## Assignment 2 - Group 26

## Exercise 1 - 20-time

As the extra feature, we've decided to implement PowerUps. PowerUps are entities that execute a certain effect on the PlayingField/PlayingFish once a PlayerFish has eaten it.

These following are the CRC cards are for the main classes of the PowerUp feature.

PowerUp		
The parent class for all PowerUps	Is located in: PlayingField	
Produces an unkown effect on PlayingFields	Makes changes to: PlayingField	
	Makes changes to: PlayerFish	
Can interact with PlayerFishes	Parent of: DurationPowerUp and all other PowerUp types	

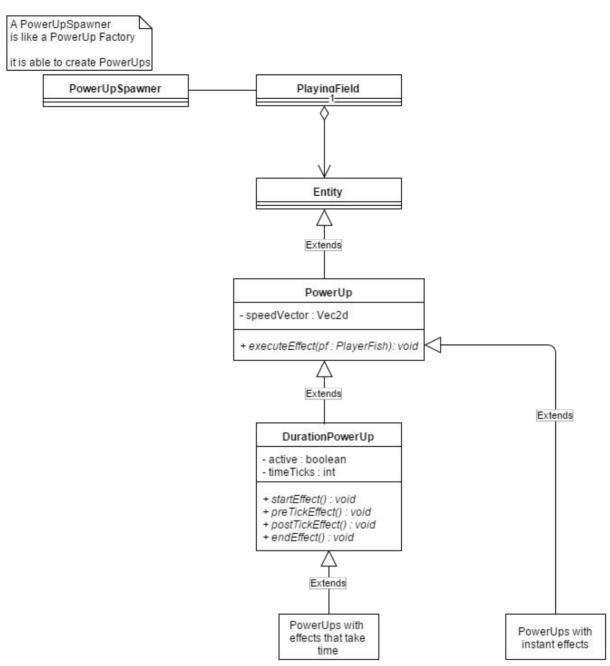
PowerUpSpawner		
Able to create different PowerUps	Is located in: PlayingField	
Spawns PowerUps repeatedly on the PlayingField	Creates: PowerUp	
	Implements: TickListener	

DurationPowerUp		
Can help child classes of	Child of: PowerUp	
PowerUp do something with		
durations or time	Implements: TickListener	
	Parent of: Any child class of	
	PowerUp that want	
	something to do with	
	durations or time	

There are several other classes for the PowerUp feature, but these are only different children of the PowerUp class that implement the unknown effect.

An example child of the PowerUp class is PuExtraLife. This class has the simple effect that it gives a PlayerFish an extra life. A different Powerup child is PuFreeze, which freezes all EnemyFishes on the PlayingField for ten seconds. Because there are more PowerUp children that last for a certain duration, it's simpler to make one class that does this for them. That is the responsibility of the DurationPowerUp class. It can play a different effect at the start, each tick and once the duration is over.

The (simplified) UML can be found on the next page.



Note: the classes PowerUp, DurationPowerUp and all their child classes are located in a seperate package com.github.fishio.power\_ups.

## Exercise 2 - Your wish is my command

As the TA assignment, we had to split up the PlayingField class into 3 parts: GameThread, Renderer and PlayingField. The reason for this is that it has too many responsibilities.

We first created CRC cards for these 3 classes, in order to decide what responsibilities would be assigned to each class.

PlayingField	
The class that represents a playing field.	Is created by: SinglePlayerController
Creates a GameThread and Renderer, and starts/stops them when needed.	Parent of: SinglePlayerPlayingField
Holds all objects that are in the game.	

Renderer		
he class that renders the game.	Is created by: PlayingField	
eracts with PlayingField	Implements: Listenable	
Renders alive Drawables	The second secon	
Renders death animations for		
dead Drawables		

GameThread		
The class that runs the game.	Is called by: PlayingField	
Interacts with PlayingField - Moves movables - Calls EnemyFishFactory to add entities - Checks for player collisions - Cleans up dead entities	Makes changes to: PlayingField Implements: Runnable, Listenable	

We decided that the PlayingField would still create the GameThread and the Renderer, and act as an interaction point to them (start/stop them), but that the PlayingField would no longer be tasked with the actual game or rendering itself.

The (simplified) UML can be found on the next page.

