Genetic Algorithms For Obstacle Avoidance: Multiple Approaches

Assaad Oussama ZEGHINA and Oussama ZOUBIA

Department of Computer Science, University Mustafa Benboulaid Batna 2

Abstract

Genetic algorithms are bio-inspired methods which are based on Darwin's theory of evolution by natural selection, they represent the idea of passing the genes of the fittest members of a population to their offspring; these offspring will then replace the older less fit members of the population. Genetic algorithms can be introduction methods to collective intelligence and artificial intelligence because they are quite simple and they demonstrate the concepts of intelligence in reduced complexity, its usage ranges from simple optimization problems to complex and advanced robotics. In this work we first introduced the concept of genetic algorithms, how they are used, their different components and parameters and how they are represented, then we defined a famous problem known as the Path Finding/Obstacles avoidance problem. Different algorithms were proposed to solve such problem that we tested and had inspiration from. After understanding and testing the algorithms on different problem models we proposed a method that can solve it, it surpassed the other methods in complex models of the path finding problem but failed to catch up in smaller and simple problem models, we finished the work by visualizing and discussing the test results that we obtained from the comparison of different methods and our method, we concluded the work by mentioning a future work and a developed version of our proposed method.

Keywords– Artificial Intelligence, Collective Intelligence, Intelligent Agents, Genetic Algorithms, Game.

I. INTRODUCTION

Artificial intelligence methods have always been inspired from nature, the first ideas of AI methods are all inspired from the biological and psychological (behavior) mechanisms of the living creatures, they were based on the observation of humans at their environment and were continuously innovated and updated from relatively recent discoveries and theories presented by the scientific community. We can see the influence of the prior approaches even in today's state of the art methods in the new era of artificial intelligence which adapt machine learning and deep learning methods to achieve human level understanding and interactions with the environment, it could be seen in methods like Convolutional neural networks which are a class of neural networks heavily inspired from the

analysis of the visual inputs by the human brain. Nowadays, the first invented bio-inspired methods are being used in parallel with the recent ones to achieve high efficiency when tackling complex problems that require interactions with the environment; we can consider reinforcement learning as a concrete example of such applications.

One of the fields that were first bio-inspired is the evolutionary algorithms that consist of optimizing certain parameters by iteration, and one major sub field and the most popular among them is Genetic algorithms which our article focuses on. Genetic algorithms (GA) represent a computational model having its roots in evolutionary sciences and it's usually used for optimization problems to acquire the best fit-to-the-problem solution, and unlike traditional mathematical optimization methods they do not improve a single solution but a set of solutions, a so-called population (each individual of the population can be considered as a hypothesis). The GAs produce successor hypotheses (individuals) by mutation and recombination (crossover) of the best currently known hypotheses. Thus, at each iteration a part of the current population is replaced by offspring of the fittest individuals in the current population.

In this work we will give a brief introduction into this field and demonstrate its roots and relations with collective intelligence and AI in general, and we will test a couple of optimization techniques to solve one of the most famous problems in AI which is path finding and obstacle avoidance using a genetic algorithm and an evolutionary approach.

II. Background Knowledge

Before we fully define the problem and dive into our proposed method we will dedicate the current section to give an overview of the field which is Genetic algorithms and define its roots and some of the different methods used in it and its relationship with collective intelligence and artificial intelligence.

In the second part of this section we'll talk about the tool we'll be using to implement our proposed method which is the Unity game engine, first we'll discuss the structure and the philosophy of the game engine, its components and its most outstanding features, then we finish by highlighting the standard project and files structure for the implementation of genetic algorithms

A. Artificial Intelligence

Artificial intelligence is a branch of computer science that deals with the development of algorithms and techniques that can simulate or even recreate the capabilities of the human mind, giving the machine the ability the do complex and advanced tasks. The first insights of this domain goes back to second World War with the creation of The Bombe machine by Alan Turing in the purpose of deciphering the Enigma messages of the Germans, According to Turing, a machine that could interact with humans without the humans identifying it as a machine, could be said to be "intelligent", but the term artificial intelligence was first introduced only in the year 1956 by John McCarthy in Dartmouth Conference, the idea of AI inspired the researchers around the globe to start taking it as a proper research domain. In the 1960s, researchers gave special importance to developing algorithms to solve mathematical problems and geometrical theorems using AI methods. In the late 1960s, computer scientists introduced Computer Vision, and started developing machine learning for robots. However, computer scientists found it incredibly difficult to create intelligent machines which led to a slowdown in the progress of this field (what's known as AI winters).

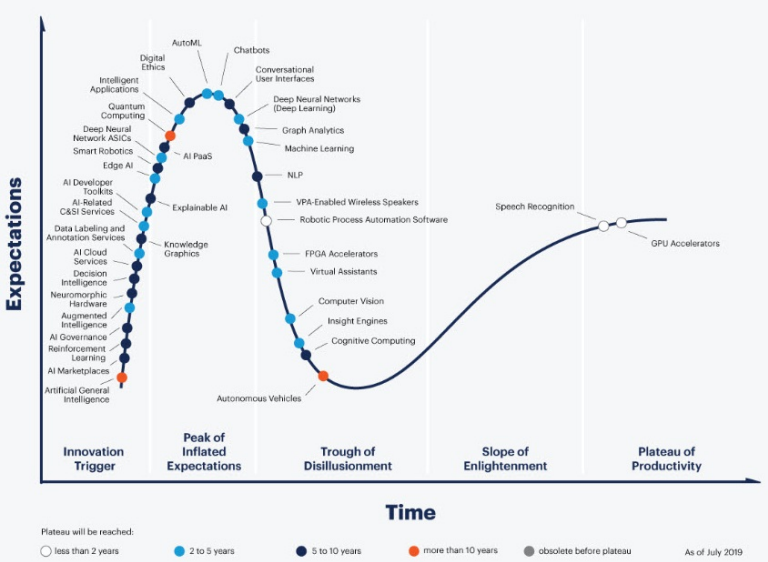

Fig. 1. Major advancements in AI since its birth. [1]

In the late 1990s, AI development and research came back on due to major improvement in computer hardware along with the plans of the Japanese government to develop a fifth generation computer, and the uprising interest by the American corporations in machine learning. In the early 2000s AI continued to develop and progress thanks to an even bigger advancement in computer hardware precisely in Graphical processing units (GPU).In the last 15 years major economical companies took advantage of AI and machine learning to their huge commercial advantage, other than processing user data, these companies are continuously and constantly pushing the advancement in AI research and they are considered its pillars since all major advancements and state of the art methods are developed by laboratories associated with these companies. Currently AI is used in various domains and it consists of many

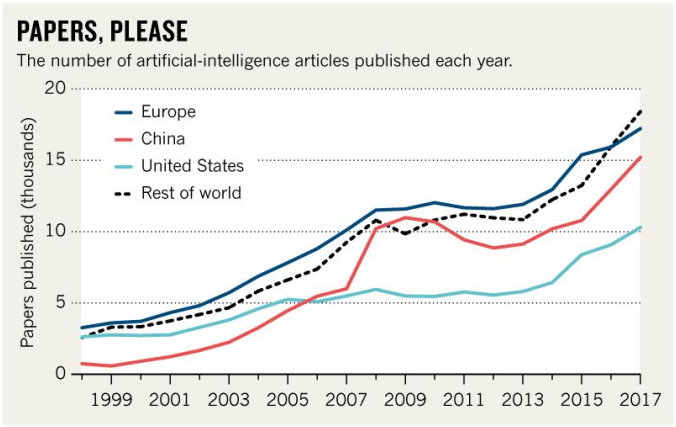

Fig. 2. Number of AI articles Published in recent years. [9]

sub-fields using a variety of techniques, such as:

Machine Learning – e.g. supervised, unsupervised learning, reinforcement learning.

Evolutionary Computation – e.g. genetic algorithms, genetic programming, they are another bio-inspired application of AI which are used for optimization problems.

Vision: object recognition, objects localization, image understanding.

Social collective intelligence – e.g. agents coordination, prediction, parameters optimization.

Robotics – e.g. intelligent control, autonomous exploration.

Expert Systems – e.g. decision support systems, teaching systems.

Natural Language Processing – e.g. machine translation, speech recognition and production.

Planning – e.g. scheduling, game playing.

Many categorizations have been proposed by different scientists, one of the interesting categorization is presented in figure 3, and the sub-fields that are most relevant to the problem discussed in this work are the ones highlighted in red, the **evolutionary computation** and **collective intelligence**.

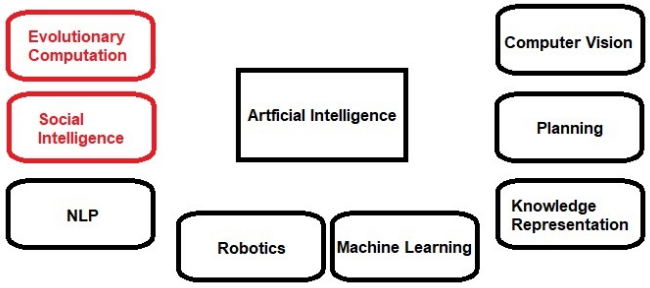

Fig. 3. Categorization of AI

B. Collective Intelligence

Collective Intelligence's effects can be observed in a variety of aspects in life and nature such as Sociobiology, Psychology, Economics and Computer Science… research surrounding this area did not flourish until recent years with the current advances in social technologies and AI although its roots go back to Aristotle's era. Pierre Levy defined collective intelligence as "a form of universally distributed intelligence, constantly enhanced, coordinated in real time, and resulting in the effective mobilisation of skills. I'll add the following indispensable characteristic to this definition: the basis and goal of collective intelligence is mutual recognition and enrichment of individuals rather than the cult of fetishised or hypostatised communities " [6].Collective Intelligence is sometimes referred to as the "Wisdom of Crowds," a term was introduced by James Surowicki's 2005 in his book titled with the same name [11], where he argued that a group of people could achieve better decisions and predictions rather than a single individual.

However, in the era of digital innovation, with the advancements of artificial intelligence and increasing interconnectivity, a new frontier in Collective Intelligence has emerged – Social Collective Intelligence. This is defined

as an emergent property where a group of agents act together to combine their knowledge and insight.Various examples of collective intelligence can be seen within the natural world. One of the most notable is the way ant colonies work together to build nests, collect food and defend against predators. Other examples include migrating of birds, animal herding, bacterial growth, fish schooling and microbial intelligence.

Collaborative Learning is the main goal of collective intelligence and data processing skills in general. Agents add new beneficial knowledge to the common knowledge base. They convey their knowledge in particular contexts in a form of clear propositions, or narratives, or visuals, etc. the knowledge that the agents contributed to the other members of the population is taken in consideration to determine the best decision possible. While CI is built around the idea that groups of citizens or experts can be smarter and more effective than individuals, scaling CI initiatives can be difficult. This is largely because of the transaction costs involved in CI. Unlike the more automated processes of AI which needs minimum or no human intervention. If implemented effectively, automation through AI could indeed save time and effort, leading to what we call “Augmented Collective Intelligence” [13].

As mentioned in their paper [10], Emil Scarlat and Iulia Maries state that “Collective intelligence improves competitiveness within organizations (or a group of intelligent agents) in the context of a global problematic situation, and can represent a critical factor in the organization’s development”, this statement fall into the same definition as a genetic algorithm.

C. Genetic Algorithms

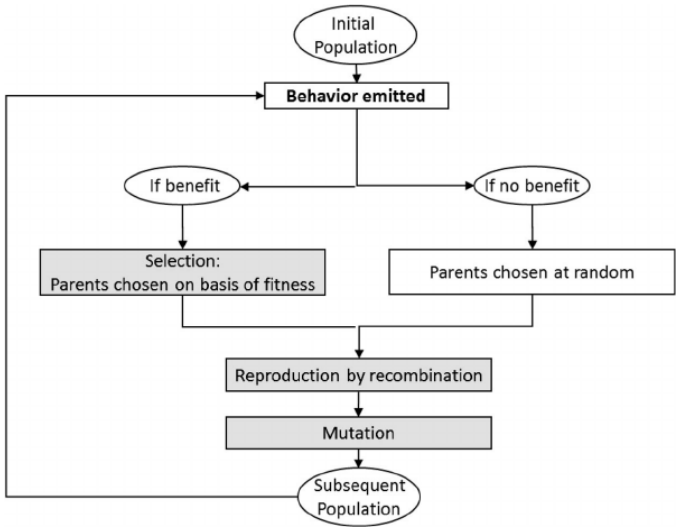

Fig. 4. Schematic diagram of the evolutionary theory. At every time tick a behavior is emitted from the population of potential behaviors. If the behavior results in benefit to the organism, then the left pathway in the diagram is followed, otherwise the right pathway is followed. Shading identifies the three rules of the evolutionary theory. [7]

Charles Darwin (1809 – 1882) is famous for his work on evolution by natural selection. The idea is that fitter individuals will naturally tend to live longer and produce more children, and hence after many generations

a population will automatically emerge with good innate properties. This has resulted in brains that have much structure, or even knowledge, built in at birth. This gives them at the advantage over simple artificial neural network systems that have to learn everything. Computers are finally becoming powerful enough that we can simulate evolution and evolve good AI systems. We can now even evolve systems so that they are good at learning. A related field called genetic algorithms has had some success in evolving programs, rather than programming them by hand. Evolutionary algorithms have three main characteristics:

1) Population-Based: Evolutionary algorithms are to optimize a process in which current solutions are bad to generate new better solutions. The set of current solutions from which new solutions are to be generated is called the population [3].
2) Fitness-Oriented: If there are some several solutions, how to say that one solution is better than another? There is a fitness value associated with each individual solution calculated from a fitness function. Such fitness value reflects how good the solution is [3].
3) Variation-Driven: If there is no acceptable solution in the current population according to the fitness function calculated from each individual, we should make something to generate new better solutions. As a result, individual solutions will undergo a number of variations to generate new solutions [3].

Genetic algorithms (GAs) have been firstly introduced by Holland in the 1960s, but it was not possible to implement them due to high computational complexity, until the 1990s with the major achievement in the field of computer hardware. In 1993 a paper was published [2] that gave a general presentation of GAs, some mathematical analysis about how GAs work and how best to use them, and some applications in modeling several natural evolutionary systems, including immune systems.GAs were able to even successfully solve problems concerning molecular dynamics where conventional techniques had failed.

1) Definition: Genetic Algorithms are a random search-based optimization techniques based on the principles of Darwin's theory of natural evolution and natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. GAs are a slow gradual processes that make slight changes to the solutions iteratively until reaching the optimal or near-optimal solution to complex problems that would require much longer time to be solved with other techniques.

GAs start with a pool of population (group of agents, parameters, hypotheses) randomly initialized, and then a series of breedings (crossover) and mutations is applied to the successive population (generations of populations) to produce the optimal offsprings (the optimal solution), the selection of the optimal individuals for breeding is based on a function called a fitness function. Fitness functions associate to each member of the population an index value based on its fitness to the problem or how close it is to the optimal solution, these values are called "fitness scores". This whole process can be summarized and demonstrated in figure 5.

Genetic algorithms gained so much popularity due to their high performance and efficiency and their ability to deliver good-enough solutions in a fast-enough period of time compared to other methods; they provide a list of multiple solutions to the problem unlike other optimization methods. With good population initializations and a good definition of the fitness function the probability of reaching a solution (if there is one) is almost guaranteed. On top of that the genetic algorithms are able to solve NP-Hard problems with less specification of the solution method

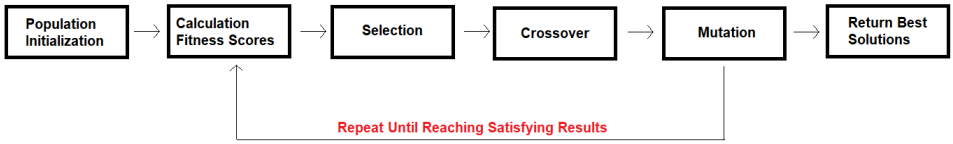

Fig. 5. The Process of Optimisation Using a Genetic Algorithm

(e.g. maze solving problem). One last major advantage over the classical mathematical optimization technique such as gradient based optimization techniques is its immunity to suffer from local optima problems (figure 6).

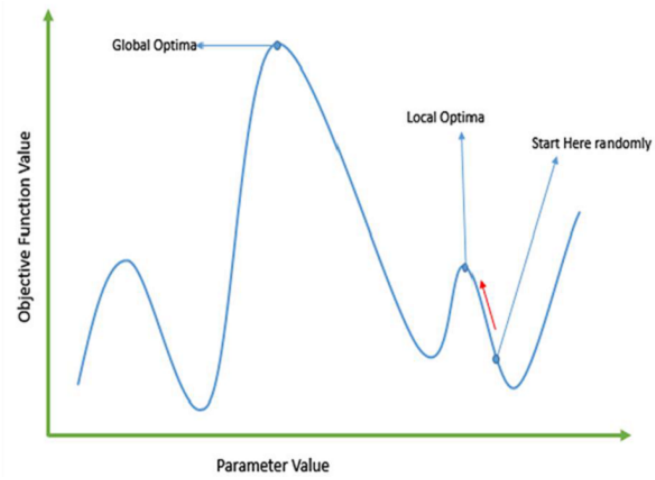

Fig. 6. Problem of Local Optima encountered by Gradient based optimisation techniques. [12]

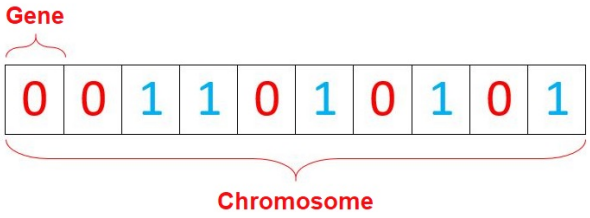

Fig. 7. Genes and Chromosomes Represenation

2) Genes: One of the most critical decisions to make while implementing a genetic algorithm is choosing a proper representation of our chromosome and genes for the individuals; these chromosomes and genes mimic the functionality of real biological chromosomes and genes, each gene contains semantic information that define the behavior of our individual,. The chromosome can be defined as a structure that contains all different genes for

the individual Figure 7. There isn't fixed rule to choose or defining the gene representation and it is all problem specific. There are various representations of the gene in a GA; we'll represent some of the most commonly used:

1) Binary representation: this representation is used when the problem is simple and the problem space only consists of Boolean decision variables (yes or no), our chromosome in this case will only contain genes of only 2 possible values. When we are faced with a more complex problem we may need to use other advanced representations of the chromosome7.
2) Integer representation: in this representation the genes are defined as integer values for example: genes of a character could be (1: up, 2: down, 3: left, 4: right)8.

2	4	5	1	0	2	8	0	6	5

Fig. 8. Integer Representation of Genes and Chromosomes

3) Real value representation: In some cases we may need to define our genes using continuous or real values rather than discrete values, this representation is considered the most natural representation9.

0.32	0.9	1.5	1.3	0	1.19	0.5	1.7	2.2	1

Fig. 9. Real Value Representation of Genes and Chromosomes

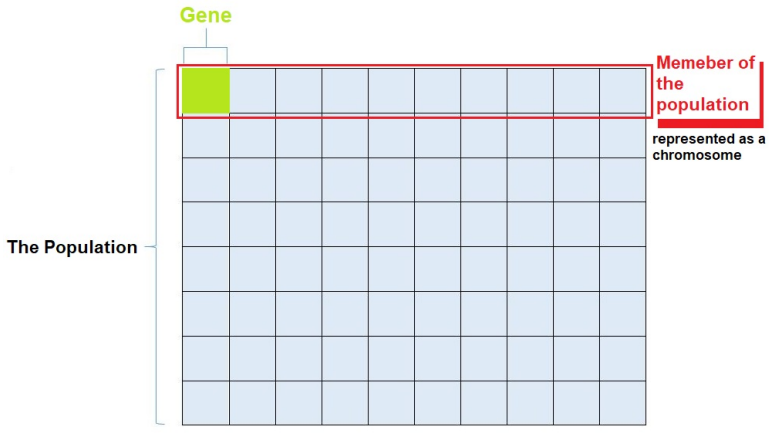

Fig. 10. A population can be considered as a group of potential solution or chromosomes, each is a group of genes

3) Population: The population can be defined as set of all the possible solutions (individuals) to the problem. It is similar to human population but instead of humans, we have Candidate Solutions to the problem, the population

can be structured as a 2 dimensional array of size of the population and size of the chromosome(Figure 10). There are multiple methods to initialize a population in a genetic algorithm:

1) Random initialization: which is the most commonly used initialization method that uses completely random values.
2) Heuristic initialization: it uses a known heuristics for the problem(the definition of a heuristic initialization function is required).

The set of each new population is called a generation, it will replace the old population members with new ones (offspring), bred from the most fit members of the prior population based on a fitness value defined by the fitness function, with each generation created the population members will be more fit to reaching the solution.

4) Fitness Function: The fitness function is the core of a genetic algorithm, it takes an individual and determines how well it fulfills whatever criteria the algorithm is optimizing for, a fitness score value is returned by the function which determines if an individual is good enough to be selected and passed to the next generation. This fitness function is calculated for each individual in the population across all generations and thus, we need to implement it in an optimized way because it can affect the speed of our GA In some cases, the complexity of the problem makes it really hard to calculate the fitness function, in such cases it is preferable to calculate a fitness approximation for our individuals.

Defining a fitness function for the individuals is the hardest part when formulating a genetic algorithm and it all depends on the problem that we are trying to solve; therefore there is not a constant way to define it. There are multiple criteria which need to be satisfied by the fitness function that we need to take in consideration:

- The fitness function needs to be defined clearly to be able to be understood clearly.
- It should be implemented efficiently and optimized. Because it could become a bottleneck and affect the whole genetic algorithm.
- It should measure how a given an individual is solving the problem.

5) Parent Selection: Parent selection is the phase in Genetic algorithms where a parent (individual) is chosen and allowed to breed and pass its genes to the next generation; the selection is based on the fitness score calculated by the fitness function which we discussed in the previous chapter. This phase is very important and crucial to the convergence of the genetic algorithm as choosing good parents will result in creating fitter and closer generation to the solution.

In some cases, an extremely fit individual can take over the population, this will create new solutions close to each other in the problem space, and as a result it would affect the diversity in the population. Having diverse solutions in the population is necessary to the convergence of the GA; this whole process is called “Premature convergence”. There are many Parent selection methods(Figure 11) to choose from:

- Fitness Proportionate: this method is one of the most commonly used parent selection methods, its idea is that the individuals has a probability of being selected which is proportionated to their fitness score which means that the individuals with higher fitness are the ones to be selected. There are two implementation of this method: Roulette Wheel Selection and Stochastic Universal Sampling.

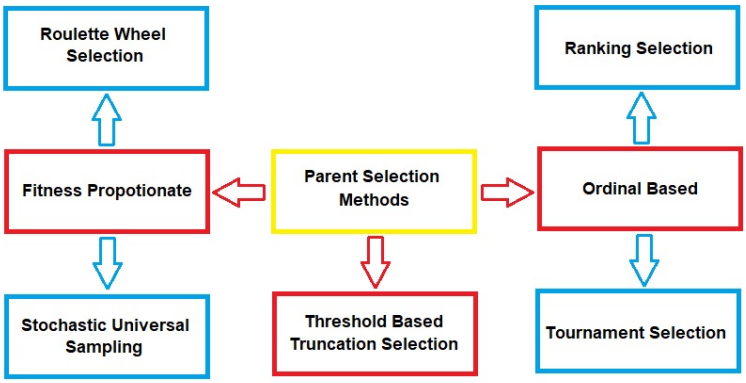

Fig. 11. The Choice of a Parent Selection Method depends on the given problem, the choice is considered a Hyper-parameter, and can only be determined by tests

- Ordinal Based Selection: we have two types of ordinal selection which are: Tournament Selection: tournament k, in this selection we choose a k value from the population randomly and select the best of these individuals to become a parent, the process is repeated to select the next parent Rank selection: it works by first giving the individuals a ranking and then every individual receives his fitness from this ranking, this selection is used to avoid premature convergence.
- threshold based (truncation selection): in this selection the individuals are sorted based on their fitness score and those among them who have the highest score are the ones selected.

6) Crossover: The crossover operation is similar to the biological reproduction operation where both parents combine their genes to create offspring, in GA we don't use a combination of the parents' genes but there are a variety of methods to choose from which would depend on the problem at hand. This operation is very important as it will result in generating fitter and closer offspring to the solution. There are multiple crossover methods to apply to the parents' genes that we discuss below:

- One point crossover: in this method we pick a random point, the genes that are on the right (tail) of that point are swapped between the two parents, as a result we would have an offspring which has genes from both parents(Figure 12).

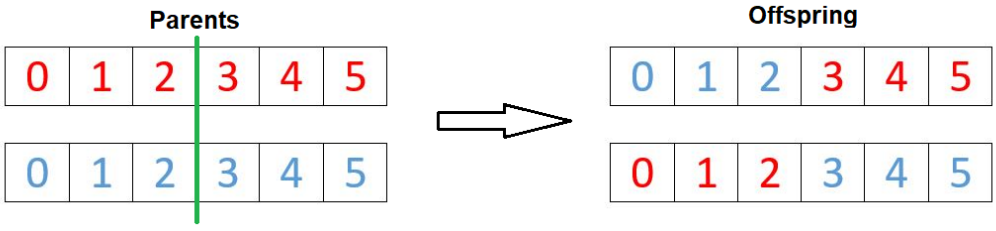

Fig. 12. One Point Cross Over Demonstration

- Multi point crossover: similar to the one point crossover but instead we pick two points; the genes which are between the two points would be swapped between the two parents(Figure 13).

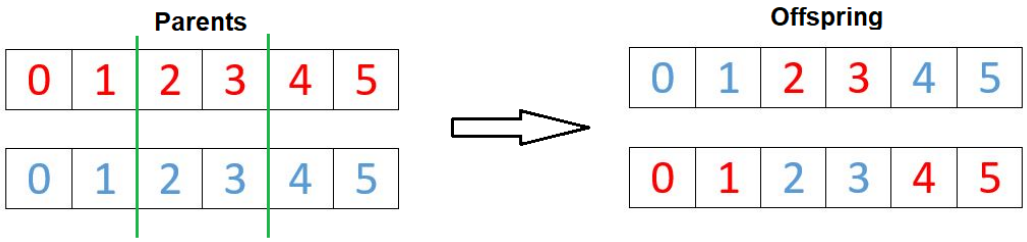

Fig. 13. Multi Point Cross Over Demonstration

- Uniform crossover: in this method we do not pick a point or points to divide our chromosome but instead we treat each gene separately, deciding whether to swap the parents genes is left to a probability of a coin flip(Figure 14).

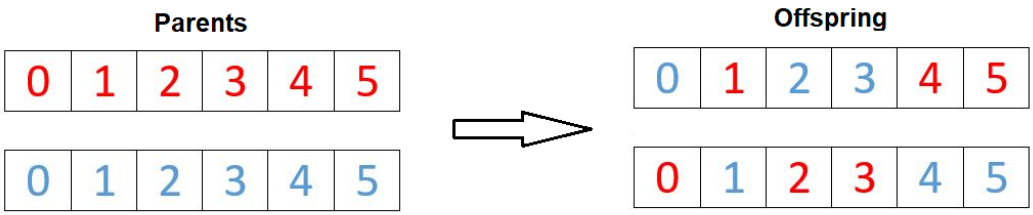

Fig. 14. Uniform Crossover Demonstration

- Beside these crossover methods, there are other such as: Whole Arithmetic Recombination, Davis' Order Crossover (OX1)...

7) Mutation: Mutation can be defined as a probability rate used to change the initial value of the gene and it is primarily used to add more diversity in the population and prevent falling into local minima problem and premature convergence, the mutation rate is usually very small, if a high rate is used then the genetic algorithm may not converge and fall into a random search problem. Mutation is essential to the convergence of the algorithm and it is used to explore the search space. We will demonstrate some of the most commonly used mutation methods below:

- Bit Flip Mutation: We simply pick one or more random bits and invert them. This is used for binary genes.
- Random Resetting mutation: It is an extension of the bit flip mutation. We select a random value from the range of values and assign it to a randomly chosen gene.
- Swap Mutation: We select two random genes, and we swap their values. This is common in permutation based encodings.

- Scramble Mutation: We select a set of genes and scramble them on the chromosome; the chosen genes may not be contiguous
- Inversion Mutation: We select a set of contiguous genes on the chromosome and reverse their order.

Choosing how to apply the mutation to the individuals depends on the representation of the genes and the defined problem.

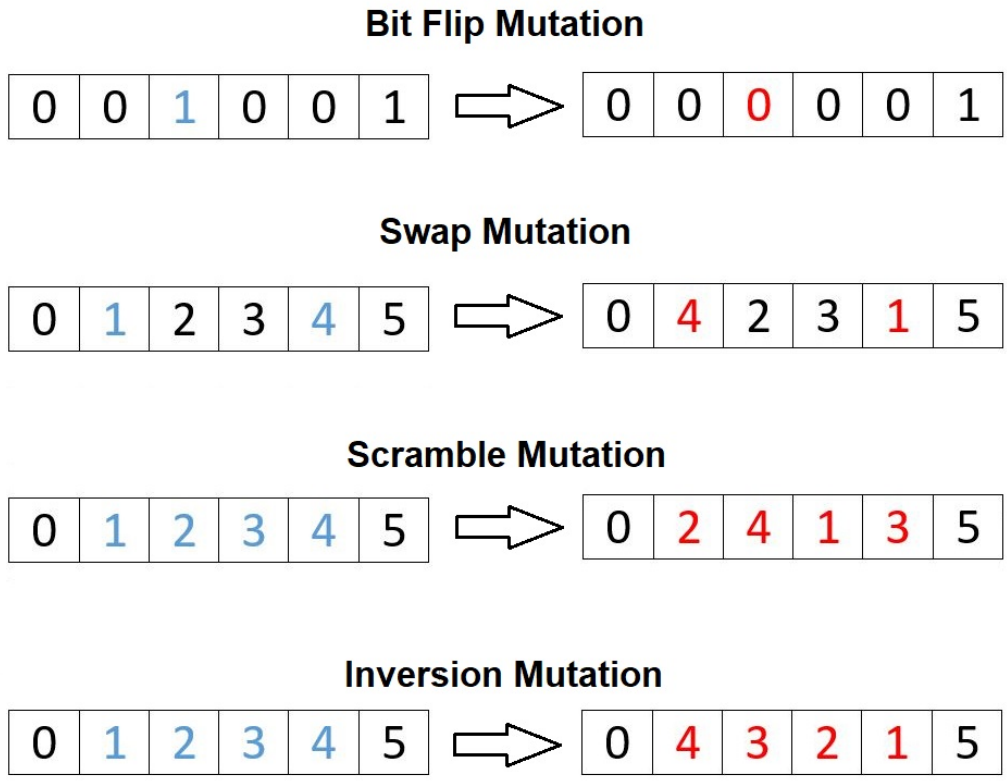

Fig. 15. Mutation Methods Demonstration

D. Unity Engine

The Unity engine is a game engine developed by Unity Technologies, which gained popularity in the recent years due to its simplicity and power when it comes to developing video games for desktop platforms, consoles, mobile and web; it can also be used as a tool to visual deep learning and machine learning models... It gain all that popularity because it makes the developing phase fairly easy compared to other game engines out there (such as Unreal Engine 4) and it offers a set of features that makes it stand up (like supporting more than 25 platforms), the scripting language that Unity use is the C# programming language, an object oriented language. In this section we will briefly present the Unity Engine, its simple interface and some of the features that it offers:

Unity engine interface(Figure 16): when opening a new unity project, you are welcomed by a simple user friendly interface, a default unity interface layout contains multiple windows:

- The Hierarchy window: which contains all of our game objects in the scene.
- The project window: which contains the different files of our unity project (scenes, assets, prefabs, scripts…).
- The scene window: which contains the scene or the visual representation of all the game objects and assets that are in our hierarchy window.
- The game window: which is the execution of the scene in a game mode view.
- The inspector window: when selecting a game object in the scene window or the project window, all of its properties (including its position, physics, scripts attached to it…) are shown in the inspector window.

Unity engine features:The unity game engine is powered by a set of features that makes it easy to use which we mention:

- GameObjects: they are the building bloc of a unity game scene which could be a character, a building, a ground or in case of neural networks a neuron.
- Scripts: scripts are C-sharp language code files that define the behavior of a GameObject.
- Scene: they are a small executable or playable parts of a game.
- Prefabs: they are copies of the GameObjects stored for reusing.
- Rigidbody: a Rigidbody is a parameter given to a GameObject that gives it a physics-like interaction (mass, gravity…).
- BoxCollider: it's a parameter given to a GameObject that gives it the ability to interact with other game objects (preventing GameObjects from passing through each other).

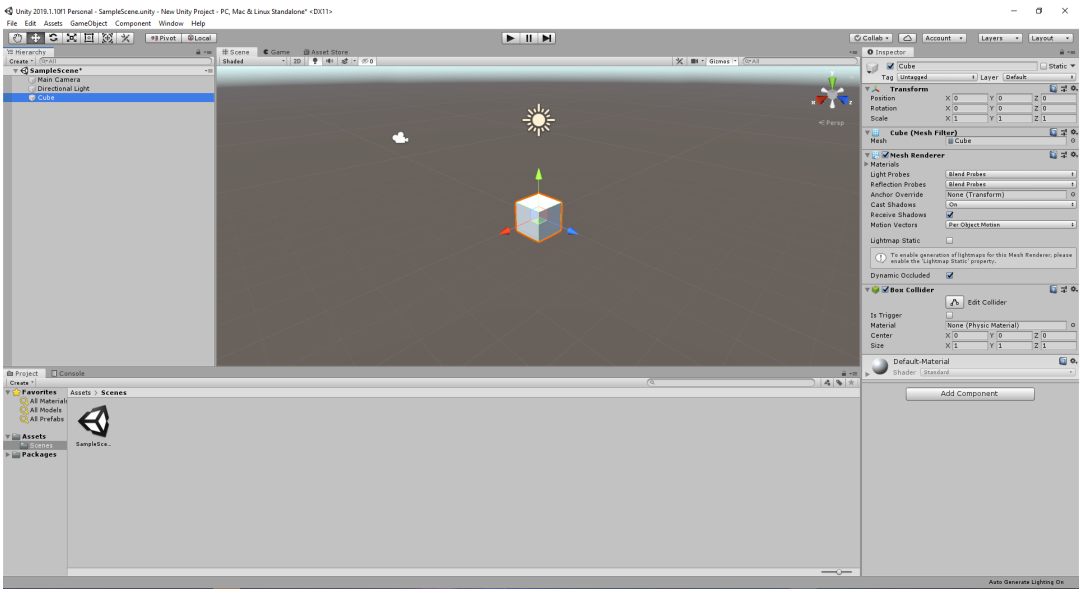

Fig. 16. Unity Game Engine Interface

E. Genetic Algorithms In Unity Engine

The field of evolutionary computing examines intelligence through environmental adaption and survival; it attempts to simulate the process of evolution by implementing concepts such as selection, reproduction and mutation. In short, it endeavors to replicate the genetic process involved in biological evolution computationally. Genetics concentrates on the transmission of traits from parents to offspring. Evolutionary computing simulates the combining of chromosomes to produce offspring, a population is created with anywhere between 100 and many thousands of individual organisms where each individual is represented usually by a single chromosome. The number of genes in an organism will depend on its application. The population is put through its paces in a testing environment in which the organisms must perform at the end of the test.

Each organism is evaluated on how well it performed; its level of performance is measured by a fitness test. This test might be based on how fast the organism completed a certain task. How many weapons it has accumulated or how many human players it has defeated in a game. The test can be whatever the program considers the best judgment of a fit organism. Once the test is complete the failures get killed off and the best organisms remain. These organisms are then bred to create new organisms which make up a new second generation, once breeding is complete first generation organisms are discarded and the new generation is put through its paces before being tested again and then bred.The process continues until an optimal population has been achieved.

In recent year, The Unity engine has gained a lot of popularity in the field of applications of AI and machine learning algorithms, due to its reduced complexity compared to simulation programs, and to the control that it provides to the programmer.

Applications of genetic algorithms in unity engine can be applied in so many models organized in different project templates, the one that we will be using consist of two main parts; code and visuals, visuals will contain the files used in the scene building, code files will be the part used to apply the GAs.

Starting by visual files, they are usually prefabs and GameObjects, either created outside of unity and imported or created by a drawing/design software then imported into the project, after that they are used to represent the individuals of the population or the environment.

Code files are the main part, and where most of the work is done, we can mention three main files: Population manager, Brain and the DNA files. We start off by the DNA file, which is as its name indicates the file containing the genes for the population members; it can be viewed as the file that contains the parameters or the solution to the problem in a data structure as a table, list or even simple variables. This file is usually bound to the individual prefab, and it is usually a class that every population member initiate from, it contains in addition to the DNA, some useful functions, like the ones used to initialize the chromosomes, or crossover functions, and may even contain the fitness function, but it is less probable, because we usually implement it in the population manager.

The second file is the brain which is responsible for movement of the agent, it is always bound to the agent's prefab, and it contains variables used by the population manager to calculate the individual fitness score, variables such as the time the agent lived, or the distance it traveled. The file also contains functions usually dedicated to its interaction with the environment and movement if the problem requires it.

The last file is the population manager which is the main engine behind the GAs implementation; it contains the function responsible of creating the population, the calculation of the fitness score for the population members, the selection of the fittest members, the mutation and the creation of new generations. The population manager is usually bound to a game object called with the same name, and it is usually set invisible. Since it is used to control the population, so we pass it the prefab and the brain and the DNA codes, and it initialize the population and call the functions contained in the other two files.

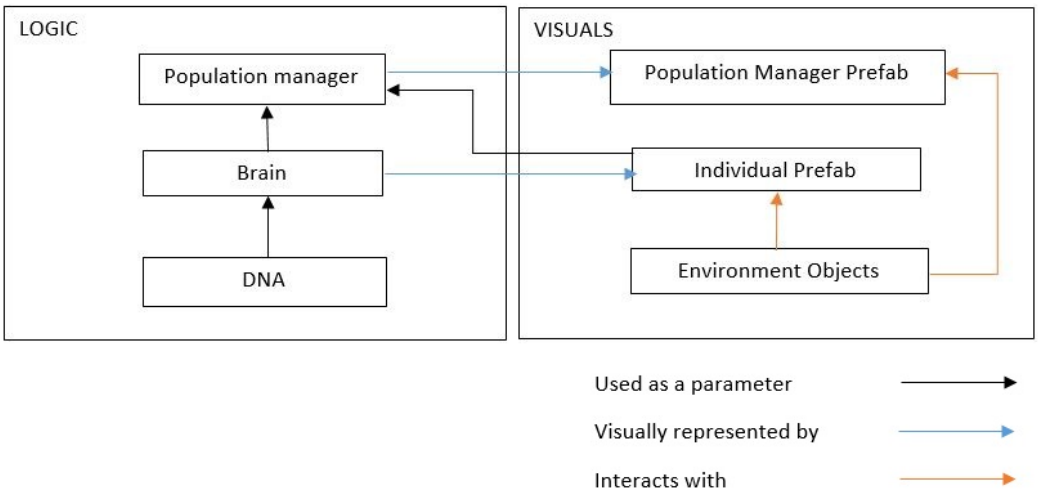

Fig. 17. Diagram summarizes the implementation of Genetic Algorithms in Unity Engine and the relation between the Components.

III. Related Works

The problem that we are aiming to solve in this project is known as Obstacle avoidance/Path finding; basically it consists of a population of agents trying to find a path to a destination (solution) while avoiding the obstacles that come across their way; it is a simple problem but it requires all the characteristics of Genetic algorithms that we discussed in the previous chapters (such as crossover and mutation), this simple problem is a good example at demonstrating the flexibility and potential of Genetic algorithm at converging and reaching the solution. Solving this problem using simple methods or algorithms is a near to impossible task, since we are faced with the difficulty of representing the problem and if we try to represent it using primitive approaches it would alter the nature of the initial problem; since the number of possible steps that could be taken by the agent is near infinite, thus the use of a sophisticated approach is a requirement; and that is the reason that Genetic algorithms are the go-to approach in such case.

Many variations were proposed the solve this type of problems; some of them consist of a population of agents moving randomly in the problem space until reaching the solution; while others take advantage of the information collected while exploring the problem space to reach their solution, which can lead to a faster convergence.

A simple approach was proposed by “jodiDL” [4], the results were acceptable and applied on the benchmark that he presented, the algorithm could always find a path to the target. The fitness function that he defined consists of calculating the distance between the agent and target.

Another approach that captured our interest is a work of “nj-AllAboutCode” [8] who was able to get some really good result in really short period of time; his approach focused on rewarding the agents for getting closer or finishing and penalizing them for hitting an obstacle by multiplying the fitness score by either a value larger than 1 in the case of a reward or a value between 0 and 1 in the case of a penalty.

The last approach that really left us intrigued was the method proposed by “jp-varela” [5], this method added more variable to the calculation of the fitness score compared to the previous ones, the agent is rewarded when it dies closer to the destination or if it reaches the goal with less number of steps, for two individuals reaching the same spot, the one that reaches it with less steps is rewarded more than the other, but we find that these new variables added to the complexity of the fitness function without major improvements in the result.

- The fitness function proposed by “jodiDL”:

  $distance \leftarrow dist(Agent, Destination)$

  $fitness\_score \leftarrow 1/distance$

- The fitness function proposed by “nj-AllAboutCode”:

  if $recordedDis < 1$ **then**

  $recorded \leftarrow 1$

  end if

  $fitness \leftarrow (1/(finishTime \times recordedDis \times recordedDis))$

  $fitness \leftarrow fitness^4$

  if $hitObstacle$ **then**

  $fitness \leftarrow fitness \times 0.4$

  end if

  if $hitTarget$ **then**

  $fitness \leftarrow fitness \times 3$

  end if

- The fitness function proposed by “jp-varela”:

  $fitness \leftarrow (1/(distance))$

  if $distance < Mindist$ **then**

  $Mindist \leftarrow distance$

  $fitness \leftarrow fitness \times 1.5$

  end if

  if $steps < Minsteps$ **then**

  $Minsteps \leftarrow steps$

  $fitness \leftarrow fitness \times 2$

end if

if $hitTarget$ **then**

$fitness \leftarrow fitness \times 2$

end if

Each one of the previous approaches was tested separately in a different environment, in the next section we will try to imitate the most interesting between them, propose our own complex environment, then we will start comparing all the previous 3 methods with our improved method in all the available environments.

Fig. 18. Environment test proposed by jodiDL

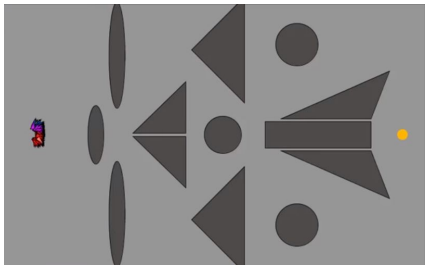

Fig. 19. Environment test proposed by jp-varela

Fig. 20. Environment test proposed by nj-AllAboutCode

IV. Our Method

We proposed a platform for all the previous genetic algorithms and their fitness function to be tested on; this platform consists of the set of parameters for the population and the agent that we mentioned above and the same an environment made up of obstacles and a destination for the population.

The set of parameters or rules that we define for the population object(the file is named: Population Controller) consists of: the population size: which is the number of agents in the population, the chromosome length: which is the number of genes that our chromosome contains, Cutoff: which is the parent selection percentage, the mutation rate: define the mutation rate for the population. The first thing that we improved was defining a list (called survivors to keep) as a parameter in the population manager; in this list we keep the top five fit survivors (or agents) to keep these good genes and guarantee that they are not altered in the next generation. The parameters values that we chose are defined in the figure 21.

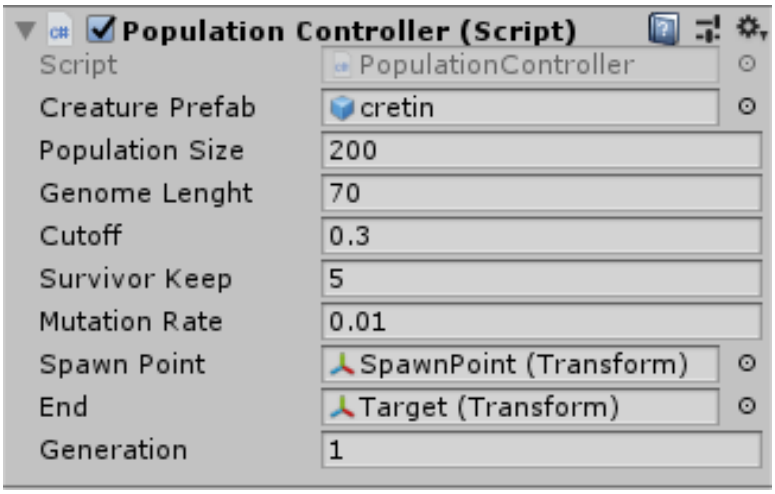

Fig. 21. Population Manager Parameters

The set of parameters that we define for the agent(contained in the brain, the file is named: Genetic Pathfinder) consists of: the creature speed: which is the speed of the agent when moving, Path multiplier: which is the distance travelled by the agent with each step. The parameters values that we chose are defined in the figure.

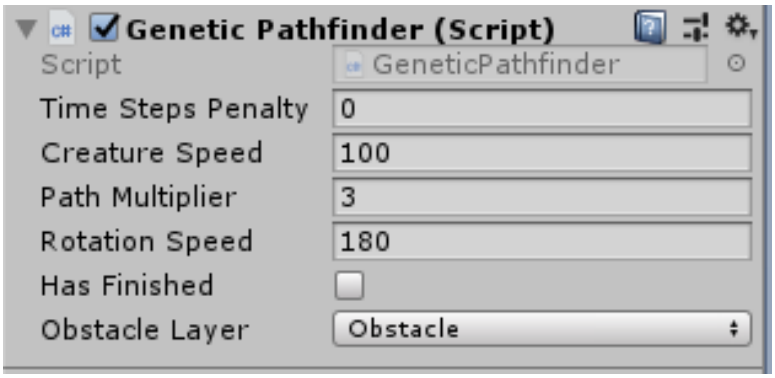

Fig. 22. Population Manager Parameters

Our fitness function is similar to the previous ones but we calculated the fitness score in a different way, the score value is not in the range of 0 and 1, instead we used other values to highlight the big difference between good

agents and bad ones, in addition our algorithm penalizes the agent when hitting obstacles by multiplying the fitness score by 0.5, a major improvement that we proposed is penalizing the agents that see multiple obstacles between them and the destination, for example a good agent can see one obstacle to the solution (it means it is closer to the solution) when a bad agent can see 3 obstacles to the solution (which means it is far from the solution).

Our function Pseudo-code:

$distance \leftarrow dist(Agent, Destination)$

if $distance = 0$ **then**

$distance \leftarrow 0.0001$

end if

$fitness \leftarrow (60/distance)$

$obstacleMultiplier \leftarrow 1 - 0.1 \times number\_of\_obstacle\_between\_agent\_and\_target$

$fitness \leftarrow fitness \times ObstacleMultiplier$

if $hitObstacle$ **then**

$fitness \leftarrow fitness \times 0.5$

end if

return $fitness$

V. Tests and Discussion

The tests of all the algorithms (including the method we proposed and the previous algorithms) were made on a platform of 3 different models: the first model was simple made of only one obstacle in the way to the destination, the second model was complicated, it had more obstacles compared to the first one, and the last model was more complicated than both of the previous ones although it has the same number of obstacles than the second model).

We tested all three fitness functions and our proposed one on all the same models multiple tests (10 tests of each function on each model), throughout the different generations, then we observed whether the algorithm can converge and reach the solution or not, and if it did then how many generations it took to reach it. Then we collected the results of the observation of each algorithm. We will discuss the different results that we had in the next part. We can refer to the fitness function of the first algorithm as Fit1, the fitness function of the second algorithm as Fit2, the fitness function of the third algorithm as Fit3, and we can refer to our proposed fitness function as Fit0.

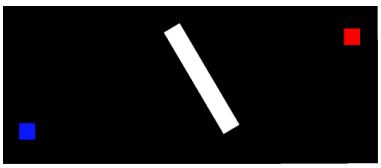

Fig. 23. First Environment model

Fig. 24. Second Environment model

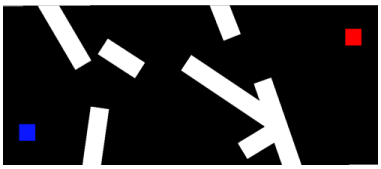

Fig. 25. Third Environment model

The first model was only chosen as a simple benchmark to test the ability of the algorithms in reaching the goal, Fit0 (the one that only uses the distance to the goal as a metric) was the one that had the best results in this

problem, Fit2, Fit3 and our proposed fitness function were slightly less performing (reached it in more generation) due to the unnecessary complexity in this simple model that only has one obstacle. All of the fitness functions are able to reach the target in all the tests. We captured the tests results and we plotted them as seen in the figures.

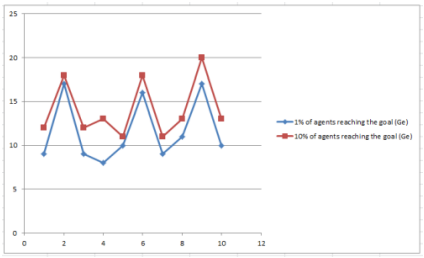

Fig. 26. Results of the first fitness function in the first environment model

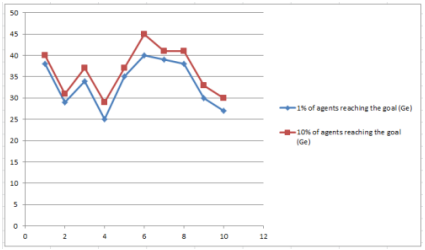

Fig. 27. Results of the second fitness function in the first environment model

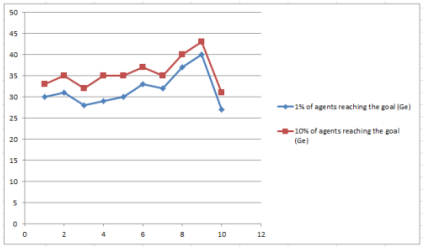

Fig. 28. Results of the third fitness function in the first environment model

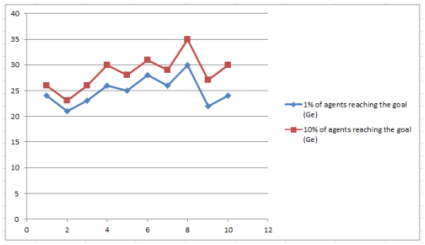

Fig. 29. Results of the proposed fitness function in the first environment model

TABLE I

CONVERGENCE RATE FOR THE FIRST MODEL

Fitness Function	Fit1	Fit2	Fit3	Our method
Convergence Rate	10/10	10/10	10/10	10/10

Even though the second model was more complicated than the first one, the results were quite similar, with the Fit1 converging in lesser generations compared to the other Fitness functions, this goes back to the design of the model as it represented the problem space with obstacles that are easy to avoid (as seen in the model 2 there enough space between and around the obstacles), and the Fit1 only uses the distance as a metric enabling the agents to search the problem space freely, getting around the obstacles to the solution. The Fit2 had the worst performance among all the fitness functions, as it took longer generations to reach the target, and 3 out of 10 times it was stuck and did not converge at all, this is because Fit2 has multiple requirements, and that it wastes many of its possible

moves doing unnecessary movements or going through a longer path, and in some cases it uses all of its possible moves which can cause it to fail, as we can see in the figure 30(where we are at 81 generations and it did not reach the target, the unnecessary movement is shown by the red circles).

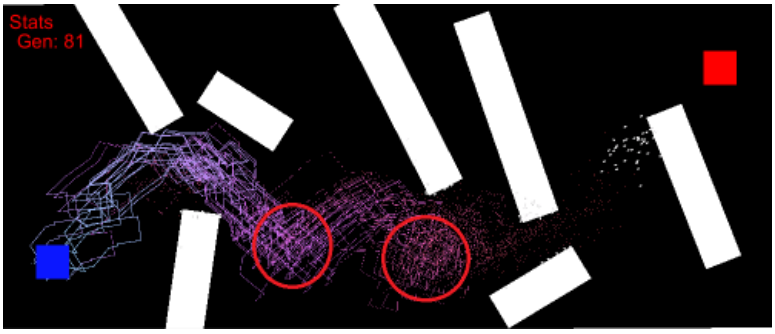

Fig. 30. The red circles highlight the unnecessary movement.

The Fit3 required more generations to reach the goal compared the Fit1 but had approximately the same as Fit2; it did not converge in 1 times out of 10, this function suffered from the same problems as the previous one (Fit2) but showed slightly better performance compared to it; Both Fit2 and Fit3 show fast performance towards the goal in the first generations and then slow down as shown in the figure 31(traveled a long way by generation 21, but then the improvement rate drops significantly). Our proposed fitness function took longer generations to reach the goal compared to Fit1 and approximately the same as Fit2 and Fit3 dues to the unnecessary movements, but it always converges and reaches the targets. The captured tests results are shown in the figures.

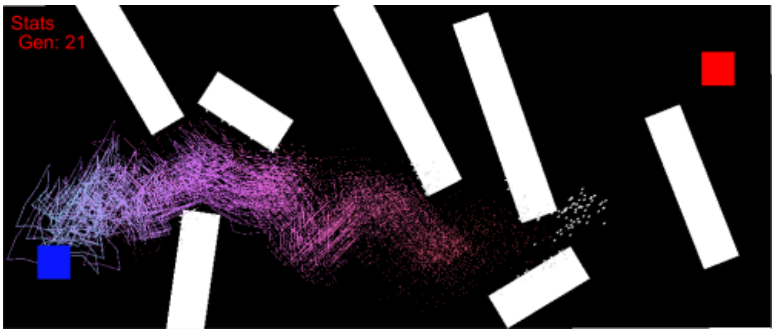

Fig. 31. The red circles highlight the unnecessary movement.

The third model is more complicated than the previous two because it contains a tricky spot as we like to call the absorbing corner as shown in the figure 40, but this is where our proposed fitness function shines unlike the other ones, The Fit1 only reached the goal in 3 out of 10 times in approximately 28 generations, the rest of the times it got stuck in the absorbing corner and did not converge at all. Fit2 and Fit3 had almost the same performance as Fit1, as they only converged in 8 out of 10 times. Our proposed Fitness function was able to reach the solution

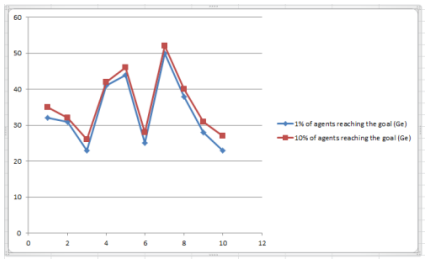

Fig. 32. Results of the first fitness function in the second environment model

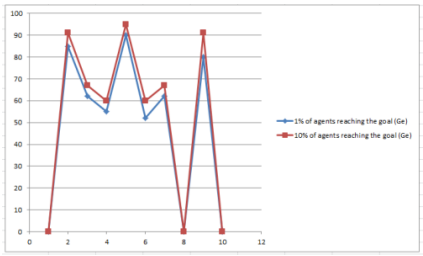

Fig. 33. Results of the second fitness function in the second environment model

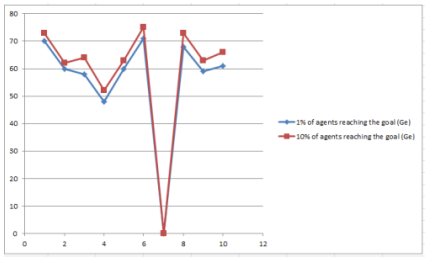

Fig. 34. Results of the third fitness function in the second environment model

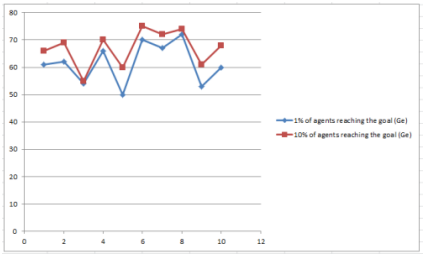

Fig. 35. Results of the proposed fitness function in the second environment model

TABLE II

CONVERGENCE RATE FOR THE SECOND MODEL

Fitness Function	Fit1	Fit2	Fit3	Our method
Convergence Rate	10/10	07/10	09/10	10/10

without getting stuck in the absorbing corner (it only failed 1 time out of 10), due to its ability to choose the path that contains less obstacles (because the absorbing corner contains 2 obstacles and the path above contains only one obstacle), it took a number of generation between 130 and 60 to converge. The test results are shown in the figures.

TABLE III

CONVERGENCE RATE FOR THE THIRD MODEL

Fitness Function	Fit1	Fit2	Fit3	Our method
Convergence Rate	03/10	03/10	02/10	09/10

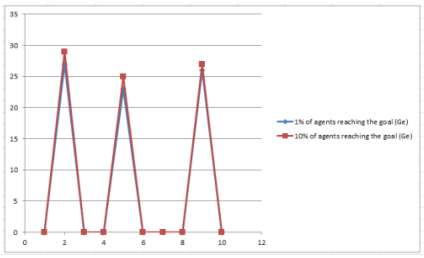

Fig. 36. Results of the first fitness function in the third environment model

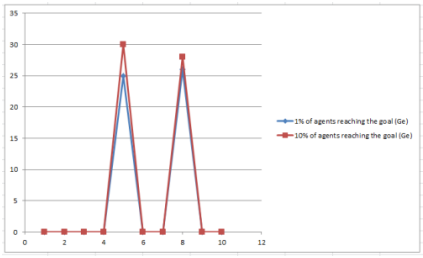

Fig. 37. Results of the second fitness function in the third environment model

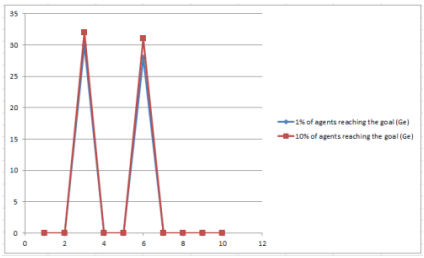

Fig. 38. Results of the third fitness function in the third environment model

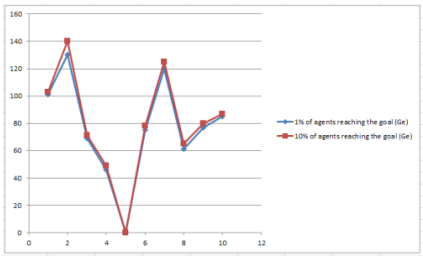

Fig. 39. Results of the proposed fitness function in the third environment model

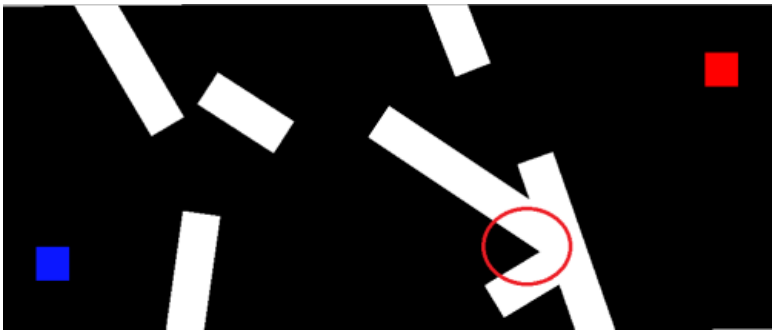

Fig. 40. Absorbing Corner in the third model.

VI. Conclusion and Perspective

Our presented method showed its robustness in the most complex model, but failed in the less complex one, same result obtained by the other methods. We can take as a conclusion from this work that GAs are used to avoid complexity, defining complex functions to solve a simple problem could lead to bad results which is in the case of the simple fitness function that was used (which gave the best results in the first simple model), and in the case of a really complex problem, a redefinition of the core of the problem is a necessity. For more complex environments, even our proposed function will fail, precisely in the case of having different obstacles between a local exit and the

goal, this will lead the agents to avoid the exist, even though if they pass through it, avoiding the other obstacles would be a trivial task. The shortcoming of our function is due to the global vision of the agents, as a solution we propose a further definition of the problem, by binding the penalty rate not only by the number of obstacles between the agent and the target, but also with the distance between the agent and the obstacle, the further the obstacle is the lesser the penalty associated with it is. The new pseudo-code of the improved function:

$distance \leftarrow dist(Agent, Destination)$
if $distance = 0$ **then**
    $distance \leftarrow 0.0001$
end if
$fitness \leftarrow (60/distance)$
for $obstacle\_between\_agent\_and\_target$ **do**
    $fitness \leftarrow fitness \times (1 - \frac{1}{dist(agent,obstacle)})$
end for
if $hitObstacle$ **then**
    $fitness \leftarrow fitness \times 0.5$
end if
return $fitness$

Unfortunately, due to the lack of time, we couldn't test this new approach, and we're going to leave it as a future work.

REFERENCES

[1] What's new in gartner's hype cycle for ai, 2019. https://softwarestrategiesblog.com/tag/artificial-intelligence/.

[2] S Forrest. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

[3] Ahmed Gad. Introduction to optimization with genetic algorithm. https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b, 2018.

[4] jodiDL. Shortest path genetic algorithm. https://github.com/jodiDL/shortest-path-genetic-algorithm-, 2018.

[5] jp varela. Genetic-algorithm-pathfinder. https://github.com/jp-varela/Genetic-Algorithm-PathFinder, 2017.

[6] Pierre Levy. Education and training: New technologies and collective intelligence. *Prospects*, 27(2):248–263, 1997.

[7] Jack Mcdowell. A quantitative evolutionary theory of adaptive behavior dynamics. *Psychological review*, 120:731–50, 10 2013.

[8] nj AllAboutCode. Ai-maze-genetic-algorithm-smart-agents. https://github.com/nj-AllAboutCode/AI-Maze-Genetic-algorithm-smart-agents, 2018.

[9] Sarah O'Meara. Ai researchers in china want to keep the global-sharing culture alive. https://www.nature.com/articles/d41586-019-01681-x, 2019.

[10] Emil Scarlat and Iulia Maries. A genetic algorithm for community formation based on collective intelligence capacity. In James O'Shea, Ngoc Thanh Nguyen, Keeley Crockett, Robert J. Howlett, and Lakhmi C. Jain, editors, *Agent and Multi-Agent Systems: Technologies and Applications*, pages 271–279, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[11] James Surowiecki. *The Wisdom of Crowds*. Anchor, 2005.

[12] TutorialsPoint. Genetic algorithms - introduction. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fundamentals.htm, 2019.

[13] Stefaan G. Verhulst. Where and when ai and ci meet: exploring the intersection of artificial and collective intelligence towards the goal of innovating how we govern. *AI & SOCIETY*, 33(2):293–297, May 2018.