Programmeertheorie
Design Document

Why?

To get you and your group started on your epic journey in the field of heuristics and truly hard problems, your group has to compose a design document. You are faced with a daunting assignment and a blank sheet, and at the end of this course we expect epic results. To get you going, we ask you to rack your brain, and capture what you think you need to build. The result of this process should be a document that captures *design decisions* you have made, a document that helps you start coding and most importantly, one that allows you and your group to divide the work.

What?

Your design document should capture any and all design decisions that you make before (or while) you start your implementation. The format of this design document is up to you. Which means that pure text is fine, but pictures of whiteboards could do just as well. There's only one constraint, it should be understandable for you, the future you, your group and even outsiders like us, the staff. Reasonable to include in your document is:

1. A list of classes and functions/methods (and their return types and/or arguments).
2. Your choice of representations.
3. What functionality you want to implement. For instance, you might want to write results to a text file, or maybe even build some kind of (graphical) user interface.

How?

Get your group together and find a place where you can talk. The next step, you have guessed it, start talking. Ask yourselves the important questions in life, such as how do you represent a house in Amstelhaege, how do you move a vehicle in Rush Hour? When the fiery discussion calms down, and a decision has been made, write it down. Once you have written some design decisions down, iterate on them. For instance, given your representation of a tile in the tile fitting case, how do you ensure tiles do not overlap? If this requires you to change your representation of tiles, then go back and change it. For in this point in time, changing your representation requires you to just change a few lines of text in a document or the retaking of a whiteboard picture, and not rewriting many lines of code.

Do not concern yourself too much on how to solve your case, as you will have plenty of time to do so in the weeks to come. Instead focus on the essentials you need to get cracking. Furthermore, focus on the overall design. For instance, it is (very) likely your program will change over time, thus it is probably wise to prepare for such an event. You could create *interfaces* to access parts of your program, such as methods to retrieve vehicles from the field in Rush Hour or methods to access layers in Chips & Circuits. By doing so, you abstract away from the actual implementation. Thus, when you change the representation of some part, you do not need to change the remainder of your program.

In a lesser design changing one section of code requires you to change two more elsewhere, which then require you to change four more, and so on. An important design decision for combating this flooding effect, is keeping parts of your implementation *loosely coupled*. For instance, when building user interfaces it is common practice to separate the visualisation from the application logic itself, as shown in figure 1. Not only does this make testing your application easier, it also becomes easier to modify both your application and user interface. Heck, you could even plug in multiple user interfaces by keeping the user interface loosely coupled.

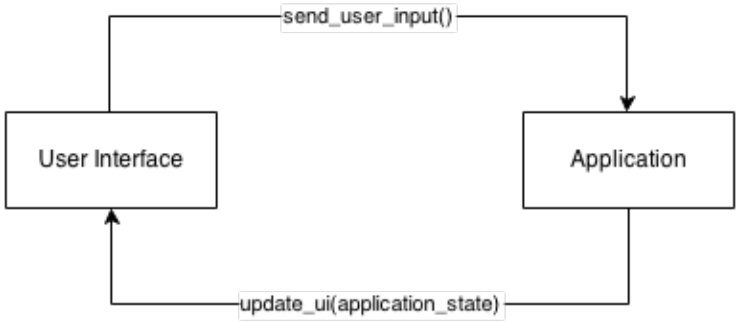

Figure 1: Seperation of User Interface from Application.

Remember, an important aspect of this document is to help you get started and divide the work among members of your group. Therefore, you should try to identify parts that can be developed separately.

When?

Your group should finish your design document before your first meeting, so we can discuss its content during. And no, you will not receive a grade for your document, but we do require you to submit one.

Where?

You can submit your document by putting it in your git repository in a directory called doc. Please remember to share your git repository with the staff!