www.editada.org

International Journal of Combinatorial Optimization Problems and Informatics, 12(1), Jan-April 2021, 9-20. ISSN: 2007-1558.

Application of genetic algorithm on autonomous agents for virtual navigation in Unity3D

Jesús Iván Rubio-Sandoval, Adolfo Josué Rodríguez-Rodríguez, José Lázaro Martínez-Rodríguez, David Tomás Vargas-Requena, Juan Carlos Huerta-Mendoza, Wenceslao Eduardo Rodríguez-Rodríguez

Maestría en Ciencias y Tecnologías Computacionales de la Unidad Académica Multidisciplinaria Reynosa Rodhe-Universidad Autónoma de Tamaulipas. Carretera Reynosa - San Fernando, cruce con Canal Rodhe Col. Arcoiris, Cd. Reynosa Tamaulipas, C.P. 88779.
jesus.rubio.svl@gmail.com, arodriguez@docentes.uat.edu.mx, lazaro.martinez@uat.edu.mx, dvargas@docentes.uat.edu.mx, jchuerta@docentes.uat.edu.mx, wrodriguez@docentes.uat.edu.mx

Abstract. This paper presents a software implementation of genetic algorithms on autonomous agents that can navigate a virtual environment created in Unity3D to find an indicated destination. The fitness of the autonomous agents is calculated mainly through a proposed fitness function, which evaluates the performance to heuristically find a destination on three different scenarios with different difficulty and simulation parameters. Based on the obtained results, it is determined that the complexity of the scenarios, the established parameters, and the randomness of the algorithm affect the performance of individuals in finding their destination, but without the need to establish a map of predefined routes.

Keywords: **g**enetic algorithms, virtual environments, unity3D, autonomous navigation.

Article Info
Received: Ago 20, 2020.
Accepted: Dec 13, 2020.

1 Introduction

Nowadays, the solution of mathematical and/or optimization problems is linked to computer systems that can perform multiple simulations in the shortest possible time, using many combinations of input variables. To explore all possible solutions, the use of Genetic Algorithms is needed, which involves a stochastic search technique based on the mechanism of natural and genetic selection. The central research theme of genetic algorithms is to maintain a balance between exploitation and exploration in search of optimal solutions to survive in different environments [1]. The application of genetic algorithms in videogame development or virtual reality gives rise to adaptive autonomous agents that can adapt to their environment and improve their performance over time [2].

The virtual environments in three dimensions are tools that have increased in utility to visualize, manipulate, and process information with the creation of simulators throughout various fields such as education, science, and engineering. In the area of video games and virtual reality, scenarios are characterized by the ability to be navigated or explored. They can become relatively complex, putting the spatial skills of users to the test [3]. Although there is a difference in the design of these environments, living beings can adapt relatively easily, unlike robotic agents with artificial intelligence, which must be configured to navigate them, either by programming defined routes, points of interest or in an adaptive manner.

Various works have been developed to create autonomous agents in virtual environments that adapt to various factors. In [4] the authors use genetic algorithms for the development of uncontrollable agents that adapt to three types of gaming behaviour: survival, combat efficiency (to defeat other agents) and a combination of both, resulting in better artificial intelligence than in their better conditions increases up to 5 times its survival time and victory in combat with respect to agents to which the evolutionary algorithms were not applied. In [5], a virtual vehicle was developed that is controlled automatically using neural networks, and that uses genetic algorithms for its training. Its results are satisfactory when managing to drive on a track without colliding with walls. In [6], it is explained the process of developing genetic algorithms and their implementation for finding routes within a maze without the use of pathfinding algorithms.

This article proposes the implementation of a genetic algorithm applied over autonomous agents for heuristic navigation within a virtual scenario in Unity3D, allowing users to find a destination without the establishment of predefined routes. The use of a genetic algorithm in solving navigation problems allows finding more than one route without the need for it to be the shortest and for autonomous agents to reach their destination with the possibility of wandering on their way.

2 Theoretical foundations

2.1 Genetic Algorithms

Genetic algorithms are a class of adaptive stochastic optimization algorithms that use natural evolution principles to calculate optimal solutions to search problems. It is based on three fundamental principles described by Darwin: reproduction, natural selection, and diversity of individuals. Genetic algorithms use a population of solutions, everyone in the population represents a candidate solution, and its properties are found on the individuals' chromosome or genotype. Based on these properties, an evaluation function called "fitness function" is performed, which calculates a score based on the individuals' possibilities of solving a type of problem, which can be minimization, where a lower score is sought or a maximization problem, where higher scores are preferred. Over time the population evolves, which occurs with the formation of new individuals through the crossing process. In this process, two individuals are selected according to their aptitude and using a selection method. One or more children are the product of this cross and contain characteristics of both parents. Like nature itself, a mutation can occur in one of these children at low probabilities, which causes its genotype to change partially or completely, to maintain a varied population that can expand the search to the desired solution. All new children represent a new generation and replace the old population. Genetic algorithms are an iterative process, and each iteration is called a generation. The generations will continue to occur until a condition is met, either obtaining the optimal solution or a specific number of iterations have occurred [7].

2.2 Navigation in virtual environments

Navigation is a task that has been relevant for thousands of years. Its objective is to explore an environment to search for new paths, transfer from different points, and optimize routes through shortcuts. The navigation is performed by applying strategies that include the integration of roads, trackers, points of interest, route navigation, and map navigation. Each of these strategies uses knowledge that involves the spatial ability of individuals, the recognition of their environment, the location of their starting point and destination, the learning of a sequence of places and actions and/or the use of a map of the spatial positions of the known locations [8]. Within the area of autonomous system applications, self-localization has become a significant challenge for autonomous navigation of autonomous devices. Autonomous navigation requires obstacle detection and collision avoidance, tracking objects around them, and real-time monitoring of the position and orientation of the device in question [9]. In virtual environments such as simulators and video games, the need for autonomous navigation is applied in non-controllable agents to create artificial intelligence capable of navigating in its environment. It is common to use pathfinding algorithms, which search for the optimal path between two points using a mapping system or the use of graphs.

2.3 Unity 3D

Unity is a video game development tool created by the Unity Technologies company that has tools for rendering images, 2D / 3D physics, audio, animations, multiplayer networking, NavMesh navigation tools for Artificial Intelligence or support for Virtual reality. One of the significant advantages or virtues of Unity is the large community of users it has, not only within the Unity forums but throughout the Internet [10]. Unity 3D is available in most of the latest generation platforms and operating systems, being able to reuse the same code for all the equipment. This game engine allows programming in three programming languages, all object-oriented: C #, JavaScript and Boo (obsolete) [11]. Traditionally, in Unity to solve navigation problems, a component called NavMeshAgent is used, which is placed in an agent on the stage and navigates on a surface using a Navigation Mesh (NavMesh) [12]. This component was integrated into Unity and other game engines using a library called RVO2 with which the user can specify static obstacles, agents and their speeds [13].

3 Methodology

3.1. Genetic algorithms Methodology

The description of the proposed project is based on the requirements for the genetic algorithms: Definition of the problem to be optimized, generation of an initial population, natural selection, crossing of individuals, and mutation. The applied algorithm uses object-oriented programming techniques; particularly, individuals are governed by a class that contains the characteristics of the chromosome of each and methods to perform the actions described above. At the beginning of the simulation, input values described in section 4 of this document are introduced and give rise to the instantiation of individuals at the starting point of the scenario. When completing an N number of steps described in the configuration screen, the agents proceed to the procedure of the genetic algorithm.

3.1.1. Definition of the problem to be optimized

This project is based on the creation of autonomous agents who can adapt to a virtual environment and navigate in a heuristic way to reach a defined destination. The navigation performance of individuals within the map is evaluated according to the proposed fitness equation (1) and the code in Algorithm 1. This function uses the final agent position at the end of a generation to measure a remaining distance to the destination in question. This distance is related to the total distance between the start and destination points through a division to obtain a percentage of the remaining distance traveled. The average speed and time of movement are part of the individual's genes established at the moment of instantiation of the individual on stage and serve to reward those agents that present more significant time and speed of movement. The final resultant value comprises a value from 0 to 1, where 0 indicates that the target was not approached while 1 is the optimal approach value, returning to the situation in a maximization problem.

$$\text{Fitness} = [1-(d/D)] + (\text{speed}_{\mathbf{av}} / 200) + (\text{Timer}_{\mathbf{av}} / 300) \tag{1}$$

Where:
D = Distance between an origin point and end point.
d = distance between the final position of the agent and the end point.
$\text{speed}_{\mathbf{av}}$ = Average movement speed
$\text{Timer}_{\mathbf{av}}$ = Average movement time.

Algorithm 1. Fitness function proposed for the algorithm.
```
//Fitness function
public void Fitness(Vector3 target){
      float   distFit   =   1   -   (Vector3.Distance(this.transform.position,
target)/Vector3.Distance(positions[0], target));
      float sumMovSpeed = 0;
      float sumMovTime = 0;

      foreach(Genes gen in genes){
            sumMovSpeed += gen.moveSpeed;
            sumMovTime += gen.moveTime;}

      float promMovSpeed = sumMovSpeed / genes.Length;
      float promMovTime = sumMovTime / genes.Length;

      fitness = distFit + (promMovSpeed/(2*100)) + (promMovTime/(3*100));
      fitness = Mathf.Clamp(fitness, 0, 1); }
```

3.1.2 Initial population

The initial population is composed of several individuals that are randomly generated within a range of possible solutions. Each individual (chromosome) has a number of genes that are determined in the configuration screen, specified in detail in section 4.2

within a variable called "Max steps per generation". The composition of the individual can be seen in Figure 1, and each of the genes comprises:

- A (motion) vector that determines the direction in which the agent will move. The values in X and Z in Vector3 are randomly set between -1f and 1f, while the Y value remains at position 0.
- A speed of movement that determines how fast it will move. It goes in a random range between 0 and 2 floating.
- Movement time determines how long it will move in one direction. It is in a random range between 0.5 and 3 float.

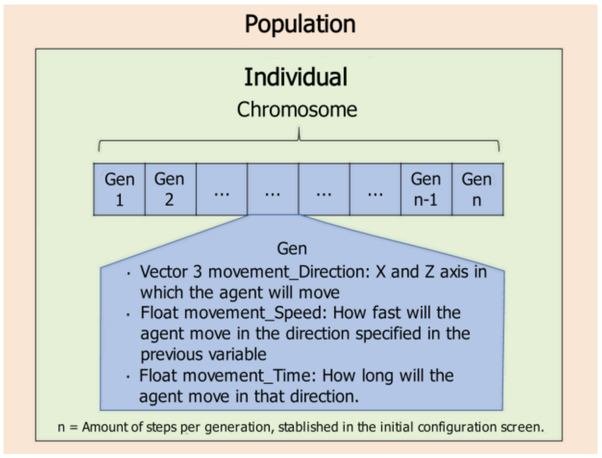
Population
Individual
Chromosome
Gen 1
Gen 2
...
...
...
...
Gen n-1
Gen n
Gen
· Vector 3 movement_Direction: X and Z axis in which the agent will move
· Float movement_Speed: How fast will the agent move in the direction specified in the previous variable
· Float movement_Time: How long will the agent move in that direction.
n = Amount of steps per generation, stablished in the initial configuration screen.

Fig. 1. Components of an individual. It is composed of a vector, speed, and time of movement.

3.1.3 Natural selection

Natural selection is made after evaluating the fitness function and is responsible for selecting a pair of individuals to cross and generate a new individual. For the parent selection, an individual is randomly from the entire population,
and its suitability is assessed against a random value ranging from 0 to the best fitness. The individual is selected as a parent if its fitness value is higher than the random value; otherwise, it is discarded, and another individual is selected for comparison. The flow of selection can be seen in Figure 2 and the steps of Algorithm 2.

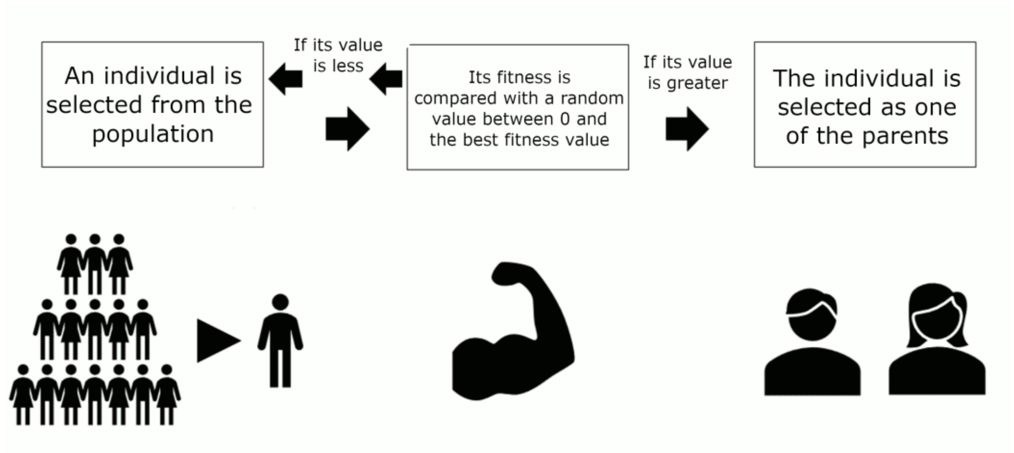
An individual is selected from the population
If its value is less
Its fitness is compared with a random value between 0 and the best fitness value
If its value is greater
The individual is selected as one of the parents

Fig. 2. Natural selection flow chart to select the parents to mate.

3.1.4 Crossing phase

Once having two parents, the cross is made, which generates an individual that combines the genes of both. The crossing performed is called "uniform crossing", which consists of each gene on the chromosome being transmitted from the father with the same probability. A representation of the uniform cross is shown in Figure 3.

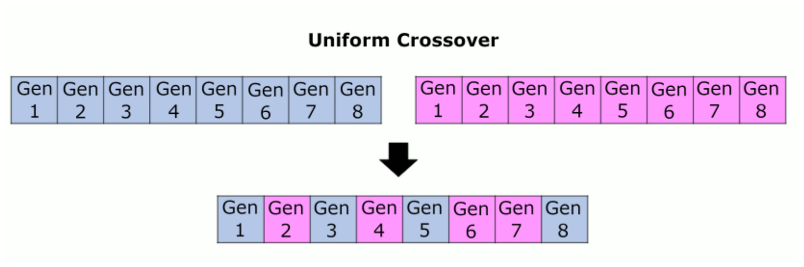

Fig. 3. Crossing of two chromosomes by means of uniform crossing.

Algorithm 2. Selection of parents for the generation of a new individual, adapted from Daniel Shiffman (2016), *Nature of Code,* Github [14].

```
       //Breeding
       public void CreateNewGeneration(){
             PFDNA[] newPopulation = new PFDNA[population.Length];
             for(int i = 0; i < population.Length;i++){
                   PFDNA father = acceptReject();
                   PFDNA mother = acceptReject();
                   PFDNA child = father.Crossover(mother);
                   child.Mutation(mutationRate);
                   newPopulation[i] = child; }

             for(int i = 0; i < population.Length;i++){
                   Destroy(population[i].gameObject);
                   population[i] = newPopulation[i];}
            generations++;}

      private PFDNA acceptReject(){
             while(true){
                   int i = Random.Range(0, population.Length);
                   float r = Random.Range(0, getBestFitness);
                   PFDNA partner = this.population[i];
                   if(partner.fitness > r){
                         return partner;}
                         }
             }
```

3.1.5 Mutation

Finally, the individual resulting from the cross undergoes a mutation, which has a low probability of occurring and has the function of altering part of the individual's genes in order to recreate nature itself. Thanks to this, it is more difficult for individuals to converge to the same DNA, maintaining variety in the population. The probability of mutation and other parameters in this project are determined through a configuration screen before the simulation. In this project, each gene of each individual has an X probability of mutating. If this is the case, the gene takes new random values among the possible solutions established in the initial population. Algorithm 3 shows the proposed mutation.

Algorithm 3. Mutation function of the newly generated individual.

```
          //Mutation Function
          public void Mutation(float mutationRate){
              float rnd = Random.Range(0f,1f);
                  if(rnd < mutationRate){
                      float rnd3 = Random.Range(1, 101);
                      if(rnd3 > 50){
                          int rnd2 = Random.Range(0, genes.Length-1);
                          genes[rnd2].direction = new Vector3(Random.Range(-1f,
      1f), 0, Random.Range(-1f,1f));
                          genes[rnd2].moveSpeed = Random.Range(0f, 2f);
                          genes[rnd2].moveTime = Random.Range(0.5f, 3f);
                      }

                  }
          }
```

4 Design and Experimentation

4.1 Technical requirements

This project was developed on a PC with Windows 10 x64 operating system with 8GB of RAM and Intel Graphics HD (R) 4000. The Unity version used is 2019.2.17f1, in the C # Monobehavior environment.

4.2 Design of the experiment

To test the evolutionary algorithm, three scenarios of different difficulty were created for its navigation as shown in Figures 4, 5 and 6. Each of the tests consists of: A starting point, a target point, the individuals in the population, and obstacles that characterize the different scenarios.

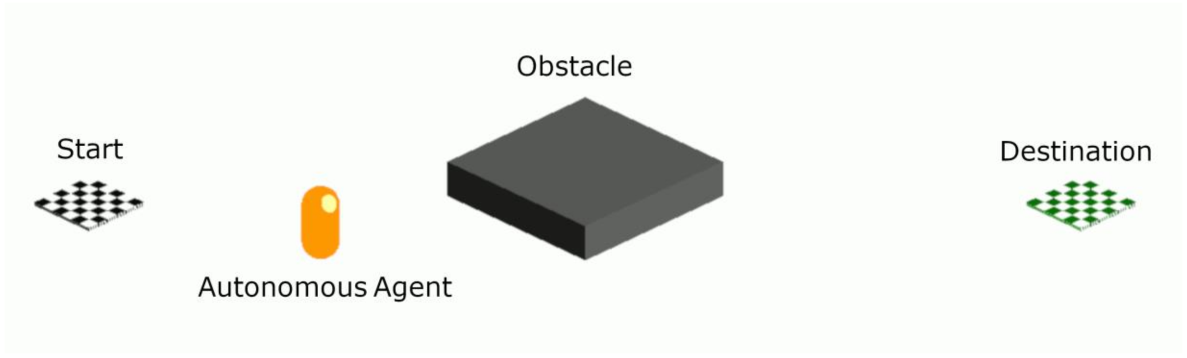

Fig. 4. Simulation scenario 1. It consists of navigating around a single obstacle. The starting point is the grid surface on the left. The destination is the squared surface on the right, the individual is the cylindrical object, and the obstacles are the dark cubes. Difficulty: Easy.

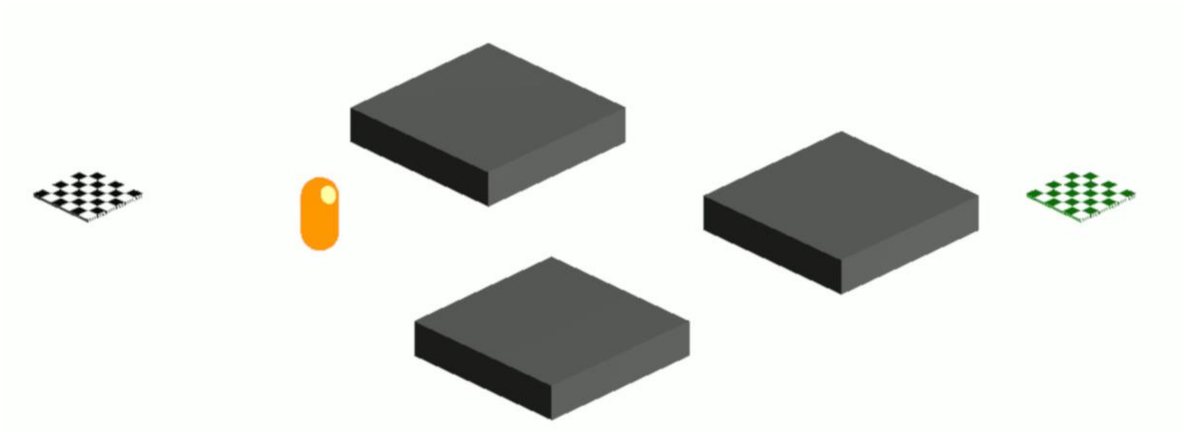
Fig. 5. Simulation scenario 2. It consists of navigating between various obstacles. Difficulty: Medium.

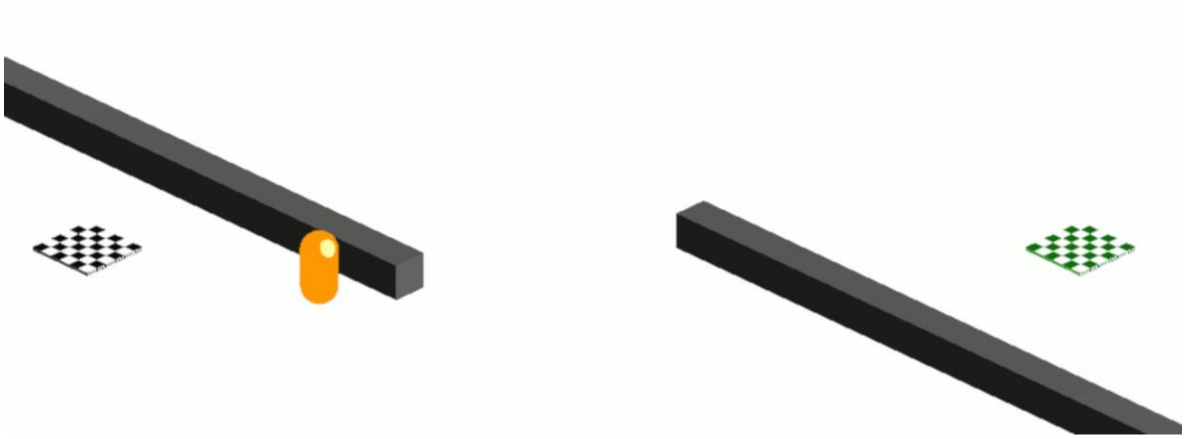
Fig. 6. Simulation Scenario 3. It consists of navigating between two obstacles, being necessary to learn how to go around them. Difficulty: Hard.

4.3 Configuration screen

At the beginning of the software simulation, an initial configuration screen is shown (Figure 7) that allows the user to adjust parameters directly affecting the simulation and the genetic algorithms. This screen allows the user to modify the values of:

- Population Amount: indicates the number of individuals with whom the simulations will be done.
- Mutation Rate: indicates the likelihood that an individual will change its genetics.
- Max Steps per Generation: indicates the length of the individual's genes.
- Speed Scale: alters the speed of the simulations.
- Finish Condition: 3 options are offered: until the objective is found, after x successive generations or never.
- Select Level: 3 differently designed stages are offered.

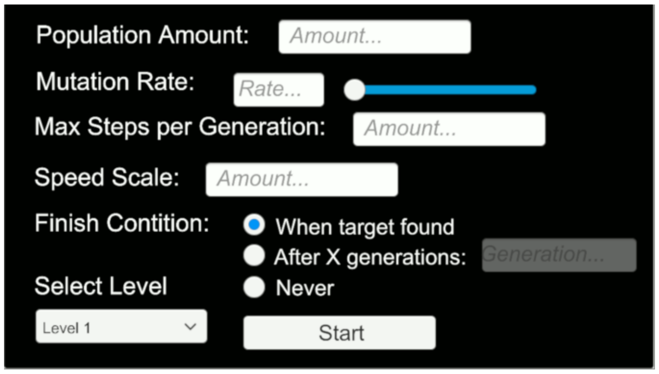
Population Amount: Amount...
Mutation Rate: Rate...
Max Steps per Generation: Amount...
Speed Scale: Amount...
Finish Contition: [✓ When target found] After X generations: Generation... Never
Select Level: Level 1

Fig. 7. Initial configuration screen.

4.4 Scenario configuration

It is proposed to execute three tests in each scenario to analyze the performance of the algorithm with the following initial settings:
- Population amount: 50, 75 and 100.
- Mutation rate: 2 %.
- Max Steps per Generation: 25.
- Speed scale: 10.
- Finish Condition: When target found

To evaluate the performance of the algorithm on the autonomous agents, the best values of each generation, and their comparison in each simulation will be considered.

5 Results

Figure 8 shows the path traveled by the individual with the best fitness within scenario 1, showing three reference generations (the first two blurred for comparison): initial, an intermediary, and the final. Figure 9 and Table 1 show numerical results obtained in simulations of the first scenario. In these tests, the simulation with 75 individuals presented a favourable evolution more quickly, managing to find their destination in generation 5, unlike the scenarios with 50 and 100 individuals who found their destination in generation 6 (even though the population with 50 individuals started as the most suitable).

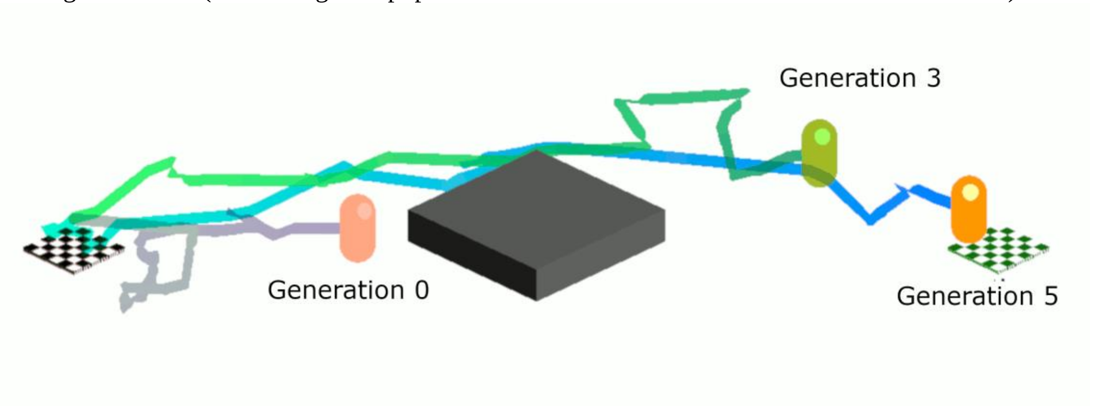

Fig. 8. Route performed by the agents with the best fitness in generation 0, 3 and 5, with a population of 75 individuals.

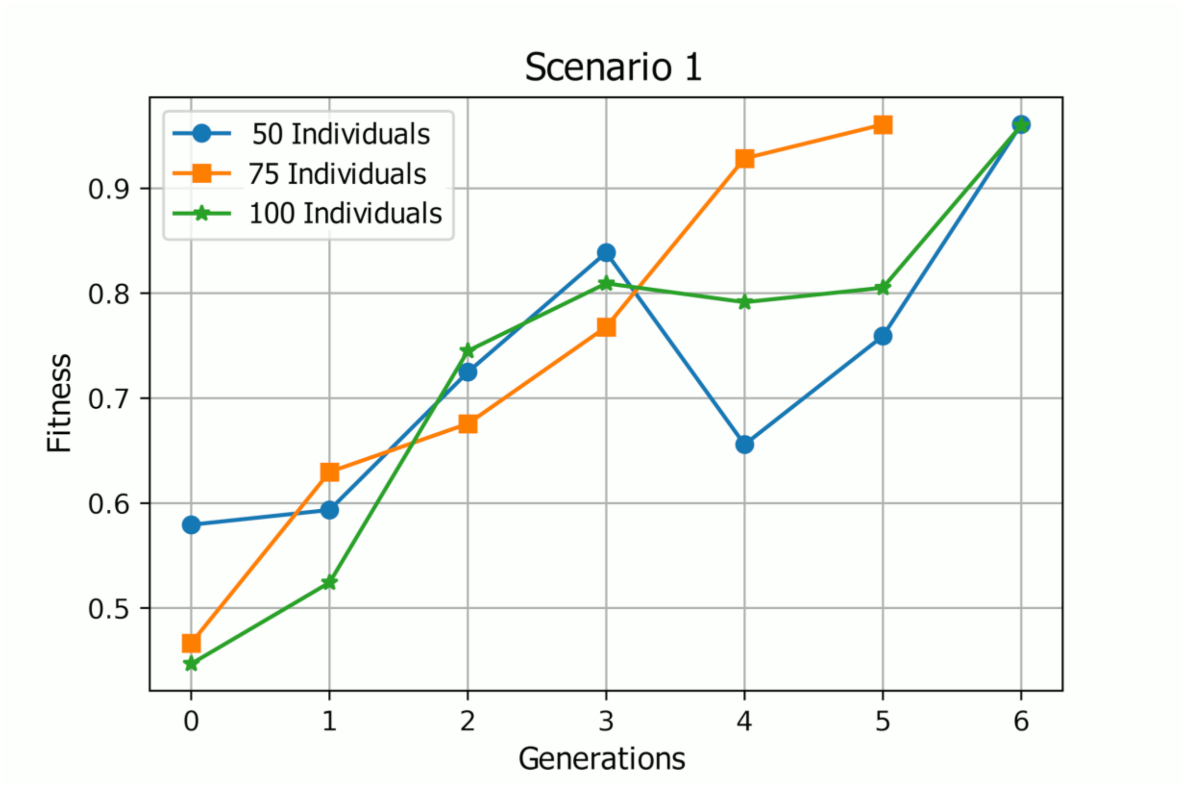

Figure 9. Graph of the simulation in the first scenario. A comparison of the performance of the tests with different populations is represented in relation to the best fitness obtained (Y) of each generation (X). The initial population is considered as generation 0.

Table 1. Summary of the values obtained in the simulation of the first scenario indicating the number of individuals in the population, the number of generations it took to find their fitness, the largest increase and smallest increase in fitness, an average increase. The best aptitude with which the simulation started and ended.

Population	Number of generations	Highest growth	Lower growth	Average growth	Initial fitness	Final fitness
50	7	0.2018	-0.1830	0.0636	0.5794	0.9610
75	6	0.1631	0.0323	0.0988	0.4666	0.9608
100	7	0.2208	-0.0179	0.0854	0.4469	0.9598

In the second scenario, a similar performance to the previous one was presented, being the population with 75 individuals the one that first found the destination, taking it until generation 4. The route taken can be seen in Figure 10. Both the population of 50 and 100 showed a lower performance, taking them until generation 8 and 9, respectively. Both showed regressions in their evolution, negatively affecting their final result. Figure 11 and Table 2 show in more detail the results obtained.

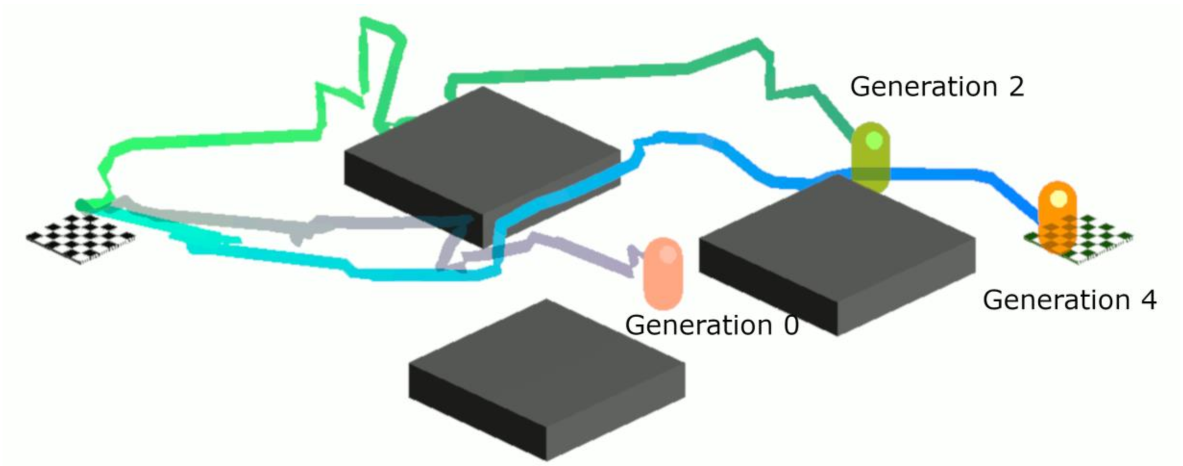

Fig. 10. Route performed by the agent with the best aptitude in generation 0, 2 and 4, with a population of 75 individuals.

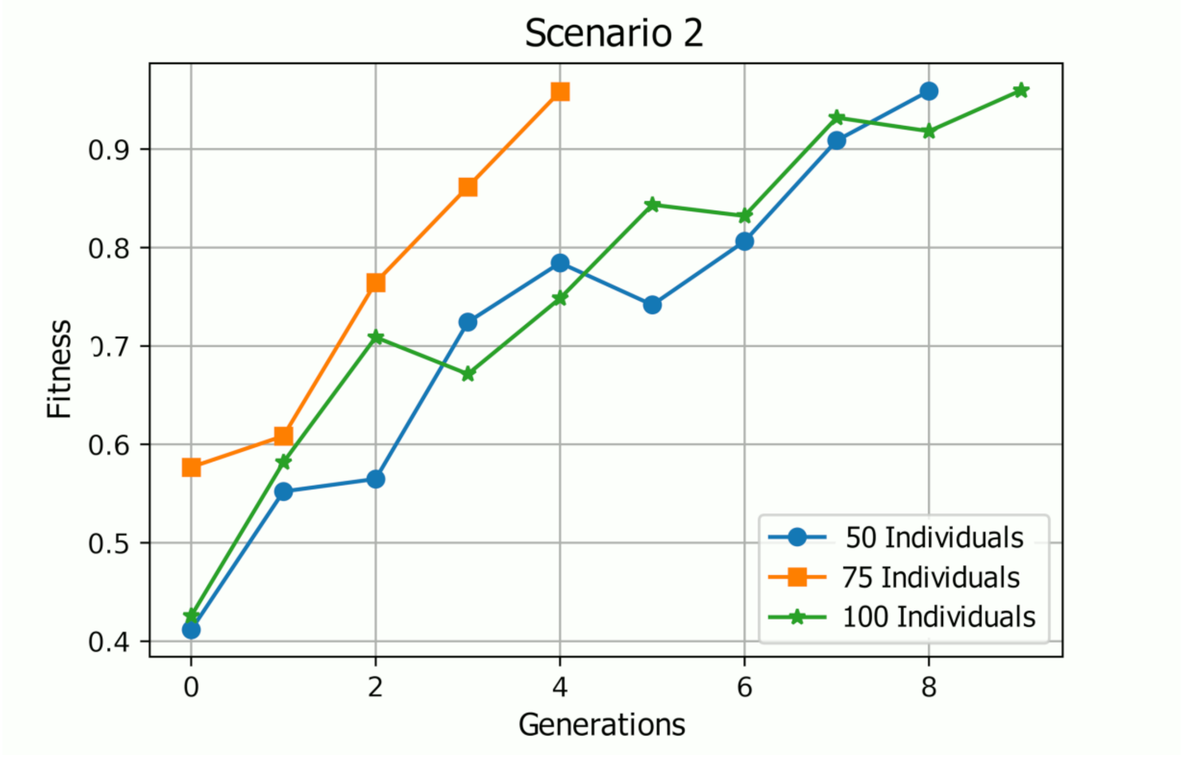

Fig. 11. Graph of the simulation in the second scenario.

Table 2. Summary of the values obtained in the simulation of the second scenario.

Population	Number of generations	Highest growth	Lower growth	Average growth	Initial fitness	Final fitness
50	9	0.1594	-0.0427	0.0684	0.4116	0.9592
75	5	0.1560	0.0317	0.0954	0.5769	0.9587
100	10	0.1562	-0.0374	0.0593	0.4257	0.9599

Again, the population with 75 individuals presented a higher average increase and without presenting any type of regression. The simulation with 100 individuals shows the lowest growth, this might be due to the randomness of the selection and crossing process. In the last scenario, the simulations with a smaller population present a more significant problem to navigate the scenario. The population with the best performance was that of 100, and the one that presented a low performance was that of 50. Figures 12, 13, and Table 3 detail the path is taken and the performance of the simulations.

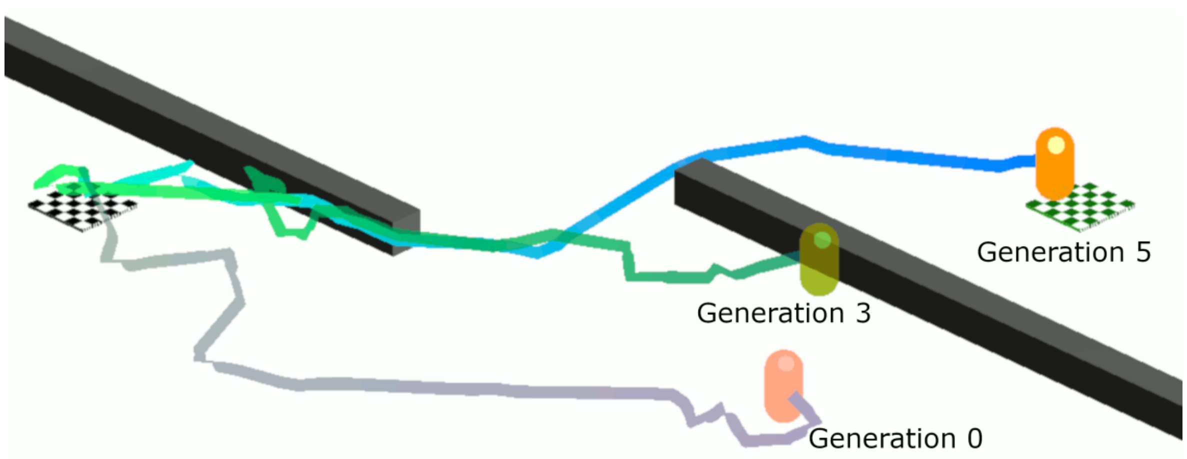

Fig. 12. Route performed by the agent with the best aptitude in scenario 3, with a population of 100 individuals.

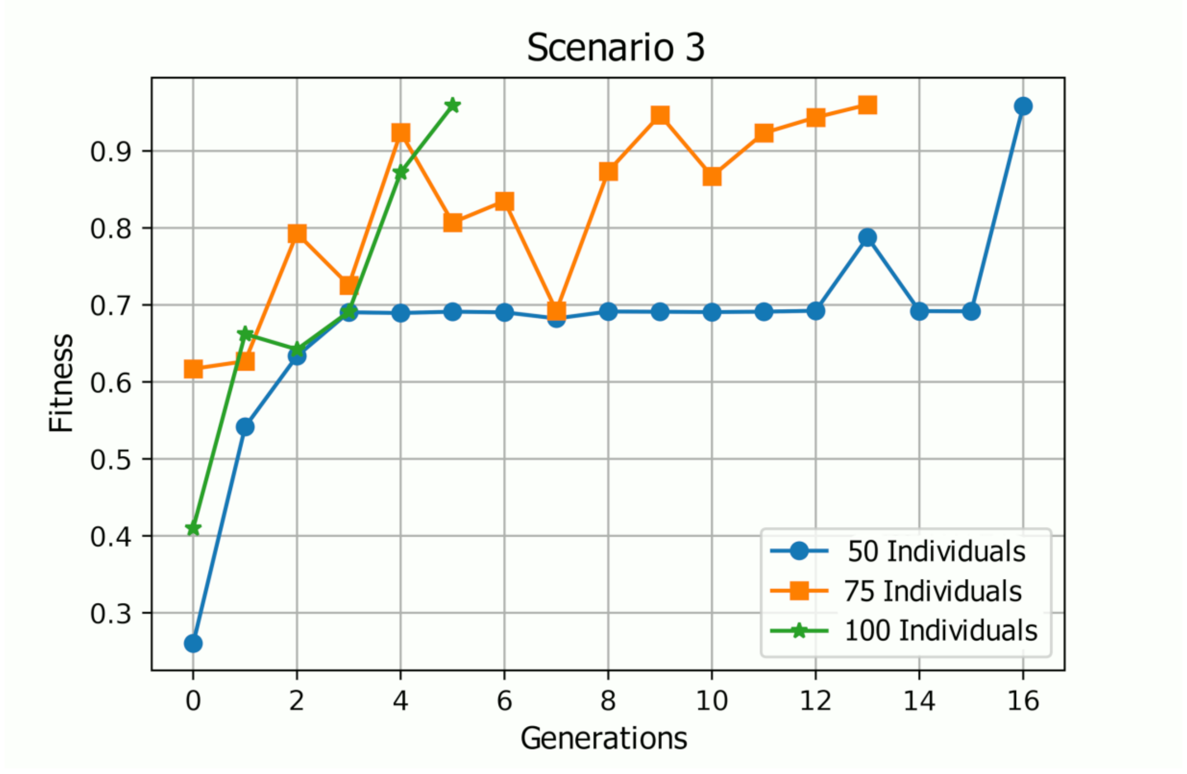

Fig. 13. Graph of the simulation in the third scenario.

Table 3. Summary of the values obtained in the simulation of the third scenario.

Population	Number of generations	Highest growth	Lower growth	Average growth	Initial fitness	Final fitness
50	17	0.2813	-0.0950	0.0436	0.2603	0.9584
75	14	0.1985	-0.1427	0.0263	0.6170	0.9601
100	6	0.2521	-0.0200	0.10978	0.4101	0.9590

The peculiarity of the third scenario is to force individuals to "go back" for a moment in order to overcome the second obstacle. By not going around it, they can get stuck at one point without being able to continue moving towards their destination. Thus, this is demonstrated in the simulation with a population of 50 individuals around 12 generations to get out of that problem completely. The population with 100 individuals, having a greater variety of genes allowed to overcome that obstacle more quickly.

6 Conclusions

According to the obtained results, it can be determined that the objective function fulfills the purpose of creating autonomous agents that can navigate in a virtual scenario in a heuristic way. The autonomous agents can adapt to their environment. However, due to the random nature of these algorithms, favourable conditions must be presented so that the population evolves in the same way. With the initial configuration, the simulations can be completed at a relatively high speed as it does not take many generations to find the destination. The modification of the parameters of initial population size and/or the number of steps significantly affect the simulations because with a smaller population it is possible that you do not have a lower number of solutions, while a lower number of steps forces individuals to always move in the optimal way possible or otherwise a regression is reached and may never find their destination.

Despite its success, the complexity of the scenarios may present a problem for assessing the fitness function, as it evaluates favourably the closer the individual is to the destination. If a maze-type scenario is made, which requires individuals to move away from the destination to avoid obstacles, it would be necessary to increase the number of steps and hope that individuals do not get stuck on dead ends. To solve the previous situation in a more agile way, route search or pathfinding algorithms can be applied, which apply graph theory to create a route map. These algorithms solve this type of problem more efficiently. Its combination with genetic algorithms allows us to find routes that are the shortest, requiring less effort or less traveled.

References

1. Gen, M. & Lin, L.: Genetic Algorithms. In: B.W. Wah (ed.): Wiley Encyclopedia of Computer Science and Engineering, John Wiley & Sons, Inc, pp. 1-15(2008).
2. Hoekstra C.: Adaptive Artificially Intelligent Agents in Video Games: A Survey, UNIAI-06, pp. 7-11 (2006).
3. De Castell S.; Larios H.; Jenson J.: Gender, videogames and navigation in virtual space, Acta Psychologica, Vol. 119, pp. 1-20 (2019)doi: 10.1016/j.actpsy.2019.102895.
4. Esparcia-Alcazar A. I.; Martinez-Garcia A.; Merelo J. J.; Garcia-Sanchez P.: Controlling bots in a First Person Shooter Game using Genetic Algorithms, IEEE Congress on Evolutionary Computation, pp. 1-8 (2010) doi: 10.1109/CEC.2010.5586059.
5. Vaidya A.: Self Driving Car using Genetic Algorithm in Unity Engine. Ashwin Vaidya, https://ashwinvaidya.com/blog/self-driving-car-using-genetic-algorithm-in-unity/ (2018), Accessed 20 July 2020.
6. Galbo G.: Path Finding With Genetic Algorithms. #Yolo Programming. https://yoloprogramming.com/post/2017/01/11/path-finding-with-genetic-algorithms (2007), Accessed 20 July 2020.
7. Karakatič S.; Podgorelec V.: A survey of genetic algorithms for solving multi depot vehicle routing problem, Applied Soft Computing. Vol 27, pp. 519-532 (2015) doi: 10.1016/j.asoc.2014.11.005.
8. Warren W. H.: Non-Euclidean Navigation, Journal of Experiment Biology 222, pp. 1-10 (2019) doi: 10.1242/jeb.187971.
9. Mohamed S. A. S.; Haghbayan M.; Westerlund T.; Heikkonen J.; Tenhunen H.; Plosila J.: A Survey on Odometry for Autonomous Navigation Systems, IEEE Access, Vol. 7, pp. 97466-97486 (2019) doi: 10.1109/ACCESS.2019.2929133.
10. García D. E.: Qué es Unity y características principales, OpenWebinars, https://openwebinars.net/blog/que-es-unity/ (2019), Accessed 22 July 2020.
11. Yeeply: Desarrollo de juegos con Unity 3D ¿Cómo funciona esta herramienta?, Yeeply, https://www.yeeply.com/blog/desarrollo-de-juegos-con-unity-3d/ (2014), Accessed 22 July 2020.
12. Juhola O. P.: Creating Self-Learning AI using Unity Machine Learning, JAMK University of Applied Sciences, pp. 1-61 (2019).
13. Snape J.; Guy S. J.; Vembar D.; Lake A.; Lin M. C.; Manocha D.: Reciprocal Collision Avoidance and Navigation for Video Games, Intel, pp. 1-14 (2012).
14. Shiffman D.: Genetic Algorithm, Evolving Shakespeare, The Nature of Code [Source Code], https://github.com/nature-of-code/noc-examples-p5.js/blob/master/chp09_ga/NOC_9_01_GA_Shakespeare/Population.js (2016), Accessed 2 May 2020.