Final Design Rationale Document assignment 3 by Tony Nguyen

**Vehicle**  **& Map Traversal–**

Other design Documents:

* Class Diagram UML is here however
* Class Sequence UML for Traversing the map

Assumptions:

* This will differ from exit since exit is used for movements by 1 space from what I can tell
* That to actually choose to move it would be in the options menu (thus an action)
* That the creation of the vehicle would have the destination of which in mind
* That a vehicle would have to be created on both maps since it is an item

Classes to be Created:

Vehicle (Extends Item) –

* To be the interface and tell us where to move

TraverseAction (Extends Action) –

* To do the actual movement between maps

Other Classes Involved:

Item (Extended)

Action (Extended)

Location (A Variable to be passed)

How It Should Work:

Creating a Vehicle requires the Location at which it would travel to as well as the name of where it is traveling to, along with this it needs all the fields that the Item class requires. This Vehicle Item has the Action of TraverseAction in its allowable actions (only accessible if you are standing on the vehicle) which should be created when the vehicle is created. TraverseAction will move the player/actor and should display an appropriate message depending on the context of moving between the maps.

Design Ideas/Design Rationale:

*Single purpose Principle*

I feel as though this Task can be seen as made easy if placed inside another Class (potentially an extended GameMap). However I think it may be better for me to just create New classes, when thinking about it GameMap just represents the Map and the Things to maintain it (Such as tick method) and so having something that Acts outside the map (Traversing between them) feels like bad practice. Yes, there may be exits but I think I may want to follow Open/Closed Principle as a what if in the future we need additional things for Vehicle (Such as a fuel gauge, a life bar for items or potentially other actors traversing).

*Don’t Repeat yourself principle*

In this task I will try to not need to create new classes that don’t extend anything. I feel as though that the tools to make traversing between maps are already there (moveActor from maps should help). My understanding is that all I need to really do is make it so I have an action (and how to get that action) that can give the player the option to, which is why I think extending Item and Extending Action would be the best choice.

*Reducing Dependencies*

It may include some bad practice since TraverseAction may have some dependencies across packages (Since it needs a Location, GameMap and actor and that’s going across an encapsulation boundary). Problem is that due to the nature of the task being a traversal which requires Classes from engine, the problem is I can only add/change things in the Game package along with trying to follow the first Single Purpose Principle. Therefore, they may be necessary even if it may be bad practice.

*Encapsulation Boundaries*

If I could have made changes to Engine package, I would’ve Placed TraverseAction in it so that I would follow the principle of Grouping related elements together in the same Encapsulation boundary and thus would also follow the principle of reducing dependencies across Encapsulation boundaries. However, I cannot but this is what I would’ve done.

**Mambo Marie –**

Other Design Documents:

* UML class Diagram
* UML sequence diagram for Mambo Marie summoning zombies

Assumptions:

* Mambo Marie herself isn’t a Zombie
* It is random on what edge she spawns at
* Zombies can be spawned on any able place of the map and don’t need to be close to Mambo herself
* There is only 1 MamboMarie at all in the game and she will stop spawning at all if you kill her

Classes to be Created:

NewGameMap (ExtendsGameMap) –

* To perform the calls to do Marie summoning in its tick method

Mambo Marie (Extends ZombieActor) –

* To represent Mambo Marie herself

MamboTeleport –

* This should handle Mambo appearing on the field and disappearing off the field

SummonZombieBehaviour (Extends Behaviour) –

* A behaviour which determines if it is the correct time to summon zombies

SummonZombieAction (Extends Action) –

* The action which should actually summon zombies onto the map

Other Classes Involved:

GameMap (Extended)

ZombieActor (Extended)

Behaviour (Extended)

Action (Extended)

Zombie (Created)

How it should Work:

Inherently the NewGameMap should have a Class called MamboTeleport which would deal with the appearance and disappearance of Mambo marie along with the spawn chances. NewGameMap will only be created so that it Extends GameMap, not modified just extended. MamboTeleport should have a variable which is Mambo and is created at start up. Mambo herself extends ZombieActor and has a new behaviour called SummonZombieBehaviour (Which implements behaviour). That SummonZombieBehaviour should decide when to summon zombies and when it does return a new instance of the SummonZombieAction where that will handle placing Zombies onto the map itself.

Design Ideas/Design Rationale:

*Single purpose principle*

The reason I decided to have so many classes for this MamboMarie is that I feel as though this particular task requires multiple parts that have different purposes that although they are related to each other should be focused on their specific purpose. Therefore, this task should in fact utilise multiple classes as that would follow this principle. Along with this NewGameMap should only be used for extending the Tick() method as to follow this.

*Justifying that what I am doing to GameMap is just extending*

I am only extending the GameMap, not modifiying it. I will not be changing the current implementations of the GameMap, in fact the only thing we are doing to it are adding new methods to extend it and for the Methods of the original we need to add onto we will always call super() at the start to ensure that we are merely extending/adding onto the current implementation rather than changing it. This in part should Keep the principle of Open/Closed where I am only extending rather than modifying.

*Don’t Repeat Yourself principle*

Majority of it seems to just be extending/implementing off of classes that are already there (such as Actor). The only Class that wouldn’t really do this would be MamboTeleport as that may need to be built from the ground up. Other than that, though it would make sense for me to just extend instead of building all of them from the ground up like a mad man. Thus, Mambo extending Actor, SummonZombieBehavior should implement behaviour with the respective action extending action and extending GameMap should be the wisest choice.

*MamboTeleport Encapsulation boundaries and declaring in the tightest scope possible*

This class deals with coordinates, checks, random and potentially a lot of loops. Due to this I feel as though I have to be careful as to where to declare variables. I should check whether the loop has need of this certain variable or if it is required throughout, along with this I should ensure that our Class variables be private or at least protected in the advent that we could inherit from the class in the future.

**Ending the Game –**

Other design Documents:

* UML Class Diagram
* UML Sequence Diagram for ending the game if no humans/zombies

Assumptions:

* In the case that we start the game with no humans we have 1 turn before the game ends (This is due to the fact that this can only be fixed in the world but can’t change that)
* There is only 1 player (Since this is a rogue like this is very likely)
* Only the player can quit

Classes to be Created:

NewGameMap (ExtendsGameMap) –

* This is an extention of gameMap and adds what we need to perform ending the game, it does a call to check the end of the game

QuitAction(Extends Action) –

* This actually removes the player to quit the game

EndGame –

* This checks the end of the game and makes the calls to end the game should a certain condition be met

Other Classes Involved:

Action (Extended)

GameMap (Extended)

Display (Created Once and used forever)

Player (Removes Player to end game)

Actor (Checks which type of actor)

ActorLocations (Passed Variable)

MamboTeleport (Passing a variable to which tells us if mambo is still alive)

How it should work:

(From the humans/zombies winning)

The NewGameMap should have method calls that invoke checking the end of the game. This should be done in the tick method or at least have tick method call another method designated to checking, along with this it should call MamboTeleport to get a Boolean to tell us if Mambo is alive. The class EndGame should Check whether or not the conditions for ending the game have been met or not and if it has should display a message regarding who wins and then create the QuitGame Action to actually end the game (Which removes the player actor to do so). Since it is in tick however, creating a game where only zombies and the player will have 1 turn before ending. This can be fixed by having the check game end method somewhere around the world.run(), but I can’t change world so unfortunately the bug may persist. (I’ve Justified why I can’t do this in the last section of design ideas).

(From the player Quiting)

The player should have an QuitAction added to their allowable actions to make it so the player has a choice of quitting, I feel like this should be added to player before the player has their turn and so I think I should place it in the playTurn method before the actual player picking. The problem here is I am kinda breaking Single Responsibility as playTurn should only be for processing actor turns, not adding them to action. With that said however this is a compromise as the only other place I could’ve put the action this was in world but as stated in the other ending the game condition I cannot change. (Again, this is justified in the last section of design ideas).

Design Ideas/Design Rationale:

*Open/Closed Principle*

Attempt to make it follow the Open to Extending/Closed for modification principle.

The idea is to make it so that in the future we can add more stuff to here (Such as displaying additional stuff or asking for player input after death) but cannot modify what we currently have (cannot change checking ending the game without breaking it in some way). Therefore, I should make a designation for checking the end conditions and another for actually ending the game.

*Potential bad Practice with instanceof (and justification for use)*

May Potentially have to do some bad practice and check the instanceof certain actors (bad practice as it just works against the whole polymorphism as Actor is an abstract class and normally we’d make use Actor as a broad term for its subclasses but in this case it doesn’t make use of it necessarily) but I feel as though it is needed as we may need identification checks of the subclasses themselves and we are actually more interested in the subclasses rather than the actor classes (Although I feel a little iffy about this). In hindsight This may also technically go against Liskov’s Substitution principle, but I still stand by that this is necessary. In fact, if my implementation handles the subclasses in a way that regardless of input and even if there is no response to it technically still follows Liskovs as that technically handles every type of subclass.

*Single Responsibility*

Along with this I must try to make it where It follows the single responsibility principle. Seeing as how I am extending the GameMap the purposes of which don’t align with ending the game, thus it wouldn’t make sense to put ending the game methods in it. Therefore, I should make another class to handle ending the game and only utilise GameMap for its ability for keeping track of every single turn and what to do for each turn, instead of checking the conditions to ending the game and the subsequent actions. EndGame and QuitAction should follow this by EndGame being the Class to check and respond whereas QuitAction will actually do the Ending of the Game.

*Trying to avoid Literals principle*

I have to somehow display who wins and such, I feel like there may be multiple times where the phrase “\_\_\_\_ wins” could come up or something similar is needed through out. Therefore, I should make a string variable with the keyword final to make it so I avoid String related literals.

*Justifying why I don’t make changes to world to fix bug and adding the quit action to the player (Also why it can’t be like NewGameMap)*

The reason why I cannot change world is that I would need to directly change the code of world to allow for these fixes, the code in the world however are set in stone and even calling super() then adding onto won’t work as:

* In the case of Quitting, place it before the playturn() call in processActorTurn() but that can’t be done without modifying
* In the case of the 1 turn bug I would have to put it somewhere before processActorTurn() in run but again’ can’t be done without modifying it

This goes directly against the Open/Closed Principle as this is just straight up needing to change it. With that said this situation differs from NewGameMap because NewGameMap only extends it. It only adds New methods and the ones from the previous that are in NewGameMap always call super() and so are actually just adding onto what was already there, thus NewGameMap follows Open/Closed and World doesn’t.