Test Plan Document

Requirements:

For the testing of this software, one of the requirements for this project is that it should operate in iterations in which each of the goats can move to an adjacent intersection given that the intersection is not occupied by another goat, or the intersection is not a border intersection. Any type of collision detection can be used, but the goats must not have knowledge of the location of other goats, nor the border of the grid. The goat should only be made aware of the fact that travel in that direction results in a collision. Also, if a goat finds the exit, the goat should be removed so that collisions outside of the grid do not happen. The program should be able to operate with a grid of size 13x13. It must also utilize the GPU to parallelize the solution and be able to operate with over half of the intersections occupied (ceiling(intersections/2)). Lastly, when testing the software, the program must produce a movement file at the end of simulation which lists each of the goat's movements in the form of: goatNumber.

Integration tests:

To begin our integration tests, we first started off simulating our code with one goat and a grid size of 5 x 5 to make sure our code is accomplishing what it is supposed to be. After simulating, we found that the test passed. When simulating, the output showed the start position of the goat, the grid coordinates for each cell, as well as the coordinates for the exit. It also showed that the goat made 32 moves before reaching the exit. The simulation also listed the goat’s movement when it was trying to find the exit, and it showed the movement even if the goat tried to move into the border of the grid. All of these movements and collisions were listed in x and y coordinates. After testing with one goat, we then integration tested the program with two goats, and a grid size of again 5 x 5. When testing this, we were able to see the start position of both goats listed in x and y coordinates. As with the first test, it also showed the coordinates of each cell in the grid, as well as the coordinates for the exit. When looking at the tests, goat one was able to find the exit in 27 moves, while goat two was able to find the exit in 7 moves. Again, the simulation first listed all the moves made by goat one first, and then the moves made by goat two after. This test fulfilled the requirements and thus was a pass. Our last test was testing the program to fulfill all of the requirements. To do this, we made the grid size 13 x 13. We also made the number of goats 7. When running the simulation, it showcased the coordinates of each cell of the grid, as well as the coordinates of the exit cell. It also showed the movements of each goat in x and y coordinates, as well as how many moves it took each goat to find the exit. It also listed the moves of the goats that resulted in collisions, such as a goat running into the border of the grid or into another goat. Lastly, whenever a goat did find the exit, it was removed from the grid, so that other goats could fill the exit cell, which fulfilled the requirements.

Unit tests:

When conducting unit tests, we first started with array generation. To test whether the arrays could be generated in a certain size, we first set the grid size to 5 x 5. When doing this, the output showed the grid size, as well as the coordinates for each of the cells in the array. Our next test for array generation was testing a 13 x 13 grid, which was the requirement. When testing this, it showed all thirteen cells in the array listed in a grid format with x and y coordinates for each of the cells, so this test passed. Our next unit test was goat placement. We had to make sure that if there was more than one goat, each goat would not get placed in the same cell as another goat, as that is not allowed. When running this unit test, we first tested it with one goat just to make sure the goat spawned correctly inside of one of the cells on the grid, which it did. Next, we tried to unit test the goats’ placements with two goats to make sure they did not share a cell. When we simulated multiple times, both goats always spawned in two different cells and never shared the same cell. Lastly, we tested the goats’ placements with 6 goats in a 13 x 13 grid. When doing this, it showed that each goat spawned in a different cell, and they never shared the same cell, which passes the test. Then, we unit tested goat movement and collisions. We first tried this with one goat. When simulating the goat movement and collision detection, it showed that whenever the goat tried to move in the direction of the border of the grid, it would output the goat's movement along with the showing that it is trying to collide with the border, but it couldn’t. It would output the goat’s movement in x and y coordinates until the goat found the exit, at which point the goat would be deleted from the grid. Also, when testing the goat’s movement, it showed each of the goats’ movements in x and y coordinates and how many moves it took to reach the exit. Next, we tested goat movement and collision with 5 goats. When unit testing this, none of the goats were ever able to collide with one another, nor were they able to collide with the border of the grid, which makes this test a pass. Lastly, we tested the goat's exiting the pen. For this text, whenever a goat gets to the exit, it needs to be deleted so that other goats can get to the exit, as the exit is another cell, and more than one goat cannot share the same cell. When testing this, we first started with one goat. When testing this, it showed the coordinates of the exit. When the goat reached the exit, it was deleted from the grid. Next, we tested with two goats. When testing, it showed that when goat one reached the exit, it was deleted, and it allowed the other goat to reach the exit as well. Lastly, we tested with 5 goats. When simulating, whenever one goat reached the exit, it was deleted. Two or more goats were never able to reach the exit at the same time, and each one was deleted after reaching the exit, which passes the test.