Paradise: Designing Perfect Worlds With Genetic Algorithms

Benjamin Cook

Dept. of Computer Science, Faculty of Science, University of Calgary,
2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4
benjamin.cook@ucalgary.ca

April 2020

1 Abstract

The following is an in-depth analysis of a term project for the University of Calgary Course CPSC 565: Emergent Computing. The goal of this project was to turn the traditional understanding of evolution (dynamically changing creatures adapting to a static world) on it's head. Given a static (unchanging) simulated "animal", the goal was to use a genetic algorithm to design the ideal world for these creatures to thrive on. The paper will discuss implementation details, the techniques used to design the simulation, and the findings of running said simulation on various different creatures.

2 Introduction

Traditional evolution in the real world is characterized by organisms adapting to best suit their environment over the course of many generations. Giraffes with slightly longer necks have access to higher food sources, and are therefore more likely to survive, breed and pass these traits off to their offspring. These offspring follow this same cycle and eventually are able to develop and exploit these traits in order to best survive in their environment.

Although the Earth as an environment has gone through many global changes throughout its history, generally speaking this "environment" is relatively static when compared to the evolution of these organisms. In particular, the environment that these organisms live in does not change as a result of evolutionary processes. The temperature of the planet does not increase or decrease in response to wolves developing thicker coats of fur for example. For simplicity's sake, we will define "traditional evolution" as dynamically adapting agents evolving to

fit statically defined environments.

The goal of the Paradise project was to reverse the roles of the organism and environment as defined in traditional evolution. Instead of the organism evolving to survive best in the static environment, the goal is to create a static creature, and have the environment evolve in a way that allows the organism to thrive. The process that we are describing could be thought of as "Terraforming" a world to best suit the needs of a particular organism. Although this term is usually used in a science fiction context, there are already many papers discussing the feasibility of Terraforming Mars to better meet the needs of human life [1].

The following paper will discuss the goal of using a genetic algorithm to design a perfectly Terraformed world for a given creature. It will then cover the methodology of building and testing the simulation, followed by an analysis of the findings. The paper will also call for further experimentation using more complex creatures, and worlds.

3 Project Goal

The primary goal of this project then, is to observe the effectiveness of a genetic algorithm as a method to design the "perfect world" for a static organism. The environment, as well as the organisms and their behaviors will be simplified down as much as possible in order to test this concept. I created a "population" of these simplified worlds, and simulated them for a discrete amount of time. Once this time is up, I then judge each world's "fitness" based on an evaluation function, and select the best worlds. These worlds were then "combined" and "mutated" in order to generate a new population of worlds to be simulated. After a given number of generations, the algorithm should produce worlds that enable the organism to thrive much better than the initially randomly generated ones.

4 Prior Work

After searching several databases for related work I was not able to find a project with the exact goal of developing a "perfect world". There are many papers and projects with the goal of simulating traditional evolution (see [7] for example). These projects use a similar methodology, while achieving a different goal; adapting a creature to an environment.

The most similar work/existing systems are artificial intelligence that are designed to change the environment based on an agents input/state. Systems in video games, such as "Left 4 Dead"'s AI director [6] and "Rimworld"s "AI Storytellers" [5] manipulate the environment of the player based on their state.

If the player's health is low, then the AI may place a health pack in the next room. Whereas if the player's health is high and his/her shooting accuracy is high, the AI may decide to put a difficult enemy in the room instead.

The final related work I've found that focuses on using artificial intelligence to design environments was in the paper "Digital-Techniques-Applied-to-Design-Processes" [4]. Chapter two of this paper discusses the success in using genetic algorithms to develop layouts for structures which is very similar to the environmental development goal of my project.

5 Methods

The Single World simulation was implemented in Netlogo[2] which allowed for easy creation and viewing of simple simulated 2D agents and worlds. Furthermore, the LevelSpace Extension[3] allowed me to run multiple single world simulations, and have them report information to a master simulation containing the genetic algorithm

5.1 Single World Simulation Details and Implementation

The following section highlights how a single world in simulated in Netlogo.

5.1.1 Organisms

The organisms are represented using Netlogo's turtle system. Each world will start with an initial number of turtles. These turtles have the following properties:

- Energy: This is decreased each time the turtle moves, and is increased/decreased any time the turtle runs over a non-black patch based on the set parameters as discussed in 5.1.3.
- Randomize movement: Each tick of the simulation, the turtle will pick a random direction and move forward once.
- Death Mechanics: If the turtle's energy falls below 0, then the turtle will "die" (removed from the simulation).
- Reproduction Mechanics: If the turtle gains energy greater than or equal to a certain number, then it will spawn a new turtle, and have some of it's energy reduced.

5.1.2 Worlds

The worlds are represented using Netlogo's Patch system. Each world is a 17x17 grid of Squares. Each Square can be either Yellow, Red or Green which represents different food sources. Each of these different food sources is removed

temporarily when an organism steps on the patch. When this happens, the organism will gain or lose energy depending on the parameters set and discussed in 5.1.3. After a certain duration, the patch will regenerate with the same colour as before. Time passed on the worlds will be represented using Ticks in Netlogo.

5.1.3 Parameters and Reporters

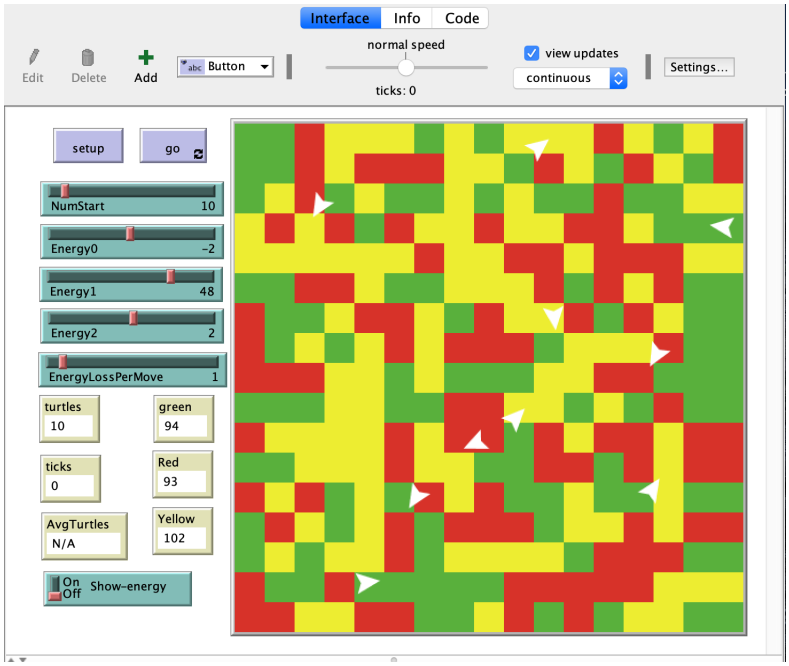
The single world simulation has the following Parameters that can be modified:

- NumStart: The number of organisms that are initially placed into the world upon it's creation.
- Energy0: The amount of energy that an organism gains (or loses) when running over green patches
- Energy1: The amount of energy that an organism gains (or loses) when running over red patches
- Energy2: The amount of energy that an organism gains (or loses) when running over yellow patches
- EnergyLossPerMove: The amount of energy that an organism loses each time it moves.

The single world simulation also has the following reporters and monitors:

- Turtles: Indicates the number of turtles on screen at a given time
- Green: Indicates the number of green patches on the screen at a given time
- Red: Indicates the number of red patches on the screen at a given time
- Yellow: Indicates the number of yellow patches on the screen at a given time
- Ticks: Indicates the amount of time (in discrete ticks) that have passed since setting up the world.
- AvgTurtles: Indicates the average "turtles-per-tick" of a world. This is calculated by:
  $\frac{\sum_{n=1}^{EndTime} Turtles_n}{n}$
  where n is the time in ticks, $Turtles_n$ is the number of turtles at time n, and EndTime is number of ticks before the simulation stops. As we will discuss in section 5.2.2, this is the best way to evaluate the "fitness" of a given world.

5.1.4 Single World Simulation Example



5.2 Master Simulation: Genetic Algorithm Details

The master simulation contains the Genetic algorithm, as well as a graphical representation of the fitness functions of each world as they run. The master simulation runs each generation of worlds for 300 ticks, and then asks the worlds to return their evaluation function, and uses the genetic algorithm to create the next generation.

Each world is represented by a list of size 288. where the 0th element represents the top left corner, and the last element represents the bottom right, (counting up by 1 as we go left to right).

5.2.1 Population, Number of Generations and Initialization

The population (number) of worlds remains constant for each generation and stays constant for the duration of the simulation. The number of generations to run is determined by the GenerationSize slider. The number of generations to run is determined by the numGenerations Slider.

When the simulation is initialized, the master simulation generates an initial random population of worlds and begins simulating them.

5.2.2 Evaluation Function and Selection

Throughout the testing process I attempted to use many different evaluation functions. Many of which were unsuccessful.

- Number of Organisms: The most obvious evaluation function of how fit a world is would simply be to count the number of organisms in each world. The problem with this approach however is that the number of turtles in each world fluctuates wildly throughout the duration of each simulation. Even the worst suited world has moments where there are more turtles than the best. Since the time of each simulation stops arbitrarily at 300 ticks, it means that there is a high probability that the best suited worlds have less turtles than others, and will therefore not be evaluated properly due to the random nature of their cycles of population growth and decline.
- Sum Of Energy: Another way I attempted to evaluate each world is through finding the total energy of the all the turtles remaining after 300 ticks. This worked better than the number of organisms method as the amount of energy seemed to fluctuate less. However it suffered from the same problem of being too susceptible to randomness.

As stated above in 5.1.3. The final and most effective method for evaluating how well a world is suited was to take the average number of turtles per tick. This allowed the evaluation function to more consistently and accurately score the better worlds higher. It also eliminates the issue of ending the simulation at an arbitrary time of 300 ticks, since it takes into account the fitness of the entire duration of the simulation, rather than just the final moment.

The genetic algorithm portion takes these evaluation functions, and sorts them in descending order to select them using a roulette based selection. I initially started by just taking the top two worlds without any randomness, however this often resulted in highly homogeneous gene pools which drastically slowed the exploration rates. This dramatically increased the number of generations required to arrive at good solutions. As a result, a weight-based probabilistic selection method was much more effective in exploring more world configurations quickly, while still choosing the better worlds much more often. The degree of randomness in the selection is dependant on the roulette factor slider. The higher the roulette factor, the more randomness is introduced into the selection. The selection function chooses two parents who will be used in the crossover function to generate a new population.

5.2.3 Crossover

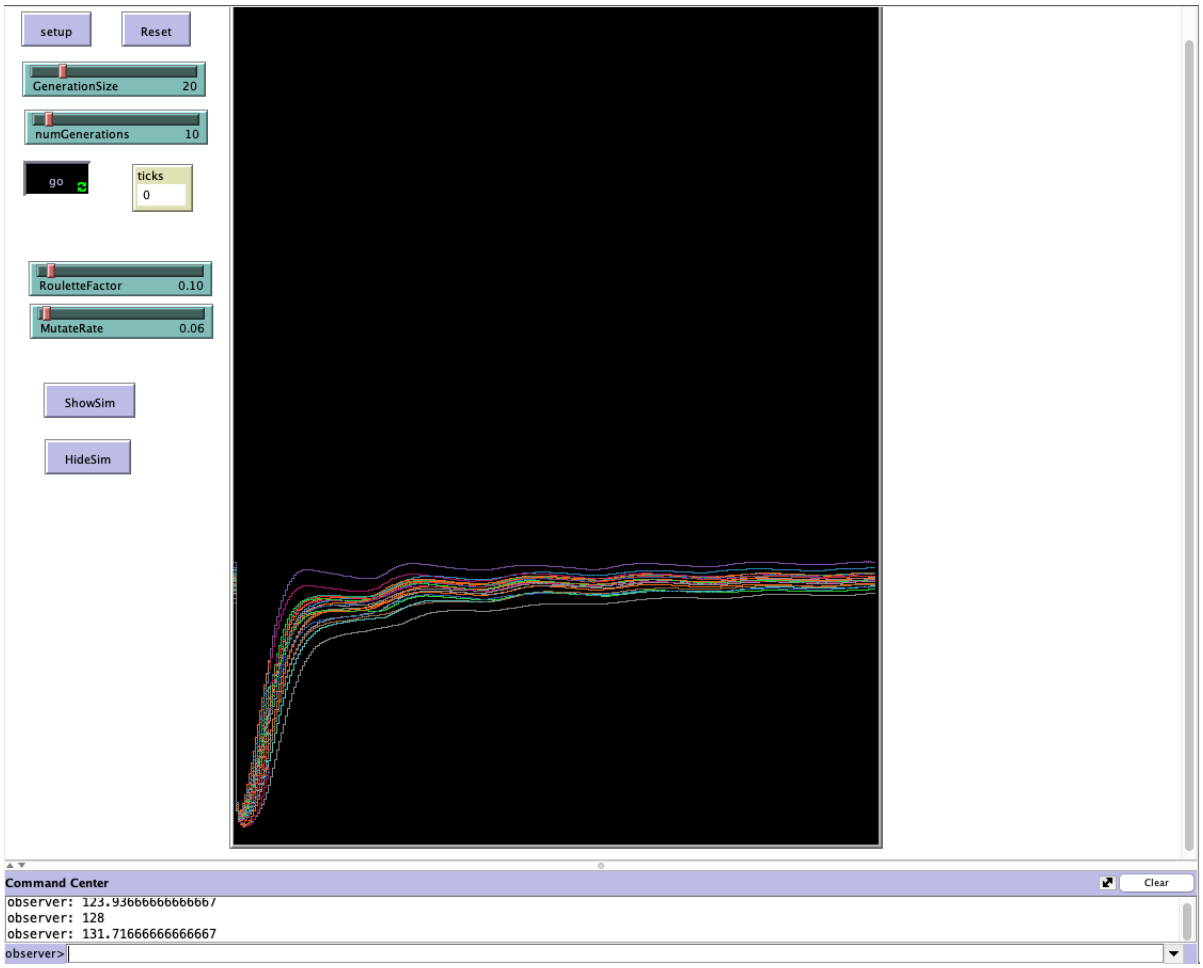
The crossover function takes two random indices of the list i_1 and i_2. "Genes" (single patch of the board) are taken from the first parent from index 0 to i_1, and from i_2 to the end. The genes between these two indexes are taken from the second parent in order to construct a "child" world. This process is repeated

until the generation has been completely repopulated. The parents are then discarded (does not use elitism to preserve parents between generations).

5.2.4 Mutate

Each gene of each child has a chance to mutate after being created from the crossover function. This is determined by the mutation rate Slider. This may seem like a high degree of randomness, however after extensive experimentation, this method of mutation was very effective specifically due to the fact that the mutation rate can be changed mid-simulation. Initially, the mutation rate can start at a fairly high number (0.10-0.15) and then lowered by the user (to around 0.01-0.03) as the solutions improve. This was the most effective way at cutting down the number of generations to arrive at good worlds.

5.2.5 Master (Genetic Algorithm) Simulation Example



5.3 Experimentation

The following single world simulation parameters produced results worth mentioning. The specific details, thought process and results of each experiment will be analyze, contrasted and compared in the findings section.

ExperimentNum	Energy0	Energy1	Energy2	LossPerMove
1	-2	48	2	1
2	48	48	2	1
3	48	48	-48	1
4	-1	40	-38	1

The overall goal of these experiments is to determine if it is possible to develop a perfect world for a variety of different creatures. As long as the creature is able to reliably sustain itself on a randomly generated world, then the algorithm should be able to optimize their growth by manipulating their environment. At this point in the experimentation process, I was unsure as to how quickly, (in terms of number of generations, and population required) it would take to arrive at a truly optimize world. I was also unsure if the algorithm would be capable of generating a "perfect" world, or if it would merely improve their survival rate by developing a "slightly better" world.

6 Findings

Before discussing the findings of each of the individual notable experiments described in section 5.3. I will first discuss the most effective strategies when manipulating the parameters of the genetic algorithm itself:

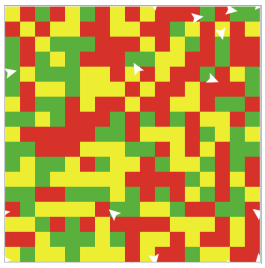
- A generation size of 20 was always sufficient to generating and testing a reasonable number of worlds while also maintaining fairly fast run time.
- A roulette factor of 0.10 was the most effective in allowing the algorithm to explore a variety of solutions while also consistently picking the top one's to breed.
- It was almost always effective to first begin the algorithm with a high mutation rate (0.10) (for the first 60-100 generations) and then continue running the algorithm with a mutation rate of 0.01 for the remainder.

These strategies were used in all four of the following experiments:

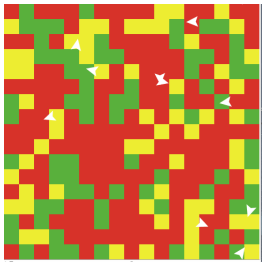
6.1 Experiment 1

The first experiment was to create a highly "specialized" creature who gains a significant amount of energy from a single specific (red) food. And gains very little/loses energy from the other food types.

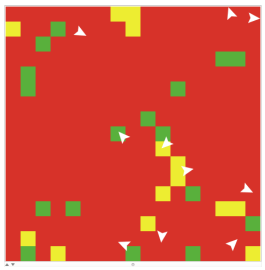
The initial population of worlds began with the average turtle count of 119.83



After running the algorithm for 80 generations with a roulette factor of 0.1 and a Mutation rate of 0.1. The evaluation function has increased to 172.75.



After running the genetic algorithm for another 100 generations, with a lower mutation rate of 0.01, the world has developed the ability to sustain 302.40 average turtles, with a board that is almost entirely red.

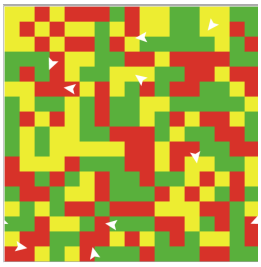


Improvement from this point on slows significantly, only making incremental progress. In the end, I was able to nearly triple the average number of organisms in the world. Although the world is not necessarily "perfect", as there are still inefficiencies in the remaining green and yellow patches, it was able to significantly improve over a very short amount of time.

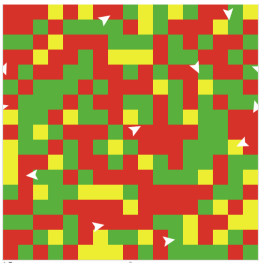
6.2 Experiment 2

The second experiment experiment was to take similar creatures as to in experiment 1, however I decided to make them "less specialized" by also having them gain a significant amount of energy from green food. In this version, there should be a larger pool of potentially good worlds as the creature is more versatile.

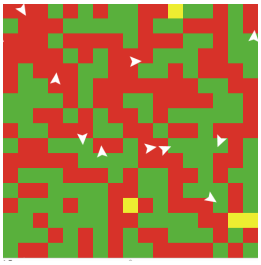
The initial population of worlds began with an average turtle count of 241.41. This is significantly higher than in experiment 1 which makes sense considering that these creatures are objectively "better" than those in experiment 1.



After running the algorithm again for 80 generations with a roulette factor of 0.1 and a mutation rate of 0.1. It is clear that there is significantly less yellow spaces in the world, whereas the "good" green and red spaces are much more prominent. The evaluation function has increased to 290.02. This increase is extremely similar to the first 80 generations of experiment 1. Both of these runs increase the evaluation function by around 50.



After running the genetic algorithm for another 100 generations, once again lowering the mutation rate to 0.01, the world developed the ability to sustain 330.51 organisms. This is an extremely "near perfect world" with only 4 "sub-optimal" yellow patches.



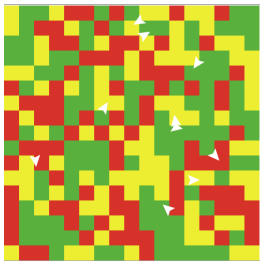
From this point forward, progress was again very slow, which is obvious considering there is very little room for improvement. I was never able to produce a result with 0 inefficiencies, even when turning the mutation rate to 0. However I was able to reduce the world to a single yellow square. This version of the algorithm ended significantly better than Experiment 1. This is likely due to the

fact that since there are now 2 "good" choices for food, there is a significantly higher ($\frac{2}{3}$ vs $\frac{1}{3}$) chance of a beneficial mutation.

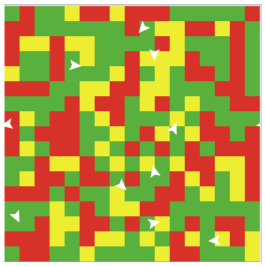
6.3 Experiment 3

The purpose of this experiment was to determine if a kind of food that is extremely detrimental to the creature (in this case yellow) has an affect on the ability develop a perfect world.

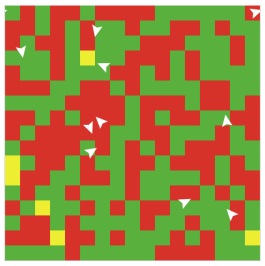
The initial population of worlds began with the highest average organism count of 187.47.



I then once again ran the simulation for 80 generations with a mutation rate of 0.1 and a roulette factor of 0.1 which resulted in an evaluation function of 252.04. (an increase of about 65).



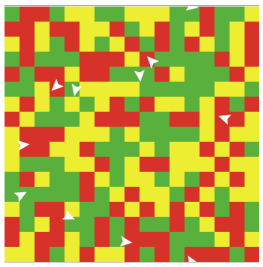
I then ran the simulation for another 100 generations and turned the mutation rate down to 0.01 as with the other simulations, and ended with an evaluation function of 331.07. This final result were not significantly different than the experiment 2 results despite the higher penalty values of the yellow food.



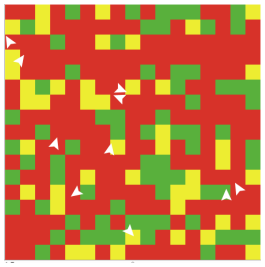
6.4 Experiment 4

The goal of the final experiment was to attempt to develop a perfect world for the most inefficient creature that can still reliably sustain itself and won't immediately die out.

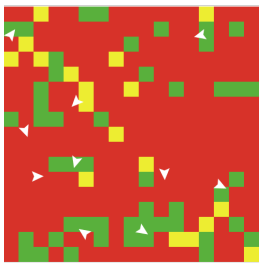
The best of the initial population averaged 31.9.



As with the previous experiments, the simulation was ran with 80 generations, 0.1 mutation and 0.1 roulette factor. After running this algorithm, the world was able to evolve to support 132.35 organisms; an increase of over 400%



After lowering the mutation rate to 0.01 and running the algorithm for 100 iterations again, the resulting board ended on an evaluation function of 187.67, and still has large areas to potentially improve.



7 Discussion of Findings

Overall the performance of the algorithm in terms of both time, as well as effectiveness exceeded my expectations. Experiment 2 in particular was a massive success; generating a near perfect world in less than 200 generations. I expect that the ability to manipulate the parameters of the genetic algorithm mid-simulation contributed greatly to exploring early to find "good" worlds, and then slightly tweaking them later to alter them into near perfect ones.

8 Conclusion

Overall I am extremely proud of the progress that was made on this project. The ability to generate near-perfect worlds in this extremely simplified model confirms that genetic algorithms can be effective tools to accomplish this task.

9 Future Work

I believe that there are two major areas that must be expanded upon:

The first of which is more complexity added to the single world simulation. The simplified model was an excellent proof of concept that this is a potential method of developing perfect worlds. The next step would be to add new elements to both the creature's behavior, as well as the world generation. Even on a 2D simplified board, potential additions include:

- Terrain Generation: Different terrain such as mud, water, grass, rocks etc. These terrain elements can give the creatures effects such as faster/slower movement speeds, energy expenditure, or can increase/decrease the regrowth time of food in it's vicinity.
- Overpopulation Mechanics: Mechanics that involve punishing worlds that produce too many organisms may produce interesting results. Currently the "fitness" of whether a world is a directly proportional to the number of creatures it is able to sustain. However in reality, an overpopulated world in not necessarily the best world.
- Predator and Prey Mechanics: The creatures can be programmed to be hunted by certain other creatures, or hunt others as another means of food. Interesting behavior could result from this, such as the world purposely inserting a predator into the world in order to reduce the previously mentioned overpopulation penalty.
- Urges towards certain foods and food types: It may be interesting to see how the genetic algorithm would use these urges towards certain kinds of food to it's advantage.

Furthermore, another logical progression of this simulation would be to increase the complexity of the creature themselves. Instead of each creature being represented by a single "turtle" entity in Netlogo, the ability to create a more complex multi-system creature and evolve a world to fit their needs may produce unique results. This may logically progress into converting the premise of this project into a 3-d simulation using an engine such as Unity. Overall I believe that there are many potential additional complexity elements that can produce interesting results.

10 References

1. http://www.marspapers.org/paper/Mole_1999.pdf
2. https://ccl.northwestern.edu/netlogo/
3. Hjorth, A. Head, B. Wilensky, U. (2015). "LevelSpace NetLogo extension". http://ccl.northwestern.edu/rp/levelspace/index.shtml Evanston, IL: Center for Connected Learning and Computer Based Modeling, Northwestern University.
4.http://www.ufjf.br/domvs/files/2013/11/DIGITAL-TECHNIQUES-APPLIED-TO-DESIGN-PROCESS.pdf
5. https://rimworldwiki.com/wiki/AI_Storytellers
6. https://left4dead.fandom.com/wiki/The_Director
7. https://www.sciencedirect.com/science/article/pii/S0097849301001571