CSCI 3081
Braitenberg Vehicle Project
Design Document

Kadin Schermers

This design document explores the pros and cons of three different implementation styles for the Entity objects in the Braitenberg Vehicle project assigned to us in CSCI 3081. The three implementation options are as follows:

- The instantiation of entities in the provided code.
- The use of an abstract Factory class and derived factories
- The use of a single concrete Factory class that is responsible for the instantiation of all Entity types.

We will explore each of these options independently from one another, but their ideas will be talked about amongst one another.

In-code Instantiation:

The clear and obvious benefit of instantiating within the provided code is minimal project size in terms of file count and total code. With no extra files being created for factory classes, total overhead is reduced…

The biggest drawback is the interconnectedness of the project. Each class that wants to create an entity needs to know all the information that goes into building and accessing that entity. This not only means the project requires modification in more locations when something needs to be changed, but also contains a lot of redundancies that could be avoided with the use of a factory class.

Single Concrete Factory:

The leading benefit of using a single factory design pattern is that it allows us to remove the need of direct instantiation inside the program classes. This allows us to use the factory as the builder for each entity, so that each class doesn't have to know all the required information to create a new entity, and the factory is the only class that needs to have that information available to it. This also increases the security of our program, since we can hide the implementation of the entities from the program classes.

The main downside of a single factory class is when we have a variety of entity types, such as this project does, the single factory class becomes is quite broad, and we can't combine each entity types creation. This means the factory class has to carry the weight of all entity types, when it could be broken down into multiple derived factory classes.

Interface/Derived Factories:

The main pro of the interface/derived model is modularity. It allows for the use of interfaces rather than direct implementation. This means that the single factory interface can be loosely defined, and entities of different types can each have its own class that accounts for that entities uniqueness.

Main con is overhead. This method produces the most code files and documentation and testing. This can make the project harder to understand if they are looking at it with no previous knowledge.

I chose for my actual implementation to go with the Interface/Derived Factories method. It offers the best benefit of keeping modularity high so that changes along the way were easier to add. Also it reduces the dense-ness of the main program files so that they are easier to modify as well. Oh and we had to do it this way.