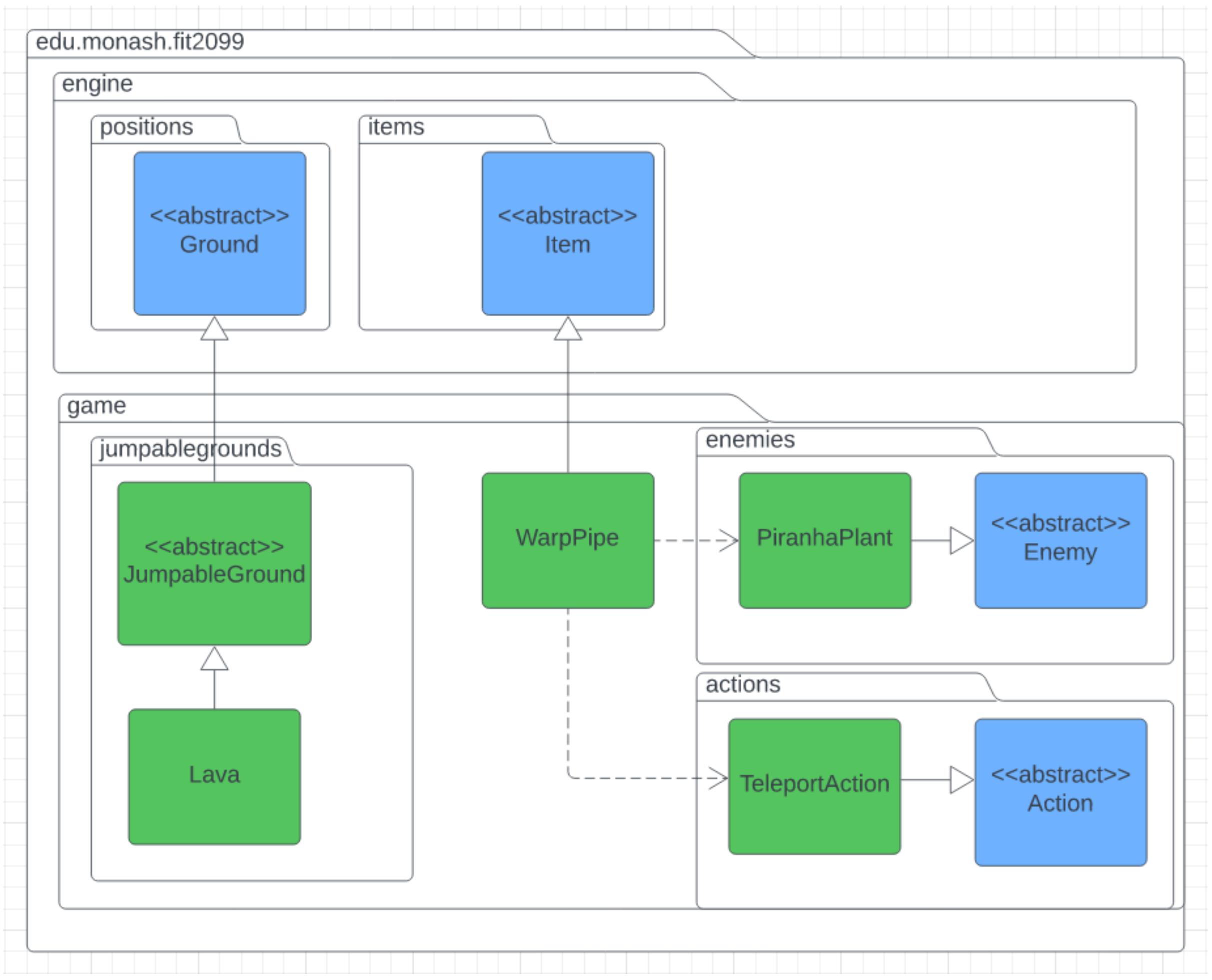
FIT2099 ASSIGNMENT 3

By Joel Atta, Jack Patton & Yepu Hou

Class diagram details:
Blue classes are classes that, as of starting that specific requirement, already existed in the project.
Conversely, green classes are classes that did not exist before starting that requirement and were added during it. Also, pre-existing classes that had their class header modified are green, such as when a non-abstract class that already existed is made abstract.
This means when a class is first created it will be in green, but if it is referenced in another UML diagram in a later requirement it will be in blue. Therefore, if you see a new class and don't understand its purpose, you can scroll up until you find the requirement in which that class is green, where it will be explained.

Req 1: Lava Zone

Class Diagram:



Design Rationale:

In similar roguelike games, the player is typically queried before they commit a dangerous action in order to ensure it wasn’t a mis inputted command, e.g. “Really jump into lava? Y/N”. I didn’t want the player to accidentally walk into lava, especially as we don’t have an inspect command that can be used to describe what the ASCII character selected actually is, meaning a player might not know that L means lava is there and as such step onto it without meaning to. While this confirmation of stepping into lava would have taken more effort than it was worth for something outside the assignment specifications, I decided to instead make Lava a child of HighGround, meaning the player would have to deliberately select a jump action jumping into Lava and could not simply walk into it. However it didn’t make much sense for something lower down to extend HighGround, so I renamed HighGround to JumpableGround. This allows for tiles that need to be jumped to that aren’t necessarily above the player, such as jumping down a pit.

Initially, the WarpPipe class was a Ground object, specifically a JumpableGround. While the requirements mentioned using a JumpAction to get on top of the WarpPipe, they didn’t mention a success chance or damage taken on a failed jump. This implied to me that Mario has a 100% chance of jumping onto the WarpPipe, so I instead decided to allow Mario to simply walk onto the pipe instead of jumping. This also allowed me to easily fix an issue where I couldn’t create a move action targeting the 2nd map - both aspects were addressed by turning WarpPipe into a child of Item instead of JumpableGround.

Req 2: More Allies and Enemies!

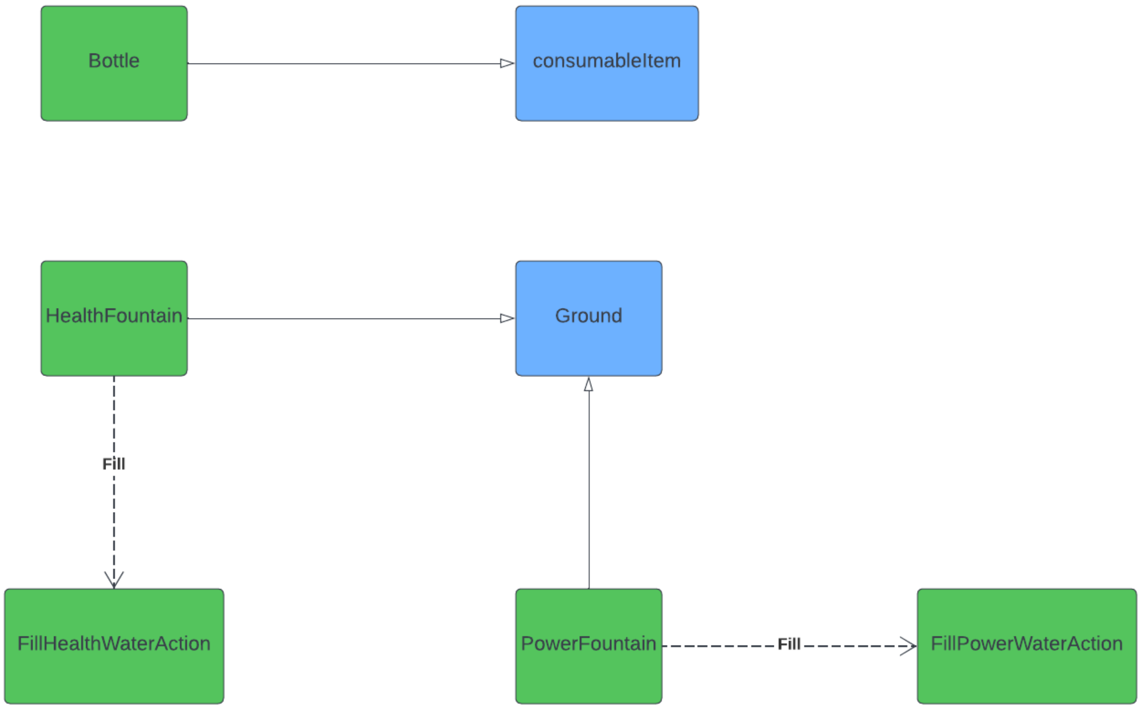
Class Diagram:

Sequence Diagram:

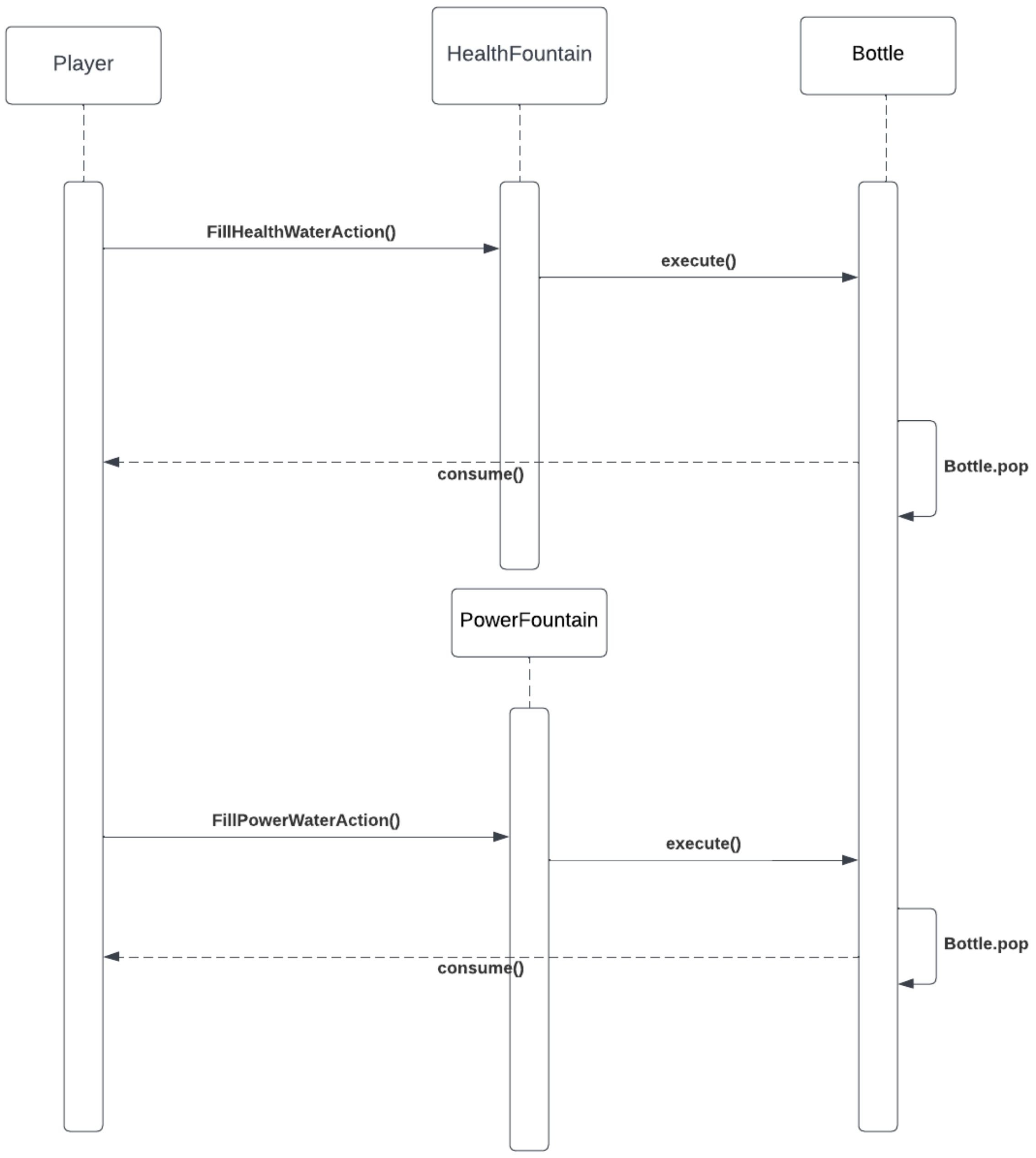
Design Rationale:

Req 3: Magical Fountain

Class Diagram:



Sequence Diagram:



Design Rationale:

The design for requirement three is having two fountains which are health fountain and power fountain. This satisfies the singularity purpose. A bottle class will be initialised as a stack so that every time we fill up the water from different fountains the water will be pushed to that bottle as a string. The bottle is extending from the consumable item this means that in the consume action it will have an if-else statement which will check whether the next element in the bottle is either power water or health water, and will give specific power according to the water that the player has consumed. There will also be two different types of fill water actions one to fill the health water and one to fill the power water. Every time this is called it will push either health water or power water inside the bottle.

Classes' purpose

HealthFountain	A fountain which extends from the ground that has a fill health water action attached to it.
PowerFountain	A fountain which extends from the ground that has a fill power water action attached to it.
Bottle	A bottle extends to the consumable items which are implemented as a stack.
FillHealthWaterAction	When the Mario goes near to health fountain he will have an action to fill up different types of water and it will be pushed in to a bottle like a stack.
FillPowerWaterAction	When the Mario goes near to power fountain he will have an action to fill up different types of water and it will be pushed in to a bottle like a stack.

Req 4: Flowers

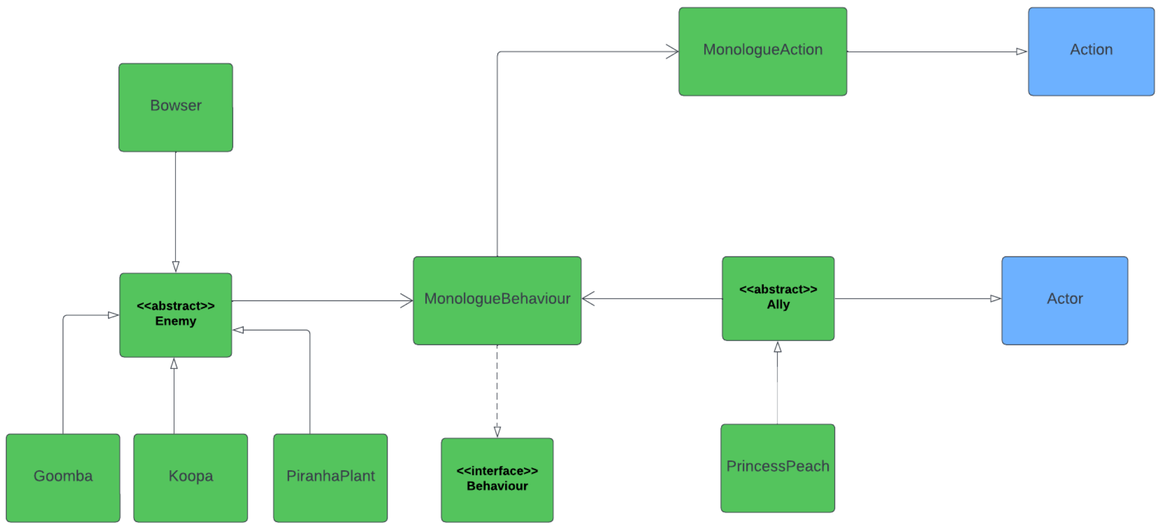
Class Diagram:

Sequence Diagram:

Design Rationale:

Req 5: Speaking

Class Diagram:



Sequence Diagram:

Design Rationale: