Design Document for Braitenberg Vehicles

Angela Almquist
University of Minnesota, Minneapolis, MN 55455
CSCI 3081W Spring 2019

March 16, 2019

Introduction

The purpose of this document is to identify the changes made to the Braitenberg Vehicle code packet provided by the University of Minnesota staff. There were three methods available for executing the desired changes to the code: the current design with hardcoded changes, a factory interface class and additional concrete classes. The pros and cons of implementing an interface verses a concrete class to add new robots to the simulation will be discussed in this report. The current condition of the code and future strategy pattern implementation will be addressed as well.

Theory

The Braitenberg vehicle simulation was first introduced by Valentino Braitenberg. The simulation can be very interesting if implemented correctly. In a successful simulation, the behavior of these vehicles are dependent on their surroundings and user inputs. These robots can be set to cower from objects, avoid other vehicles or be particularly glutenous towards food. It is desirable for there to be a user interface to add robots and change their behaviors in real time.

Pros and Cons

There were three options to consider for the development of this program: the initial implementation, an interface class or concrete classes. The initial implementation is sufficient to start the simulation; some features had to be added, but these were all easy to incorporate inside the Braitenberg object classes. The pro for this option is it was already done. The con however, would be the difficulty that would lie ahead when adding features to the Braitenberg objects. Hardcoding features inside the class could lead to undesirable coupling between functions and entities. For the interface option, the unfortunate part is it takes more code to write initially, and can lead to some complexity later on when overriding virtual functions. The benefit of using an interface is there should be less code to write overall. Features defined in an interface can be overwritten and designed specially for child classes without dealing with the extra work of writing overloading and operator constructors for each child class. Lastly, concrete classes would have been straight forward. Concrete classes would have the same effect as the interface, but may include more coding later on in development as everything would need to be initialized and defined for each concrete class. For both the interface and concrete classes, these methods allow for less coupling; features added are separate for each class and less likely to influence other parts of the program. I ended up developing a factory interface because of its cohesiveness; features should have similar formats for all derived classes. It being an unfamiliar coding style, I also wanted to learn how to develop an interface.

Design Process

A preliminary code was provided by the University of Minnesota CSCI 3081W teaching staff. The first modification to the Braitenberg code was to implement an interface called FactoryEntity. With this interface, new robots, lights and food can be instantiated in the simulation. This is the interface to three factory classes: Braitenberg Vehicle factory, Light factory and Food factory. These factories are in charge of creating new objects of their class while the interface allows for their existence in the Braitenberg arena. Currently, the interface is capable of creating objects in the arena with an .json file.

The initial code was sufficient to start a Braitenberg simulation with the help of a hard-coded .json file to define the initial conditions of the simulation however there was additional features to add. Not all behaviors were available to the robot. Color of the robot could reflect what was driving it. Both were successfully programmed in. The behaviors were defined in the BraitenbergVehicle class. The four behaviors towards light and/or food: Explore, Aggressive, Love, and Coward, were defined to cause different trajectories by affecting the robot wheel acceleration. Eventually, it will be desirable to program a strategy pattern so the behaviors are entities of the Braitenberg vehicle rather than the current hard-coded implementation.

Conclusion

Much work is left to be done. The interface method proved to be easy enough to develop, and the benefits of using this sleek method should appear in the iterations ahead. The strategy pattern will be the next feature to add to this code base. Once the behaviors are set as entities of the Braitenberg objects, this pattern should pave the way into developing dynamic conditions where the user can add, remove and edit robots and objects while the simulation is running.