Real Time Pathfinding with Genetic Algorithm

Alex Fernandes da V. Machado, Ulysses O. Santos, Higo Vale, Rubens Gonçalvez
Depto. Acadêmico de Ciência da Computação
Inst. Federal de Educ. Tec. do Sudeste de Minas
Rio Pomba/MG, Brasil
alexcataguases@hotmail.com, ulyssesdvlp@gmail.com, gbrubens@hotmail.com, higovale@hotmail.com,

Tiago Neves, Luiz Satoru Ochi, Esteban W. Gonzalez Clua.
Departamento de Computação
Universidade Federal Fluminense
Niterói/RJ, Brazil.
tiagoanz@gmail.com, satoru@ic.uff.br, esteban@ic.uff.br

***Abstract*-This paper presents a method to optimize the process of finding paths using a model based on genetic algorithms and A * for real time systems, such as video games, virtual reality environments. The proposed solution uses obstacle pattern detection based on online training system that is generally used in systems with real time requirements and in dynamic environments. The architecture, named Real Time Pathfinding with Genetic Algorithm (RTP-GA), uses a Genetic Algorithm in order to create an agent adapted to the environment that is able to optimize the search for paths even in the presence of obstacles. In specific cases, the RTP-GA architecture presents a complexity that is better than A* algorithm.**

Keywords: pathfinding, genetic algorithm, dynamic environment

I. INTRODUCTION

The real-time search of the shortest paths between two points is a fundamental problem for the area of electronic games, but it is also relevant for other areas [Stodola and Mazal, 2010]. The A * algorithm [Hart, Nillson and Raphael, 1968] (and its variations [Bjornsson et al., 2005]) is one of the key methods for finding such shortest paths. The A* combines dynamic programming and an admissible heuristic for pruning the search space. However, due to its complexity, the available computing power available in the game cycle may not be enough for running the A*. Moreover, the environments are dynamic, meaning that changes in the positions of the obstacles may occur. In such dynamic environments high time demanding algorithms, such the A*, may not be a suitable option. Since such algorithms may require many seconds to end, changes in the environment may occur while the algorithms are still running. As a result, the problem solved by the algorithm is no longer the same and consequently the solution found may no longer apply. Also, using the A* in multi-agent games may not be possible, especially in very constrained resources platforms such as tablets and cellphones.

This paper proposes an algorithm based in Genetic Algorithms [Mitchell, 1996] and in the A * to optimize the search for shortest paths in dynamic environments with real-time requirements. With this technique, the agent is able to adapt yourself in an homogeneous maze, using an heuristic that let the agent calculates an path thought a considerable number of open nodes, in a single step, without having to check the neighbors for each node visited (as demonstrated in Fig. 1).

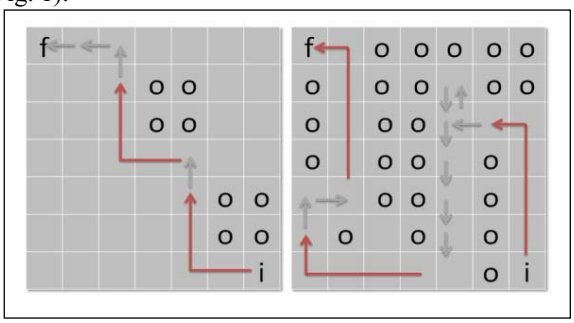


Fig 1: Two examples of obstacles adapting with Real Time Pathfinding with Genetic Algorithm (RTP-GA)

This paper, first presents an survey about the related works in path finding. After that, the RTP-GA algorithm idea is presented, followed by the algorithm specifications and features. In the end an implementation is presented with testing and comparative analysis between the RTP-GA and A* algorithm.

II. RELATED WORK

Searching for patterns in maps in order to optimize the pathfinding algorithms is a well known technique. Demyenand and Buro [2006] shows that the Triangulation Reduction A* (TRA*) method has many benefits, including accurate representation of polygonal environments, reduction of the search space and refinement of the optimal path. Although this algorithm uses real-time learning, it has severe constraints regarding the environment. To use the TRA* we must provide a description of polygonal obstacles in the environment, which can be easily applied to maps with barriers and holes but it is not suitable for labyrinths.

The use of meta-heuristics for adaptation to the environment is a topic addressed by several authors. Leigh et al. [2007] proposed the GA-Manufactured Maneuvering Algorithms (GAMMAs) that was able of finding paths more than 1000% faster than the traditional A*. In presented architecture the GA was able to adapt to the obstacles

(called risk zones), approach the human-made path and define the best way to evaluate the path. However GAMMA depends on a prior offline training, that consists in exploring some maps in order to extract some patterns before the optimization process.

Burchardt and Salomon [2006] implemented a Genetic Algorithm to search for paths for robot routing. The algorithm runs until the agent achieves its goal. There is a constant update process on the environment. The fitness function is based on the length of the path with penalization on collisions. The gene encoding follows the pattern:

Length	X_1	Y_1	X_2	Y_2	X_3	Y_3

The path is encoded as equidistant distances between the start and finish, allowing the occurrence of a deviation. However, this approach only works when a nearly direct route exists.

Bulitko and Lustrek [2006] call "lookahead" the incomplete search method, which is similar to the min-max based algorithms used in two-person games. It conducted a limited depth search, expanding of the space centered on the agent, and heuristically evaluates the distance between the agent and the destination. However, due to the lookahead pathology [Bulitko and Lustrek, 2006], determining the optimal depth for the search procedure is not an easy task.

As in [Demyenand and Buro, 2006] and [Salomon and Burchardt, 2006], this work investigates partial solutions in the map, trying to minimize the problems defined by Bulitko and Lustrek [2006] when using this incomplete search method. Intends to hold a learning through an online training (as opposed to Leigh et al. [2007]). Through a codification of the route, such as Burchardt and Salomon [2006].

III. THE RTP-GA ALGORITHM.

The proposed method generates possible sub-paths to be applied in a map randomly, evaluate the sub-paths in the map, and then adapt the sub-paths in order optimize the path traversed. Thus for standard environments with obstacles, the RTP-GA has a very low computational workload can be used in dynamic environments.

We decide to use Genetic Algorithms for solving the pathfinding problem combining two approaches of the literature, presented in Leigh et al. [2007] and in Burchardt and Salomon [2006]. The difference between our model and the one presented in Leigh et al. [2007] is that we use an online training instead of an offline one. Our approach also improves the model presented in Burchardt and Salomon [2006] since we define a chromosome able to adapt to any environment that present obstacle patterns.

IV. RTP-GA

Genetic Algorithms use techniques based on evolutionary process species in order to find good solutions for optimization problems. The algorithm iterates over a population of solutions, that are called individuals or chromosomes. On each generation (algorithm iteration), new individuals are generated by crossover and mutation process and only the most suitable individuals survive for the next generation. The individual fitness is calculated by a fitness function, or objective function. Thus, in each generation, only the individuals with the best values of objective function are selected to be in the next generation.

It is important to note that, in the presented architecture, the quality of individuals (paths) is measured not only by the objective function, but also the environment, ensuring an adaptation to changes in the map.

A. Chromosome representation

As one of the goals of the algorithm is to detect obstacle patterns in the map, the proposed Genetic Algorithm uses a model based on four movement vectors, two horizontal and two vertical. These allow you to make up four straight since a single line to a deviation in the shape of L or S of an obstacle in any one of its sides. As the maze gets narrow, the distance of the vectors (lines lenght) tends to decrease, and vice-versa. If the obstacles can be overcome by contourning them, these vectors tend to find this pattern. Each of these four vectors have a distance and a direction, as expressed in Table 1:

Type	Interval	Description
Interger	1 to 4	Distance of Movement 1 (DM1)
Interger	1 to 4	Distance of Movement 2 (DM2)
Interger	1 to 4	Distance of Movement 3 (DM3)
Interger	1 to 4	Distance of Movement 4 (DM4)
Interger	-1 to 1	Movement Direction 1 (D1)
Interger	-1 to 1	Movement Direction 2 (D2)
Interger	-1 to 1	Movement Direction 3 (D3)
Interger	-1 to 1	Movement Direction 4 (D4)

Tab. 1 – Chromosome encoding.

The interval at which movement distances may vary represent a percentage of the width (in the case of horizontal vectors) and height (in the case of vertical vectors) of the map. The directions are given by:

- M1 and M3: 0 no movement, -1 down (subtracts the y-axis) and +1 up (increase in y-axis).
- M2 and M4: 0 no movement, -1 to the left (subtracts the x-axis) and +1 to the right (increase in x-axis).

B. Fitness Function

The objective function (OF) is used to check each candidate path. It takes into consideration the following terms:

- The Manhattan Distance (MH), which has the largest weight in the formula. Its equation is given by MH = | x1-x2 | + | y1-y2 |. The higher the the value of MH the the worst situation, since it means that the agent is far from the final target;
- Total Movement (MT), calculated by the sum of half the distance moved. The weight of MH is smaller than the one of MH to serve as a tie-breaker, especially in the elimination of cycling paths;
- Convergence (C0), assumes 0 if the path cannot be traversed, meaning that an obstacle was found, and assumes 1 in case the individual represents a valid path.

This objective function must be maximized, meaning that the higher the value the better the candidate. To ensure this feature was used as a roof value given by:

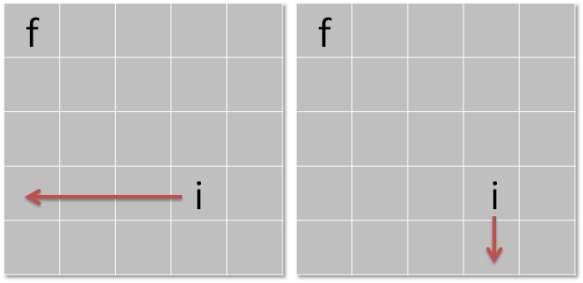
$$Roof = (Width + Height) * 3$$

Value that represents the sum of the maximum of four movement vectors plus the maximum value of the Manhattan distance. If a path is not valid the convergence ensure that the result is the lowest possible (in this case, zero). The objective function is given by:

$$FA = (Roof - MH - MT * 0,1) * C$$

C. Movement Instances

In order to show the types of changes that may happen, we will consider, with the loss of generality, a 5x5 dimension map. Also let "i" be the initial point of the agent and "f" be the final point (exit). The path under consideration will be represented by arrows in the map. The following situations may be represented in our chromosomes (Fig. 2):



(a) (b)

Fig. 2 – Straight movement.

Figure 2 shows a visual representation of the chromosomes exposed in Table 2.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,2	0,6	0,4	0.1	0	-1	0	0
0,1	0,9	0,7	0.1	-1	0	0	0

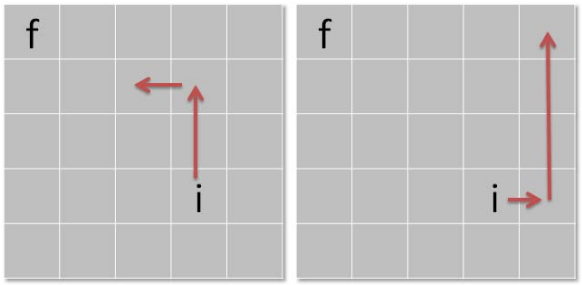
Tab. 2 – Values of straight movement.

Table 2 (line 1) presents an individual in which only the M2 field has a valid value. Thus it represents a single movement to the left on the map (due to the negative value). The distance traveled is a fraction of the map size. As DM2 (relative to M2) has value 0.6 and the map has width 5 the total distance traveled in this movement is given by 0.6 x 5 = 3 squares, as shown in Figure 2 (a) . The chromosome exposed in Table 2 (b) (line 2) also shows a movement with a single direction, presented in Fig. 2 (b). The Manhattan distance value of both chromosomes, i.e. the number of squares between the resulting point of the movement and the maze exit, are 3 and 7, respectively. The objective functions of both chromosomes are shown:

$$FA(a) = (30 – 3 – 3 * 0,1) * 1 => 26,7$$
$$FA(b) = (30 – 7 – 1 * 0,1) * 1 => 22,9$$

Since the first chromosome of Table 2 has a higher O.F. it is considered better than the second one.

In the next set of examples we will evaluate a movement with a single deviation (Figure 3) represented by the chromosomes of Table 3:



(a) (b)

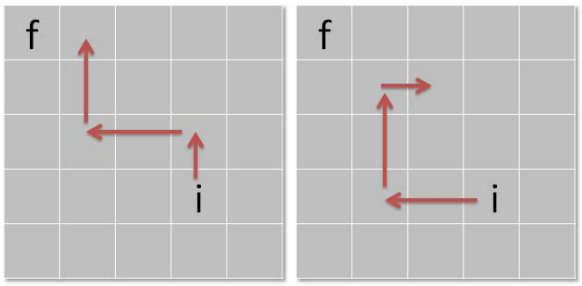
Fig. 3 – Movement with deviation.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,4	0,2	0,5	0.9	1	-1	0	0
0,6	0,2	0,8	0.1	0	1	1	0

Tab. 3 – Values of deviation movement.

The first chromosome has two nonzero directions: M1 vertically and M2 horizontally. Therefore we will have two line segments forming a path with deviation. A similar situation occur in the second chromosome. Both chromosomes are graphically exposed in Figure 3.

Table 4 presents chromosomes with two deviations, shown in Figure 4:



(a) (b)

Fig. 4 – Movement with two deviations.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,4	0,2	0,5	0.9	1	-1	1	0
0,6	0,2	0,8	0.1	0	-1	1	1

Tab. 4 – Values of the two deviations movement.

Figure 5 shows a path in which that agent collides with an obstacle:

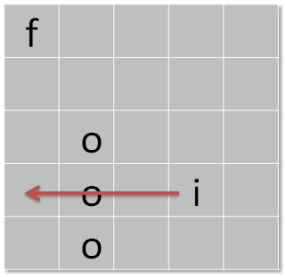


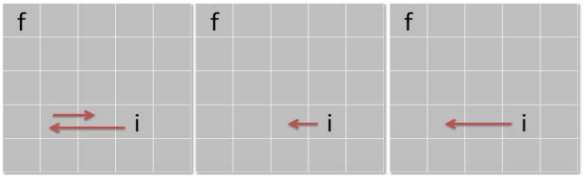
Fig. 5 – An example of an invalid path.

Considering each "o" on the map as an obstacle, it is clear that the path exposed in Figure 5 is not possible. Thus the objective function for such path is given by:

$$FA = (30 – 3 – 3 * 0,1) * 0 => 0,0$$

As can be observed, the convergence of the path is set to zero and it ensures a low value of the objective function to the chromosome related with such path. The low values in the objective functions indicate that solutions are unfeasible or poor in quality, thus the Genetic Algorithm must ensure that such solutions do not pass genetic information to future generations.

Figure 6 shows how the objective function is used to select better paths.



(a) (b) (c)

Fig. 6 – Movements with approximate fitness result.

In the three cases presented in Figure 6 the map has the same configuration. In Figure 6 (a) and Figure 6 (b) the agent movement ends in the same place, but in Figure 6 (a) makes unnecessary movements. When evaluating the chromosomes, the objective function of the three cases are given by:

$$FA(a) = (30 – 5 – 3 * 0,1) * 1 => 24,7$$
$$FA(b) = (30 – 5 – 1 * 0,1) * 1 => 24,9$$
$$FA(c) = (30 – 4 – 2 * 0,1) * 1 => 25,8$$

Therefore, although Figure 6 (a) and Figure 6 (b) have the same Manhattan distance, the total movement (MT) is responsible for appointing the best chromosome, in this case Figure 6 (b). However, when comparing Figure 6 (b) and Figure 6 (c) we see that although the MT Figure 6 (b) is smaller, the value of Manhattan distance of the chromosome in Figure 6 (c) ensures a higher value in the Objective Function revealing that this chromosome is the best among the presented cases.

D. The Genetic Algorithm

Our Genetic Algorithm starts whenever the agent collides with an obstacle. Before the collision, the greedy A * is used without considering the neighbors. This simple version of the A* is used because the GA is responsible for running a local search in the map. It is important to note that the decision of not consider the neighbors in the A* causes a significant speedup in the algorithm.

Figure 7 shows the proposed Genetic Algorithm that that perform the following steps:

1. Start with an empty population of individuals. Complete the population of size N=4 with random generated individuals.
2. Create N chromosomes using crossover operation on the initial population.
3. Create N chromosomes using mutation operation on the initial population.
4. Check the convergence of all chromosomes on the map. At this point the map may also be altered because the visited squares are marked.
5. Evaluate the chromosomes using the objective function , select the N / 2 best chromosomes (elitist criterion) and apply the selected chromosomes on

the map (at this point the agent will move through the map).

6. Greed A* is used to navigate the environment until the desired point has distance greater than manhattan's point of origin. Therefore the agent should continually check the new position on the map and mark the visited states.
7. If the comparison between the manhattan's distances show that the chosen point is not feasible to continue the flow of the algorithm using the GA module.
8. If the point chosen by A * and sub-route generated by the GA does not conflict with the stopping criterion (the "output"), take the N / 2 better, generate more N / 2 random and back in step 2.

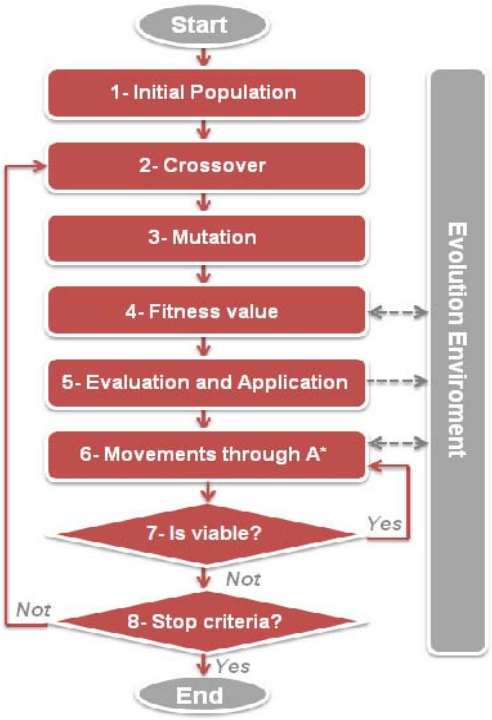


Fig. 7 – Real Time Pathfinding with Genetic Algorithm (RTP-GA).

E. Local Optimum treatment

We consider a local optimum of the path-finding problem, as modeled here, a region of the map that is relatively close to the maze exit and that all paths to the exit are unfeasible or pass through an already visited square (Fig. 8). In such cases we use the visited squares by the sub-path as obstacles only if they are the resulting point of a path, which induce the GA to search regions far from the maze exit, which can be considered a diversification strategy. To avoid the greedy A * module come back for these local optimums, it treat this sub-path as an obstacle.

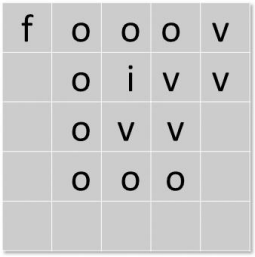


Fig. 8 – Example of a local minima
(the "v" symbol represents an visited node).

V. - IMPLEMENTATION

Since one the goals of this work is to prepare a test bed for optimization algorithms for the pathfinding problem, suitable programming language and development environment were chosen. Another factor considered in the choice of language and development environment was the possibility of exporting the developed systems to mobile devices such as tablets and mobile phones where the optimized use of the resources is highly recommended.

We performed 20 tests on three different mazes. In each test agents were placed in different positions.

A. Technology

The development technologies used were Unity 3D [UNITY TECHNOLOGIES] through the C # language.

Unity3D 3.x is a game development engine Which has the main components needed for designing games developing rapidly, such as the physics engine, collision systems, sound system and a high level programming language based on C, among others.

Unity 3D has an interface for programming in Mono. It incorporates key .NET components, including a compiler for the C# programming language and a complete suite of class libraries.

The direct assignment of a list of values to arrays in C # allowed an easy viewing of maps used in the experiments. Thus the generation of a tileset, which is the display of graphics (or sprites) defined in a two-dimensional array can be made as shown in Figure 9.

The modularization of the code and Object-Oriented paradigm allows dynamic instantiation of both the environment and the agents in Unity.

```
string[] map = new string[20]
{
   "ooooooooooooooooooo",
   "of....o...........o",
   "o..o..............o",
   "ooooooooo......ooooo",
   "ooo....o......o....o",
   "o.oo..oo......o....o",
   "o............io....o",
   "ooooo.oooooooooo....o",
   "o...o.o............o",
   "o...o.o....o.......o",
   "o...o.o....o.......o",
   "o...o.o....o.......o",
   "o...o.oooooo.ooooooo",
   "o..................o",
   "o...o.o............o",
   "o...o.o............o",
   "o...o.o............o",
   "o...o.o............o",
   "o...oio.....i......o",
   "ooooooooooooooooooo"
};
```

Fig. 9 – Dynamic tileset of an map, implemented in a c # matrix.

The result of the algorithm applied to the tileset defined in Figure 9 executed in a Tablet with Android Operational System 2.1 Eclair is exposed in Figure 10.

Fig. 10 – RTP-GA simulation with three agents in field.

B. Tests

We conducted three tests to compare the expense of memory between A* and RTP-AG algorithms. Each in a different map, with dimensions of 40 by 40.

In all, to be deterministic (and therefore not generate solutions with different memory spent in an environment without variation), the A * was performed only once. In contrast the average was calculated for 100 runs of RTP-GA in each test case.

The maps were

- Without Obstacles - to assess the direct search;
- With patterns (first Fig. 11) - to assess the advantages of the adjustment proposed by RTP-GA;
- Without patterns (Fig. 11 second) - to evaluate the worst case of RTP-GA.

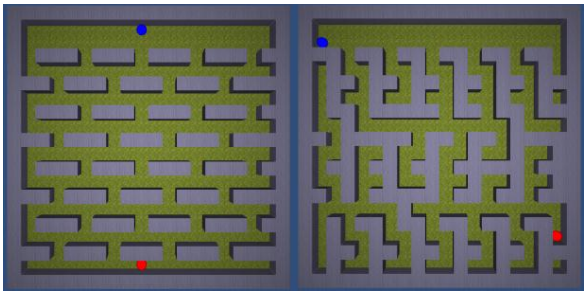


Fig. 11 – Respectively: maps with and without patterns.

C. Results and Analysis

The results show that the agents managed to adapt themselves to the surrounding environments even in the presence of obstacles.

Two situations can be observed in the Figure 1. The first map has 2x2 obstacles, the second has long corridors. In both cases the agent is able to adapt himself.

The result of the experiments can be checked in Fig. 12:

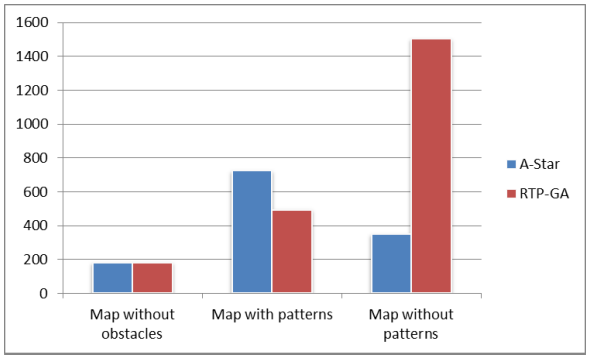


Fig. 12 – Comparison between the execution of algorithm A-Star and the average execution of RTP-AG algorithm.

By comparing the RTP-GA with A * in a map without obstacles, as predicted, there was a tie. This is due to the fact that, initially, the RTP-based GA uses the A * algorithm until an obstacle in his way.

In comparison made on a map with obstacles with patterns RTP-GA algorithm achieved satisfactory results, outperforming the A * 90% of cases this is due to the learning agent, in this type of map is used for the generation of the sub-paths that does not occur in environments with obstacles without default.

In the test with obstacles, but without a pattern, the result was not satisfactory. On average, it was much worse than A *. This is due to the fact that in this case the RTP-GA algorithm tends to generate sub-paths more frequently,

because there is no genetic advantage of the previous chromosomes which leads to an overspending of memory.

The objective of the tests is to check if the agents are able to find a path to the maze exit. In 100% of the cases the agents achieved the goal.

The results were as expected since the RTP-GA has a better performance on standardized maps with obstacles due to the adjustment provided by the Genetic Algorithm, being impracticable for maps that have no standard for the evaluation of sub-paths generated is costly.

VI. CONCLUSION AND FUTUREWORK

This work defines a new method to search the shortest path between two points in 2D maps. The RTP-GA uses the classic A* inside a Genetic Algorithm in order to find good solutions in a short period of time.

The proposed GA used lookahead based modeling to the chromosome that can be adapted to any kind of map that contains obstacle patterns. Also the lookahead pathology is avoided by the iterative nature of the algorithm.

Our experiments were conducted on large maps of this form with similar characteristics to adapt to dynamic environments, because in the second RTP-GA would have a latency of learning in small variations of the environment because genetic information.

As future work will seek improvements in this architecture to optimize the search process. Such as varying the number of segments of lines contained in the chromosome.

ACKNOWLEDGES

This work was supported by the PET-MEC (Programa de Educação Tutorial do Ministério da Educação, Brasil).

REFERENCES

UNITY TECHNOLOGIES: *Unity 3D User Manual*. At www.unity3d.com/support/documentation/manual [Accessed 10 July 2011].

LEIGH, R., LOUIS, S. J. AND MILES, C. *Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms*. In: IEEE Congress on Computational Intelligence and Games. CIG – 2007.

DEMYENAND, D. AND BURO, M. *Efficient Triangulation-Based Pathfinding*. In: AAAI'06 Proceedings of the 21st national Conference on Artificial intelligence. 2006.

BURCHARDT, H. AND SALOMON, R. *Implementation of Path Planning using Genetic Algorithms on Mobile Robots*. IEEE Congress on Evolutionary Computation. CEC 2006.

BULITKO, V., LUSTREK, M. Lookahead pathology in real-time path-finding. In Proceedings of the National Conference on Artificial Intelligence. AAAI 2006.

HART, P.E., NILLSON, N.J. AND RAPHAEL, B. *A formal basis for the heuristic determination of minimum cost paths*. IEEE Transactions on Systems Science and Cybernetics, 1968.

BJORNSSON, Y., ENZENBERGER, M., HOLTE, R.C. AND SCHAEFFER, J. *Fringe search: Beating A* at pathfinding on gamemaps*. IEEE Computational Intelligence in Games, 2005.

STODOLA, P. and MAZAL, J. *Optimal location and motion of autonomous unmanned ground vehicles*. In World Scientific and Engineering Academy and Society. WSEAS 2010.

MITCHELL, M.; *An introduction to genetic algorithms.* The MIT Press, United States; 1996.