Assignment 3

Design Rationale

Going to Town

Class added: **ImmovableItem**

Roles and responsibilities:

- Inherits Item class and set portability to false.
- Has a private attribute of type Boolean called canDrop which indicates this item can be dropped from the inventory or not.
- This represents an item that cannot be picked up but has the option to be dropped.
- Has a public method called addAction(Action action) that will add the action to the item's allowable actions.
- Override the getDropAction so that it will return a new DropItemAction if canDrop == true.

Class added: **Helipad**

Roles and responsibilities:

- Inherits ground and is represented by the symbol '*'.
- Has a private attribute of type Actor to store the reference to Player so that only the Player can enter
- Has a private attribute of type List<Item> to store the list of keys that can be used to unlock the Helicopter
- Has public method addKey(Item key) that takes in an Item and store it in the list of keys
- Has public method getKey() that returns an unmodifiable list of keys
- Override the canActorEntor(Actor actor) method so that it only allows the actor to enter if the actor is the Player.
- Override the blockThrownObjects() to return true.
- Override allowableActions(Actor actor, Location location, String direction) and to return a new EnterVehicleAction in its Actions.

Class added: **Helicopter**

Roles and responsibilities:

- Inherits ImmovableItem and is represented by the symbol '^'.

- Has a public method called setDestination(Location location, String string) that will add a new MoveActorAction to its allowable actions where the location will be used as the destination of the actor.

Class added: **EnterVehicleAction**

Roles and responsibilities:

- Has a private attribute of type Helipad that stores that reference to the helipad it will be placed on.
- Has a private attribute of type Location that stores the location of the vehicle or the helicopter.
- Has a private method hasKey(Actor actor) that will check if the actor has the keys required to enter the helicopter and return a Boolean.
- Overrides the execute method so that when this action is executed, it will move the actor to the location of the helicopter if the actor has the required key. Then it will return a string indicating that the actor has successfully unlocked the helicopter.
- Overrides the menuDescription method to return a string for the player to choose to unlock the helicopter.

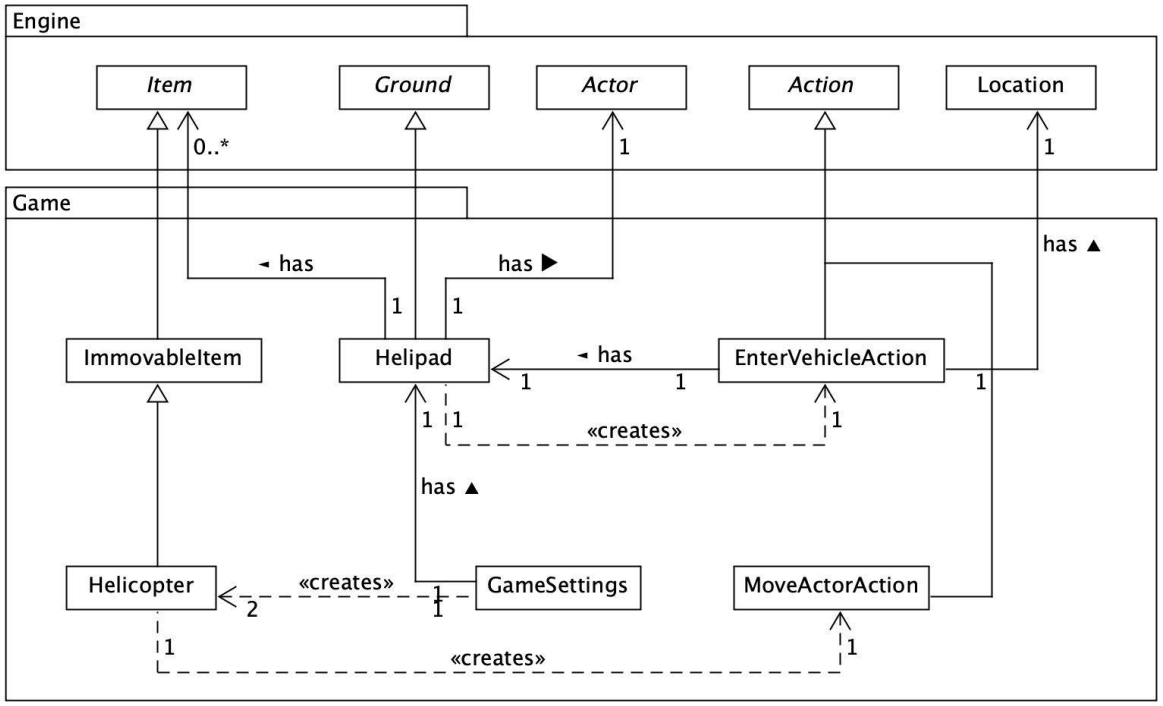
Class added: **GameSettings**

Roles and responsibilities:

- It will replace the original Application class to handle the set-up of the game and instantiate most of the objects required in the game.
- It is created to modularise the set-up process.
- Has a number of private attributes that will store the reference to the objects instantiated in this class.
- Upon creation, it will create the compound map, the town map and put the player in the compound map.
- Has a private method setUpVehicles() that will set the location of the helipad in both compound map and town map, then put a helicopter on top of each of the helipads. It will also create two keys that can be used to unlock the helicopter and then pass the reference to the helipad.
- Has a private method setUpCompoundHumans() that will create a number of humans and place them in random locations within the fence.

- Has a private method setUpCompoundZombies() that will create a number of zombies and put them in the compound map and the reference to them is put into an ArrayList<Zombie> which is a private attribute of the GameSettings class.
- Has a private method setUpKeys() that will put the keys on random zombies, the key will only be dropped when the zombie is dead.
- Has a private method setUpMerchant() that will first create a coin pouch and put it in the Player's inventory. Then, an ImmovableItem coin is created and a new PickUpCoinAction is added to each coin before putting it into each zombie's inventory. Each zombie will carry around 5-8 coins and they will be dropped to the ground when the zombie is dead. Lastly, a shop which is an ImmovableItem is created and a new VisitShopAction is placed into the shop's allowable actions. Both compound map and town map will have one shop each.
- Has a public method called setUpGame that will be called in Application class to set up the game

UML class diagram for Going to Town



How to achieve the required functionality:

1. A new class called GameSettings is created to handle the set-up of the game, replacing Application class. This is done to modularise the set-up of the game, and have each method to be responsible for one function only, therefore adhering to the single-responsibility principle (SRP).
2. Helipad which extends the ground enables the Player to interact with it when the Player is at the location adjacent to the helipad. It stores an actor which is the player

as a private attribute because it only allows the Player to enter. This is done in order to prevent the glitch where other actors step on the location of the helipad which inhibits the Player from transporting between maps. Both compound map and town map have one helipad each.

3. Two helicopters are created, one is put on the helipad in the compound map and the other one is put on the helipad in the town map. setDestination(Location, String) method is called on both helicopters to set the destination. To clarify, the helicopter placed in the compound map will be passed the location of the helipad in town map as the destination so it will add a MoveActorAction that will move the actor to the destination (helipad in town map). Likewise, the same is done to the helicopter placed in the town map with the location of the helipad in the compound map as the destination.
4. As a bonus feature, the functionality of key to unlock the helicopter is added to this system. **For complete documentation, please refer to Bonus Features.
5. To summarise, when the Player interacts with the helipad via EnterVehicleAction and the Player also has the key in its inventory, the Player will be moved to the location of the helipad where there's a helicopter on it. Then. the Player can interact with the helicopter via MoveActorAction which will move the Player to the destination predetermined in the helicopter.

Mambo Marie & Ending the Game

Class added: **NewWorld**

Roles and responsibilities:

- It extends World class to override run(), stillRunning() and endGameMessage() in World class.
- It has a private attribute of type MamboMarie to add MamboMarie into the game later.
- It also has a private attribute of type ArrayList<Location> to store locations where MamboMarie is able to appear at.
- It has 2 private attributes of type int to store the number of humans left and number of zombies in the compound map.
- It has a couple of private attributes of type boolean which are called playerLeft, quit, mamboMarieAppear. PlayerLeft attribute is used to indicate if the player is on the map.

Quit is used to check if the player decides to continue the game. MamboMarieAppear is used to check if Mambo Marie is allowed to appear on the compound map.

- Has a public override method that is called run(), that runs the game if the player is still on the map and there is at least a human alive in the game and also there is at least a zombie or a Mambo Marie on the compound map.
- Has a public override method that is called stillRunning() that checks if the game meets the conditions for the game to run. Conditions = (if the player is still on the map and there is at least a human alive in the game and also there is at least a zombie or a Mambo Marie on the compound map)
- Has a public override method that is called endGameMessage() that prints out String when the game ends.

Class added: **MamboMarie**

Roles and responsibilities:

- It inherits the ZombieActor class and is represented by the symbol 'M' when it is shown on the map.
- It has a private attribute of type ArrayList<Behaviour> that stores SpawnZombieBehaviour and WanderBehaviour which are the behaviours that Mambo Marie has.
- It has an override method called playTurn() which returns actions like spawn zombie action or move actor action if certain conditions are met.

Class added : **SpawnZombieBehaviour**

Roles and responsibilities:

- It is created to return SpawnZombieAction if certain conditions are met.
- It implements Behaviour interface to override a method called getAction().
- It has a private attribute of type int called counter that checks the number of turns needed to spawn zombies.
- Override method, getAction() returns SpawnZombieAction if counter equals to 0.

Class added : **SpawnZombieAction**

Roles and responsibilities:

- It is created to spawn zombies. .
- It extends Action class to override execute() and menuDescription().

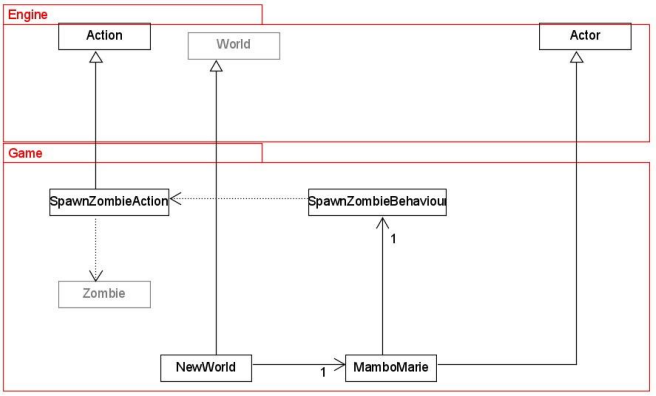
- In the execute(), 5 new zombies appear in random locations on the map. Each new spawned zombie is named as SpawnZombie1 etc.
- In the menuDescription(), it returns a string about what the actor does.

Class added : **TerminateAction**

Roles and responsibilities:

- It is created to terminate the game.
- It has a constructor with a parameter of type NewWorld . With this parameter, a method quit() from NewWorld can be called to stop the game.
- It has an override method called execute(). This method calls quit() to stop the game.

UML diagram for Mambo Marie



How to achieve the required functionality:

- Mambo Marie is a Voodoo priestess and the source of the local zombie epidemic. If she is not currently on the map, she has a 5% chance per turn of appearing. She starts at the edge of the map and wanders randomly. Every 10 turns, she will stop and spend a turn chanting. This will cause five new zombies to appear in random locations on the map. If she is not killed, she will vanish after 30 turns. Mambo Marie will keep coming back until she is killed.

1) Mambo Marie is a Voodoo priestess and the source of the local zombie epidemic.

- A new class called Mambo Marie that extends the ZombieActor class is created.

2) If she is not currently on the map, she has a 5% chance per turn of appearing. She starts at the edge of the map and wanders randomly.

- A new class called NewWorld that extends World class is created to override it's run() method.

- In the override run(), an if statement is added to check the existence of Mambo Marie on the map. For example(if (!gameMapCompound.contains(mamboMarie)) . Inside the if statement, 4 edge locations on the map are created and stored in an attribute of type ArrayList<Location> that is used to generate random locations for Mambo Marie to appear at.

- To let Mambo Marie have a 5% chance of appearing if it's not on the map, the nextInt() is imported from Random class.

3) Every 10 turns, she will stop and spend a turn chanting. This will cause five new zombies to appear in random locations on the map.

- SpawnZombieBehaviour class and SpawnZombie Action class are created. In the SpawnZombieBehaviour class, there is an override method, getAction() returns SpawnZombieAction if the attribute of type int called counter reaches 0 where it is set at 9 initially.
- The counter attribute decreases by 1 in every turn after MamboMarie appears.
- When the counter reaches 0, then Mambo Marie will return SpawnZombieAction in the tenth turn.
- In SpawnZombieAction, there is an override method called execute() that generates random locations on the map for new zombies to be spawned.

4) If she is not killed, she will vanish after 30 turns. Mambo Marie will keep coming back until she is killed.

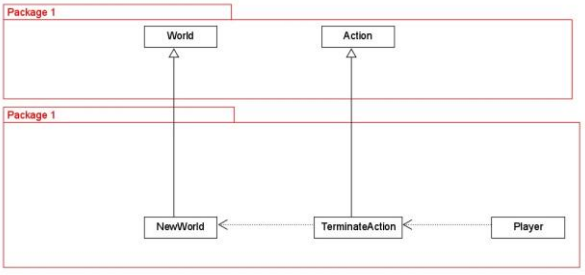
- To achieve this functionality, a NewWorld class that extends World class is created.
- There is an attribute of type int called COUNTER in NewWorld class. It is set to 0 at first.
- In NewWorld class, there is an override method called run() that runs the whole game. In this method, 3 if statements are used. One of the if-statements checks the existence of Mambo Marie, if there is Mambo Marie on the compound map, counter adds 1.

The second if- statement checks the value of the counter, if the counter is 30 which means that MamboMarie has stayed on the map for 30 turns, Mambo Marie will be removed from the map in this turn.

- Mambo Marie is able to appear again on the compound map if she is not killed. So I add a local variable of type boolean called “appear” in run(). It is used to check if the MamboMarie has been spawned before. If it has been spawned before, APPEAR is true. Otherwise, false.
- In NewWorld class, there is a private attribute of type boolean called mamboMarieAppear that checks if the Mambo Marie meets the conditions to reappear on the compound map.
- The last if-statement is to check if the Mambo Marie meets the conditions to reappear on the compound map. To reappear Mambo Marie, it has to appear on the map at least once before and it has not been killed by the player after it’s appearance on the map . To differentiate Mambo Marie from vanishing in the 30th turn or Mambo Marie is killed by the player, the easiest way is to check the existence of Mambo Marie before the 30th turn. If Mambo Marie is killed by the

  player, “mamboMarieAppear” becomes false. It means that Mambo Marie does not stand a chance to appear on the compound map anymore.

UML for Ending Game



How to achieve the required functionality:

A “quit game” option in the menu • A “player loses” ending for when the player is killed, or all the other humans in the compound are killed • A “player wins” ending for when the zombies and Mambo Marie have been wiped out and the compound is safe 1) A “quit game” option in the menu

- A new class that extends Action class called TerminateAction is created. The purpose of this class is to stop the game.
- A quit button which is an immovable item that has an action called TerminateAction. Immovable item is an item that can't be picked up. Therefore, the quit button is added into the player's inventory when the player is first set up in GameSetting class.
- So the player can quit the game whenever he wants because the player has this item in his inventory.

2) A “player loses” ending for when the player is killed, or all the other humans in the compound are killed. A “player wins” ending for when the zombies and Mambo Marie have been wiped out and the compound is safe.

- There is an override method called stillRunning() in NewWorld class. In stillRunning(), there is a for loop to iterate over every actor in actorlocations which is an attribute that stores the location of every actor and the actor.
- There are 2 private attributes of type int called “numberOfHumanLeft” and “numberOfZombieLeft”. They are set to 0 at first. In the for loop, if the actor is Human or Farmer who is still on the compound map, “numberOfHumanLeft” increases by 1. If the actor is Zombie or Mambo Marie who is still on the compound map, “numberOfZombieLeft” increases by 1.
- If the player is not on the map, there is an attribute of type boolean called “playerLeft” that becomes false. It means that the player is no more on the map.
- To end the game with player wins, “numberOfZombieLeft” is 0 and there are still players and at least 1 human on the map.
- To end the game with player loses, “ numberOfHumanLeft” is 0 or playerLeft = false.

Class Added: Ammo Class

Class added: **Ammo**

Roles and responsibilities:

- An abstract class for Ammo which is a type of Item that is used by Gun.
- It extends Item class as it is also a type of item.
- It has a private attribute of type int called ammoCount that tracks the number of ammos it has.
- Has a constructor that calls super class's constructor that initialises the name and displayChar of Ammo. It also randomly sets the amount of ammo from 5- 10.
- Contains 3 methods,
- getAmmoCount, which gets the instance variable of int ammoCount and returns it.
- setAmmoCount, which sets the instance variable of int ammoCount.
- addAmmoCount, which adds the instance variable of int ammoCount.

Class added: **Gun**

Roles and responsibilities:

- An abstract class for Guns which is also a type of weapon.
- It extends WeaponItem class as it is also a type of WeaponItem.
- It has a private attribute of type Ammo called ammoType which is the type of Ammo.
- Has a constructor that calls super class's constructor that initialises the name and displayChar, damage and verb
- Contains a methods called hasAmmo, which checks if a Gun has any ammo.

Class added: **Shotgun**

Roles and responsibilities:

- A class representing Shotgun that extends Super class Guns which is also a type of weapon.
- It extends Gun class as it is also a type of Gun.
- It has a private attribute of type ShotgunAmmo called shotgunAmmo which is the type of Ammo.
- Constructor that calls the super class's constructor to initialises the instance variables name, displayChar , damage and the verb of the Shotgun.

- Contains a methods called getAllowableActions,which returns the allowableActions of the Shotgun.

Class added: **Sniper**

Roles and responsibilities:

- A class representing Shotgun that extends Super class Guns which is also a type of weapon.
- It extends Gun class as it is also a type of Gun.
- It has a private attribute of type SniperAmmo called sniperAmmo which is the type of Ammo.
- Constructor that calls the super class's constructor to initialises the instance variables name, displayChar , damage and the verb of the Sniper.

Class added: **AmmunitionBag**

Roles and responsibilities:

- An immovable item that collects ammo for the player
- It extends ImmovableItem class.
- It has a private attribute of type int called shotgunAmmo that tracks the number of shotgunAmmos it has and also type int called sniperAmmo that tracks the number of sniperAmmos.
- Has a constructor that calls super class's constructor that initialises the name and displayChar and canDrop
- Contains 4 methods,
- addShotgunAmmoCount, which adds the instance variable of int shotgunAmmo
- addSniperAmmoCount, which adds the instance variable of int sniperAmmo
- getShotgunAmmoCount, which returns the instance variable of int shotgunAmmo
- getSniperAmmoCount, which returns the instance variable of int sniperAmmo

Class added: **PickUpAmmoAction**

Roles and responsibilities:

- An action that adds the ammo on the ground to the player's AmmunitionBag then
- remove the ammo from the map
- It extends Action abstract class.
- It has a private attribute of type called Ammo which is the ammo and also AmmunitionBag which is the bag that stores the all the ammos.
- Has a constructor that initialises the Ammo and AmmunitionBag .
- Contains 2 methods,
- Execute method that overrides Super class's execute method and picks up the ammo item and removing it from the map.
- menuDescription method that Overrides Super class's menuDescription method and returns the string to be displayed to the player

Class added: **ShotgunAmmo**

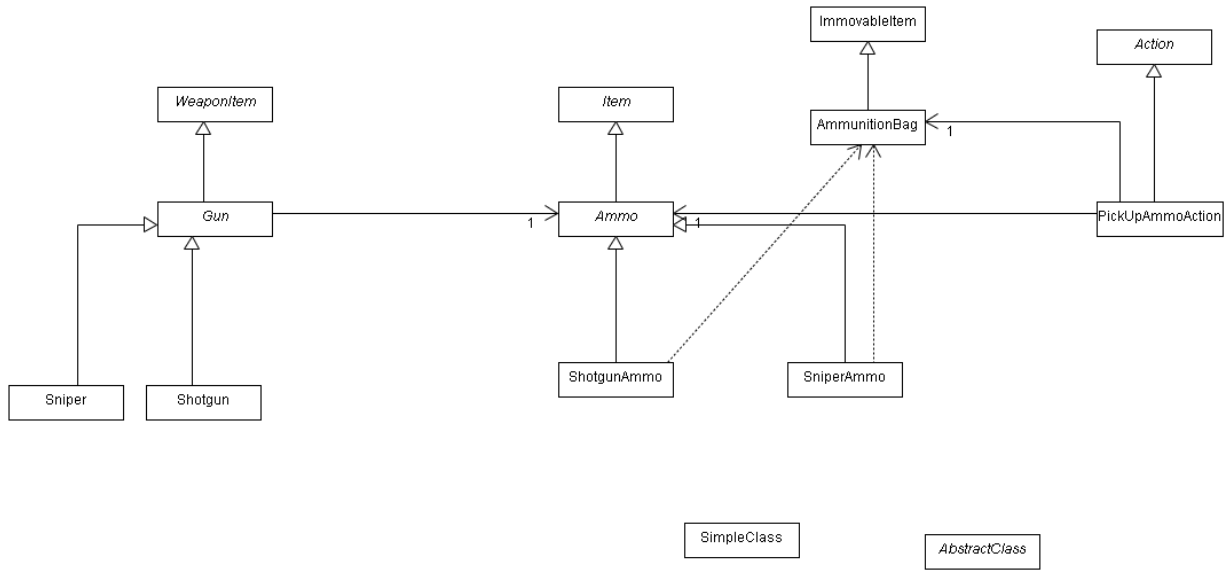
Roles and responsibilities:

- A class for ShotGunAmmo which is a type of Ammo that is required to used Shotgun weapon.
- It extends Ammo class as it is also type of ammo.
- Has a constructor that calls super class’s constructor that initialises the name and displayChar of the ShotgunAmmo item.
- This item can be picked up by player.

Class added: **SniperAmmo**

Roles and responsibilities:

- A class for SniperAmmo which is a type of Ammo that is required to used Sniper weapon.
- It extends Ammo class as it is also type of ammo.
- Has a constructor that calls super class’s constructor that initialises the name and displayChar of the SniperAmmo item.
- This item can be picked up by player.



UML Diagram for Guns and Ammo