MONASH University

FIT2099: Object Oriented Design and Implementation

Assignment 1: **Design Rationale**

Hoang Phan
Josephine Leye

Introduction

This design rationale will outline the new classes that our team has decided are necessary to fulfill the project requirements and why, as well as how these classes will be implemented into the existing system, and as a result, how it will interact with the existing system to achieve the required functionalities.

New classes

In this section we will outline the new and important classes that will need to be implemented as part of our new system. Some of them will be extended from already existing classes in the game engine package.

Goon and Ninja: Similar to the already existing Grunt class, both the Goon and Ninja class will extend the provided 'Actor' class already in the system and as such will possess the same methods and interactions as the existing Grunt class, the difference being which behaviours will be added from the ActionFactory.

Key: A Key is an extension of the Item class, and is held in the 'Inventory' of any Actor. It can be dropped or picked up by any extension of the Actor class via the DropItemAction and PickUpItemAction. An Actor must have this key in order to enter a Room on the GameMap.

Door: This will act as an obstacle preventing the Player from entering a Room, until a key is used to unlock it. Extended from the existing Ground class. The Door class should be able to check whether an Actor has a Key, as well as communicate with a Room whether it is locked or unlocked.

Room: Room will be an extension of the existing Ground class. It will require the ability to hold an Item (RocketPlan) or an Actor (Doctor Maybe), as well as allow an Actor to enter is the Door is unlocked.

Doctor Maybe: Similar to the Goon and Ninja, DoctorMaybe is an extension of the existing Actor class. He will only possess different Action and FollowBehaviour and will not move around the GameMap. He will possess the Item RocketEngine in his Inventory to start with, which can be dropped upon defeat.

Inventory: This class will store any Items that an Actor currently has. It will require methods to add and remove Items from the inventory and an ArrayList or HashMap which keeps track of each items. It will also require boolean methods for whether or not the Inventory has specific items.

Item: Include Keys, Rocket Plan, Rocket Body and Rocket Engine. Needs a field called "Name" to differentiate each items.

Rocket Pad: A special location that the player must find in order to build the rocket. Extended from the existing Ground class.

Rocket: This is an aggregation of the Rocket Body, Rocket Engine and Rocket Pad. Once built, the Player will have successfully completed the game.

Player: Represents the current user playing the game. Will require methods in order to interact with enemies, NPC as well as the environments.

Q: A unique NPC that the player can interact with. It will be an extension of the existing Actor class and can thus add or remove items from its inventory (receive RocketPlan and give RocketBody). It will require extra methods to interact with the Player such as 'Talk', and check for Items in the Player Inventory. Its Action will be based off the outcome of its interaction with the Player.

By dividing the system into separate classes such as the ones above, we will be able to divide the work amongst the team, so that classes with large dependencies are coded by the same team member.

Interactions between classes

The file FIT2099_Communication_Diagram in our git repository contains 5 different communication diagrams, each outlining a different complex interaction within the new system we are looking to implement. These diagrams are described in the section below.

Diagram 1: Unlocking a Room

This diagram walks through the interaction a Player must go through in order to gain access to a Room. Once the player attempt to pass a Door object, the system must check if their inventory contains a Key item. This will be done via a method in the Inventory class (hasItem(key)). If this method returns true, the system will check if the door is locked through a boolean attribute of the Door class (isLocked). If this returns True, and the Player possesses a Key item, the Door will be unlocked. Only once Door is unlocked can the Player enter Room, and interact with Items and Actors in the Room.

Diagram 2: Defeating an enemy

This diagram walks through the course of action after the defeat of an enemy. Firstly, upon encountering either an enemy, the Player will have some form of attacking action (hurt()), once this action is performed, the Actor class should have a boolean method to check whether it is conscious (isConscious()), if this returns false, the enemy must drop a key from its inventory using dropItemAction(). From this, the Player can pick up the key and add it to its inventory using addItemAction().

Diagram 3: Creating the Rocket

This diagram walks through the steps that must be taken in order to create the Rocket. First the Player must check if their inventory contains the correct items needed to build the rocket, the RocketBody and the Rocket Engine, this will be done via a method in the inventory class (hasItem(RocketBody), hasItem(RocketEngine)). Once these return true, the Player object must have a method to check whether its location on the GameMap is the RocketPad. Only when all of these return true can the player create a Rocket Item.

Diagram 4: Interaction with Q

This diagram walks through the interaction between the Player and Q in order to retrieve the RocketBody in exchange for the RocketPlan. Once a Player encounters Q, they will have menu options supporting the actions to givePlans() and talk(). If Q is spoken to, the system will check if the Player has the RocketPlan in their inventory. If the Player does not have the Rocket Plan in their inventory, Q will display a message saying he can only help if you have the Rocket Plan. If the Player does have Rocket Plans in their inventory, Q will display a message to hand the plans over. The Player will then be presented with the option to hand over the plan. Upon receiving the Rocket Plan, Q will drop the Rocket Body from his Inventory using the dropItemAction(), for the player to pick up using addItemAction().

Diagram 5: Ninja's interaction with player
This diagram walks through the course of action between the Player and the Ninja class once the Player is in range of the Ninja. Before each move the Ninja makes it will have to check if it is in range of the Player (inRange()). If this returns true, the Ninja uses its method getAllowableActions() from the ActionFactory in order to play its turn (PlayTurn()), in the Ninjas case, this should be StunAndMove(). This action has a 50% chance of stunning the player, before moving the Ninja one space away from them. When the Player has to playTurn(), this should result in a skipTurnAction() if the Player has been stunned.