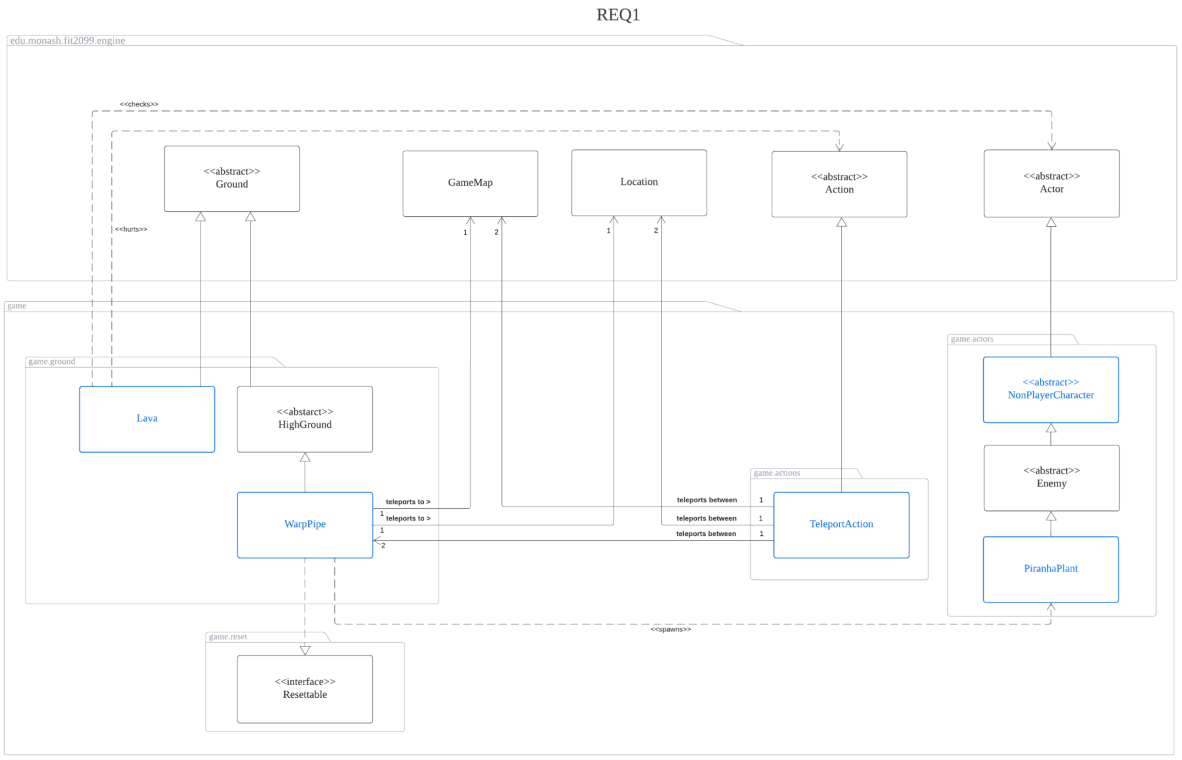
REQ1

Class Diagram:



Requirement:

- New map with lava ground
- Lava ground that inflicts 15 damage per turn to the player standing on it. Enemies cannot enter lava.
- Warp Pipe that would spawn a Piranha Plant in the second round. After Piranha Plant is dead, it allows teleportation from the first map to lava map and vice versa for the player.
- Resetting the game would make warp pipe spawn Piranha Plant again.
- Piranha Plant that cannot move and stays on a warp pipe.
- When an actor teleports from one warp pipe to another in another map, the actor should be able to teleport back from that warp pipe in the second map back to the warp pipe he used. If the actor repeats the action again using another warp pipe, teleporting back from the second map will teleport the actor to the second warp pipe he used.

Design Rationale:

We decided to create a class called **Lava** which extends Ground class. There is one attribute called burnDamage in the class. We make a constructor which would initialise its display character and set burnDamage to 15. This follows **SRP** as by extending ground class, it is very easy to make a lava ground. In the tick method, we would hurt the actor standing on it. Instead of just putting the damage in the tick function, we make burnDamage as an attribute as it brings clearer meaning and it is easier to maintain.

Next, a **PiranhaPlant** class which extends the Enemy class is created. A **WarpPipe** class extending **HighGround** and implementing **Resettable** is made. Being a highground, warp pipe is set to have 100 jump success rate and 0 fall damage. If the actor is in invincible state, it will destroy warp pipe upon jumping onto it. There are 4 attributes in the WarpPipe class, including destinationMap, destinationLocation, destinationWarpPipe and boolean hasSpawned. In the requirement, we need to keep track of the warp pipe, location and map where the actor teleports from so when the actor teleports back from the second map to the first map, it can teleport to the previous warp pipe. In our design, we decided to record this information in the warp pipe using the 3 attributes destinationMap, destinationLocation and destinationWarpPipe. The hasSpawned attribute is used to track if the PiranhaPlant has been spawned as we can only spawn PiranhaPlant in the second round. During the second round and every time the game gets reseted, hasSpawned will be set to false and it will spawn again on warp pipe. Player needs to kill PiranhaPlant in order to stand on the warp pipe and teleport. In the allowableActions() method, it will first call `super.allowableActions()` from highGround, this follows **DRY** because we do not need to repeat the same code for all highGround. Next, it will check if a player is standing on it and add a new **teleportAction** to the second map, information of 3 destination details will be passed in. We make one 0-parameter constructor and a 3-parameter constructor which have 3 destination attributes for warp pipe class. This follows the principle that **classes should be responsible for their own properties.** The 3 destination information are closely related and change together, so they are placed together in the constructor and in passing parameters of **teleportAction**. **WarpPipe** class is also responsible for spawning the warpPipe thus it has hasSpawned attribute to keep track of the spawning.

To enable teleportation, we added a **TeleportAction** class extending Action class. The extension follows **OCP** as adding a new action is very easy without affecting the code of other actions. There are 6 attributes in the class, destinationMap, destinationLocation, destinationWarpPipe, sourceLocation, sourceWarpPipe, sourceMap. We make a constructor which accepts and initialises these 3 attributes. In the `execute()` method, we will first check if the destination location has an actor. If there is, it will be PiranhaPlant in the current design and we will destroy it by removing it from the map. Then, we would move the actor to the destinationLocation, which is on a warp pipe. After that, we would update the destinationWarpPipe, destinationLocation and destinationWarpPipe attributes of the destination warpPipe to the ones the actor teleports from, which are the 3 source information. This design follows the principle that **classes should be responsible for their own properties.** The 3 attributes of the destination warp pipe are updated together using the 3 source information attributes. **SRP** applies here too. The TeleportAction class has only 1 feature, teleporting actors from a warpPipe in a map to another. No excess unnecessary features are added here.

To create a new map with lava ground, we make a lavaGroundFactory in application class. We then make new maps filled with a lot of lava grounds. Next, we initialise a few warp pipes in the first map in random locations using the 3-parameters constructor in warp pipe class. In this case, all warp pipes in the first map will allow teleportation to the only warp pipe in the top left corner of the lava map. After the first teleportation, the warp pipe in lava map will record the information the actor teleports from. Thus, the system works well. Instead of initialising the 3 destination attributes in warp pipe class which requires a static method and static variables for the initial values, we decided that it is not good design. So we do the work to initialise the warp pipe and the 3 attributes in application class. This follows the principle of **avoiding excessive use of literals.**