**Creature Simulation Test Plan**

The following tests were performed in order to ensure the simulation is working correctly. While it has not been completed yet, an automated test mode is being constructed to run through many of these tests automatically to verify that the results remain consistent as future changes to code are implemented.

1. **Verified all potential configuration parameters are being read correctly.** In this test, debug mode was turned on (by changing DEBUG=1) in the simulation.h file. Various parameters were changed and it was ensured that after the simulation instance was created, the variables were set correctly by dumping all of the relevant variable data using the displayCfg function in the simulation.cc file.
2. **Single car, travel from start to finish with no creatures.** In this test, trace files are used to verify that the car moves according to the Nagel-Sh. Algorithm. First the car is tested with a speed of one and the speed is varied up to 11 to ensure the behaviour is as expected. Again the DEBUG flag in “simulation.h” is set to true so that extra output can be seen (as the car advances from cell to cell, a line is printed stating the speed and what cell it moved from and to). When the car reaches the end of the highway it should be removed and a line should be output in the trace file with all of the relevant car information.

**Problems Found:** Cars didn’t actually reach maxspeed, were only going to maxspeed-1. This has been resolved.

1. **Many Cars, no creatures.** This test is the same as the previous test, however now we ensure that no cars collide with each other and that there are randomized speeds at the start between 0 and max\_speed. We also check that when cars leave the highway, the relevant information is dumped to the log file (creation time, movement trace, etc.**)**
2. **Single car, single creature.** This test is similar to the single car test, however, the creature is now added. Trace files and debug output are used to step through what conditions the creature sees and whether or not it decides to move (for both the naiive and f&d algorithms). This test also verifies that creatures can successfully cross and are hit under the right circumstances. This is controlled by varying the speeds, crosspoints and the “fuzzy” velocity and proximities so that various conditions are encountered. For example, if we have 3 5 7 for both velocity and proximity, we can use a single car and fix the speed to values that fall between these ranges to test to see if the creature is hit or is successful crossing under either of these conditions depending on where the crosspoint is.

**Bug:** When only a single creature is used, it gets stuck on the highway and is unable to move because of a state variable that only allows a creature to move one lane during a timestep. This has now been fixed.

**Bug:** Cars in the first cell (cell zero) were not appearing in the creatures perception area due to a loop condition (should have been >= and was originally >). The bug did not affect any positions other than immediately at the start of the highway.

We have verified that various speeds and proximities are being perceived correctly by the single creature based on the entries in the KB after the simulation is run. With speed and proximity “fuzzy” values of 3 5 7 we tested with crosspoints, speeds and proximities of 2 4 6 8. With all combinations of these, it was possible to see entries in all possible configurations in the KB table, which means the creature is able to perceive the entire range of states correctly.

1. **Many cars, Single creature.** This test is similar to the single car test above but extends it to include many cars. This case is somewhat useful in cases where the crosspoint is far from the start of the highway. This allows some interaction between the random initial speeds of the cars so that some slightly different configurations are possible for the single crossing creature.

We have verified that this test works using similar techniques to the previous test.

1. **Many cars, many creatures.** This test is essentially the full simulation. In this test, the learning algorithms are tested and verified.

To test the naiive approach (intel=0) any configurations where a creature is hit once should be avoided. This can be easily tested by running several configurations and verifying that in fact this is the case in the KB result at the end.

To test the fear/desire approach, there are several test cases. First we can check that the fear/desire approach behaves the same as the naiive case when the fear and desire are set to zero in the configuration file. If this is the case, in all of the dice roll scenarios, fear and desire will be zero, meaning that it is equivalent to naiive. Next we can test with a case where fear is greater than desire. We should notice few creatures being hit in this case, because they will behave more closely to the naiive approach and avoid crossing in dangerous locations. Conversely, we can test with desire greater than fear. In this case we should notice more creatures being hit because they are taking more risk.

We have verified that indeed the behaviour of the two approaches are the same when fear and desire are set to zero. Furthermore, the behaviour is as expected for fear > desire and for fear < desire.

1. **Multiple Lanes.** This test repeats all of the above tests, but with multiple lanes. It ensures that the same expected behaviour occurs consistently with the extra lane.
2. **Traffic in Opposite Direction**. In order to set a lane to move in the opposite direction (right to left), a flag must be set for each the lane so that it works in the opposite direction. For instance, to make the lane closest to the creature move backwards, LANE\_BACKWARDS 0 is set in the config file. Each lane which should move backwards must be individually set in this manner. This resulted in many code changes. First, the car movement algorithm needed to be modified to check for the direction of the lane. If the adjacent lane is in the opposite direction from the current lane, cars may not pass using the adjacent lane. Similarly, the creature algorithms must be changed. When a creature estimates the distance and proximity, it must know the direction of the lane the cars are travelling on for the estimate. Collision detection between the creature and car must similarly take into account the direction of the lane.

This is all tested by repeating many of the above tests to ensure they still work with multiple directions.

After initial testing, the simulation works with traffic in both directions for a single car, and multiple cars.

1. **Position Changing of Creature**

* **Only move after so long at one location**
* **Can only move a couple of cells at once**
* **100-150m over several timesteps side to side**
* **Number of cells in one timestep configurable**
* **Randomness in direction of creature (left or right)**

**About the Fear & Desire Algorithm:**

Originally there were two approaches to fear and desire for the creatures. Initially, each creature could have a random fear and desire each bounded by the constants set in the configuration file. This was later altered so that all creatures had the same static fear and desire for all creatures. The most recent iteration of the fear and desire has been altered with coin flips. Each creature has its own pair of random number generators that are initialized when the creature is instantiated. Then fear and desire is set according to how the coins are flipped.

* If the result is “TF”, the fear value from the configuration file is taken.
* If the result is “FT” the desire value from the configuration file is taken.
* If the result is “TT” the desire – fear value is taken
* If the result is “FF” 0 is taken (the creature has neither fear or desire)