

Crowd Simulator using Genetic Algorithm

Mr. Tanay Prashant Joshi¹, Ms. Shubhada Labde² and Mr. Pauras Ranade³

¹⁻³Dept. of Computer Engineering, K.J. Somaiya Institute of Engineering and Information Technology
Mumbai, India

Email: tanay.joshi@somaiya.edu, shubhada.l@somaiya.edu, pauras.r@somaiya.edu

Abstract—Crowd management is a problem that needs to be solved in an optimized way. Several algorithms exist which save time to reach the destinations. This paper proposes the Genetic Algorithm as an alternative to designing a conventional controller via algorithmic means. The paper aims to showcase the proposal of a type of crowd simulator by the use of genetic algorithms to select for a simulator with desirable properties, such as low collision and staying on the path. The crowd simulator is a waypoint-based implementation that navigates a constantly populating crowd to their destination in an urban setting, described as a potential field. The crowd is steered by a common controller, which instructs each individual based on its genetic code. The genetic code is measured for Fitness based on the total collisions and the time spent across the potential field.

Index Terms— Genetic Algorithm, Crowd Controller, Individuals, Chromosomes, Waypoints, Potential Field.

I. INTRODUCTION

There are several useful applications to computer-generated simulations of crowds. Accurate crowd simulations are important to risks and evacuation planning. The entertainment industry has an abundance of applications including the video game industry, which needs crowds for a variety of types of games that are not necessarily simulation oriented.

The creation of realistic looking, believable behaving and real-time rendered crowds is an exigent task and is often handled via an application-specific way, with traditional algorithms used to design the simulators. We aim to demonstrate a new way to design a crowd simulator, via the use of modern, non-conventional algorithms.

Genetic Algorithms (GAs) are a type of evolutionary algorithms, that are modelled after natural selection, using biologically inspired operations such as gene mutation, crossover and reproduction, to generate high-quality solutions to optimization problems. The size of the population is dictated by the nature of the problem. We seek to showcase the design of a particular type of crowd simulator, consisting of a waypoint driven crowd in a field of obstacles and valid paths. The simulator, which we are calling a crowd controller, is responsible for directing the crowd based on their current positions and velocities, understands how to interpret its inputs based on the “genetic code”, which is a binary string representing an implementation of the controller. This implementation is simulated and the evaluation of simulation is associated with the genetic code. The genetic algorithm then selects these based on the evaluation value, to produce the next generation of simulators. This is iterated to produce an optimal simulator with desirable qualities. Natural crowds contain types of pedestrians such as hasty, relaxed and polite. These nuances make handling crowds a considerably complex problem. Designing one controller for all

of them is quite infeasible due to the random nature of genetic algorithms. A better solution is using multiple controllers within the simulated environment [7]. Trees and other hierarchical data structures could be used for collision detection. Since crowds have a large number of objects, we have to have trade-offs for accuracy and time required for computing collisions [8].

Our project aims to demonstrate this as an alternative approach for the design of crowd simulators, reducing the complexity of designing the controller by the use of computational power. A genetic algorithm is a search algorithm that searches a set of solutions that can optimally solve a problem [3]. The genetic algorithm creates a set of individuals or chromosomes as they are called in genetic terms and allow them to evolve to find a better solution to the problem. A single solution in the population is referred to as an individual or a chromosome. The individuals' fitness is its ability to solve the problem.

II. PROPOSED METHODOLOGY

The proposed system consists of two substructures: The Genetic algorithm and the Unity Engine.

A. The Genetic Algorithm

GAs are parallel while most of the other algorithms are sequential i.e. they search for a solution (not to be confused with an individual in a population which is also known as a solution or a chromosome) in a single direction. If the solution found by traditional algorithms is not globally optimal, they start searching in a new direction thus discarding all of the previous work. In GAs, since multiple individuals are working concurrently, they search for solution space in multiple directions. One of the solutions may arrive at an impasse. In such a situation, the algorithm starts to look for one in other possible horizons thus increasing the chance of finding an optimal solution. GAs are capable of changing multiple properties or genes simultaneously. Furthermore, a chromosome is capable of changing one gene while other chromosomes optimize other genes which help generate multiple good solutions [4].

The algorithm deals with the generation of genetic codes, mutation, crossover and selection. These genetic codes describe the crowd controller and the fitness values are obtained by simulation of the same. There are various selection methods like Roulette Wheel Selection, Steady State Selection to choose from. These come with their respective advantages and demerits and there always exists a trade-off between their efficacy and limitations. The type of selection proposed is Steady State Selection which involves selecting two best individuals as parents to generate two offspring. These generated offspring replace the two worst individuals in the population thus keeping population size constant. The Steady State GA enjoys an advantage of lower computational task over Generational GA. However, at the same time it takes large generations to converge.

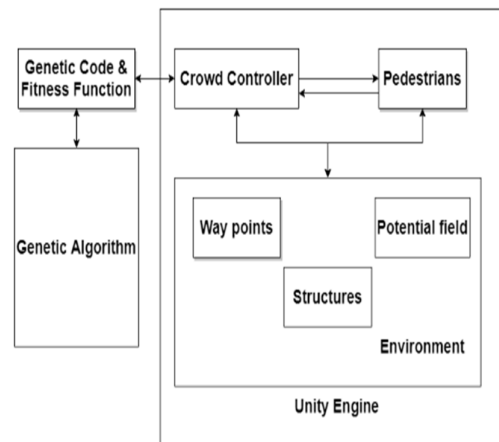


Fig 1. System Design: GA generates weights/genetic codes acting as an input to the crowd controller which is simulated by the Unity Engine by setting up an environment with waypoints, structures and a spawning system for pedestrians

B. The Unity Engine

The Unity Engine simulates the crowd controller by setting up an environment with waypoints and a spawning system for pedestrians which will be directed by the crowd controller as per its genetic code. The engine detects collisions of pedestrians, as well as providing them with updated waypoints. It also updates the fitness value

based on the potential field. Every few frames, the crowd controller calculates updates in the velocity for each pedestrian. At the end of the simulation, the engine calculates the final fitness value for the crowd controller.

1) Crowd Controller

The Crowd Controller is responsible for working with multiple things which entails spawning new pedestrians periodically, and keeping track of them during the simulation. A unit crowd controller controls a group of pedestrians. In our simulation, we are having multiple crowd controllers in one single generation. A pivotal part of it involves calculating the updated velocities for the pedestrians as they move through the environment and reach waypoints. It uses a weighted vector sum function, which takes the inputs: position, velocity, and the next waypoint from the pedestrian objects. It applies vectorized sum for each pedestrian given by the equation:

$$V_o = f(\sum w_i A_i) \quad (1)$$

- V_o is the output velocity vector for the current pedestrian.
- f is a thresholding function limiting the sum to valid velocities for a pedestrian.
- w_i is the weight obtained from the genetic code, with value in $[-1,1]$.
- A_i is a vector from the inputs and can be the position vector, velocity vector or next waypoint, for the current pedestrian, or other pedestrians.

$$\begin{bmatrix} V_{ox} \\ V_{oy} \\ V_{oz} \end{bmatrix} = w_1 \begin{bmatrix} A_{1x} \\ A_{1y} \\ A_{1z} \end{bmatrix} + w_2 \begin{bmatrix} A_{2x} \\ A_{2y} \\ A_{2z} \end{bmatrix} + w_3 \begin{bmatrix} A_{3x} \\ A_{3y} \\ A_{3z} \end{bmatrix}$$

Fig. 1. Expansion of Eq (1) : A_1 , A_2 and A_3 are current pedestrian's position, velocity and the next waypoint vectors respectively in the x, y and z axes which are being multiplied by weights w_1 , w_2 and w_3 generated by the GA thus producing the output velocities in each axis.

The above figure demonstrates the working of equation (1). It performs the weighted sum to calculate output velocity. Consider the simplification of equation (1) without the function f and only three inputs. V_{ox} , V_{oy} , V_{oz} are the output velocities of a pedestrian object in each of the components x, y and z respectively. Similarly, A_1 , A_2 and A_3 are the vectors for the current pedestrian's position, velocity, and next waypoint vectors respectively. This equation can further be extended till A_n to include more vectors depending on the factors being considered. Including additional features will increase the complexity of the algorithm. For the sake of simplicity, we are only considering three inputs. Think of these vectors as features. As mentioned earlier, w_i is a weight with which every A_i vector is being multiplied. w_i is nothing but a gene. Note that the crowd controller is responsible for outputting velocities for the i^{th} pedestrian. To calculate the velocity, it makes use of the weights w_i . w_i is in turn calculated by the GA. This process is applied to every pedestrian object to compute its velocities.

2) Pedestrians

The Pedestrians are 3-dimensional objects with required animations to interact with the environment, colliders etc. Colliders are used to detect collisions between pedestrians. This is brought about by the built-in methods provided by the Unity Engine. Pedestrians move as per their velocity, updating their position every frame as they move towards the direction instructed by the crowd controller.

3) Environment

The Environment refers to the simulated environment in which the pedestrians follow their designated path and produce the evaluation of the genetic code.

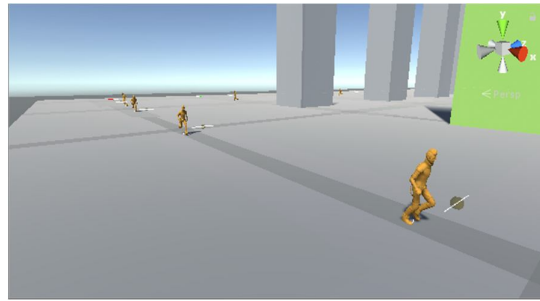


Fig. 3. A crowd being directed in an environment by crowd controller who is in turn receiving weights from the Genetic Algorithm

The above figure is a typical example of what the environment looks like and thus consists of structures like buildings and footpaths and an interactive crowd.

The environment consists of the following parts:

a) *Waypoints*

Waypoints are a method to describe the path a pedestrian takes when passing through the environment, as checkpoints to determine whether the path has been taken correctly, as well as provide guidance to the pedestrian as to where they should go next. There are a few special waypoints, the spawn point, where the pedestrians load in, and the goal, where the pedestrians are required to reach. On reaching the goal, they are removed from the simulated environment, and this is noted for the evaluation.

b) *Potential Field*

This is our way to describe the idea that each part of the surface that a pedestrian is allowed to walk on, need not be appropriate. For example, walking on roads should not be permitted, while walking on footpaths is desirable while one of the possible easier approaches is simply keeping the pedestrians from walking on the road. However, that defeats the purpose of controlling the crowd through GA. It could be desirable to make roads temporarily allowed such as for road crossings or using a traffic signal.

c) *Structures*

This refers to the objects present in the environment such as buildings, roads, footpaths, trees, poles etc. They do not serve any direct purpose but are either obstacles for pedestrians to avoid or surfaces to walk on.

4) *Structure of the genetic code*

The genetic code is in the form of a binary string and describes the weights of the crowd controller. It consists of genes, each describing one weight. There may be more genes added to add bias terms to the vector sum. The gene length will describe the precision of values available for that weight. 8 bits of precision allow for 256 values between -1.0 to 1.0, roughly 0.008 apart. 16 bits allow for 65536 values, roughly 0.00003 apart, and so on. This is to reduce the length of the genetic code, as well as avoid the use of floats directly which would have unused exponent bits since our values are limited to [-1,1]. For 8 bits the allowed range is [0, 255]. Clipping can be performed by subtracting 128 from the value to bring the value between [-128, 127] and finally dividing the whole by 128 to limit the range to [-1, 1].

5) *Calculation of the fitness function*

Fitness value measures how good or bad an individual is in performing the task in question. Higher the fitness value, finer is the chances of being chosen as a parent while 'unfit' individuals in the population are replaced by newly generated children. The children undergo the process of 'crossover' and 'mutation' to make them diverse from their genitors i.e. introduce variety in the solution space to explore different avenues for solving the problem. This process is repeated for several iterations. A larger and diverse population increases the likelihood of finding good solutions. But there exists a trade-off as it may significantly increase the run-time and memory requirements. While the exact proportions are not fully determined, these factors will determine the fitness of each simulation:

- No. of pedestrians that reached the target.
- No. of waypoints covered by pedestrians that did not reach the target. Total of collisions of all pedestrians with each other or the environment.
- Time spent by each pedestrian in positive potential and negative potential fields.

III. RESULTS

The final fitness obtained after running simulation for 10 generations and 20 crowd controllers per generation was -4736. We believe that running the simulation for longer generations and larger crowd controllers per generation could produce satisfactory results. Thus, an improvement in the fitness function was observed but no workable controller could be produced given limited resources and time constraints.

IV. CONCLUSION

Designing Crowd Simulations is a labyrinthine task, and making conventional algorithms to make one takes significant time, resources and human effort. Our results from running the simulation show that a GA shows potential of producing comparable crowd simulator, with significantly increased runtime, larger generations, more crowd controllers per generation or maybe better selection of parameters to produce the genetic code that produces the required behavior.

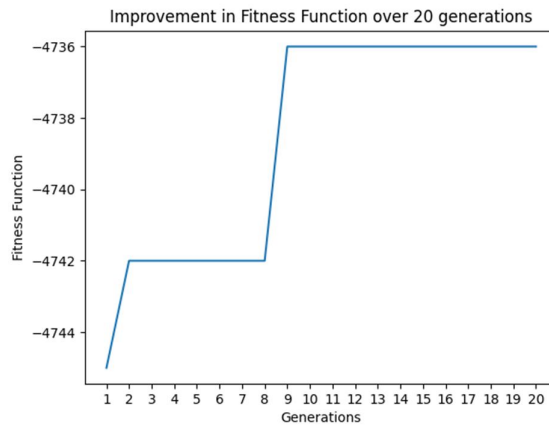


Fig. 4. An improvement in fitness function can be seen over 20 generations with 10 crowd controllers per generation

V. FUTURE SCOPE

The fitness function could be improved by considering different parameters or methods of generating fitness. Other types of selection methods and different designs of crowd controller could be investigated. In our system, we are restricted to simulating and continuously monitoring the pedestrian crowd. The system can be extended to include additional complex components like the inclusion of vehicles on the roads, manholes and barriers to make the environment more practical. Also, pedestrians have restricted movements. They can only walk. We could provide more degrees of freedom like the ability of jumping and climbing stairs. In our project, as stated, we are using Steady State Selection. However, the performance of the system could be compared to different types of selection methods to solve this particular problem. Other aspects which could make the system more realistic is by including a variety of types of pedestrians and designing controllers for vehicles along with the one for the pedestrians.

REFERENCES

- [1] Sharma, Poonam & Wadhwa, Amit & Komal, Ms. (2014). Analysis of Selection Schemes for Solving an Optimization Problem in Genetic Algorithm. *International Journal of Computer Applications*. 93. 1-3. 10.5120/16256-5714.
- [2] Review of Selection Methods in Genetic Algorithms Nisha Saini Research Scholar, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, *International Journal Of Engineering And Computer Science* ISSN:2319-7242 Volume 6 Issue 12 December 2017, Page No. 22261-22263 Index Copernicus value (2015): 58.10, 76.25 (2016) DOI: 10.18535/ijecs/v6i12.04
- [3] Thede, Scott. (2004). An introduction to genetic algorithms. *Journal of Computing Sciences in Colleges*. 20.
- [4] BAJPAI, PRATIBHA & Kumar, M.. (2010). Genetic Algorithm - an Approach to Solve Global Optimization Problems. *Indian Journal of Computer Science and Engineering*. 1. 199-206.
- [5] Anand, Smit & Afreen, Nishat & Yazdani, Shama. (2015). A Novel and Efficient Selection Method in Genetic Algorithm. *International Journal of Computer Applications*. 129. 7-12. 10.5120/ijca2015907067.
- [6] A. Yadav and C. Nuthong, "Traffic Signal Timings optimization Based on Genetic Algorithm and Gradient Descent," 2020 5th International Conference on Computer and Communication Systems (ICCCS), Shanghai, China, 2020, pp. 670-674, doi: 10.1109/ICCCS49078.2020.9118450.
- [7] N. Pelechano, J. M. Allbeck, and N. I. Badler. 2007. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '07)*. Eurographics Association, Goslar, DEU, 99–108.
- [8] Tecchia, Franco & Loscos, Céline & Chrysanthou, Yiorgos. (2002). Visualizing Crowds in Real-Time. Tecchia, F. and Loscos, C. and Chrysanthou, Y. (2002) Visualizing crowds in real-time. *Computer Graphics Forum*, 21 (4). pp. 753-765. ISSN 01677055. 21. 10.1111/1467-8659.00633.