Team: 13 Hacker Defense
Names: Jake Mitchell, Ryan Craig, Phil Leonowens

1)

Implemented

Key: BR = Business Requirement
UR = User Requirements
F = Functional Requirements
NF = Non Functional Requirements
- = Part 2 Requirements
+ = Added Requirements

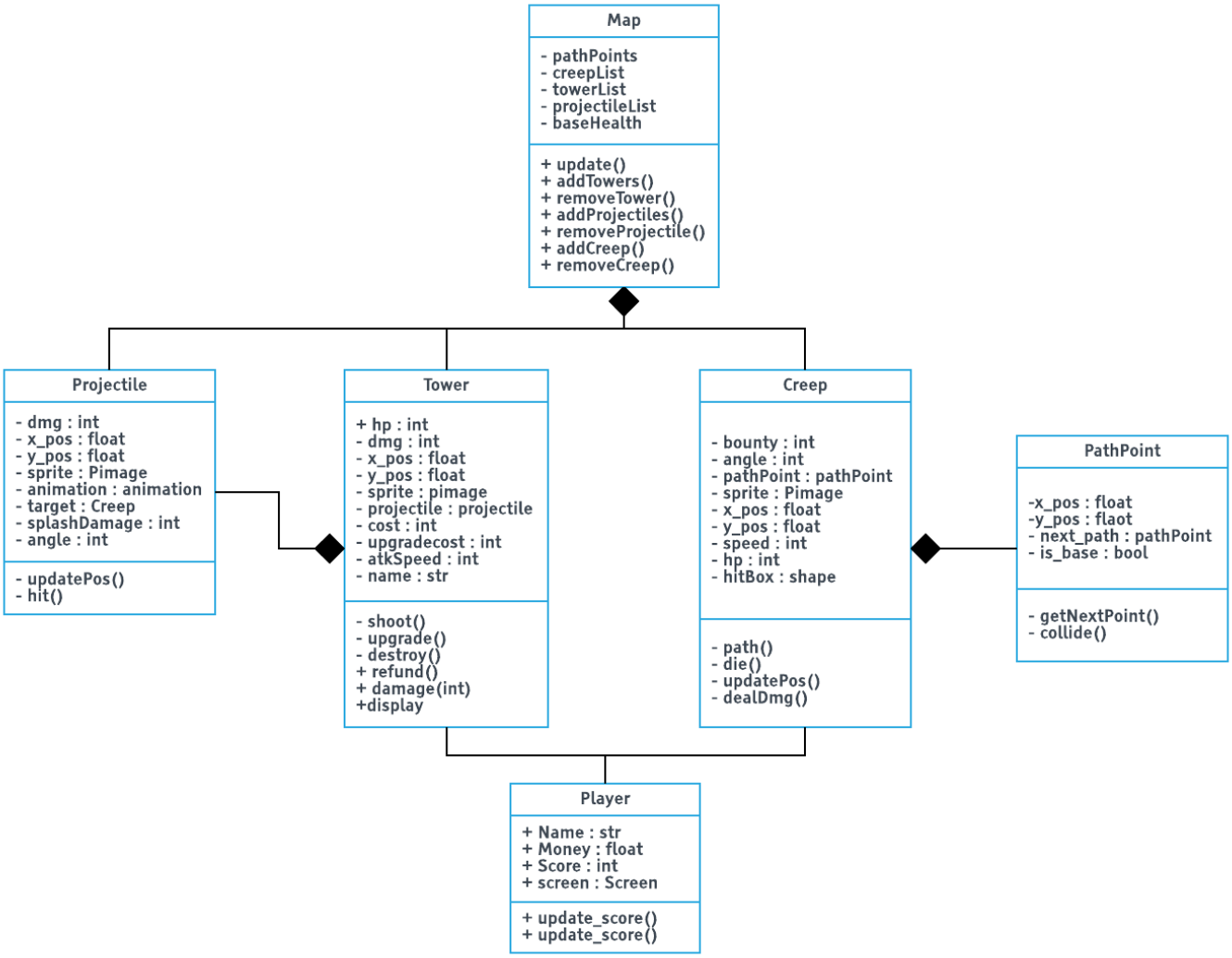
ID	Requirements	Success Metric	Priority
BR-0	Create a desirable game that people will play and that is engaging	Successful feedback from user testing	Medium
BR-1	The game is an appropriate difficulty	Our user testers are able to beat the game, but not all of them win	Medium
UR-2	A variety of towers need to be able to be constructed by a player	We have multiple varieties of towers that a user can place.	High
F-3	A variety of enemies need to spawn with a number of different characteristics abilities and properties	Five different kinds of enemies, with varying speeds and skills exist	Critical
F-4	The game can be lost	There are losing conditions, whether it's a base HP, or a certain number of creeps reaching base	Critical
UR-5	Users must be able to see the projectiles in flight	Projectiles fly smoothly towards their targets and perform an animation on hit	High
F-6	Creeps can be damaged and killed by projectiles	Projectiles can interact with creeps and deal damage to	High

		them	
NF-7	Deliverable will be small enough to distribute	Can be hosted online successfully in a manner that the average user can download within 120 seconds	Medium
NF - 8	Resolutions up to 720p are supported	Adjustable resolutions up to 720p work and render well and legibly	Medium
NF - 9	Runs at a steady 30fps	Our computers are able to run the game at 30 frames per second without many lag spikes	High
UR+10	User is able to enter a “name”	User’s name can be entered and changed, while being displayed	Medium
UR+11	User is able to submit their score to be considered for a high score	Users score can be stored and called up at a later time	Medium
UR+12	User is able to see the high scores	High score screen shows the top 5 scores and their respective names	Low
F+13	There are multiple rounds of enemies	Multiple waves of enemies spawn and steadily get tougher	Low

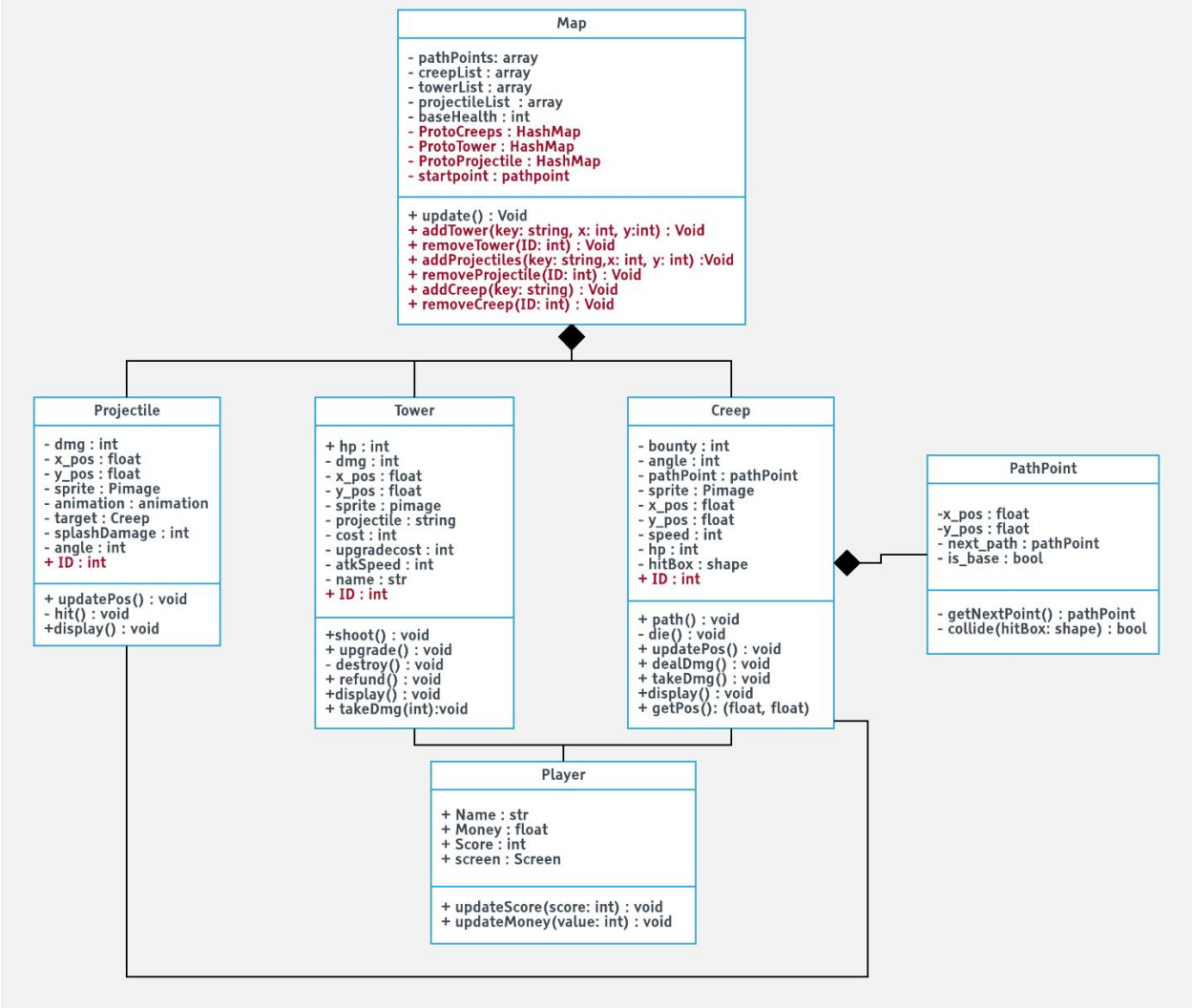
1) **Unimplemented From Part 2**

ID	**Requirements**	**Success Metric**	**Priority**
UR-3	Towers can be upgraded	¾ of the towers have at least one upgrade path	low

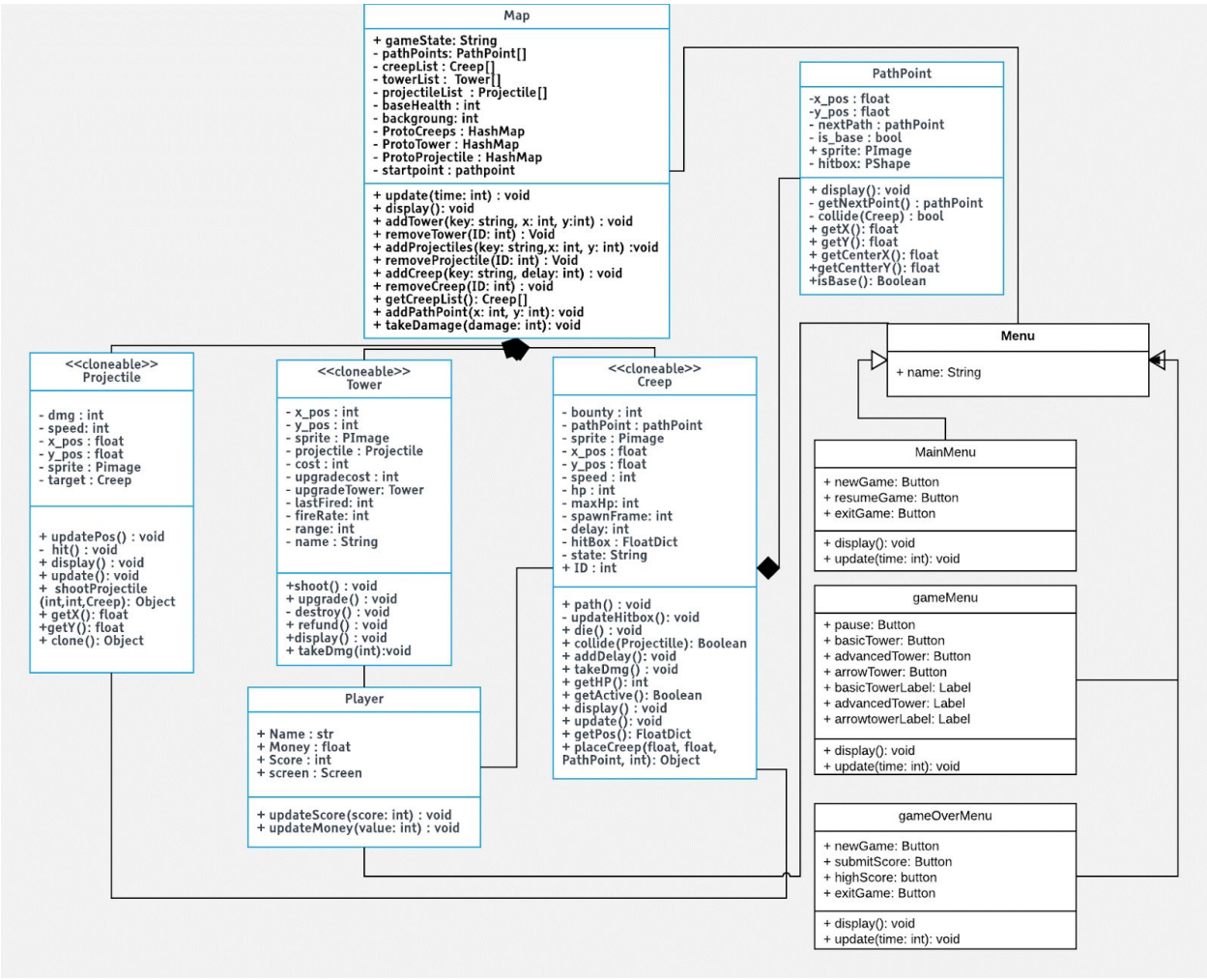
2) Part 2 Class Diagram



Part 3 Class Diagram: (red is prototype additions)



Final Class Diagram

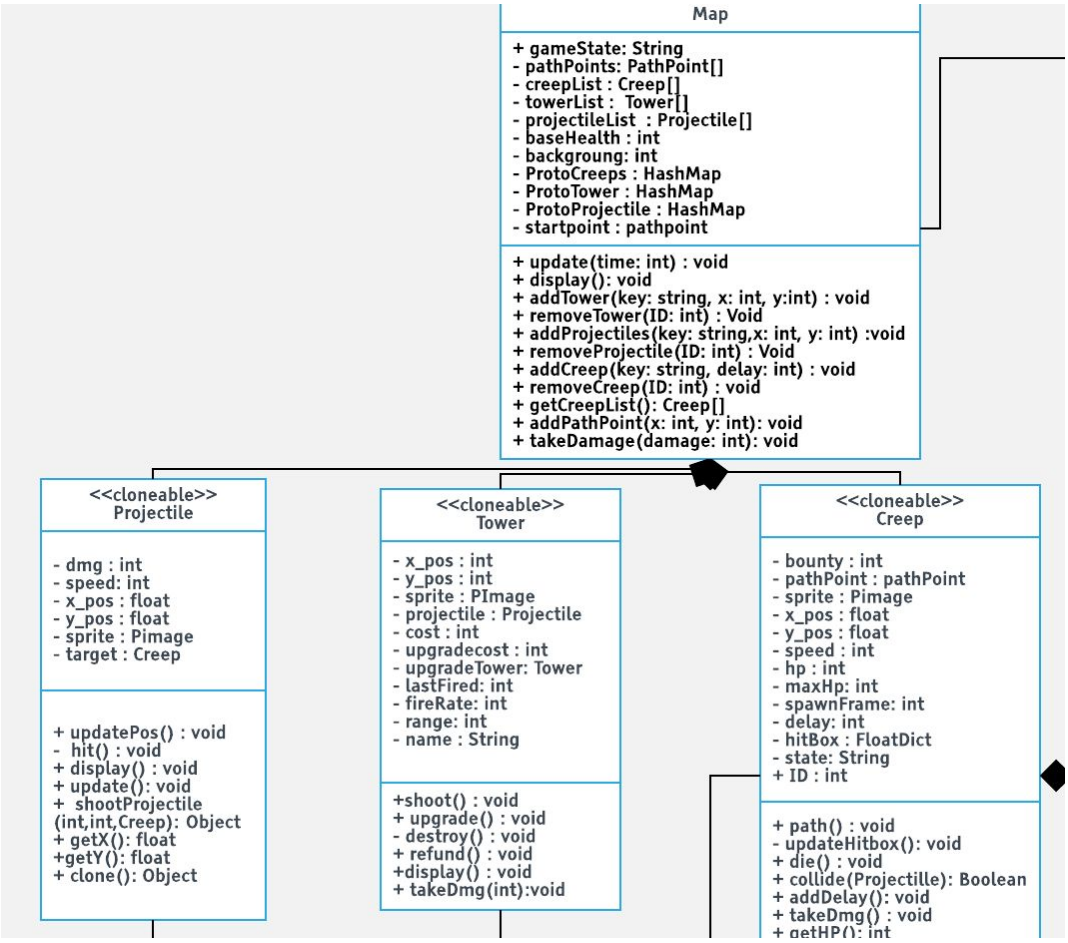


What Changed?

At the refactoring stage we changed up to make map responsible for the projectiles. We also decided to take on a prototype design type for creeps, towers, and projectiles. Another change we made was adding in more functions in each class. There were a lot of helper functions that ended up coming in handy and we wrote to help us work around Processing's implementation of certain aspects. For instance, we thought Processing would have more helpful collision detection built in, but we ended up having to write a lot of it ourselves. Creeps and Map got a lot more functions than originally planned and this was to help us out with our communication between the classes. Lastly one of the biggest changes was implementing Menus. This helped us display buttons over the game at different states in the game. The newly added menus allow the user to start new games, see high scores, and place towers, and we needed to write a subclass of each menu.

4)

For our design pattern, we implemented a prototype design pattern. We needed to spawn a lot of towers which had the same attributes (projectile, hp ,sprite, range), creeps (bounty, hp, speed, sprite), and Projectiles (sprite, speed, damage). After some thought, we decided a prototype pattern would be the best way to accomplish this, as no other approach, like subclassing for each type of tower or creep and making flyweight instances for each, made any sense. We made these classes implement the cloneable interface so that we can create new shallow copies of them. To make the base copies of these objects, we created a HashMap for each one of the objects in the Map class in which we inserted the base instance we would be duplicating. We called these ProtoTower, ProtoCreeps and ProtoProjectile and in each map, we fill them with all of our prototypes that we decided should exist. We store a key (ex: “Arrow Tower”) and associate it with a base instance of what we want the arrow tower to look like, giving it a certain projectile, range and cost. When we want to spawn an arrow tower, after performing the checks that make sure we are allowed to do this, we get the base instance in the HashMap, and clone it, then give it a few new properties, like the X and Y it will reside at and be displayed at. In doing so we create a shallow copy that now exists in the towerList. A similar process is used to instantiate new projectiles and new creeps. The creepList, towerList, and projectileList will then all be populated, and these lists are iterated through at runtime in order to have them be displayed and updated every frame.



5) What have you learned about the process of analysis and design now that you have stepped through the process to create, design, and implement a system?

While working through the design and analysis of this project through the semester we learned a lot about which aspects were really helpful and where our time can best be spent in creating a great final product. Without the initial planning and class diagrams, a project like ours would have been a nightmare to create. The analysis stage was extremely useful because it made us sit down and think through what a user would need and what the system would need to do in order to fulfill those needs. The class diagram created in Part 2 was one of our most referenced documents throughout the semester and we are all convinced by its usefulness. Some of the other diagrams were not as useful to us in designing our final product. Charts and documents such as use case diagrams, activity diagrams, and sequence diagrams seemed much less useful for the design of a system. They seem more like analysis tools for after a system has been created.

Designing an object oriented system like we did comes down to the quality of refactoring as it all comes together. The initial creation of the requirements table, the use case documents, and the class diagram were invaluable to us during this project. We have learned that the time spent on these saves a ton of time when writing the program. More important than just having them, is updating and modifying them as we refactored the classes and responsibilities. The final implementation of a project such as this comes down to proper planning using some of the documents and refactoring it as it gets close to completion.