Peer Review

The project has an MVC structure and uses the Observer pattern to notify a change in the model to the view without making the model dependent on the view. Neat. It also follows design principles, but occasionally lacks a consistent coding style.

The pattern template is implemented through abstract classes. This promotes reusable code and avoids repetitive code, eg. AbstractVehicle.

VehicleFactory and ScreenFactory are two instances of factory pattern in the code. VehicleFactory returns the creation of a car and should have full control of the implementations of cars. Currently SportsCars has a public constructor which allows their creation outside of the factory. This defeats the purpose of using the factory which acts as the sole creator and common denominator of a group of objects.

In AbstractVehicle the design pattern Defensive Copying is used to avoid alias problematics. It does however feel redundant since the immutable class Vector2 can’t be changed anyway and can’t have any alias problems to begin with. AbstractVehicle also uses a public setter for the position of the vehicle which makes it possible for all classes to teleport a vehicle anywhere on the map.

There are a lot of public methods even though it is written in the SDD that they should be limited. This makes the code prone to weird dependencies.

Private variables are the standard practice in the code but Vector2 has public final variables. Getters are used in conjunction with private and final in pretty much all other parts of the code and this would be prefered for Vector2 as well.

Documentation

The code is overall well documented. Every class is commented and almost all of the public methods are documented with JavaDoc. Some methods have explanations for their parameters but not of the method itself. This results in some inconsistency in the documentation of the code.There is also one comment which we presume is misplaced in GameModel. While the explanations all in all are to the point, sometimes they are too short or general. In addition some JavaDoc-comments have a space between the method and the comment, eg. AbstractPowerUp.

Proper names are used in the code for the variables, methods and classes. All classes have big letters and all variables and packages have small letters as the first letter. The project has a good structure with proper naming. When it comes to testing there are three tests in the project. That could be expanded upon with more tests on more classes.

Abstractions and modular design

There are well implemented abstractions in the code where needed. Examples are abstract classes for vehicles and power ups as well as an interface made for all kinds of vehicles, not only the sports car.

The code is modular, but some packages are almost too small. The Player class has its own package and consequently needs to make all methods public, which feels unnecessary.

The code in this project is very reusable. Every class has a clear purpose and they can be reused in other situations. They also have a good structure and great modularity in the code which makes it easy to maintain. The only exceptions to this is the frequent use of public methods which could lead to unpredictable side effects.

Improvements

The collision between the car and the ground is quite buggy and creates lag. Especially when you boost at the same time. When the car is spinning it looks like the map isn't moving with it. The reason for this might be that the car isn't centered around itself correctly.

To decrease the amount of negative dependencies between the model and the view we recommend a facade which only presents the necessary methods to the view. This would mean that the model still can have public methods readily available to the controller while view only has access to a limited and carefully chosen few of these methods. For example the view can now use gas() which should only be able to be called in the gameController when the correct key is pressed.

Right now the music is part of the controller, but should rather reside in the view to follow MVC guidelines better.

The map is hard coded and reversing at the start makes you fall off the map, while driving too far forward makes you fall off the map as well. So it would be nice to either have a clear goal or an infinite map.

A final thought, which is not directly related to the code itself, is that it in its current state feels like the game lacks a purpose. Implementing a win condition or a point system would make the game more interesting and would give a clearer sense of direction for the project.