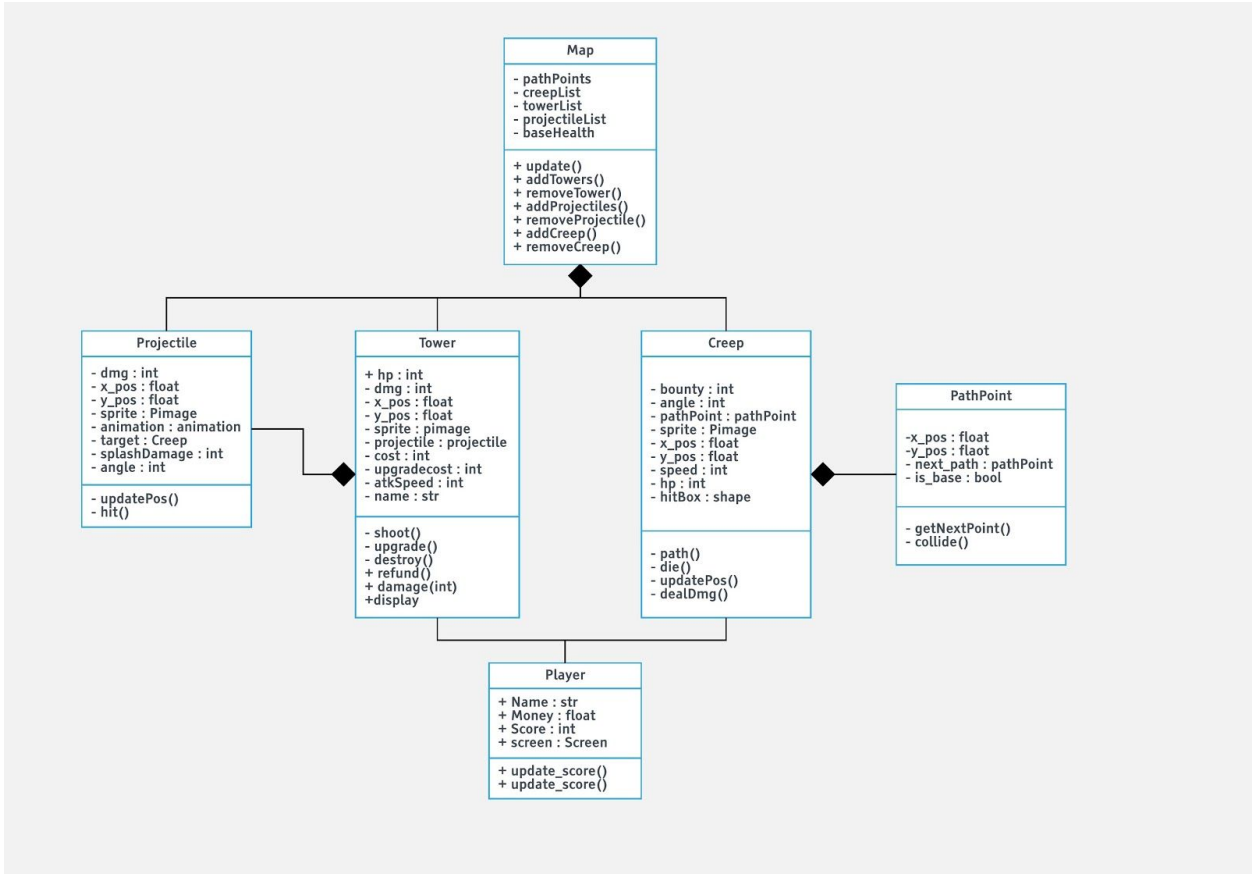
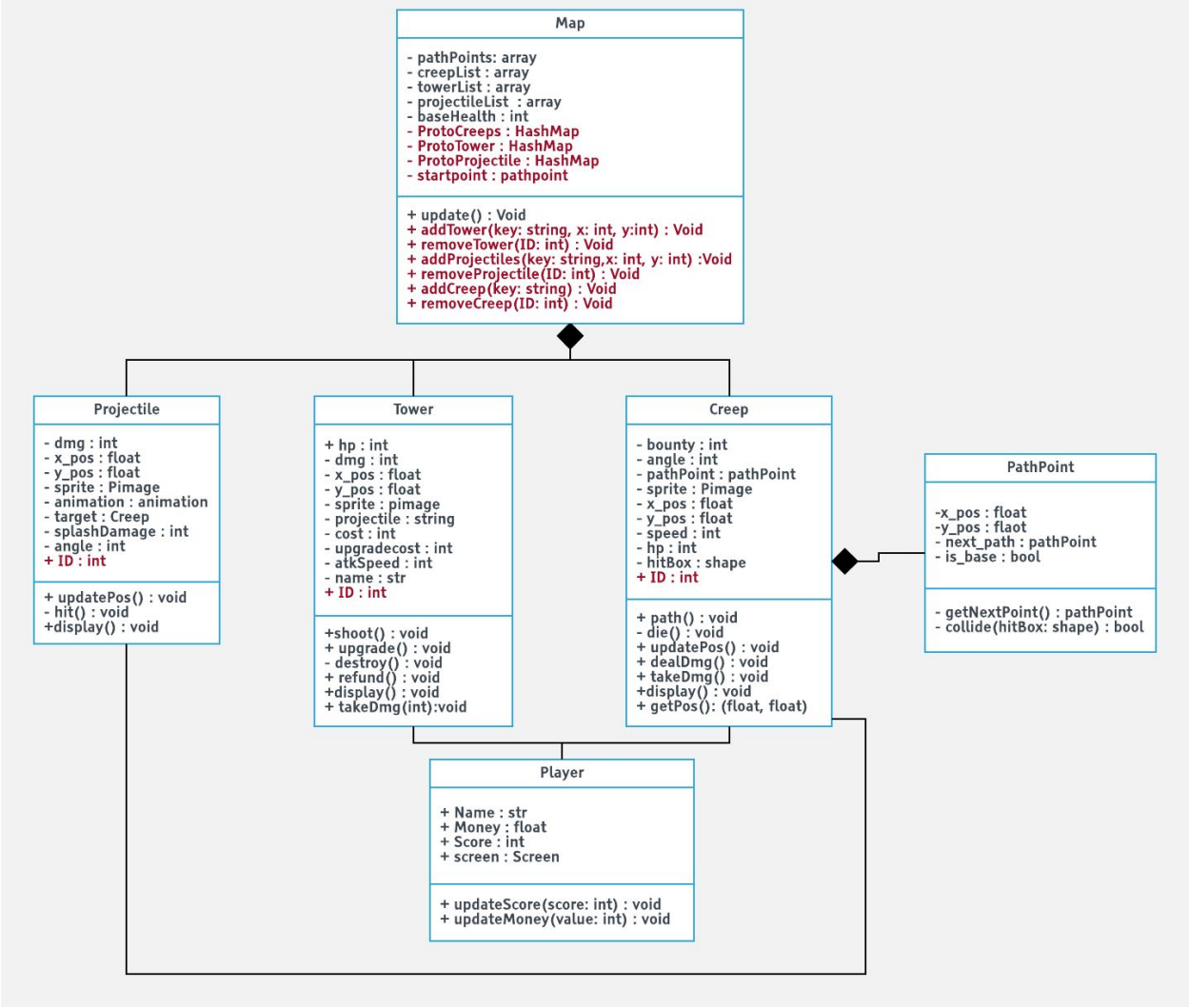

Map
- pathPoints
- creepList
- towerList
- projectileList
- baseHealth
+ update()
+ addTowers()
+ removeTower()
+ addProjectiles()
+ removeProjectile()
+ addCreep()
+ removeCreep()
Projectile
- dmg : int
- x_pos : float
- y_pos : float
- sprite : Pimage
- animation : animation
- target : Creep
- splashDamage : int
- angle : int
- updatePos()
- hit()
Tower
+ hp : int
- dmg : int
- x_pos : float
- y_pos : float
- sprite : pimage
- projectile : projectile
- cost : int
- upgradecost : int
- atkSpeed : int
- name : str
- shoot()
- upgrade()
- destroy()
+ refund()
+ damage(int)
+display
Creep
- bounty : int
- angle : int
- pathPoint : pathPoint
- sprite : Pimage
- x_pos : float
- y_pos : float
- speed : int
- hp : int
- hitBox : shape
- path()
- die()
- updatePos()
- dealDmg()
PathPoint
-x_pos : float
-y_pos : flaot
- next_path : pathPoint
- is_base : bool
- getNextPoint()
- collide()
Player
+ Name : str
+ Money : float
+ Score : int
+ screen : Screen
+ update_score()
+ update_score()




**Updates between versions:**

We added all of the parameters and return types required for the class methods, as well as updated them to reflect if they should be public or private. The main changes we made will be explained below. We were unsure what was meant by the feedback on the original that we did not have enough classes to complete the use cases, there was no direction as to what use cases were not clear as to how they could be completed using the classes we already have.

**Adding the Prototype design pattern:**

Because of the large number of different types of creeps / towers we are going to have, instead of cluttering our codebase with subclasses for each creep, which will only contain different values for health, speed, and other instance variables; we decided to use the prototype design pattern and have a preset list of creeps that can be instantiated from. We are storing a

hashmap in the Map which will contain all the different creeps as objects, which we can reference via key (name) and clone from whenever we want to create a new creep or tower. From there, we can give the clone an x, y position and add it to the TowerList or CreepList and our code will know to render it on the next pass through.

**Fixing Collide from PathPoint:**

In our original design the collide() function within the PathPoint class the collide method was not very well thought out. We have updated it so that any object could call the function to see if they have collided with the PathPoint. This is mostly going to be used by instances of Creep that use the PathPoints to follow their path to the end. It will return a boolean value depending on if the hitbox passed in in the argument contains the PathPoint. Before we had the idea that it would handle some of the creep pathing, but that would violate the principle of single responsibility..

**How a tower would target a creep, create a projectile, and the projectile would path to and kill a creep:**

Processing works on a frame by frame event loop, and so we need functions to be run once per frame and update the locations and such of all of the objects. When a tower is ready to shoot at a creep, it will search through the creepList in map, and target the closest one to the end that is within the towers range. Once it has a target, it with spawn a projectile using the projectile prototype. Every frame the projectile will update its position towards the current position of the creep. Once it updates it position and its hitbox collides with the creeps, it will call the hit() method within projectile. That method will call dealDmg on the creep with the damage value stored in the projectile. If that damage is enough to kill the creep, the creep will remove itself from the creep list and delete itself.