Path-IT - Using Genetic Algorithms to perform Path Finding over defined constraints

Tuesday 18th July, 2023 - 12:58

Tinouert Alexandre
University of Luxembourg
Email: alexandre.tinouert.001@student.uni.lu

This report has been produced under the supervision of:
Pr. Dr. Luis Leiva
University of Luxembourg
Email: luis.leiva@uni.lu

Abstract

In the wake of modern uses of Artificial Intelligence, particularly the practices of Machine Learning, new techniques proved new ways of enhancing automated tasks by learning. One such related technique is the general realm of Genetic Algorithms, as their purpose are based on the Darwinian Evolutionary Principle, which is a relevant way to understand and produce new softwares based on Genetic Algorithms. However, the efficiency of Genetic Algorithm might be uncertain, as this approach might seem random in itself, especially over small predicates or conditions. This development will lead to prove that Genetic Algorithms are nonetheless efficient. Both the theoretical and practical model show promising results to the use of Genetic Algorithm as a doctrine, although it seemed that more or less random mutations coupled to crossovers might lead to convergences to unwanted behaviors. Furthermore, Genetic Algorithms tend to converge quickly to solutions. The results encourage the use of Genetic Algorithm as a convenient way to assess a task's difficulty as well, showing some disparities in efficiency but in the end, converging to the correct, or expected result in a task. This paper anticipates not only the efficiency of a Genetic Algorithm model in a certain task, but how it can be made to be adapted to any other task following some degree of constraints and/or difficulty. For example, the model can be replicated over some other tasks with similar inputs, and the task itself can have different constraints or initial conditions.

1. Introduction

This report presents the second Bachelor Semester Project performed by Tinouert Alexandre during the Summer Semester of 2023. It informs about the developed scientific and technical deliverables, meaning a mathematical and scientific demonstration as well as a developed software to answer the main question that this BSP raised: "Are Genetic Algorithms efficient to perform tasks like pathfinding?".

The main setup use case would be as follows, individual creatures in the form of ships (as "individuals", "creatures" and "ships" will refer to the same entities in this BSP), have to roam around a map and find the exit of the map. Their knowledge of the map is only their coordinates, the ones of the exit of the map, and eventual obstacles near them. Their speed is constant and can use Neural Networks to make moves, like turning left or right, and Crossover and Mutations to pass their characteristics if they manage to find the exit of the map, hence if they were the best creatures of their respective group.

Although using algorithms like A^* or Dijkstra's algorithm would be useful for path-finding, the purpose of this BSP resides in the following primitive Design of the Scientific Deliverable.

As a Design for the Scientific Deliverables, this BSP aims to define the basics of Genetic Algorithms. Design definitions will help demonstrate a use case with Genetic Algorithms, proving its utility in such a use case or similar ones. This use case regards pathfinding, with grid-located obstacles but more free moves for the cursor. A fitness function will evaluate the individuals according to the epoch/generation and select the best individuals, which are the ones that arrived at the best time or the ones that came the closest to an arbitrary exit of the map. In this case, Genetic Algorithms can be used as they converge to solutions quickly thanks to parenting genes and mutations, which will be explained mathematically.

Technically, using the Scientific setup, the purpose would be to create an application allowing to perform training and testing over those constraints. As such, running multiple generations would allow the creation of a set of individuals capable of achieving maps with a relatable good efficiency. This program will be a web-based program, written in JavaScript, with the notable help of Canvas' integrated API to render the base game on which Genetic Algorithms will be used upon training and testing.

To sum up the deliverables:

- Scientific Deliverable:
  - "Are Genetic Algorithms efficient to perform tasks such as pathfinding?"
- Technical Deliverable:
  - Development of Path-IT, a software based on a game

environment/engine with its constraints, as well as an interface to perform Machine Learning.

2. Project description

2.1. Domains

2.1.1. Scientific. This BSP will cover the basics of Genetic Algorithms. With the use case, the main computation as well as the purpose and general structure will be seen.

2.1.2. Technical. The development of Path-IT covers the fundamentals of JavaScript, in the context of Object Oriented Programming, such as classes, methods, and inheritance, all the way to JSON.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. As in the question, a direct subquestion emerges. "What are Genetics Algorithms?". As Genetic Algorithms imply Machine Learning, precisions have to be made about the use, hence structure, and the semantics of the inputs, processes, and outputs. Moreover, in what context will Genetic Algorithms be used? As suggested in the question and Introduction, mainly pathfinding. Hence, how to define the use case mathematically, and what are its capacities, moves for individuals, and definitions such as tick? This suggests a deeper definition of the use case than provided in the Introduction. As such, defining the use case may be a rule of thumb, but more precise constraints will have to be cited in the Scientific Deliverable demonstration to understand the context. Next, definitions of Genetic Algorithms will be addressed in the Pre-requisites and the demonstration itself. It is however expected to have at the end of the demonstration, an individual with good efficiency in pathfinding with desired constraints.

2.2.2. Technical deliverables. To assess Genetic Algorithms' efficiency in path finding, first would be needed the development of a game with the cited constraints. This game would have defined controls, obstacles and endpoints. Next, as for Machine Learning, a medium dataset of maps will have to be made in order to make path finding usable no matter the combination of beginning and end states, and of course obstacles, omitting impossible ones (the ones that, any creature from the starting point cannot reach the endpoint). Hence, the development of Neural Networks for creatures will have to be made with its definitions (propagation, functions and structure). Finally, the application should render waves of creatures, which are epochs, or generations, and perform Machine Learning in the scheme of Genetic Algorithms, and be able to save the abilities of the yet best creature.

3. Pre-requisites

Here are the respective pre-requisites of this BSP, representing the knowledge that is known before beginning this BSP.

3.1. Scientific pre-requisites

For the Scientific Deliverable, definitions of Machine Learning, and Neural Networks are needed. Especially, the main structure of a Neural Network has to be defined, accompanied by basic algebraic definitions that follow. Consequently, matrix multiplications and manipulations are needed. Next, the concept of forward propagation is also a must as it is the main computation between the different layers of the network, with basic weighted and biased sums, and their assigned weights and biases, thus constructing the structure of neurons in a Neural Network. Algebraic usual functions such as exponential or logarithms are also needed for the definition of the activation function. Those definitions will be adapted to the use of Genetic Algorithms, aside

3.2. Technical pre-requisites

Next, for the Technical Deliverable, medium knowledge of JavaScript is needed. For example, data structures such as JSON are a must, but knowledge of Canvas is also much needed for the development of the game, as Canvas graphically renders the interface. Other than this, doctrines like Object Oriented Programming will also be needed for objects like individuals and eventually Machine Learning process parts.

4. Genetic Algorithms - an efficient way to perform Machine Learning

This section covers the Scientific Deliverable of this BSP. The purpose of it is to demonstrate that Genetic Algorithms are efficient regarding Machine Learning.

4.1. Requirements

Answering the question "Are Genetic Algorithms efficient to perform tasks such as path finding?" is vague, as definitions of "Genetic Algorithms" have to be made, but also a correlation between Genetic Algorithms and the tasks performed. To ease the whole process of the Scientific Deliverable, here are the requirements to meet.

4.1.1. Define a use case.

Before tackling more abstract definitions (Genetic Algorithms), defining a use case is more appropriate. The use case needs to be tangible and relatable, here in the span of pathfinding, although it would work for any other span, as long as it defines the limits of the subject or the study. As such, tasks need to be straightforward with an input, and an output. Hence, a general starting point, and a general ending point, since those tasks will be repeated over and over, assuming that they will be studied for the sake of Machine Learning. In this perspective, the outlook of induction can be used in such a finite amount of tasks, to define a base case, then a general transition from a task n to task $n + 1$ as tangible and naive output values such as efficiency are easily tractable.

4.1.2. Recall basics of Machine Learning and introduce Genetic Algorithms.

At this point, defining the limits of the definitions of Machine Learning and consequently Genetic Algorithms is needed. The goal is to define the structure and the process of how Genetic Algorithms behave in the scheme of Machine Learning. This will be an extension of the use case requirement, as the use of Genetic Algorithms will impact the behavior of the use case itself (namely some of the imposed move constraints in pathfinding). At the end of this requirement, enough definitions will be cited to assemble the complete pieces of the chosen base case and follow induction.

4.1.3. Perform induction on tasks.

This requirement showcases the main computations from the base case, to the transition from case/task n to $n+1$ as a finite amount of cases can be defined to not overload the general use case. The definitions made from the first two requirements are vital. This induction takes a random fitness score and its goal is to determine that behavioral convergence happens and can be avoided using random mutations.

4.1.4. Analyse the quantifiable yet naive general output.

After the induction, general values are expected to be obtained. Although an "efficiency" value cannot be directly output from the last requirement, others can define it properly, such as time for example. However, in this Scientific Deliverable, only the completion of the task itself is considered. The goal of this requirement is to define efficiency regarding the use case first, then more broadly concerning Genetic Algorithms, to assess its general efficiency and answer the question.

4.2. Design & Production

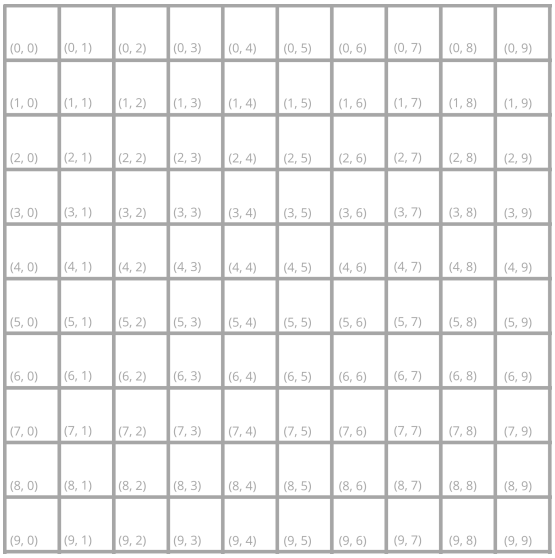
4.2.1. Use case definitions and limits.

For the sake of simplicity, the use case of pathfinding will be made over simple and rigorous limits and constraints.

Let the map be a $10 \cdot 10$ grid with requirements as such:

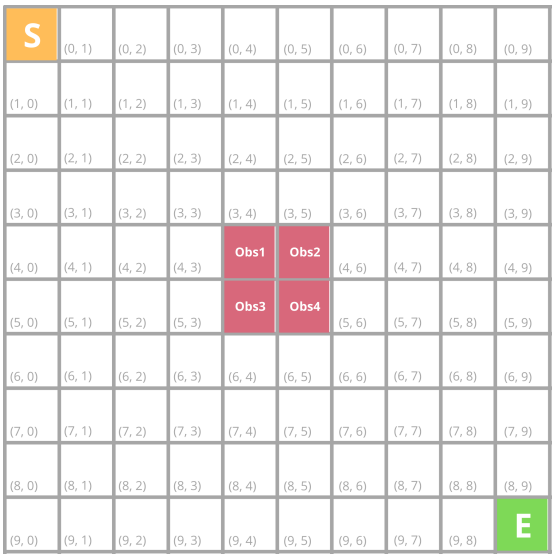
- One starting point S, shown in yellow.
- One ending point E, shown in green.
- Multiple obstacles with their coordinates arbitrarily registered in an array $Obs = [Obs_1, Obs_2, ... \ Obs_n]$ with $Obs_i \ \forall i \in \mathbb{N}$ being tuples of their respective x and y coordinates, shown in red.
- The rest of them are empty slots in white.

Here is an example of the invalid empty $\emptyset$ map:



Empty "$\emptyset$" map

And, as an example of a valid map, here is the $debug\_1$ map used in the Technical Deliverable (Path-IT):



"$debug\_1$" map

In this case, assume that the map used is doable, i.e. no strip of obstacles completely separating the S and E, but there can be as many obstacles as long as E is "reachable" from S, diagonal cut-offs also count as a continuous strip of free slots must exist.

From now on, the goal would be to spawn creatures on S and make them arrive at E. Assume those creatures/ships are objects that are relatively small compared to slots.

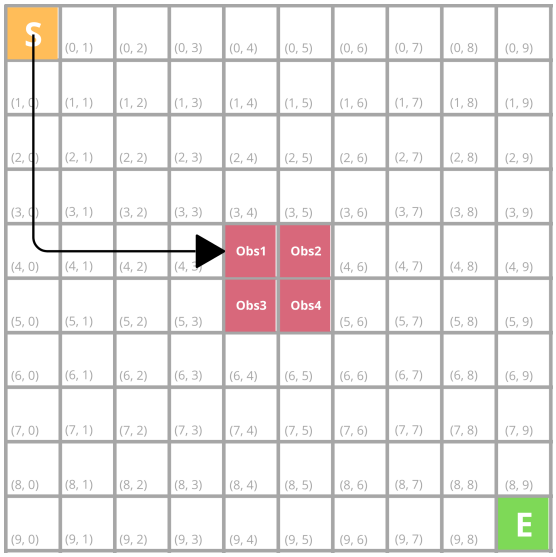
For the use case to be more developed, those creatures can "freely" roam at a constant speed, and turn at a constant pace. Those values are not much relevant in the use case, so assume

a tick rate of 20 fps (frames per second) at which the situation will update, meaning that ships can make a three-prompted decision, either:

- Go straight ahead at speed v.
- Turn left at angle $+\theta$.
- Turn right at angle $-\theta$.

Again, assume v and θ are not much relevant for the scientific deliverable as Genetic Algorithms can adapt to those.

Consider now a run/task/case, being a run of the simulation. Here is an example of a potential random run with a ship going from S, with a randomized erratic behavior:



Example of run on "$debug\_1$" map

Here, the ship departed from S, turned right, and hit Obs_1, failing the map. If the ship managed to get to E, the ship would have succeeded in the map.

This design-oriented subsection brings the necessary limits of the use case, especially physics and general goals and objects, that will be used in the following subsection.

4.2.2. Genetic Algorithms' Introduction and Context.

Genetic Algorithms are a section of Machine Learning, based on the Evolutionary Principle by Charles Darwin, stating that creatures evolve to match and master their environment. With this broad definition, assuming the Evolutionary Principle, Genetic Algorithms rely on trial and error. There should be a creature trying and failing while having an indicator of its general score to favor the best behaviors and genes possible.

In the realm of Computer Science, Genetic Algorithms often base themselves on the topics of learning algorithms and simulations. As such, the imposed constraints will greatly impact the creatures subjected to them, such as obstacles, traits, and other characteristics.

Creatures begin to roam in their environment with initialized random traits. For creatures to evolve, they perform tasks in "Generations", so spawning multiple creatures at the same time and measuring their activity. The quality of the activity is measured by a "Fitness function" (i.e. the time their survived or the fastest one to perform a task), here based on the completion rate. In this case, the main Fitness function used here is based on the A* algorithm, which allows us to measure the general progression of the ship throughout the task. Hence, the "best" creature can be selected through this process. At this point, there may be multiple patterns of learning according to the process, but in this particular case, assume that the best creature selected can pass its genes to the next generation. This ship will survive along with other creatures, which are children of the best creature, and the other creatures who will disappear. As such, those creatures will be equipped with 50% of the best creature's genes and 50% of its other parent creature's genes. This process of creating a new generation is called "Crossover". Before spawning the next generation, random mutations can be made on the population's genes to avoid behavioral convergence in case of trick or unwanted behaviors being the norm.

To sum up, Genetic Algorithms' process repeated for as long as wanted is as such:

- Spawn a generation of creatures (randomly initialized if it is the first).
- Creatures do actions and are graded on their quality.
- Select the best creature among the generation.
- Crossover with the best creature and the other ones. The best one will survive.
- Mutate randomly to avoid creatures being stuck in particular cases.

4.2.3. Use case outlook on Genetic Algorithms.

In this studied case, assume ships are creatures, spawned over generations on S. Ships roam around the map and their goal is to hit E. Ships are ranked from the closest one of E to the farthest. If two ships arrive at E, the one that arrived the earliest is selected.

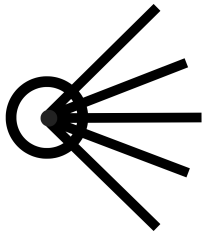
However, at this point, determining the traits to be passed is impossible to determine, as no tangible genes have been created and assigned to ships.

In this case, ships can measure their coordinates (absolute position, x and y), the coordinates of E, and sense their environment at a limited distance. For the sake of simplicity, assume the ships can sense obstacles and collisions for as much as three times their width. Defined values will be mentioned in the Technical Deliverable, as Genetic Algorithm can bypass defined values over generalized values, as long as those traits are similar in form, assume differences will be negligible. This model will be seen in the next subsection.

When all ships have died or are in a locked state of looping, selection occurs, and the best ship is selected, and crossover happens over all other ships, with some random mutation. Hence, how to model the ships' genes can be answered using Neural Networks

4.2.4. Neural Networks modeling for traits - Input.

Assume ships to be of this general shape:



Example of a ship's shape

This ship possesses 5 captors with a certain nonnull length, of around 2 to 3 times the ship's length as an arbitrary value. Again, for the sake of simplicity, if one of the ship's captors hits, its assigned "hit value" will be 1, or 0 if no hit is detected.

By generalizing this factor, ships now have 5 captors of a certain length, registered in an array $CapTouch$, which also registers the "hit value" of those said captors.

Next, ships must be able to turn left or right each tick of the simulation or decide to go straight. The intensity of turning is arbitrary, just as the length of the captors is, as long as a ship can turn around in a square unit of a map.

Finally, assume that ships know their exact coordinates on the map. That is absolute coordinates. As in the Technical Deliverable, although the map is a 10×10 grid, the map is rendered as a 480×480 pixels square. As such, some ships may have coordinates with x and y ranging from 0 to 480, with the top left corner being the origin. Note that those coordinates are ordered just as in the map example, with x increasing going from left to right, but y increasing from up to down. Furthermore, to avoid ambiguity, knowing coordinates does not allow the ship to know the location of objects.

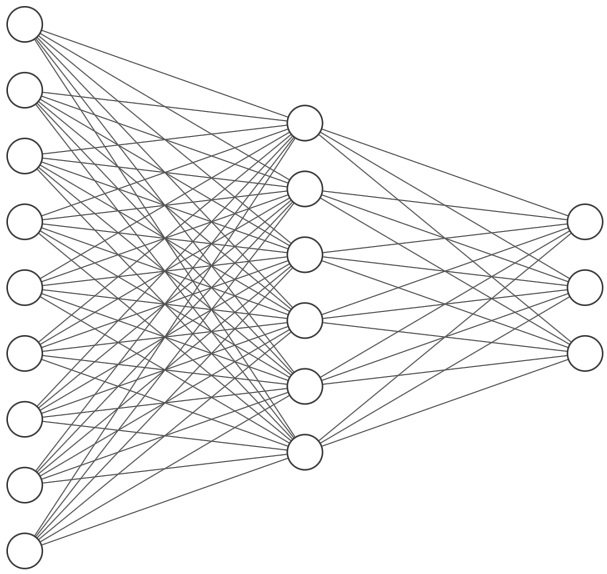
However, and to top it all, ships are allowed to acknowledge the endpoint's coordinates, as this will be useful as guidance for the ships later on.

With all those determined traits and defined limits for ships, the obtained input value for ships is a 9-array of normalized floating values. Hence, before compiling those values, they are divided by the highest possible obtainable value to be normalized between 0 and 1:

$$Input = [x_{ship},\ y_{ship}, CapTouch[1],\ CapTouch[2],\ CapTouch[3],\ CapTouch[4],\ CapTouch[5],\ x_{end},\ y_{end}]$$

4.2.5. Neural Networks modeling for traits - Process & Output.

Process-wise, for ships to be able to fully brace their senses, the design of the fully-connected Neural Network is as follows:

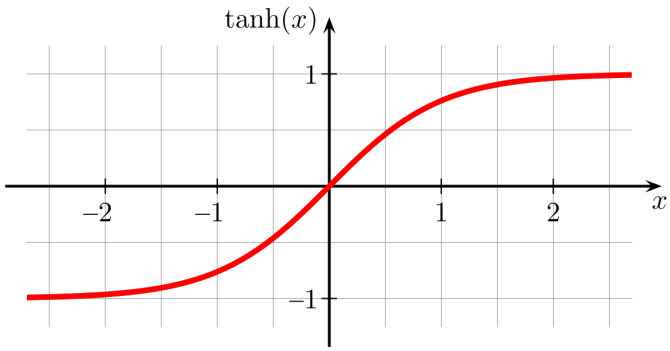


Neural Network's fully-connected used structure.

This $9 \times 6 \times 3$ Neural Network relates the process of propagation for the ships' behaviour while simulating and accounts for a total of 3 layers.

The first "Input Layer" consists of values obtained from the $Input$ array, hence 9 values. The second "Hidden Layer" contains intermediary values for the process issued from the operations, biased sum, and reduction of the first layer. The final layer contains the general output, with its purpose described later.

Other operations occurs between the first and second layers. Let $A_i,\ \forall i \in [1,\ 9]$ the input values from the $Input$ array. By compiling those values, the matrix A can be obtained, which is a 1×9 matrix. Hence, as the obtained values for the second layer are compiled in a 1×6 matrix, the needed weight values are compiled in a 9×6 matrix. Trivially, bias values are compiled in a 1×6 to be added to the matrix. In the end, the hyperbolic tangent is used to normalize the new values in such a span:



Hyperbolic tangent image's span from -1 to 1

Hence, the intermediary values' matrix β can be defined as such:

$$\beta_{1\times 6} = tanh(A_{1\times 9}\ W1_{6\times 9} + B1_{1\times 6})$$

From the second to the third layer, the process stays the same, but the sizes change. The output values' matrix C can be defined as such:

$$C_{1\times 3} = tanh(\beta_{1\times 6}\ W2_{3\times 6} + B2_{1\times 3})$$

The purpose of the matrix C is to serve as the indicatory move for its assigned ship. If the $argmax$ or the highest value of the matrix is the first value, then the ship goes left. If it is the second value, the ship will go forward without changing direction and if it is the third value, the ship will steer right.

In the scheme of simulation, propagation is done at every tick, with each time the updated respective values (coordinate, hit values).

To conclude this part of the Deliverable, notice that this model accurately represents the sole behavior of a ship and how it senses its environment with precise and arbitrary limits, which will be challenged in the next part and the Technical Deliverable as well.

4.2.6. Genetic Algorithm - Process.

Here is the suggested implementation modeled on the Technical Deliverable one. On a random doable map are defined generations of 5 ships initialized with random weights and biases. The fit score is different from the Technical Deliverable, due to technical limitations. Assume the fit score ranges from 0 to 11, with 0 to 10 being the progression of the ship through the map, and 10 to 11 being the "leaderboard" with 11 being the earliest possible on an arbitrary scale. For example, if the ship manages to finish the map in 0 seconds, then its score will be 11. But each tick lowers the "finish fit" to 10 being an infinitely long progression and lower means that the ship did not manage to finish the map.

Hence, at a generation n of ships that have been running for some time, here is a ranking of the ships ordered from the best to worse, using the aforementioned fit score issued from a function, like $f(Ship_1) = b$:

$Ships$	$Fitness$
$Ship_1$	b
$Ship_2$	w
$Ship_3$	x
$Ship_4$	y
$Ship_5$	z

Note that the ship's names are subjective, just to name them according to their ranking.

During the general task, ships' success was measured using a function based on the A* algorithm, as such for the fitness function (s is a ship, s_p is the starting point, the A* algorithm is always done between the parameter and the end cell):

$$f(s) = 1 - \frac{aStar(s)}{aStar(s_p)}$$

aStar(x) returns the number of cells separating the object "x" and the endpoint. The higher the score f(ship) returned, the better the ship performed.

Here, selection occurs, and since $b > w > x > y > z$, then $Ship_1$ is selected as the best ship and can survive and spread its genes.

Crossover happens between $Ship_1$ and the other ships. In our case, $Ship_1$ survives the next generation. But new ships have to be created from the general crossover. Here, let the new ships $Ship_5$, $Ship_6$, $Ship_7$ and $Ship_8$ emerging from the crossover. Their weights and biases are issued from the mean of the weights of the selected ship ($Ship_1$) and their other parent ship.

For example:

$$W_{Ship5} = \frac{W_{Ship1} + W_{Ship2}}{2}$$

$$then$$

$$B_{Ship5} = \frac{B_{Ship1} + B_{Ship2}}{2}$$

Those calculations initialize the weights and biases for $Ship_5$. The other ships have another second parent than $Ship_2$.

At this point, the new expected fit score from $Ship_5$, again as an example is $\frac{b+x}{2}$ as the expected traits from $Ship_1$ and $Ship_2$ have been merged. Although this is not exactly true for most maps, having gained traits from a more successful ship

with the same conditions as the other parent ship means that $Ship_5$ will perform better than $Ship_2$, but not certainly better than $Ship_1$, and so is for every other ship. This "behavioral convergence" occurs from the crossover.

This raises a new interrogation, such as what happens when all ships converge with the best behavior, without finishing the map. In that case, the use of mutation is crucial, as it allows ships to defer from this stocking convergence. The mutation rate defines a certain portion of weights and biases to be modified into randomly generated, yet normalized values from -1 to 1. Hence, a mutation happens to the newly created ships, but not the best.

The purpose of mutation can be inscribed in such a case. If all five ships have the same fit score, then have the same behavior and potentially similar or same weights and biases, resulting from behavioral convergence, then having a 5% mutation rate on 4/5 ships helps them define new behaviors. Again, for the sake of simplicity, this means that their new fit score will be +/- 5%. On a large scale of generations, this breaks the behavioral convergence by creating new best ships that are theoretically 5% better at some generation.

Following this principle, the mutation is applied to every newly created ship, hence $Ship_5$, $Ship_6$, $Ship_7$, and $Ship_8$, with $Ship_1$ unchanged from the beginning.

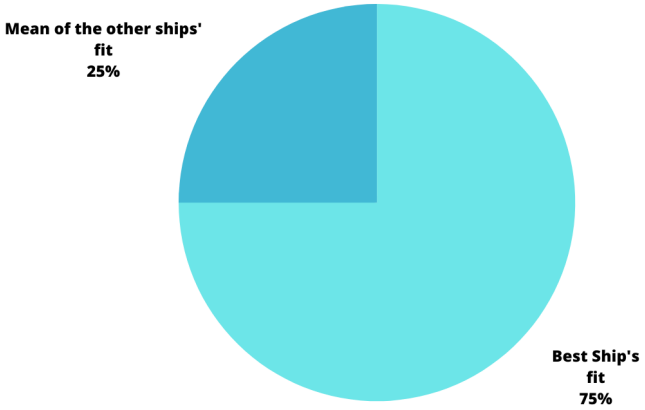
The new generation $n+1$ is then composed of $Ship_1$, $Ship_5$, $Ship_6$ $Ship_7$ and $Ship_8$, with ships that have a different yet more accurate behavior than generation n, thanks to crossover and mutation.

4.2.7. Efficiency.

As such, Genetic Algorithms are generally reputed to have a slow start solving tasks. However, the process of behavioral convergence allows ships to quickly come to the level of the best one. Then, thanks to mutations, ships can still progress despite behavioral convergence, thus according to the mutation rate. This model can be replicated as much as wanted, for as many constraints as desired, making Genetic Algorithms adaptable to any change of conditions and constraints.

In this model, thankfully, the study of efficiency correlates with the general fitness score of a ship. However, if the efficiency score was only reflecting the best ship's score, then all the mutable potential of the other ship is lost. As such, the study of efficiency needs to account for the score of other ships. Again, this precision allows the reflection of behavioral convergence of other ships, as the efficiency score needs to be higher when other ships are completing the task as well.

The study of efficiency can be summed up in the following pie chart:



Pie chart for efficiency

Hence, the mathematical function for efficiency can be as such (S is a list of multiple ships and b is the best ship):

$$eff(S) = 0.75 \cdot f(b) + \sum_{i=0}^{len(S)} \frac{f(S_i)}{4 \cdot (len(S) - 1)}$$

This efficiency function is expected to mimic the general behavior of the best ship when finishing a task "alone" at around 75%, or all ships dying at 75% completion.

4.2.8. Map complexity.

This subsection's purpose is to establish a complexity order for maps. As maps have different layouts, the complexity is also variable. Here is a function that determines the complexity index of a map. The higher it is, the harder the map is (s_p is for the starting point, w_p is/are for the points where the aStar score is the higher than s_p and are the highest of their path, d if for the object density). For a map m:

$$O(m) = (aStar(s_p) + \sum aStar(w_p)) \cdot d$$

With aStar(start) being the aStar score from the start of the map, aStar(worst) is the highest possible score, hence from the the "worst point" to arrive on and $d = \frac{\#objects}{100}$.

For some appreciation of some benchmarks for the designed standard map, please refer to the Appendix section.

4.3. Assessment

4.3.1. Theoretical Analysis.

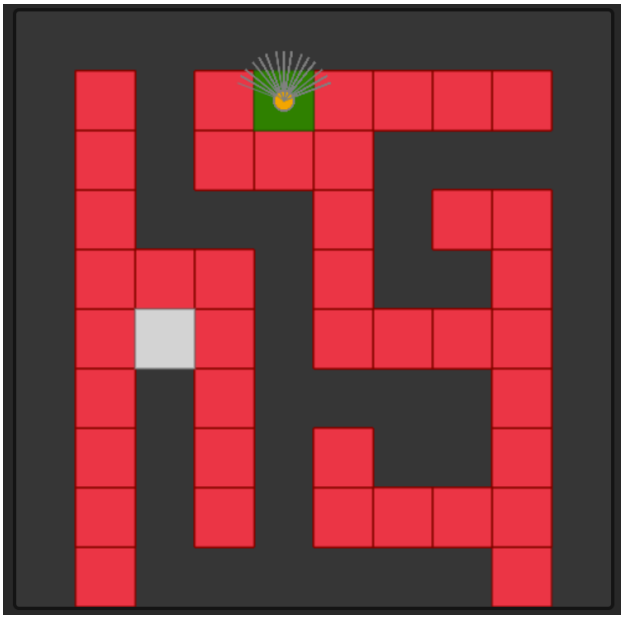
To assess the Scientific Deliverable, using the general efficiency value is relevant. In this case, some data from the Technical Delivery itself will be taken, as assessments for both Scientific Deliverables and Technical Deliverables are similar

due to studying the structure of the Genetic Algorithm itself and its development.

Naively, using this created model, leading to one best ship being preserved and favoring behavioral convergence at first, then favoring mutation to encourage small yet persistently increasing results in the general fit, hence efficiency score happened to be verifiable in theory. Using a general mathematical demonstration on a pseudo-induction case leads to confirming the feasibility of this model to a scaled-up Technical Delivery.

4.3.2. Analysis on efficiency values.

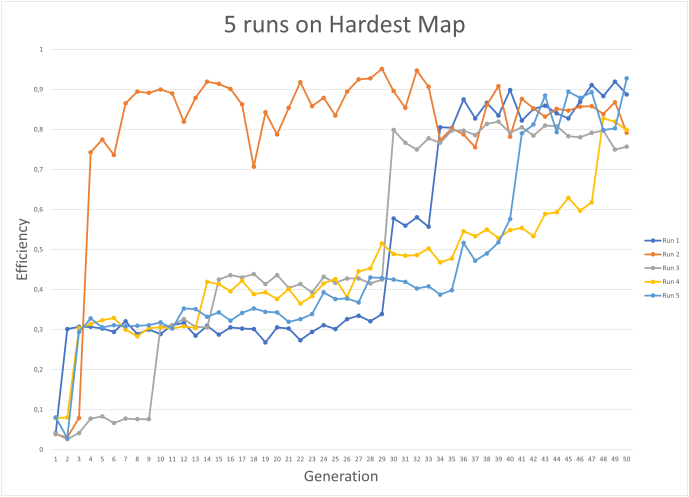
Assume, in this case, the following initialization. Each generation is constituted of 10 ships, that were initialized with random weights and biases. During the following runs, the following map was used:



”hard_map” layout from Path-IT’s GUI with starting point (green), ending point (white), and obstacles (red).

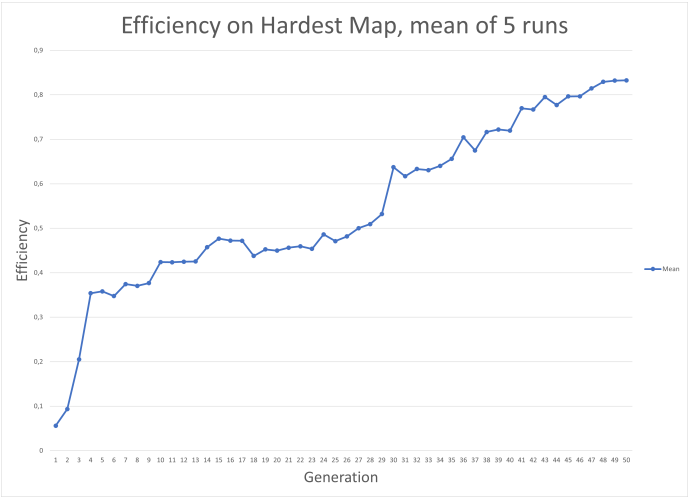
This map, denoted as ”hard_map”, has been defined as the hardest map yet. Although the complexity order was naive, it is still an appropriate indicator of a map’s difficulty. The complexity order of this map being equal to 44, and being the highest among ”standard” maps shows that ”hard_map” is the hardest of the batch.

On this map, using the described Genetic Algorithm’s model, 5 runs were executed, each consisting of exactly 50 generations. In each generation, the efficiency is calculated and stored in an array to be retrieved. Here is the result of the efficiency score for those 5 runs:



Plot of the 5 runs on the hardest map, efficiency on each generation.

On this graph, some recurrent patterns can occur, like some sudden jumps in efficiency from 10% to around 40%, or from 40% to 80% before plateauing between 80% and 90% in the long term. This can be explained by the structure of the map, having multiple traps and sudden narrow turns before running into some long hallway and encountering another similar situation with multiple possible paths to take. Those inconsistencies related to the generation’s potential or situation in behavioral convergence are less visible when comparing the mean of all those values:



Plot of the mean of the efficiency of the 5 runs, every generation.

Looking at the mean of those runs, only one straight high jump can be observed at the beginning, which is common to most runs. However, steady growth can be seen afterward, seemingly linear where all runs meet a high point between 80% and 90%. Here, the final efficiency value is 83.25% and the task has been finished by more than one ship in all runs. This practical analysis confirms that this model is viable for

such tasks, although it needs more adjustments.

4.3.3. Flaws and ways of amelioration.

Although this model proved to be efficient on the hardest standard assets of this project, here are some observed inconsistencies or ways of upgrading the model:

- Selection and Fitness function: Although selection bases itself on the A* algorithm, it is still incomplete regarding long runs and improvement when finishing the map. When a ship finishes the map, it is deemed to be the best ship for the rest of the run as it is the first ship to be analyzed among them. However, the fitness function does not account for the time made. This means that for two ships finishing the map at different times, the first ship analyzed will have the position of best, which is the best ship so far, not necessarily the fastest. Accounting for time can be made to further increase efficiency.
- Mutation can also be based on time for the same reasons, as it is based on selection and best fitness score.
- Efficiency can also account for time.

Hence, the general model is correct if analyzing the completion of the task itself, not the time it took to do it. However, this does not change the conclusion on the model itself being efficient at managing ships going to an endpoint, but arriving at a plateau when a ship finishes rather than continuing to improve time, though time was not a determinant variable in this Scientific Deliverable, its consideration could lead to further improvements and evolution of the model.

4.3.4. Verdict.

Despite some possible adjustments seen beforehand, this Scientific Deliverable can be considered a success. The production met all defined requirements while abiding by the design limits. To answer the question, due to the advantages of mutation, it is possible to nullify most of the problems of behavioral convergence like the glass ceiling that can be met, all the while saving all the advantages like converging to the best ship and meeting a standard point, then mutating. Those traits for this Genetic Algorithm can be replicated on other tasks as well. Thus, Genetic Algorithm is efficient to perform such tasks thanks to a rapid conversion in results, even for some of the hardest assets.

5. Path-IT - An app to find an optimal path

5.1. Requirements

Path-IT is a web-based application rendering a game, where Genetic Algorithms can be performed onto. With JavaScript as its main coding language, using doctrines such as Object Oriented Programming, this Deliverable will come in multiple parts for rendering and back-end calculations, coming within a defined structure. This Technical Deliverable's requirements can be defined into several subsections:

5.1.1. Create a web-based plain game.

Before performing Genetic Algorithms, a support has to be made, on which simulations will take place. This game accounts for variable constraints such as controls, obstacles, position, and so on. The structure of the game shall be as optimal as possible to allow the rapid development of maps. For example, maps can be stored in a database in a defined structure, allowing the serial creation of non-hard-coded maps. This game uses Canvas as rendering and will have a front-end process in JavaScript

5.1.2. Develop ships and properties.

Although developing ships is related to the past section, the goal of this requirement is to conciliate both the front-end and general back-end. By developing a ship as an object, the general process can be easy to make in the chain. Here, the principles of OOP will play an important, especially between the ships and their captors, but also their weights, biases, and other characteristics such as start position, which are related to maps but also intrinsic to the ships themselves and the back-end process.

5.1.3. Code the Machine Learning section.

Using the variables described before, the propagation part must be integrated into the ship's inner object. Again, the process is composed of multiple versatile functions that have to be made from the ground up and compounded into larger, more specific, and targeted functions to serve Machine Learning's purpose. Hence again, a library of such functions has to be made, and many precisions in the object parts of the ship to compute propagation, and in the main process for the Genetic Algorithm's part. At this point, a user can come and train ships on their map if they can make them up themselves and integrate them into the code. They can also rapidly modify the hard-coded mutation rate for example.

5.1.4. Add some user experience to the application.

As a final touch on the application, being sure of a certain process if it is made while watching the internal components, adding some UX (user experience) may be the glazing on top for any unexperimented and uninformed user to improve general usability and aesthetics. For example, the page may display information on the generation, some leaderboard, and/or a way to easily input some maps and some ongoing generations, and save states of it to use later or analyze its ships.

5.2. Design

The main Design choices are listed below:

5.2.1. Object Oriented Programming.

Path-IT has been developed using Object Oriented Programming for ships and captors. As such, those objects are represented as objects. For example, every method related to ships, like propagation and drawing is directly assigned to ships themselves as inherent methods. The same is done for captors with their hit value, and also spawning them via the creation of ships.

5.2.2. Special Modules and Convention.

Here are the special modules and conventions used in the project:

- Canvas, being an integrated graphical render module in HTML. Canvas has been used to render the main game on the page and acts as a drawing on a 2D scale. Canvas is used to initialize the map, spawn ships and adapt their position.
- JSON, being JavaScript Object Notation. JSON is mainly used to store data in separate files that can be accessed through functions and consist of its conventions for arrays. JSON stores maps that can be retrieved according to their indexes in the main code.
- Math is the default package in JavaScript that allows one to compute more advanced mathematical functions and formulas. Math is especially used for functions like $tanh()$, $sqrt()$ and $log()$.

5.2.3. Conventions for computations.

Throughout computations, limits and conventions were made to standardize and serialize the process. Especially, during propagation and crossover. Ships possess their alleles, or genes as weights and biases, stored as attributes in their classes. Those weights and biases are two-dimensional arrays that are the representation of matrices. Moreover, maps are also two-dimensional arrays, made up of different integers directing the general constraints (0 means nothing, the ship is free to pass through, 1 is the spawn point, 2 are obstacles, and 3 is the endpoint).

5.2.4. Layout for Back-end functions.

Back-end functions are mainly stored in doc.js. This file acts as a repository for usual functions that are expected to be used throughout multiple files, such as mean, randomization, addition, and multiplication of matrices.

5.2.5. Respective functional decomposition.

- ship.js: contains the main attribute and methods for ships, such as propagation, initialization, and drawing. It also contains the main constructor functions for its assigned captors and respective hit values.
- sensor.js: contains the hit value of the sensor, calculated from the determination of overlaying of its coordinates with an obstacle's coordinates. Those obstacles coordinates are passed down by the ship when spawning the captor along the way but are never used by the ship.
- doc.js: contains the main back-end functions used in most of the files.
- main.js: contains functions relative to the process of Genetic Algorithm, such as select, crossover, and mutate. main.js' principal execution consists of parsing the map in mapsDB.json from the given index and initializing it by drawing its layers of the spawn point, obstacles, and endpoint, and registering those coordinates. Next, for every tick, the update function checks the collision between the ships and the obstacles or the endpoint and determines the fitness score for each ship. If all ship dies or a time threshold is met, then selection, crossover, and mutate are executed, and a new generation of ships spawns.

5.3. Production

This Production section will focus on multiple code insights and development outputs regarding Path-IT. More particularly, those insights will be restricted to the main functions related to the global process of Genetic Algorithms. Here is a short selection of deeper functional descriptions of functions that will not be developed afterward.

5.3.1. Non-Exhaustive Functional Description.

- *main.setup()* uses the canvas displayed on the website and the retrieved map in mapsDB.json to draw the general layout and the starting point, obstacles, and ending point. It also registers the coordinates of obstacles into an array passed into the ship's constructor for their captors.
- *main.initNN()* calls ships.initiate() which randomizes weights and biases, used when initializing a new instance of the webpage.
- *main.update()* is the main function of main.js. It is paired with a setInterval to be called every tick. At that tick, using the initialized array of obstacles from earlier, it iterates over each ship and verifies if their position is not coinciding with the obstacles. If the ships check the conditions for dying (either collision or running out of time from a limit of ticks to not exceed per iteration), then their fitness score is updated to the correct value. main.update() also serves as updating the canvas altogether by clearing the canvas and redrawing each sprite of ships and objects at their updated coordinates.
- *main.updateValues()* and its composite functions update the value on the website's GUI.
- *main.main()* process initializes the simulation's general structure by fetching the maps info, running the initiating functions described before, creating ships with all the retrieved information, and eventually creating the setInterval for update() when pressed on the UP key.
- *doc.tanh* takes an array as a parameter and returns a new array with every entry passed through the tanh() function.

- *doc.matMult()* operates matrix multiplication between two compatible matrices (hence arrays). For every entry of the new matrix, the result from the old matrices is calculated with the general formula: $newMat_{i,j} = Mat1_{i,k} * Mat2_{k,j}$ thanks to the overall iterations that browse the matrices.
- *doc.matAdd()* operates matrix addition between two compatible matrices, the operation is straightforward with the new entries being the sum of the corresponding two entries from their respective matrices.
- *doc.argmax()* returns the index of the highest value of an array.
- *doc.meanW() and doc.meanB()* will return a new matrix, with its entries being the mean of the corresponding entries from the respective matrices.
- *doc.aStar()* takes a map and an index as parameter. Since the map is a square array ($n \cdot n$ array), then the A* algorithm is done from the index to the ending point (depicted as a 3 on a map) and will return the smallest number of cells separating the index to the ending point.

Those functions are the main components for the global website running process, thus game and simulation. But some functions are directly depicted from the model defined previously in the Scientific Deliverable, as they are worth proper highlights.

5.3.2. Ship's propagation.

As ships read the general environment from their captors, they need to read their weights and biases, interpret their surroundings, and act. From the propagation section of the Scientific Deliverable, the general structure can be largely elucidated and thus developed.

```
1  propagate(xEnd, yEnd) {
2
3          let W1 = this.weights[0];
4          let W2 = this.weights[1];
5          let B1 = this.biases[0];
6          let B2 = this.biases[1];
7
8          let input = [[].concat(this.x / 240, this.y
               / 240, this.capTouch, xEnd / 480, yEnd /
                480)];
9          let weigthedA = doc.matMult(input, W1);
10         let finalA = doc.matAdd(weigthedA, B1);
11         let newInput = [doc.tanh(finalA)];
12         let weigthedB = doc.matMult(newInput, W2);
13         let finalB = doc.matAdd(weigthedB, B2);
14         let final = doc.tanh(finalB);
15         let decision = doc.argmax(final);
16         return decision;
17     }
```

Listing 1: propagate function

Hence from this function, the semantics are almost straightforward. On line 8, the general input array is constructed, being composed of the ships' coordinates, the ending point's coordinates, and the hit values returned from the captors. Next, each line from line 9 can be interpreted as the propagation method used. Hence, from lines 9 to 11, propagation between the first and second layers happens, and from lines 12 to 14, propagation from the second to the third and final layers is done. On line 15, *doc.argmax()* is used to get the index of the biggest element found, which equates to the decision made by the ship. 0 implies going left, 1 going straightforward and 2 going right. Again, the structure of the *final* is of length 3 and corresponds to the weighted sum and normalization.

5.3.3. Initialization of Generations.

```
ships = [p1, p2, p3, p4, p5, p6, p7, p8, p9, p10];   1
```

Listing 2: Initialization of Gen.

Here is an example of an initialized array of creatures, hence ships. Assume, after the task has been done, that every ship has at least a fitness score defined.

5.3.4. Selection.

At this point, from the Scientific Deliverable, selection can be applied as such:

```
function select(ships) {                                  1
    var best fit = -1;                                    2
    var bestFitIdx = -1;                                  3
    for (let i = 0; i < ships.length; i++) {              4
        if (ships[i].fit > bestFit) {                     5
            bestFit = ships[i].fit;                       6
            bestFitIdx = i;                               7
        }                                                 8
    }                                                     9
    console.log(ships[bestFitIdx].id + " has been         10
        selected");
    selectedFit = [ships[bestFitIdx].fit];                11
    ships[bestFitIdx].id = "currentBest";                 12
    return bestFitIdx;                                    13
}                                                         14
```

Listing 3: select function

select() is a forthright way of getting both the best ship and the best fitness score. It will iterate through the array and save the ship and the best fitness score if it has been the highest ever seen in the array. In the end, a console.log allows for eventual debugging, and some assignment to other values retrieved in other functions is done. All the while, the index of the best ship is returned and the corresponding ship is tagged to the "currentBest" ship.

5.3.5. Crossover.

Having now the knowledge about general fitness scores, and having determined the best ship among others, the crossover can be applied. Hence, a new generation of ships will be created in the image of the best ship and every other while preserving the best ship.

```
1  function crossover(best, ships, xSpawn, ySpawn,
       angle, xObs, yObs) {
2      let newShips = [];
```

```
3     for (let i = 0; i < ships.length; i++) {
4         if (best != ships[i]) {
5             let newShip = new play.Ship(xSpawn,
                  ySpawn, angle, [], [], xObs, yObs, "
                  p" + (i).toString());
6             newShip.weights = doc.meanW(best.weights
                  , ships[i].weights);
7             newShip.biases = doc.meanB(best.biases,
                  ships[i].biases);
8             newShips = [].concat(newShips, newShip);
9         }
10    }
11    best = new play.Ship(xSpawn, ySpawn, angle, best
          .weights, best.biases, xObs, yObs, "
          currentBest");
12    newShips = [].concat(best, newShips);
13    return newShips;}
```

Listing 4: crossover function

From line 2 to line 10, the general process is to create a new array that will foster the new generation of ships, then iterate over the existing one and apply functions between the iterated ship and the best ship. Here, doc.meanW takes the weights of the best ships and the iterated ship and creates a new set of weights which is the mean of the best's and iterated weights. The process is the same for meanB, this time with biases. Those new parameters allow the creation of new ships that can be put in an array. From lines 11 to 12, the preservation of the best ship dictates that the best ship should also be part of the new generation, and is also pushed in the array.

Note that the ship's constructor needs multiple variables, such as xSpawn, ySpawn, the starting angle, and the passing obstacles for their captors, hence explaining the high number of parameters, heavy yet essential for this function called once per generation: for spawning the new one.

5.3.6. Mutation.

The general final step of the Genetic Algorithm generation's birth is mutation.

```
1  function mutation(ships) {
2      let mutationRate = (1 - selectedFit) / 2.5;
3      if (mutationRate < 0.03) {
4          mutationRate += 0.02;
5      }
6      for (let i = 0; i < ships.length; i++) {
7          for (let j = 0; j < ships[i].weights.length;
               j++) {
8              for (let k = 0; k < ships[i].weights[j].
                   length; k++) {
9                  for (let l = 0; l < ships[i].weights
                       [j][k].length; l++) {
10                     if (Math.random() <=
                           mutationRate && !ships[i].
                           best) {
11                         ships[i].weights[j][k][l] =
                               (-1) ** doc.getRandomInt
                               (2) * Math.random();
12                     }
13                 }
14             }
15         }
16         for (let j = 0; j < ships[i].biases.length;
               j++) {
17             for (let k = 0; k < ships[i].biases[j].
                   length; k++) {
18                 if (Math.random() <= mutationRate) {
19                     ships[i].biases[j][k] = (-1) **
                           doc.getRandomInt(2) * Math.
                           random();
20                 }
21             }
22         }
23     }
24     return mutationRate
25 }
```

Listing 5: mutation function

In the first lines of the function, a definition of the mutationRate is done, with it being relative to the selectedFit, which is the best fit, issued from the select() function. If the mutationRate happens to be low, a plateau is made at mutationRate = 0.02 being the lowest possible value, to avoid behavioral convergence that would not be beneficial to the simulation as shown in the Scientific Deliverable. Next, huge loops are made (from lines 6 to 15 and lines 16 to 23) for the mutation of the weights and biases. Since the dimensions of weights are different from the biased ones, the structure is different, but the process is the same. For each entry in either weights or biases, a random number is generated. If this randomly generated number is lower than the mutationRate, then the entry is modified to a random number. Finally, the mutationRate is returned to give information to the user on the website and that can be retrieved for any derived usage.

5.4. Assessment

This Technical Deliverable met all the requirements, and its outcome and results impacted the Scientific Deliverable with impartial and accurate values resulting from the model itself it ought to imitate. Moreover, the code is general enough to change most of the conditions of the game, such as the number of ships per iteration (generation), or the maps themselves using the onboard GUI on the website. As such, not only does Path-IT satisfies the generic use concerning the main question of this BSP and the Scientific Deliverable, but its versatility allows for more cases on different maps and conditions, and its intrinsic model can be replicated in other types of simulation in the scheme of Genetic Algorithm. Hence, the Technical Deliverable is a success as well.

6. Acknowledgment

The author would like to thank Pr. Dr. Leiva for his insights and tutoring regarding the project and the studied topics.

7. Conclusion

This section concludes this Bachelor Semester Project, which had as objectives the elaboration of a Genetic Algorithm model regarding Path-finding with its assessment on efficiency, and the creation of software, comprised of a

web application provided with Machine Learning capabilities allowing the creation of a Genetic Algorithm program. The model developed in the Scientific Deliverable is a success, as for this use case, although Genetic Algorithms may have a slow start solving tasks, having a trained model is not that long as Genetic Algorithm converges quickly to such solutions. Following this assessment, the software developed in the Technical Deliverable is also a success as it allows users to test and train the model on different distinct maps with varying levels of difficulty and thus constraints, knowing that the overall rendering is flawless in that case with no related detected bug or misconception.

8. Plagiarism statement

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] Sofer. (2000). An Introduction to Genetic Algorithms . Melanie Mitchell. The Quarterly Review of Biology. Available on **doi.org/10.1086/393289** *Deeper explanations regarding Genetic Algorithms' process, general use and efficiency.*

[2] Ding, S., Su, C. & Yu, J. An optimizing BP neural network algorithm based on genetic algorithm. Artif Intell Rev 36, 153–162 (2011). Available on **doi.org/10.1007/s10462-011-9208-z** *Example of Genetic Algorithm's use regarding Neural Networks.*

[3] Mitchell, M. (1995), Genetic algorithms: An overview. Complexity, 1: 31-39. Available on **doi.org/10.1002/cplx.6130010108** *Synthetic introductive report on Genetic Algorithm's purpose.*

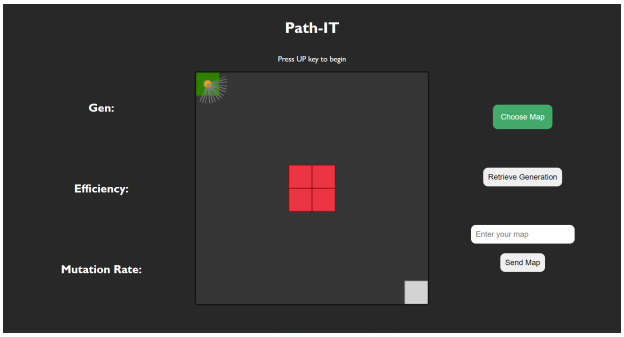
[4] Genetic Algorithms: Principles of Natural Selection Applied to Computation, Forrest, Stephanie, 1993/08/13, American Association for the Advancement of Science. Available on **doi.org/10.1126/science.8346439** *Precisions on concepts of "selection" related to real natural selection, with mutation analysis.*

9. Appendix

9.1. Path-IT Software

Path-IT is an open-access website that can be reached from **this link**. Here are two screenshots from Path-IT:

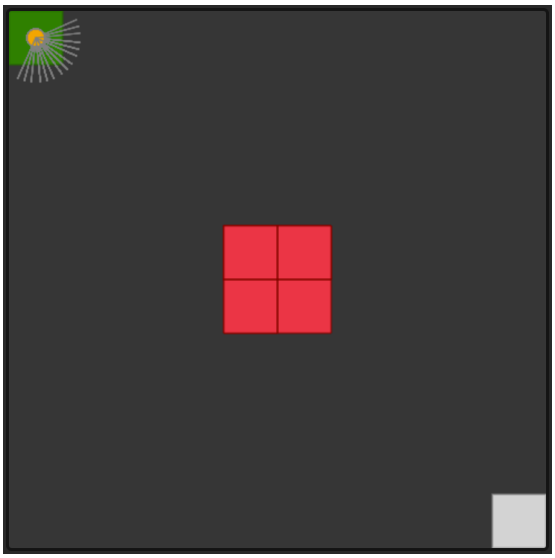
Path-IT: no map selected



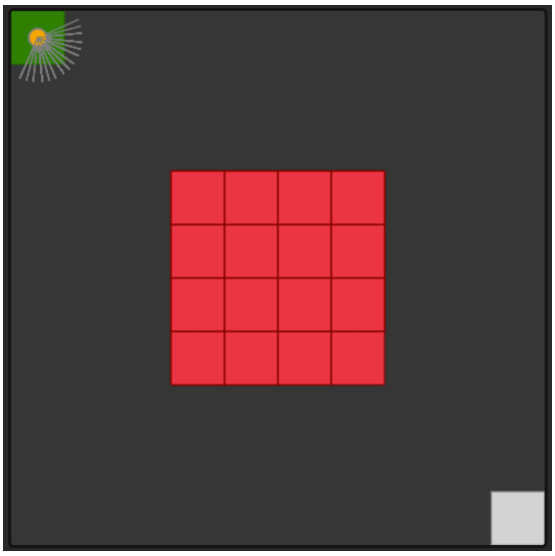
Path-IT: debug1 selected

9.2. Map examples and Complexity

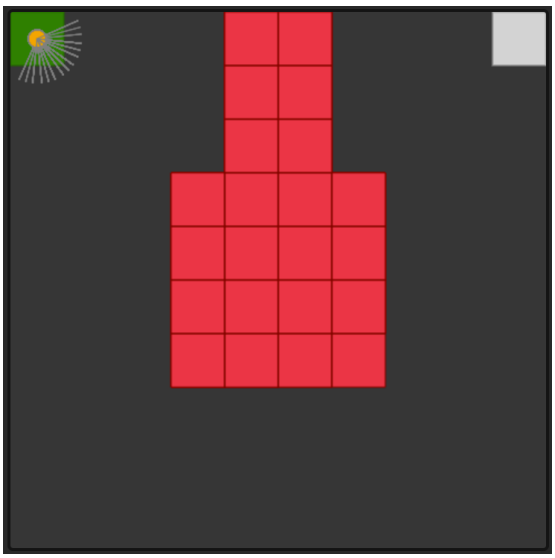
As described earlier, a batch of map was developed for Path-IT. Despite limitations, it is possible to create "hard maps", refer to the Scientific Deliverable for the complexity determination function, finding a map's difficulty index. Again, the higher the index is, the harder the map is. The scale is naive.



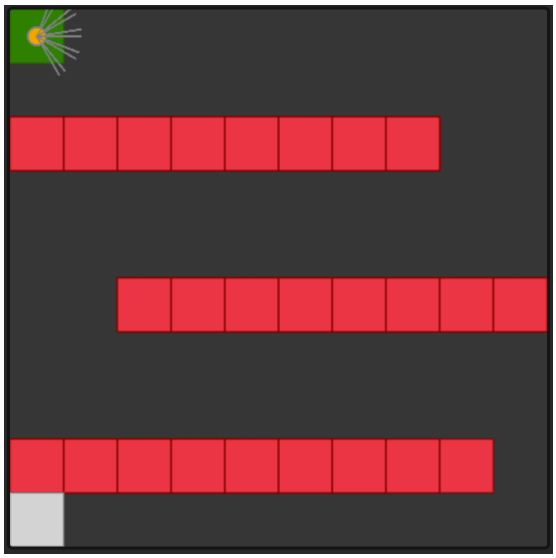
debug1 map: Complexity = 1.44



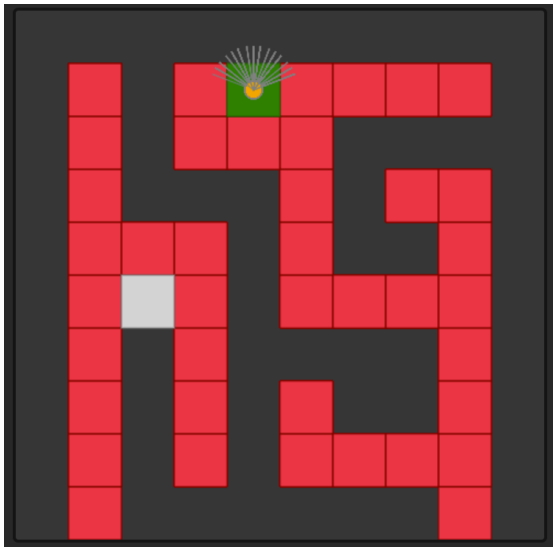
debug2 map: Complexity = 5.76



roundabout map: Complexity = 10.12



zigzag map: Complexity = 20.5



hard map: Complexity = $(20 + 32 + 31 + 27) \cdot 0.4 = 44$