Design Rationale
Newly created/modified classes with new relationships

FireFlower
Extends the ConsumableItems class. Allows actors to consume it to have fire breathing capabilities

Tree
Slightly modified to spawn FireFlowers

Fire
Extends Item class. Originally we implemented it to extend Ground class, but by doing so we need to keep track of the ground overwritten, to spawn back the ground overwritten after the fire has dissipated after 3 ticks, so we avoided association between Fire and Ground.

DefaultAttackAction
Extends AttackAction. The normal attack done by either the player or most of the enemies. Adheres to DRY principle

FireAttackAction
Extends AttackAction. The fire attack done by Bowser or when the player has consumed a FireFlower that will launch a Fire at the target's location. Adheres to DRY principle

AttackBehaviour
Slightly modified to check if actors have fire breathing capabilities, if so then launch a fire attack, else default attack
AttackAction
Modified to be an abstract class. In assignment2, we did not foresee that there would be different types of attacks available. Back then, our implementation was split into 2 parts: how to hurt the enemy, and what to do if the enemy is dead after hurting it. We know that there is only one way that an actor can die, which is when it is not conscious. We also know what happens when it dies, which is to drop all its items and remove them from the map. The only thing that differs between different types of attacks is the way the attack is carried out. For example, normal attacks just hurt the target, whereas fire attack hurts the target and launches a fire at them. Hence, AttackAction’s execute method just has the implementation to check for when the target is not conscious, drop all its items and remove them from the map. This way, we don’t violate the Open-Closed Principle, because different types of attacks can just implement different logic on how to hurt the enemy, and use AttackAction’s execute method to implement dropping items when the enemy is dead and removing them from the map without changing anything from AttackAction’s execute method. Also, since we are not lumping up all different types of attack into just one single AttackAction class, it also adheres to Single Responsibility Principle, since each child class of AttackAction only has the responsibility of implementing how to hurt the enemy, rather than one AttackAction that needs to check all sorts of logic to attack enemies in different ways.

“For every growing stage of the tree (Sprout -> Sapling and Sapling -> Mature), it has a 50% chance to spawn (or grow) a Fire Flower on its location”
In Tree class, we can create a method called attemptSpawnAFireFlower which if lucky will spawn a FireFlower at its current location. Sprout and Sapling class can use this method in their tick method if they are growing to the next phase to spawn a FireFlower if lucky.

“Mario can consume this fire flower to use Fire Attack.”

Since player can consume it, we made FireFlower extend ConsumableItem. In FireFlower’s consume method, we make the actor have the capability of fire breathing.

“Once the actor consumes the Fire Flower, it can use fire attack action on the enemy. Attacking will drop a fire `v` at the target's ground.”
In the enemy's allowableActions, if the player has the capability of fire breathing, launch a fire attack. In FireAttackAction, launch a fire at the target's location.

“This "fire attack" effect will last for 20 turns”
FireFlower’s ticker is set to 20, and will decrease when consumed, by 1 in tick method

“The fire will stay on the ground for three turns”
Fire’s ticker is set to 3, and will decrease every tick by 1

“The fire will deal 20 damage per turn. “
Can utilize Fire’s tick method. If there is currently an actor standing on it, hurt it by 20 hp.