Project Implementation

The main goal of our simulation program is to find the control plan which allows the car to traverse the terrain in the optimal amount of time. This is done by carrying out a series of runs, each with a unique Control Plan. By running every possible Control Plan and keeping track of which one resulted in the shortest traversal time, we are able to determine the optimal Control Plan for a given car and terrain. Since a Control Plan is simply an array of ones and zeroes representing “Forward torque” and “Reverse torque”, respectively, an easy way to ensure that every Control Plan is tried is by using binary counting. Since it makes more sense to start the first run of the simulation with an “All Forward” Control Plan rather than an “All Backward” Control Plan, our simulation starts Run 1 with a Control Plan composed of all one. Each successive run uses a Control Plan derived by using reverse binary counting on the previous run.

Trying every possible Control Plan is a surefire way to find the optimal solution, but this is incredibly inefficient. A better solution is to run every Control Plan that elicits a *unique* result. This will still ensure that the optimal Control Plan is found while potentially eliminating the need to run vast amounts of redundant runs. We incorporate this is by recognizing failure conditions in each run and then modifying the subsequent Control Plan accordingly. Our program recognizes when the car gets stuck and when the car falls off of the terrain. The main theory is that modifying the Control Plan at any Control Steps chronologically after a Control Step in which there was a failure point will result in a run with an identical result as the previous run. For example, consider a run in which the car follows a control plan of “All Forward” but gets stuck halfway through the run. Following the normal reverse binary counting algorithm, the next run will be composed of all “Forwards” except for the last Control Step which will be a “Reverse”. However, since the beginning of this new Control Plan is identical to the first Control Plan, we know that the car will get stuck halfway through the run again. The “Reverse” command at the end of the new Control Plan will have no effect on the result of the run and thus, this is a redundant run. By recognizing these failure points and then applying reverse binary counting at the Control Step that the failure occurred in, we are able to eliminate all of these redundant runs and continue iterating through Control Plans with unique results.

Our simulation program incorporates three different modes that modify the behavior of the simulation depending on what kind of results the user is looking for. These are as follows:

“Stop on First Success” (Mode 1)

In this mode, the program carries out runs until a solution is found and then stops the simulation. This mode is useful if you just want to find a single solution and don’t have the time or computational power available to run the other, more exhaustive, modes. This is the mode we used in the analysis of the project.

“Exhaustive” (Mode 3)

This mode executes every unique run and keeps track of the results. This mode is useful if you want to find the result of every possible Control Plan. However, even with the optimization implemented, this mode easily takes days to fully complete on an average computer system.

“Find Fastest” (Mode 2)

This mode is the same as Exhaustive Mode except that it stops runs if they exceed the run time of the most optimal run found by the program so far. Thus, this mode will not give you exhaustive details on every unique run since it will not allow many of these runs to keep going till completion. It will, however, find the optimal or “Fastest” run possible. Even with this efficient algorithm, however, this mode is entirely capable of taking days to complete on an average computer.