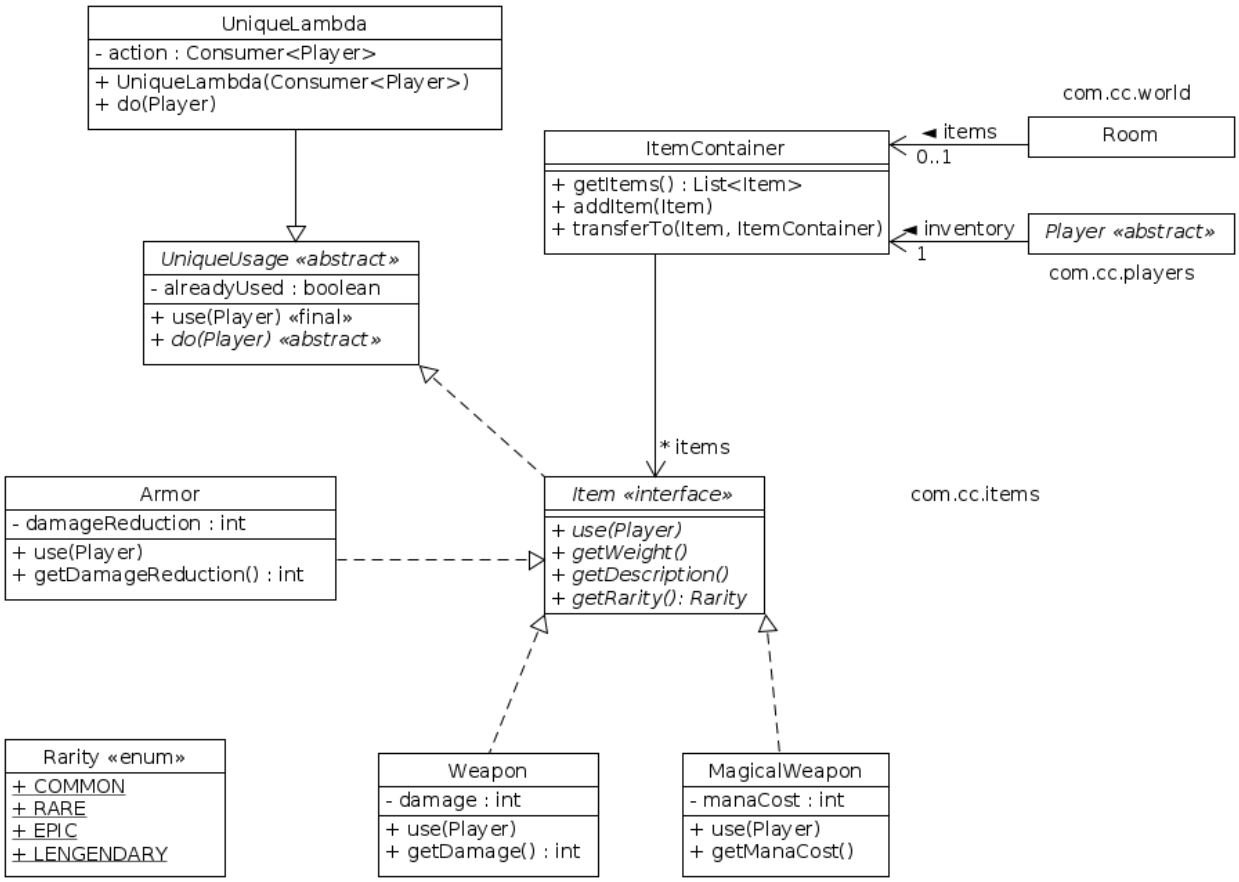
1 Not every interaction qualifies as an action; for example, moving is an action, opening the inventory is not (but using an item is).

V. USE-CASE DIAGRAM

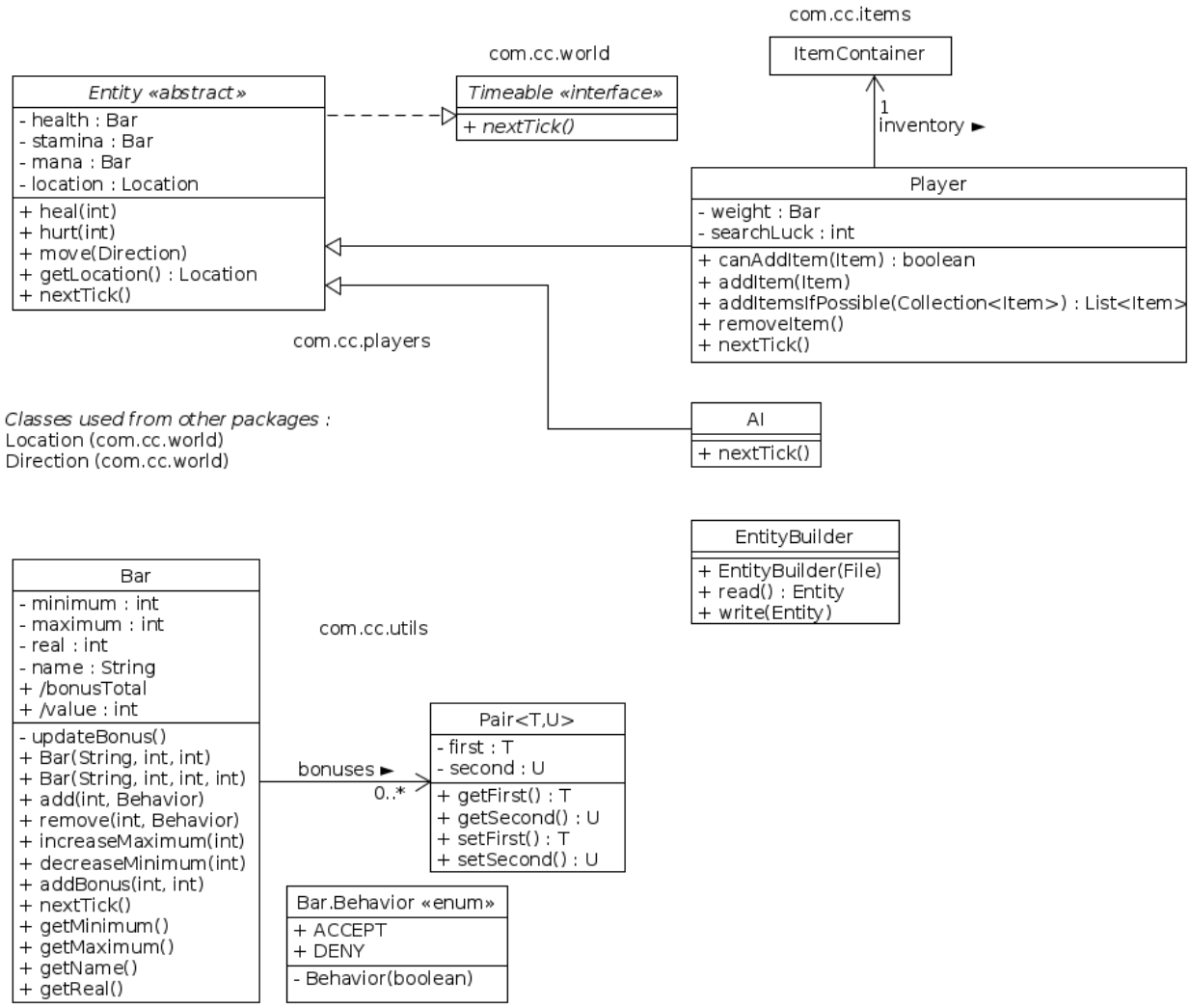


VI. Class Diagram

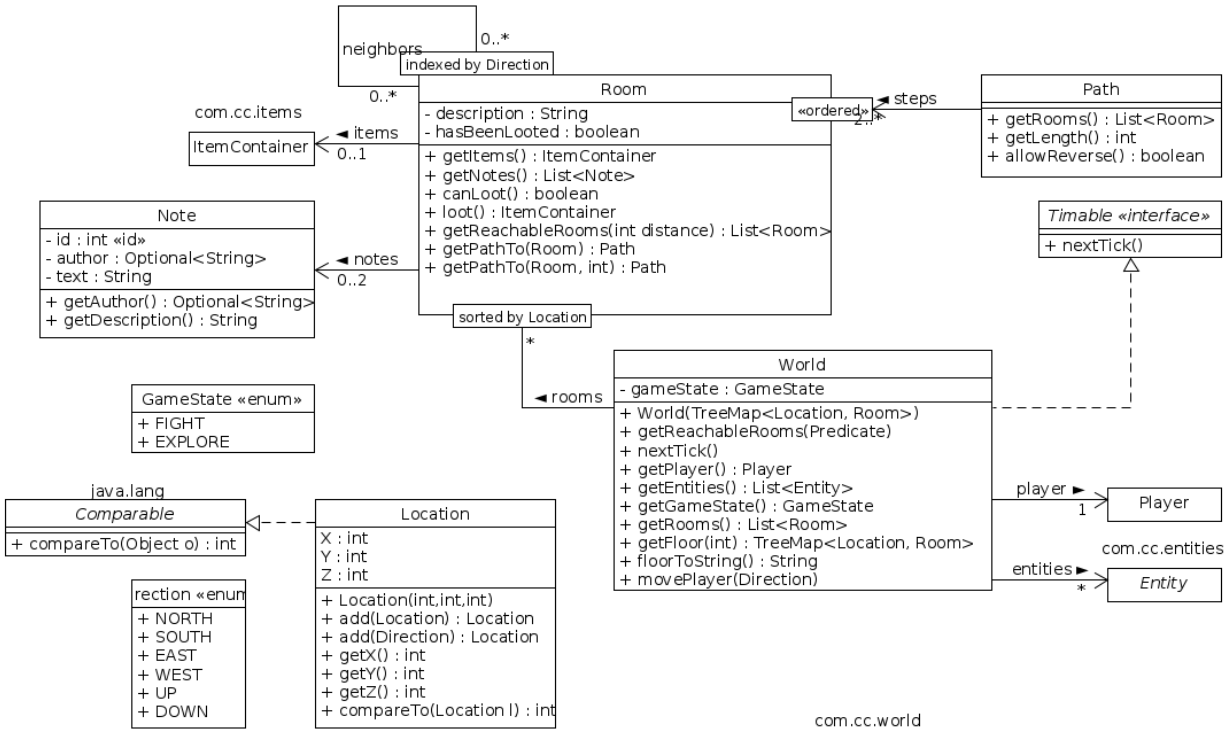
1. Items package



2. Players and Utils packages



3. World package



VII. DATA LEXICON

1. About the storing of the Rooms and about Locations

The rooms are stored in two ways;

- In the World object, that associates every location with a room,
- In each Room object, where you can find it's neighbors.

In the World object, there exists a sorted map of Location and Room, thanks to Location implementing Comparable. This allows the program that draws the Map on the screen to be very efficient, because every Location is already put in the accurate order[2] – the only missing part is to fill the blank spaces.

This is also why the Locations are made the way they are: the three coordinates, X, Y and Z, are:

- X: The value decreases when heading towards the North and increases when heading towards the South,
- Y: The value decreases when heading towards the East and increases when heading towards the West,
- Z: The value decreases when heading towards lower floors and increases when heading for upper floors.

This is made so the values, for a given floor, increase towards the bottom right – the order in which the map is drawn.

VIII. CURRENT PROGRESS

The prototype is aimed at showing the technology behind the game rather than the gameplay.

We have implemented 2 commands to interact with the game:

- `move [up|down|east|west|north|west]`: moves the player
- `exit`

The game is already able to display the map, to prove that complex functionalities are possible. Although the available possibilities are quite limited, the project is already constituted of 16 classes that constitute 943 source lines of code[3] fully documented (at this moment only the more important files are unit-tested).

2 The order is: order first by floor, then by longitude, then by latitude.

3 Excluding empty lines, obviously