Arch1
Cohort 3 Group 5

Members:

Amity Van Rooyen
Cassian Kanhukamwe
Dhruv Madan
Gilda Grimes
Jerry Anish
Matt Ritchie
Oakley Fiddler
Ruby Brown

Architecture

Description

For the project, the architecture was used as a guideline for the software developer to refer to rather than a strict instruction, allowing for the development of the game to run smoothly instead of trying to follow every component to the letter.

To represent and model the software architecture, Unified Modelling Language (UML) was used as it was the most widely used language and supported Java as it supported object-oriented programming and allowed for representation of inheritance. In order to document and display the UML architecture, PlantUML was utilised as it helped display a diagram of the UML code.

PlantUML was useful as it indicated what various classes interacted with or inherited from others. In the diagram shown in Figure 1, a white-headed arrow indicates that the class it is pointing from inherits from the class it is pointing to.

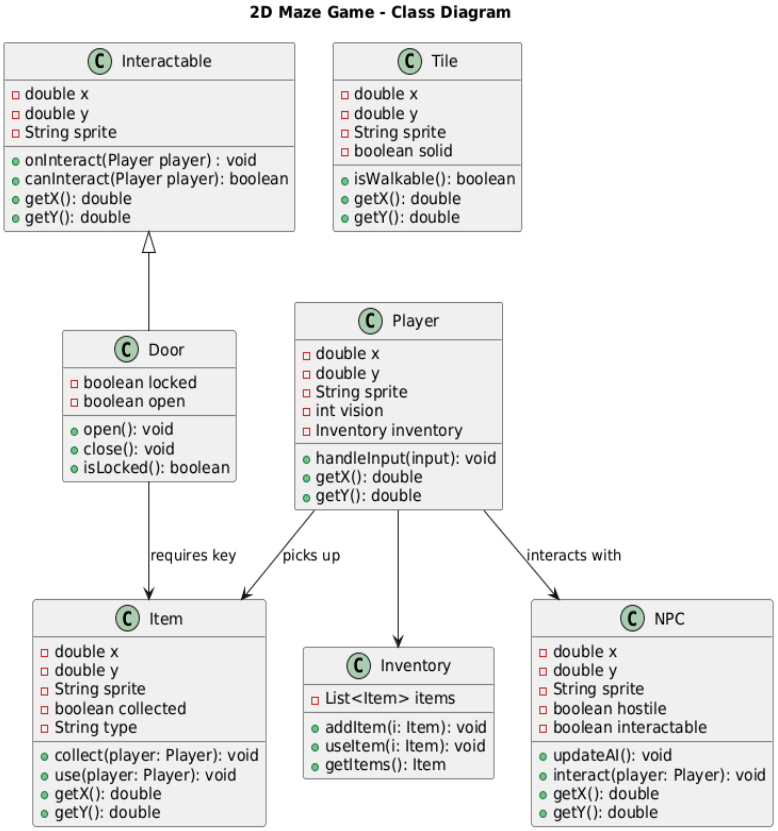
The architecture followed the instructions provided by the customer, i.e. 5 events to hinder or help the user, shown by a key required to open a door, or an NPC that may or may not be friendly.

As there were plans for having multiple interactable objects in the project, we decided to plan for the future by having the door class inherit from the Interactable class. This meant that going forward implementing further interactable objects would be more straightforward. As the game would be having a tile-based map, the tile class was made to represent each tile the player would be walking on. The plan was to have further classes inherit from this tile and to be stored in a 2d array to represent the map of the game.

The player would have its own class to store its location, with a parameter representing how many tiles the player can see. This was going to be used either with a fog of war system, or a zoom adjusted to the vision of the player. The player also had a parameter of an instantiation of the inventory class, allowing us to track what current items the player has. Putting the inventory as its own separate class allows us to implement NPCs having inventories in the future, if the need arises. Although the inventory class is quite simple, only having the functionality of a basic list, there is room for it to be expanded in the future.

The NPC class was created to allow us to create multiple NPCs for the player to interact with. The purpose of the interactable boolean parameter is so that some NPCs are able to be talked to in some sort of dialogue, whereas other wouldn't have any interaction you could do with them, for example if there was an enemy that chased you making it interactable wouldn't be the right decision. Furthermore we have the hostile boolean for if the enemy

would do damage to you upon contact.



Evolution and Adjustments

As mentioned in the previous section, not everything in the initial architecture's UML code was followed to the letter. This may have been due to disagreements with the original design, a problem with over-thinking a given feature or just for simplicity's sake, for example, the variables being of data type "double" was not needed and was changed to type "int". The reason for these simplifications is that the base game was intended to be a very barebones first implementation. Once the final first implementation of the game had been verified and tested, the UML was updated for documentation purposes.

One part of the implementation that was changed was the way the Item class works. After changing the x and y coordinates to int, as previously mentioned, the next thing we did was to add a parameter with a list of possible names. This would be initialised with the names of the classes that inherit from the item class, so that when a new instantiation is created, we can assure that the name provided for it is one of the possible names provided. There are two ways to initialise an item, one with coordinates and one without, as some items may not have a location decided for it on initialisation. We have 4 different classes inheriting from Item which will be developed in the future.