Meeting Agenda

Group: Super duper Omega gruppen
Date: 2020-09-24 13:00
Chair: Erik
Participants: Samuel, Oskar, Erik, Behroz, Sebastian

Objectives (5 min)

1. Discuss projectile particles
2. Explain improvements of tower structure to everyone
3. Discuss why Vector still exists when we have VectorD
4. Divide our big priority-2 stories into smaller parts
5. Discuss if we should remove the task “Change so that Pos represents the center of the enemy”
6. Create new tasks

Reports (15 min) from previous meeting

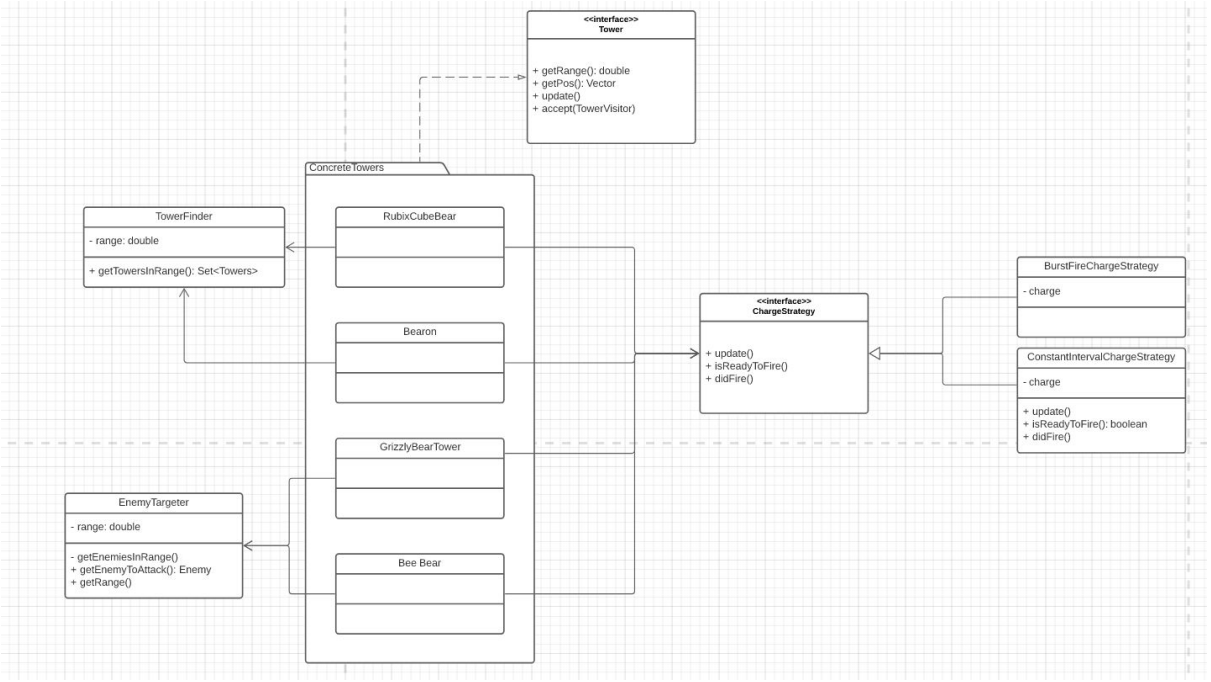
Samuel reviewed most of the current tasks (excluding the ones he participated in) that were in the review stage for model and view.

Sebastian: Made so that view shows if a tile is valid or not. Added javadoc to application, controller and some view elements. Refactored controller, view and application.

Erik: Improved the event system with what we discussed in the previous meeting. I also reviewed some tasks and made small refactoring/extensibility improvements in view.

Oskar has changed all floats to doubles and renamed VectorF to VectorD.
Fixed a bug in PathIterator after writing tests for it. The previous method would sometimes return the same value as next did.
Improved Distributions by making them all use static factory methods instead of public constructors so that we don’t sometimes need to prefix them with “new”.
Wrapped all public getters of lists that should be immutable with Collections.unmodifiableList()
Improved Vector class by adding getSquaredDist() for better performance and changing getX() and getY() to just access the field directly.

Oskar and Erik discussed how to improve the structure of towers and came up with a potentially improved design. We will discuss this design with Pelle at our next meeting.



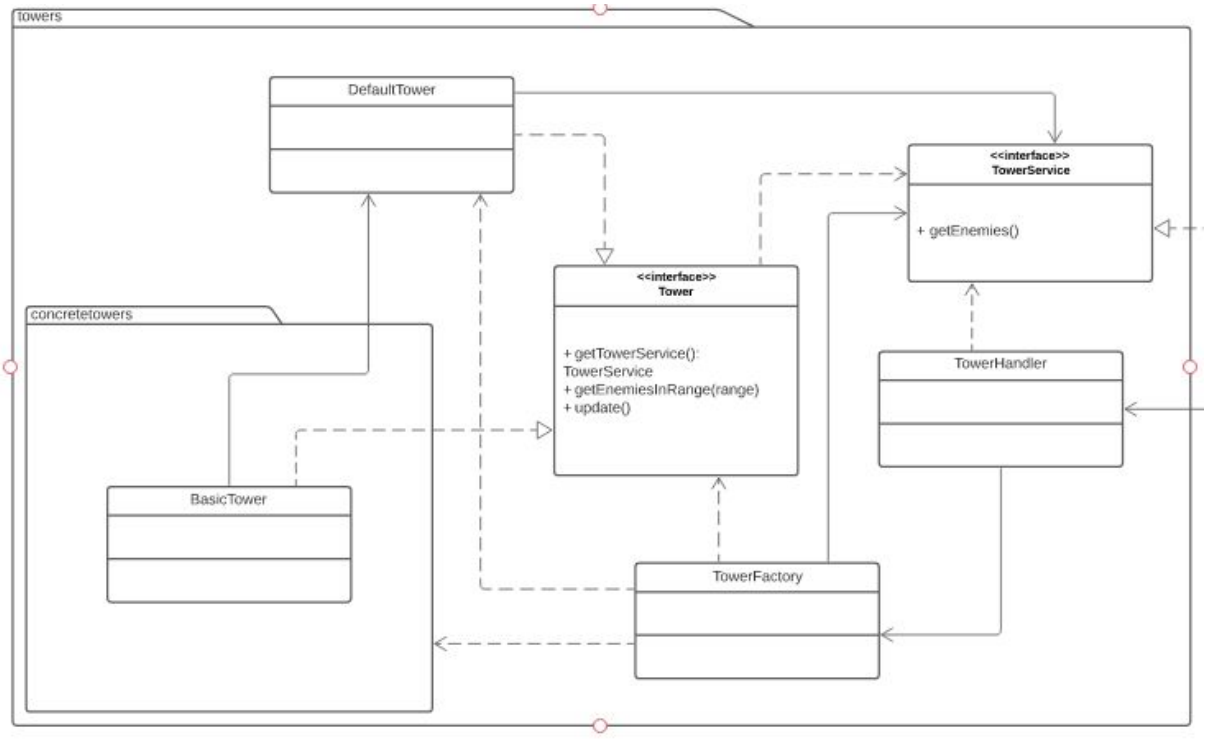
Discussion items (335 min)

1: Particles should be emitted when a rock projectile hits instead of when the grizzly bear throws it (which was just a temporary showcase anyway).

2: Tower design (report written by Oskar)

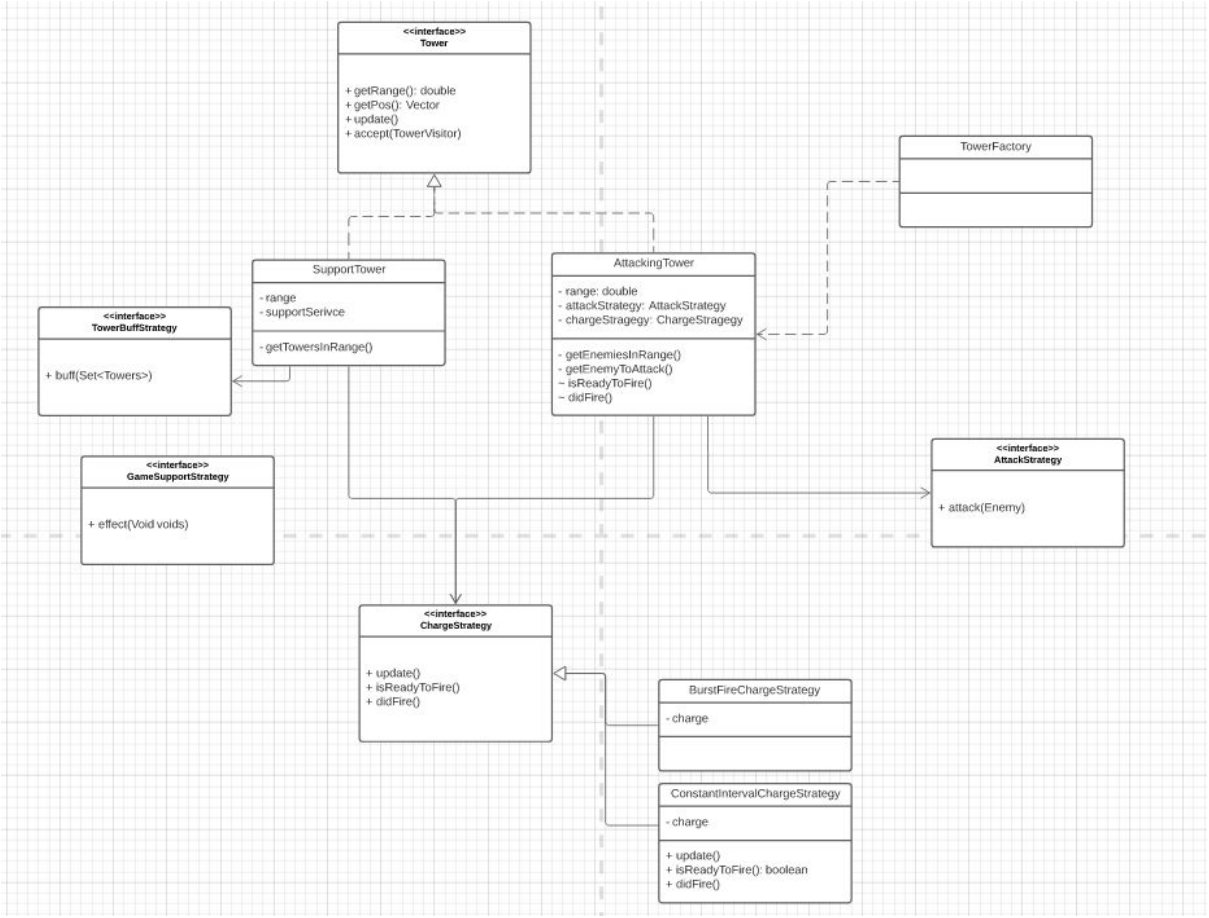
Current Tower design

Currently we try to do delegation by delegating to a baseTower from each concreteTower. This leads to big problems with the Tower interface because everything that a concreteTower needs to access (that would be protected in a normal inheritance hierarchy) needs to be public. Our way of doing delegation on Towers is very similar to a decorator pattern which it doesn't have to be. The issue with this is that there is no way for the "super type" (the baseTower, whose class i DefaultTower) to contact the concreteTower. However we have to be able to do that since the baseTower needs to tell the concreteTower to attack, which would normally be done with a template method pattern.



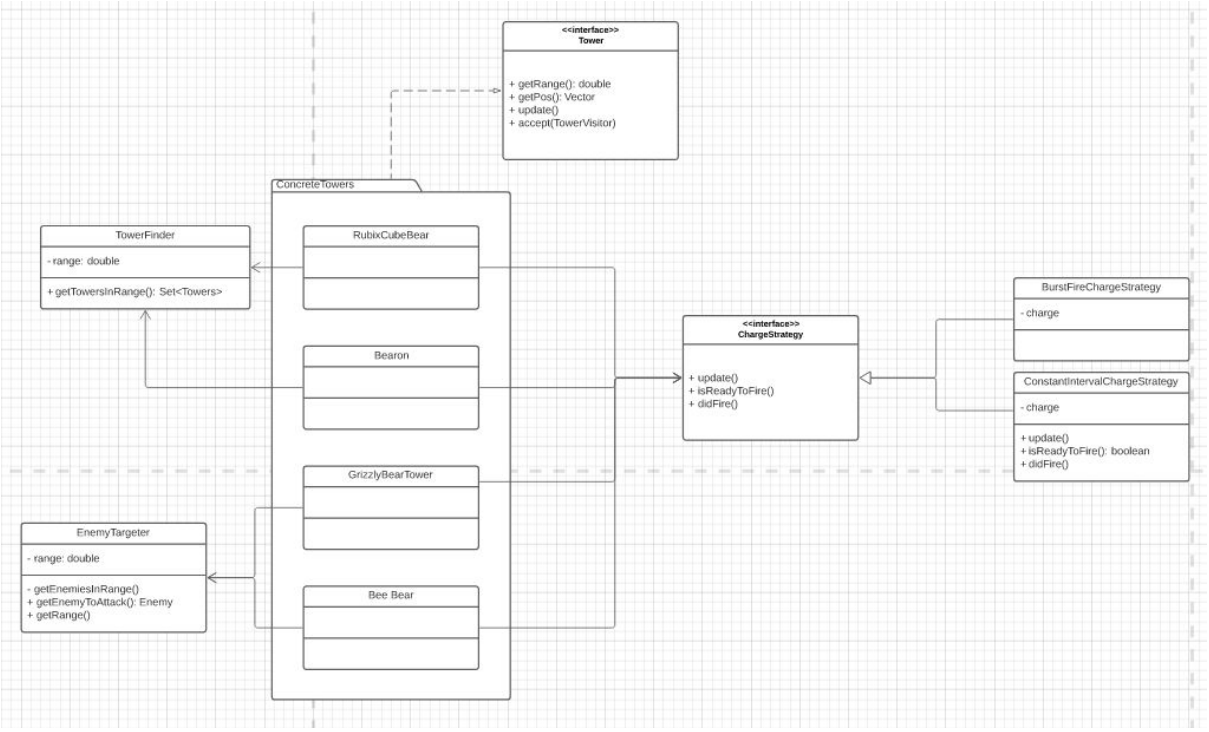
First idea to improve Tower design

We came up with another way of designing it where we have attackingTower which delegates to ChargeStrategy and AttackStrategy. Those strategies describe when to attack and what to do when attacking. Then for supporting towers we have a SupportTower class which delegates to a ChargeStrategy and a TowerSupportStrategy. This way, when the factory needs to make a GrizzlyBear it creates an AttackingTower and gives it the correct AttackingStrategy and ChargeStrategy. The biggest problem with this is that it doesn't work with our visitor pattern. The visitor requires there to be a class called GrizzlyBear in order to have a unique drawing implementation for GrizzlyBears. We could try to add an enum to describe which type of tower it is and forgo the visitor but that would require a lot of refactoring.



Second idea to improve Tower design

We could also have all the concreteTowers as Classes and then have them delegate as much as possible to other classes. Then we could have a ChargeStrategy, a TowerFinder and an EnemyTargeter to combine the common behavior of those classes. This still leads to some code duplication because each concreteTower still needs to manage classes it delegates to (like updating them, checking if chargeStrategy is ready to fire, etc). There are still some things that are unclear, such as “should the enemyTarget have the range or should the range be stored in the tower?”. Although this still seems like an improvement compared to our current design.



3: There is a problem with removing int Vector. Towers that only should have a whole number grid position should never even be able to have a Vector position with fractions.

Having both Vector and VectorD will mean that you will have a compile-time error when trying to store a VectorD as a Vector, which prevents bugs where you think a Vector is int when it's doubles. However, this will mean that there will be two types of vectors that you have to have seperate (code duplication and methods for switching between them).

Removing it will require work, changing every instance of int Vector. It also removes the type security we have from having the Vectors separate. But having only one Vector class will remove code duplication. (And Pelle wants it)

Written by Behroz:
Seems unnecessary to have an extra class that shouldn't cause trouble in this project and even if it does, it wouldn't be a noticeable hurdle to work around.

4. We divided our story "As a player I want to have a large collection of towers that I can buy because that makes the game more interesting" in three stories: One for towers with different attacks, one for towers that gives debuffs to enemies and one for towers with support abilities.

Then we got into a discussion with how flying fish should fly.

The story for creating all our enemies became two: One for basic enemies and one for creating enemies with special abilities.

5. Yes, we decided to remove it since it works now and will just be more work to change it now.

6. Problem: How should we avoid ConcurrentModificationException when Controller says to Model to buy tower.

Outcomes and assignments (5 min)

Assignments skipped. When you want to code you take what you want to code.

Wrap up

Design GUI for start screen next meeting.
Make more user stories!!!!!!!

Next meeting (with Pelle): 2020-09-25 15:00