Genetic Algorithm.

This project was to create a generationally-iterated system that could emulate natural selection processes, in order to create an AI that can determine the best location to detonate a bomb; a location with the most “bad” entities, and the least “good” entities. A program designed to reduce collateral damage and maximise the elimination of intended targets.

Implementation began with the creation of a large number of robots (stored in a vector), each with a moral alignment of either “good” or “bad”. Whether or not they are “bad” robots are determined by a random chance based on a percentage value called “evilness”, pertaining to the general moral-degradation of a selected population of robots. Then a smaller number of bombs are created and placed around the scene, starting out targeting a random “bad” robot without any bias preventing them from destroying “good” robots. These bombs each have a randomized radius, of which anything within the radius will be destroyed if the bomb was to detonate.

After the initialising, this first “generation” of bombs are then given an overall score based on how many “bad” robots they killed versus how many “good” robot casualties. This score is heavily weighted against killing “good” robots. Once every bomb has their respective scores calculated, these scores are summed up and then each bomb would be iterated against that sum in a roulette-style selection. This is done two times, one for each parent of this generation, of which would be used to create the bombs in the next generation.
Roulette-selection was used as it was relatively easy to implement, but also gave a random-but-weighted chance for the selected parents, so the gene-pool can be as diverse as possible, whilst still maintaining a higher chance for higher quality specimens.

Once two parents are chosen, the x and y positions, along with the radius of the bomb’s blast, are “crossed-over” based on a randomised cross-over point. A small chance was given for these values to “mutate”, and either add or subtract a random amount to, so that they have variance beyond what their parents allowed them.

Initially the bombs would not discriminate between “good” and “bad” robots, and would just go for the highest total number of kills, this was not an intended behaviour. After adjusting the score calculation for each bomb, and weighting it much more against the killing of “good” robots, the algorithm was able to generate more palatable results.

The display/UI of the program was implemented in OpenGL so that it as a concept would be easier to visualise (but also easier to debug and develop). Using the left mouse button, the user can activate the algorithm and detonate a bomb in the most optimal spot as determined by the AI. Using the right mouse button, the user can shuffle the positions of the robots and their respective moral-alignments. Using the mouse scroll wheel, the user can either increase or decrease the percentage of “evilness” in this population of robots, and thus changing the ratio of “good” robots versus “bad”.

Although in most executions of the AI, it is rare to have the “good” robot casualties going over one or two, sometimes the AI will chose the only best option, or the best-case scenario in a set of only bad scenarios. The very nature of the algorithm means that this will always be a possibility, and that there is no real perfect solution in this case.

In the end, the algorithm is functional, and the given problem is solvable to a best-case scenario after a series of generations, genetic cross-over and mutations.