UNIVERSITAT JAUME•I

Bachelor's Degree in Video Game Design and Development

Final Degree Project

Exploration of Genetic Algorithm Techniques for Non-Deterministic Narratives Development

Author:
Guillermo Gandía Martín

Academic Tutor:
Luis Amable García Fernández

Academic course 2018/2019

Summary

The proposal of the present Final Degree Project in Video Game Design and Development consists in the implementation of a procedural narrative system applying artificial intelligence. This work aims to inquire into genetic algorithms techniques for the creation of an automatic space of possibilities. The idea is to generate a story, based on a minimum input values sequence given by the user. In the process, the foundations of storytelling and the development of narrative archetypes will also be studied with the intention of giving rise to a structurally coherent non-deterministic discourse. The ultimate goal is to demonstrate the feasibility of a complex narrative system design through a demo implemented in a game engine such as Unity3D.

Keywords

Narrative, Artificial Intelligence, Procedural, Genetic Algorithms, Non-Deterministic Narratives.

Project link

https://github.com/spaceraptor/GeneticAlgorithmFDP

Contents

Introduction

1.1. Context and work motivation.

The idea that I propose in this Final Degree Project (FDP henceforth) did not arrive from one day to the next. In fact, before I came up with this project, I spent a couple of weeks considering a great diversity of topics.

From the beginning, it was clear to me that for this FDP I was expecting not to make a whole video game. For a game to be successful, it must be distinguished from the rest either by its aesthetic or by being technologically superior or innovative. A video game is a broad ranging work and the available amount of time and student experience is insufficient to make it stand out in any specific section.

In view of the fact that developing a generic game was not a thrilling choice for me, I decided I had to focus all my efforts on something more specific. I was expecting to obtain an innovative element I could take advantage of in advance, which might serve me to improve possible future works.

I gathered a considerable catalog of proposals for this project that covered a great amount of disciplines. Most of them, though, could easily be classified into two main categories: on the one hand, researching in the field of narrative, due to the fact that I wanted my FDP to concern narrative design without just making a video game; on the other hand, to explore unusual or alternative applications of Artificial Intelligence (AI henceforth), because I have always been highly interested in the simulation of intelligent behaviors. Eventually, it came the thought of mixing both ideas.

Studies and research that join AI and narrative fields are not numerous, placing the union between them in a very uncommon circumstance. However, the fact of being unusual and little explored makes it a fresh, different idea and also leaves the doors open to experimentation. Since my initial goal was to develop something original and appealing, as well as challenging, putting both topics together was just a matter of time.

In conclusion, the foremost reason that motivates this project lies in the intention of doing an intellectually fruitful FDP, stepping forward towards those disciplines which greatest fascination produces me. Its unconventional narrative and AI nexus makes it quite an original though demanding idea to develop. In addition, I strongly believe the fulfillment of this project may have an advantageous role in possible future works.

1.2. Objectives and expected results.

1.2.1. Expected results.

With this FDP, I pursue to fulfill the implementation of an automatic generated story and its discourse. The narrative is meant to be non-deterministic, which means that all the auto-generated stories must be unique from each other no matter how many times the user tries to repeat the same input patterns.

This project will require myself ground in artificial intelligence techniques and main archetypal narrative structures, for their later application in the construction of a non-deterministic narrative discourse displayed by the AI. It will be necessary to lay out a design to configure the desired procedural narrative system. Therefore, the result should not be seen as a software application as much as an enquiry project.

The expected procedural means are not viable to create the whole graphical content of a virtual world. Otherwise, after developing a story we would need to build a second system just for its discourse, in order to control the space, cameras, animations of characters, dialogs, timing, etc. This task would be excessively hard and ambitious. Simplicity must be taken as the key principle of visualization, avoiding any emerging complexity.

Nevertheless, discourse is still necessary to communicate the story given by the procedural system. With the purpose of minimizing the presence of complex discursive elements, the most optimal and simplest idea to communicate our story is to use text strings. Accordingly, discourse will be held by self-generated strings in a box dialog, which means that the procedural history will be told exclusively in text format. Moreover, they need to be as tailored to the viewer as possible, considering the given static information (prior general user data such as interests and preferences).

Lastly, it is expected that the narrative system generated during the development of the FDP will eventually be implemented in a game engine. Also, it will be designated the way the UI is supposed to be visualized, according to the system structure already implemented at that moment. Once this goal is achieved, several elements and adjustments might be added to improve the user experience and the quality of the generated story.

These expected results have been carried out successfully.

1.2.2. Objectives.

- To delve into the fields of knowledge involved in both the narrative and AI scenario.

It would not make sense to complete this project without having acquired the body of knowledge that is needed to carry it out:

- In the first place, to understand the narrative logic, knowing how to apply the most common structures, main archetypes, as well as learning how to manage other fundamental elements such as rhythm and suspense.

- In the second place, to take a close look to artificial intelligence techniques, as is the case of genetic algorithms, and to be able to understand them, implement them and adapt them to the needs that are opportune.

Narrative is a field that has been scarcely explored in terms of AI, given its strong creative nature and the great complexity involved in organizing such behaviors in logical sequences. Consequently, a vast part of this this FDP entails a process of research and study in the pertinent fields of development, in spite of the scarce references.

The reason why I have chosen genetic algorithms for this project lies with its flexibility creating new random and valid results, without being forced to fulfil the established criteria in a strictly perfect manner. The variety of working results obtained can be seen as possible narrative situations that characters may unleash, according to their attributes and personality.

The whole study and research process will culminate in the knowledge needed to implement genetic algorithms, adapted to the requirements of the archetypal narrative structures.

- To design a conceptual and algorithmic system of coherent non-deterministic narrative stories to be procedurally generated.

As a basis, it is essential to understand the hierarchical categories that make up the narrative structure. Starting from objects that designate the narrative unit (the smallest link of this structure), we build new objects that are increasingly complex and that are able to relate coherently with the elements of a lower category. It should not be possible to start programming without taking a prior scheme of this structure into account.

The conceptual design is the scheme intended to establish the set of relationships between the objects that configure the narrative system. From this scheme, it must be possible to generate a new story, along with its discourse. The foundations above which this design is built require a huge comprehension of the fields studied in the previous objective.

The algorithmic design is the task that is adopted once the conceptual design is finished. It takes charge of the eventual implementation of the narrative system, bringing those ideas to a level closer to the programming language. Efficiency issues are also taken into account.

The objective of the proposed designs is that they configure a narrative system, virtually capable of systematically creating a robust and coherent plot, not determined by a previously written branch. An effort has been made to open the system to the maximum possible space of possibilities. For this work, only temporary linear discourses are contemplated, that is, without temporary leaps, flashbacks or flashforwards.

- To develop a functional program of the preceding design in a game engine.

The aforementioned schemes can not be considered functional without a validation test. A demo has been developed in a game engine (Unity3D in our case), built following the indications and measures proposed in the conceptual and algorithmic design.

To consider the system feasible, the final result of this demo must generate and display a coherent narrative sequence on the screen. They should essentially be triggered by its characters, depending on their personality and other diegetic elements.

Hence, more attention has to be paid to the way in which the program use the available resources and whether the constructed context is respected. It is not about demonstrating the quality of the stories produced, but the functionality of its design.

1.2.3. Initial hypothesis.

This work can be used as a tool to create or enhance emergent narrative video games. It can both produce personalized actions given the behavior of characters and develop a succession of events accordingly.

Furthermore, the applications of this system go beyond the simple fact of building stories. It can be used, for instance, to improve interactions with NPC, making them capable of generating more dynamic and natural dialogs.

However, it has to be noted that the goal of this project is to demonstrate the viability of the design through a functional demo. The variety of situations, plot twists and the richness of the content shown will not be a requirement for this FDP. The whole set of narrative elements and grammatical variables that enhances a story can be added to the narrative system afterwards.

1.3. Environment and initial state.

1.3.1. Working method.

The work methodology has not been homogeneous, since it has been varying its dynamics according to the needs that were arising, adapting to each stage of development. It should be pointed out that the environment of the whole process took place in my own home.

The first part of the process was research and study, where the main activity was the reading and jotting relevant ideas. Documentation used was [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11] and [12]. The conceptual design phase of the narrative system consisted of two parts: assimilation of the researched content and written schematization of the narrative study under a computational perspective. The last phase of development belong to both the description of the algorithmic design and the system implementation in Unity, which are mainly focused on programming.

1.3.2. Related subjects.

VJ1203 - Programación I
VJ1208 - Programación II

VJ1215 - Algoritmos y estructuras de datos
VJ1217 - Narrativa hipermedia y análisis de videojuegos
VJ1227 - Motores de juegos
VJ1231 - Inteligencia Artificial

1.3.3. Previous knowledge.

The AI subject had not yet been completed before the project began, and my knowledge of the subject was very limited, which was a challenge since AI is one of the fundamental pillars on which the FGP is based.

The scarcity of studies and references found on the application of AI techniques in the field of narrative did not improve the situation. Few of these papers were useful, either because they were too generalist or because they aimed far from my purposes. It was clear that I had to study both disciplines separately, and rely on what I learned to put them together according to my judgment.

Luckily, the knowledge I already had regarding the narrative aspects of a video game was higher. I have always shown interest in this topic, reading or studying on my own, which subsequently allowed me to invest more time in familiarizing myself with genetic algorithms.

1.3.4. Initial state.

During the beginning of the development I had an unclear idea of what the desired result of the project should be like. I was clear that I wanted to design a system that would generate non-deterministic procedural histories, but I did not know how to get to that point, not even what elements should I show on the screen. I was not sure whether the theme I had proposed for my FDP would be too ambitious and complex, especially given the limited knowledge I had at that time. I was worried that the method to elaborate my long-awaited system of procedural narrative would be out of my reach.

However, the help received in tutorials by my tutor, Luis Amable García Fernández, was especially useful to focus the project in the direction of a specific AI technique and be able to move forward. In this way, from a series of potential candidates, the genetic algorithm was finally selected as the preferred AI type, which the entire project would rotate from that moment on. The reasons why the genetic algorithm was chosen from other AI are mentioned in the objectives section.

Planning and Resources evaluation

2.1. Planning.

2.1.1. Development stages.

The development phases carried out within the project have been sequentially divided into three stages:

- Extraction of the information (task 1).
  This stage was based on searching for information and studying deeply both the narrative and AI subjects. The main activity was the reading of books, articles, videos and other resources that could provide relevant information. Simultaneously, notes were also written down so that information could be reviewed later.

- Conceptual design (tasks 2, 3 and 4).
  The conceptual design draw the relationships and interactions of all the different narrative entities, beneath the application and adaptation of the genetic algorithms. In addition, a simple program using the genetic algorithm as an example was also implemented in Unity3D, which was useful in the following step.

- Algorithm design and final implementation (tasks 5, 6, 7 and 8).
  The last phase focused almost exclusively on the use of the computer to program. The progress took longer than expected because all the tasks of algorithmic design, final implementation and debugging were done in parallel. This was made to check the correct functioning of each inner step before moving on to the next one. Conceptual design was frequently revised and updated in view of the need to correct possible failures, inconsistencies or ambiguities.

Tasks that would make up each stage will be discussed in the following section.

2.1.2. Task planning.

At the beginning of the project, it was designated the succession of tasks to be completed and the timeline they should cover for the adequate completion of the project. These tasks have been sequentially ordered, so once a task was completed, the starting point of the following task began. Deadlines were also estimated based on its assumed difficulty.

TASKS	DEADLINE
1. Reading, initial understanding of the fields to be treated and taking notes.	- 25th February
2. Intensive comprehension of the notes taken in the previous step, in order to mix ideas from both narrative and AI fields.	6th March
3. Conceptual Design. Conceptual AI adaptation to a discourse development based on the leading narrative structure.	18th March
4. First implementation of a simplistic genetic algorithm example in a game engine.	1st April
5. Algorithmic Design. Adaptation of the genetic algorithm example already implemented, taking the Conceptual Design into account.	15th April
6. First functional prototype of non-deterministic narrative.	1st May
7. Debugging and polishing of basics aspects to improve.	13rd May
8. Final version of the procedural narrative system and verification of correct performance, efficiency, readability, improvements and possible aesthetic additions.	31st May

All tasks are directly dependent on the previous one, with the exception of the fourth, which would only depend on the first two (study and understanding). The fifth task would have as direct dependencies both the third and the fourth tasks.

2.1.3. Development delays.

In the following table we observe the list of tasks together with the corresponding deadlines that were initially assigned to each one and the date when they were eventually completed:

TASKS	DEADLINE	DELAY
1. Reading, initial investigation of the fields to be treated and taking notes.	- 25th February	25th February
2. Intensive comprehension of the notes taken in the previous step, in order to mix ideas from both narrative and AI fields.	6th March	6th March
3. Conceptual Design. Conceptual AI adaptation to a discourse development based on the leading narrative structure.	18th March	25th March
4. First implementation of a simplistic genetic algorithm example in a game engine.	1st April	15th April
5. Algorithmic Design. Adaptation of the genetic algorithm example already implemented, taking the Conceptual Design into account.	15th April	11th May
6. First functional prototype of non-deterministic narrative.	1st May	13th May
7. Debugging and polishing of basics aspects to improve.	13rd May	15th May
8. Final version of the procedural narrative system and verification of correct performance, efficiency, readability, improvements and possible aesthetic additions.	31st May	31st May

In the first place, there was a delay in the conceptual design phase, as it involved a greater amount of work and effort than initially expected, especially the third and fourth tasks. On the other hand, as already explained, delays in the fifth, sixth and seventh tasks were determined by the decision to perform them simultaneously, finishing them in very close time frames.

2.1.4. Gantt Charts.

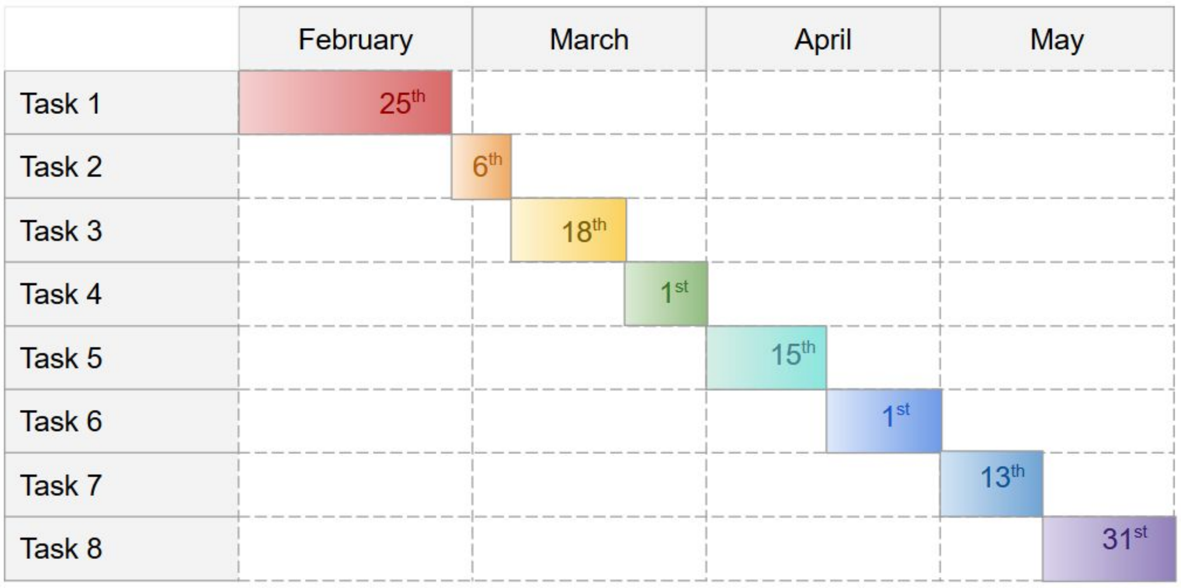


Fig. 2.1: Gantt chart about the initial planning

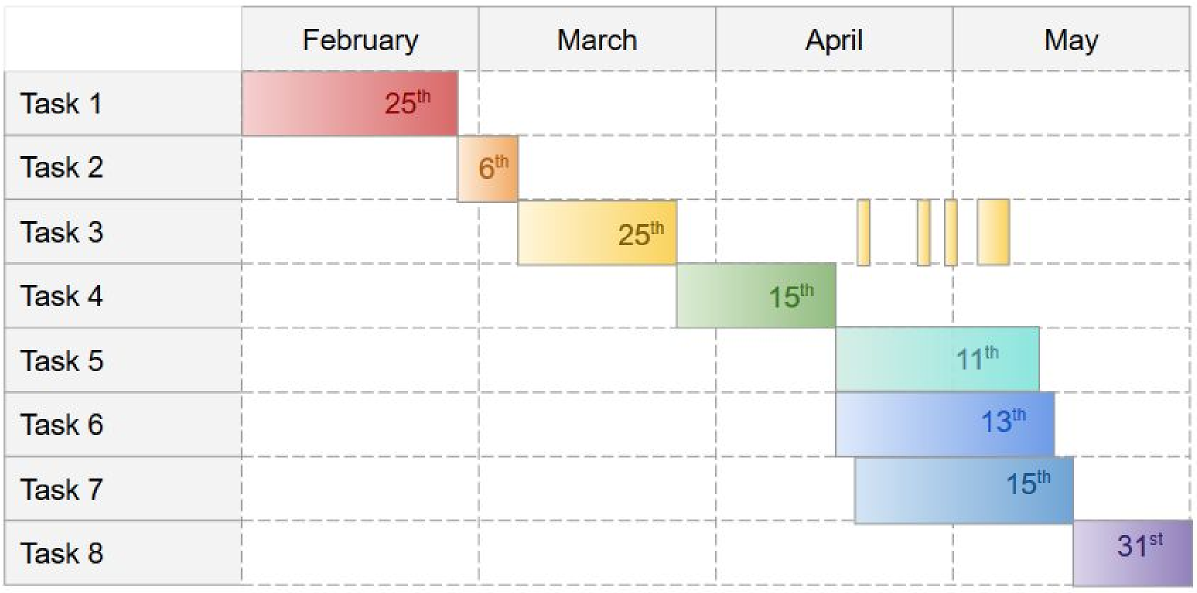


Fig. 2.2: Gantt chart about the final development process

2.2. Resources evaluation.

2.2.1. Human resources.

For the development of this FDP no other people have intervened besides my tutor.

2.2.2. Equipment resources.

- Computer or laptop
- Unity3D version 2018.3.6f1
- Microsoft Visual Studio version 2017

There has been no economic expense in acquiring material for the development of the FDP. I already had both the computer and the tools used before starting the project, besides the fact this software is free. Moreover, it is not necessary for the computer to require a powerful architecture to run the system we want to build.

System analysis and Design

3.1. System analysis.

3.1.1. Functional requirements.

- System shall store the static data provided by the user.
- System shall employ narrative elements based on the user preferences.
- System shall not generate the same output when providing the same input data.
- System shall show the best path of events, among those procedurally generated.
- System shall print a text string describing the current situation of the story.
- System shall move forward to the next step when clicking the "Next" button.
- System shall go back to the previous step when clicking the "<" button.
- System shall print the corresponding information when clicking the info buttons.
- System shall restart when clicking the "Ex" button.

3.1.2. Non-functional requirements.

- System shall be minimalist.
- System shall be efficient.
- System shall be robust.
- System shall be intuitive.

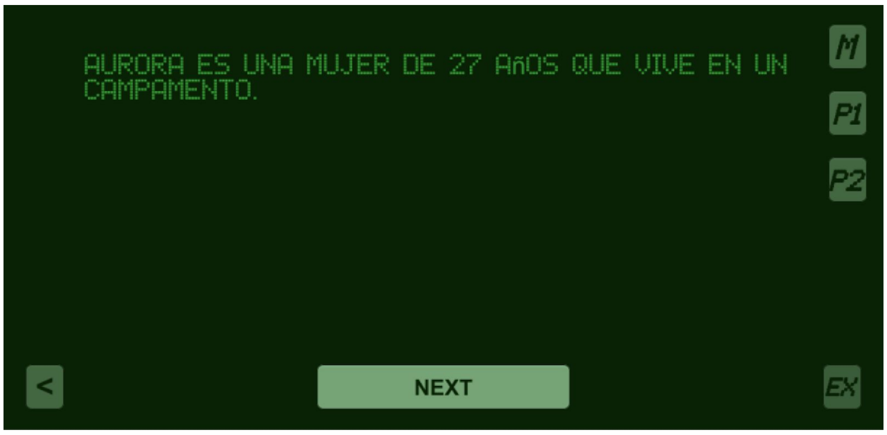
3.2. Design.

3.2.1. System architecture.

It does not require the use of internet to connect to external systems. Consequently, we can consider it a conventional or standard architecture.

3.2.2. Interface design.

The interface design is extremely minimalist, as no graphic element is needed besides the use of text strings. The whole interface is inspired in an old computer console style.



```
AURORA ES UNA MUJER DE 27 AñOS QUE VIVE EN UN
CAMPAMENTO.
```

Fig. 3.1: Image of the current interface state during the narration of a story

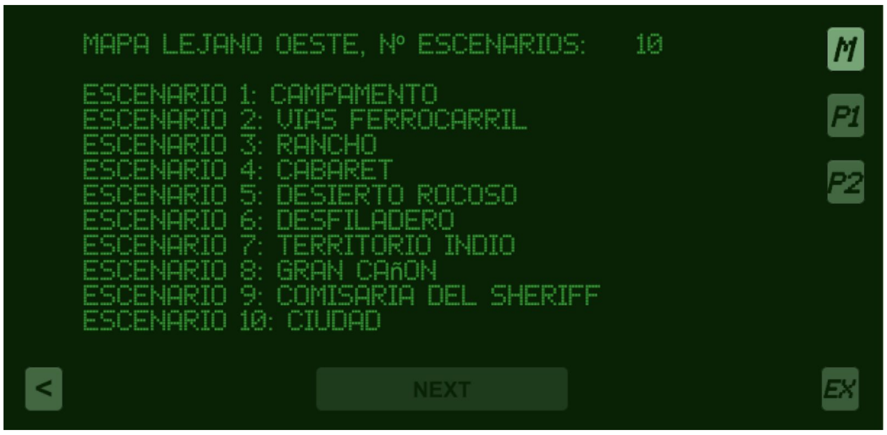
At the bottom of the screen, there are three buttons:

- The big button called "NEXT" is meant to move the step system forwards.
- The arrow button "<" on the left corner is meant to go back to the previous step.
- The button "EX" on the right corner is meant to exit and restart the story.

At the right border of the screen, there are three tiny buttons:

- The button in the top has a "M" on it and is meant to show the current map.
- The button in the middle has a "P1" on it and is meant to show character 1 data.
- The button in the middle has a "P2" on it and is meant to show character 2 data.

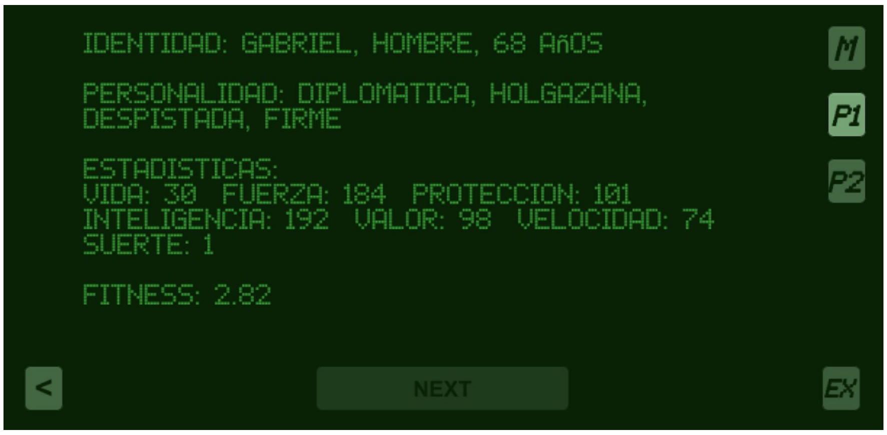
These last three buttons are intended to show every individual aspect of either the characters or the setting present in the story, as we can see on the following images.



```
MAPA LEJANO OESTE, Nº ESCENARIOS:   10

ESCENARIO 1: CAMPAMENTO
ESCENARIO 2: VIAS FERROCARRIL
ESCENARIO 3: RANCHO
ESCENARIO 4: CABARET
ESCENARIO 5: DESIERTO ROCOSO
ESCENARIO 6: DESFILADERO
ESCENARIO 7: TERRITORIO INDIO
ESCENARIO 8: GRAN CAñON
ESCENARIO 9: COMISARIA DEL SHERIFF
ESCENARIO 10: CIUDAD
```

Fig. 3.2: Image of the current map information


IDENTIDAD: GABRIEL, HOMBRE, 68 AñOS

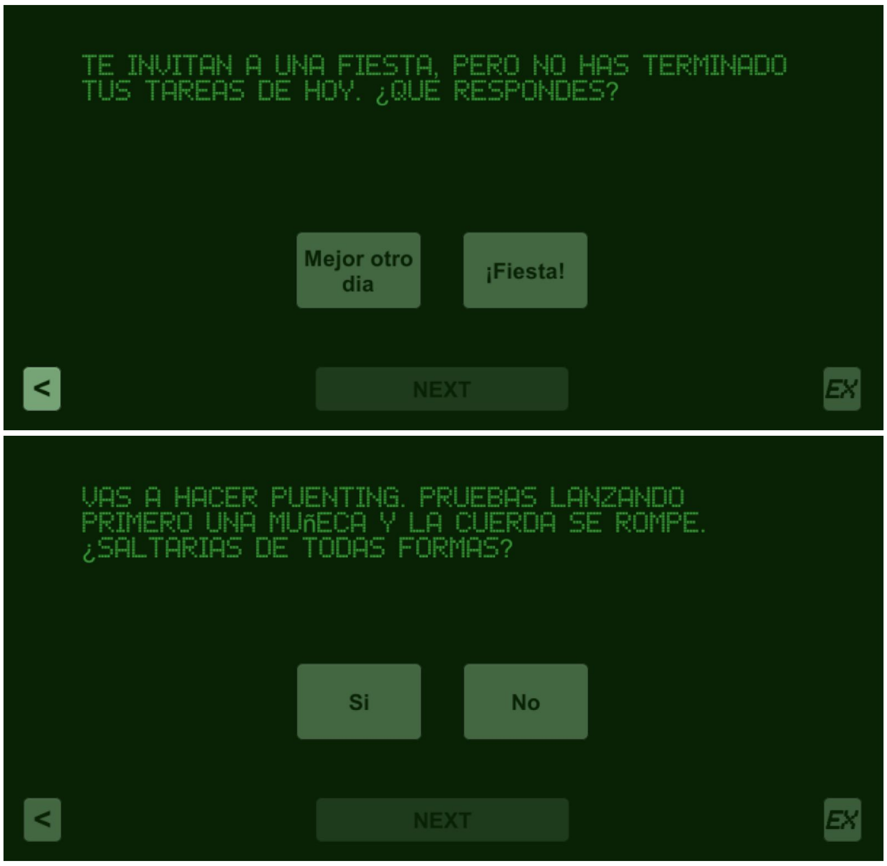
PERSONALIDAD: DIPLOMATICA, HOLGAZANA, DESPISTADA, FIRME

ESTADISTICAS:
VIDA: 30 FUERZA: 184 PROTECCION: 101
INTELIGENCIA: 192 VALOR: 98 VELOCIDAD: 74
SUERTE: 1

FITNESS: 2.82

Fig. 3.3: Image of the character 1 data

Before starting the narration, the system provides the user the choice to answer several questions (*a.k.a. static questions*). Those questions are answered through some additional buttons and text dialogs. The goal of *static questions* is to build up narrative elements according to the user preferences, such as the main character, the stages, and so on. There is also a randomized mode in which the user can skip those questions.


TE INVITAN A UNA FIESTA, PERO NO HAS TERMINADO TUS TAREAS DE HOY. ¿QUE RESPONDES?

VAS A HACER PUENTING. PRUEBAS LANZANDO PRIMERO UNA MUñECA Y LA CUERDA SE ROMPE. ¿SALTARIAS DE TODAS FORMAS?

Fig. 3.4: Images showing different examples of *static questions*

Development and Results

4.1. Brief description of genetic algorithms.

4.1.1. Overview.

Before moving on to the development of the FDP, it is considered interesting to explain what the so-mentioned genetic algorithms are and how they work.

Genetic algorithm is a AI technique inspired by the field of biology. It tries to apply the same abstract processes present in the evolution of species for calculating enough good results in the field of algorithmic optimization.

Given a population of individuals, each one representing a potential solution for the problem to solve, they are evaluated according to the quality of the solution they offer, generating a value called fitness. Best individuals, that is, those with greater fitness, will have greater probabilities of reproducing with other individuals. After this breeding process, descendants will be created from the genetic information crossover of their parents and thus obtaining a new evolved and eventually enhanced population. The algorithm also carries out mutations that can randomly change the genetic data of an individual.

An interesting point is that it allows us to select a good candidate solution, which actually does not have to be the best, but it will be the best up to this moment.

Next, it will be shown a simple example that illustrates this algorithmic process. [1]

4.1.2. Faucet example.

The implementation of this example was certainly useful for me so as to completely understand genetic algorithms and advance subsequent developing steps of this project.

[1] This program can be found in this link: https://github.com/spaceraptor/GeneticAlgorithmFDP

This example was extracted from the book [1]. Note that although it appears to easily exemplify the steps involved in the implementation of genetic algorithms; the code to perform this particular example is not provided in that book.

It is contemplated the following situation: we have a faucet with two threads, one to adjust the temperature and another to adjust the water pressure. The rotation of the threads can be represented with the values between zero and five, with zero being the coldest temperature and the weakest flow respectively, whereas for five is the hottest temperature and the strongest flow respectively.

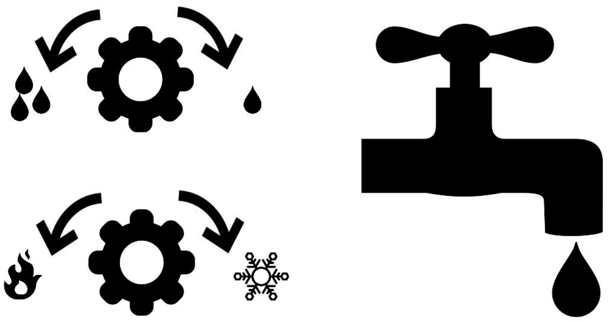


Fig. 4.1: Graphic description of the faucet example

A person wants to wash their hands with cold, not frozen water, and with a large flow, without splashing due to excessive pressure. We could state that the ideal equilibrium of values is one unit for the temperature thread and one and a half units for the pressure thread, although this information is unknown to that person. We must create a system using the genetic algorithm that simulates to some extent the behavior of this person, shuffling random values as we get closer and closer to an acceptable solution, which does not necessarily have to be perfect.

Hereunder, I will describe the elements of this problem, explaining their corresponding nuances. The key is to understand the general process and know the operations carried out.

To solve this problem we will need to create the following elements, which are the pillars of any genetic algorithm:

- Gene
- Genome
- Fitness
- Individual (faucet)
- Population
- Evolution Manager

- Gene

Genes are treated for each single variable that makes individuals different from each other. In the case of faucets there are just two different genes, temperature and pressure.

- Genome

The genome is the element that stores all the genes of an individual. Therefore, the genome of the faucet contains the available temperature and pressure genes.

- Fitness

The fitness function gives the value attached to an individual. It indicates how close a genome is to the ideal values of genes a faucet can have, or said in other words, how good is that solution. The closer the genes values are to the ideal value, the higher the fitness value will be.

It is important to set a minimum and maximum fitness value. On the one hand, a minimum value is required to limit whether a solution is valid or not. Values outside the spectrum of the established criteria will not reach the minimum fitness. On the other hand, maximum fitness can be the reference point of an ideal or perfect solution. In this example, our ideal genome could be one unit for temperature and one and a half unit for pressure, whereas the minimum could be zero.

- Individual (faucet)

An individual is the element that comprises a particular value for genome. An individual can be obtained from three different ways: from two parents (other two individuals); replicated from a single parent; and from random values for its genes. The quality of an individual is provided by using the fitness function.

An individual can mutate the values of one or more of its genes. We should set a mutation rate probability, so that for each created faucet which meets that probability, its genetic heritage can get modified for best or worst. It is not recommended to apply a very high mutation percentage, since individuals with a good genome would tend to get lost more frequently; however, it should not be too low either, so that the possibility of obtaining new better results is not null. Generally, a probability from ten to twenty percent is usually the most optimal.

In case of receiving a single faucet, the values of both genes from the parent are replicated in the son; in case of receiving two individuals, the value of each gene is inherited from a different parent in random order under the same probability (it would replicate the temperature gene of the father and the gene pressure of the mother or vice versa). Subsequently, it has to be determined whether the mutation of the genes is executed. Lastly, the fitness value is calculated.

- Population

The population is the set that groups all the individuals of a generation. It is convenient to determine the number of individuals that each generation will contain.

The first population is generated with individuals using random values for its genes. Subsequent populations are obtained from the previous generation of individuals. A selection method must be used to choose the parents given to create each new faucet. A crossover rate is held to indicate whether the faucet is going to be born from the union of two parents or simply replicates the genome of a single parent.

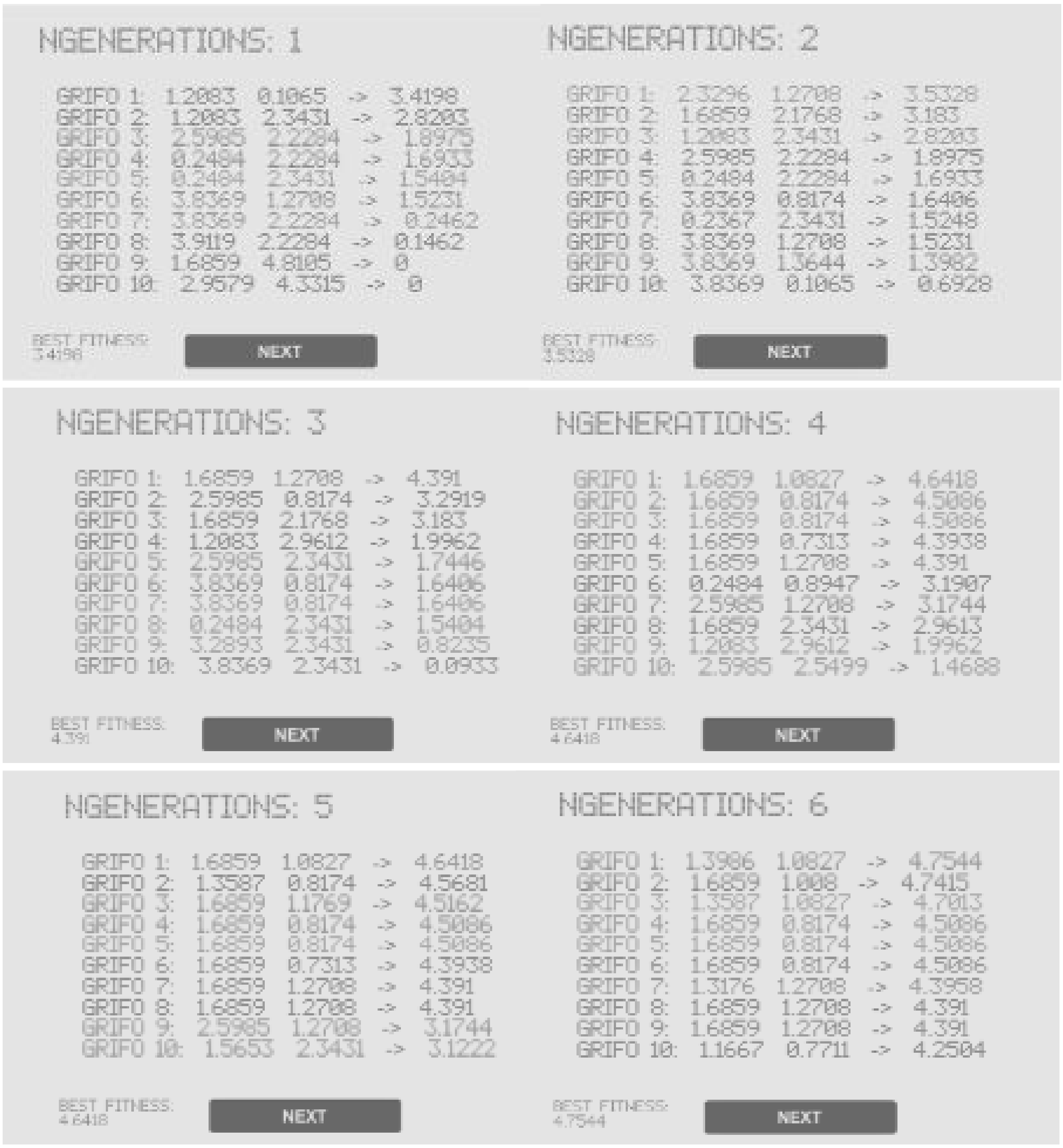
The selection method is based on a biased roulette, which consists of randomly choosing an individual, whose percentages of success depend directly on their fitness; that is, faucets at the top of the priority list would be more likely to be selected rather than those at the end for having lower fitness. However, there will not be any whose probability is nil; absolutely all individuals of the generation are likely to come out as candidates.

- Evolution Manager

It can be considered as the main programme. This is the script that manages the evolutionary process of our populations. Its duty is to arise the first generation of individuals and create the following new ones, as well as accessing the best individual of that generation and keeping the best faucet so far.

It includes a method that creates a new generation using the previous one as an argument every time it is triggered.

The images below show the evolutionary process output of this implemented example. Note how every new generation improves their fitness from the previous one, getting closer to the ideal value.



```
NGENERATIONS: 1

GRIFO 1:  1.2083  0.1065  ->  3.4198
GRIFO 2:  1.2083  2.3431  ->  2.8203
GRIFO 3:  2.5985  2.2284  ->  1.8975
GRIFO 4:  0.2484  2.2284  ->  1.6933
GRIFO 5:  0.2484  2.3431  ->  1.5404
GRIFO 6:  3.8369  1.2708  ->  1.5231
GRIFO 7:  3.8369  2.2284  ->  0.2462
GRIFO 8:  3.9119  2.2284  ->  0.1462
GRIFO 9:  1.6859  4.8105  ->  0
GRIFO 10:  2.9579  4.3315  ->  0

BEST FITNESS:
3.4198
```

```
NGENERATIONS: 2

GRIFO 1:  2.3296  1.2708  ->  3.5328
GRIFO 2:  1.6859  2.1768  ->  3.183
GRIFO 3:  1.2083  2.3431  ->  2.8203
GRIFO 4:  2.5985  2.2284  ->  1.8975
GRIFO 5:  0.2484  2.2284  ->  1.6933
GRIFO 6:  3.8369  0.8174  ->  1.6406
GRIFO 7:  0.2367  2.3431  ->  1.5248
GRIFO 8:  3.8369  1.2708  ->  1.5231
GRIFO 9:  3.8369  1.3644  ->  1.3982
GRIFO 10:  3.8369  0.1065  ->  0.6928

BEST FITNESS:
3.5328
```

```
NGENERATIONS: 3

GRIFO 1:  1.6859  1.2708  ->  4.391
GRIFO 2:  2.5985  0.8174  ->  3.2919
GRIFO 3:  1.6859  2.1768  ->  3.183
GRIFO 4:  1.2083  2.9612  ->  1.9962
GRIFO 5:  2.5985  2.3431  ->  1.7446
GRIFO 6:  3.8369  0.8174  ->  1.6406
GRIFO 7:  3.8369  0.8174  ->  1.6406
GRIFO 8:  0.2484  2.3431  ->  1.5404
GRIFO 9:  3.2893  2.3431  ->  0.8235
GRIFO 10:  3.8369  2.3431  ->  0.0933

BEST FITNESS:
4.391
```

```
NGENERATIONS: 4

GRIFO 1:  1.6859  1.0827  ->  4.6418
GRIFO 2:  1.6859  0.8174  ->  4.5086
GRIFO 3:  1.6859  0.8174  ->  4.5086
GRIFO 4:  1.6859  0.7313  ->  4.3938
GRIFO 5:  1.6859  1.2708  ->  4.391
GRIFO 6:  0.2484  0.8947  ->  3.1907
GRIFO 7:  2.5985  1.2708  ->  3.1744
GRIFO 8:  1.6859  2.3431  ->  2.9613
GRIFO 9:  1.2083  2.9612  ->  1.9962
GRIFO 10:  2.5985  2.5499  ->  1.4688

BEST FITNESS:
4.6418
```

```
NGENERATIONS: 5

GRIFO 1:  1.6859  1.0827  ->  4.6418
GRIFO 2:  1.3587  0.8174  ->  4.5681
GRIFO 3:  1.6859  1.1769  ->  4.5162
GRIFO 4:  1.6859  0.8174  ->  4.5086
GRIFO 5:  1.6859  0.8174  ->  4.5086
GRIFO 6:  1.6859  0.7313  ->  4.3938
GRIFO 7:  1.6859  1.2708  ->  4.391
GRIFO 8:  1.6859  1.2708  ->  4.391
GRIFO 9:  2.5985  1.2708  ->  3.1744
GRIFO 10:  1.5653  2.3431  ->  3.1222

BEST FITNESS:
4.6418
```

```
NGENERATIONS: 6

GRIFO 1:  1.3986  1.0827  ->  4.7544
GRIFO 2:  1.6859  1.008  ->  4.7415
GRIFO 3:  1.3587  1.0827  ->  4.7013
GRIFO 4:  1.6859  0.8174  ->  4.5086
GRIFO 5:  1.6859  0.8174  ->  4.5086
GRIFO 6:  1.6859  0.8174  ->  4.5086
GRIFO 7:  1.3176  1.2708  ->  4.3958
GRIFO 8:  1.6859  1.2708  ->  4.391
GRIFO 9:  1.6859  1.2708  ->  4.391
GRIFO 10:  1.1667  0.7711  ->  4.2504

BEST FITNESS:
4.7544
```

Fig. 4.2: Output of the faucet example program

Legend:
Ngenerations stands for the number of the current population. The first column from the left is the number of the faucet within the current generation. The second column from the left is the pressure gene value. The third is the temperature gene value. The fourth column is the corresponding fitness attached to each faucet. At the lower left corner of each image it is indicated the best fitness so far. All faucets are sorted by fitness value.

4.2. Conceptual design.

The conceptual design process consists essentially of obtaining, expressed in the form of schemes, a certain solution to a problem by synthesizing such problem into concepts and defining its related set of tasks.

The arisen problem is that of creating an algorithmic system capable of building a new story, that has not been previously written. This story must respect a series of important narrative criteria in order to be considered a well-formed story:

- Must be able to provide a beginning and an end.
- It must be consistent with the created context.
- The individuals of the story must have an active role.

The narrative system requires a user input to generate the initial background of the story, since creating it from scratch might be too complex; then, based on this information received, the machine should be able to produce new consequent information on its own. This general behavior, so simple at first sight, give rise to all the following study, which combines theoretical narrative with algorithmic logic and programming.

The conceptual design, along with the algorithmic design, that we will see later, is the result of my own reflections that have arisen during the development of this FDP. Results make possible a functional synergy between these so far so little related fields. Now, it will be introduced the main elements of the narrative system:

4.2.1. Narrative steps.

We are going to define a *step* as the minimal narrative unit of our system, which is indeed a situation or conflict involving a set of characters, a place and an action. The narrative thread could be compared to a sequence of problem-solution *steps*; that is to say, there is a connected path of *steps* that leads to the final goal. *Steps* split the story in a particular number of situations where a specific action is taken by the characters.

$$S_1 \rightarrow S_2 \xrightarrow{*} S_n$$

where S_n *means step number* n and * meaning in zero or more basic *steps*

Fig. 4.3: Narrative succession formula of *steps*

One of the most common archetypal narrative structure is known as the Hero's Journey [2]. The internal narrative structure our system follows is especially based on this outline. The hero's journey establishes twelve common points that the adventures of the hero have shared by many world mythologies. It has served as reference to great Hollywood film writers and storytellers.

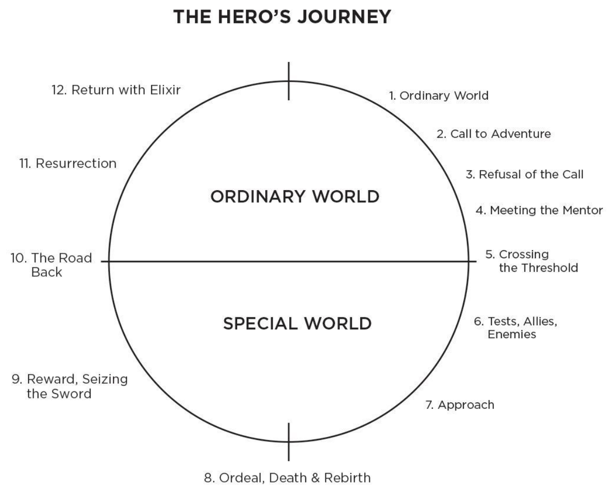


Fig. 4.4: Hero's Journey diagram

It starts in an ordinary peaceful world until that apparent stability is broken by an external force. The hero might not recognize herself as such at first, but once she decides to take part of this adventure, she will travel to the dangerous world where the evil resides to restore peace from home. There are some relevant elements involved like the mentor, allies, foes and the elixir that will bring the world to the previous state of calmness.

To simplify our eventual narration, these twelve points of the hero's journey diagram have been reduced to just eight fundamental ones.

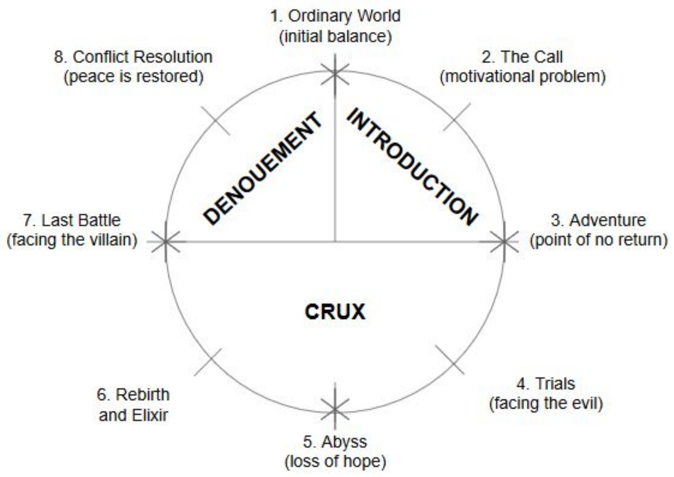

1. Ordinary World (initial balance)
8. Conflict Resolution (peace is restored)
2. The Call (motivational problem)
DENOUEMENT
INTRODUCTION
7. Last Battle (facing the villain)
3. Adventure (point of no return)
CRUX
6. Rebirth and Elixir
4. Trials (facing the evil)
5. Abyss (loss of hope)

Fig. 4.5: Simplified Hero's Journey diagram

From now on, these are going to be called Fundamental Steps (FS), as they lead the hero inside the path throughout the main storyline. Between each pair of , the system creates new smaller *steps* during the development of the story, which are going to be called Procedural Steps (PS). Each *step* brings the hero closer to the next FS.

$$FS_a \rightarrow (PS_1 \xrightarrow{*} PS_n) \rightarrow FS_b$$

where FS_n means Fundamental Step and PS_n means Procedural Step

Fig. 4.6: Narrative arc formula of *steps*

Here is where genetic algorithms comes into play. Building procedural situations for our story means that we need to generate new conflicts our characters will deal with, and derivative solutions to them. These conflicts and solutions must have sense within the constructed narrative.

Therefore, genetic algorithms require a set of criteria so that a proper solution can be selected from those which does not. We must bear in mind these criteria while planning solving:

- Previous, current and following context.
- Characters' characteristics.
- Available resources and abilities.

The first item is the most important, although it also is the most complex one. In this case, *steps* must be linked to each other to ensure consistency within the story, so there is needed a cause and effect criterion.

We define an effect as the consequence of the implemented solution to the current problem, which results to be the reason of the next problem. In other words, each procedural *step* PS_n produces an effect e_n, which is equal to the cause c_{n+1} of the following *step*. It serves as rule for the upcoming procedural *step* PS_{n+1} since it requires to continue the story from that state provided by cause c_{n+1}.

$$PS_n \rightarrow e_n = c_{n+1} \rightarrow PS_{n+1}$$

Fig. 4.7: Cause and effect equality between consecutive PS

In short, every effect set off by the current *step* will provide the actual context the following *step* needs, ensuring certain consistency. With this premise, the creation of each *step* would require the preceding node to extract its effects and, with them, determine the new cause, as we will see later.

If all the *steps* are generated from the previous *step*, the question of how to create the origin node arises. We have that the FS of the story compose the points of the structure of the hero's journey, whose purpose is to properly guide the hero in the course of her epic. These FS could be considered the backbone of the procedural story, giving some soundness to the plot. Unlike PS, they can not rely on a self-generated basis for the correct articulation of visceral moments and inflection points where the main plot, as well as the origin, is supported.

In addition, there are some narrative factors to take into account while developing a story to make it more appealing and intriguing such as suspense and rhythm. They are not meant to be essential elements to build a story, but they must be taken into account to consolidate the robustness of the work.

4.2.2. Static information.

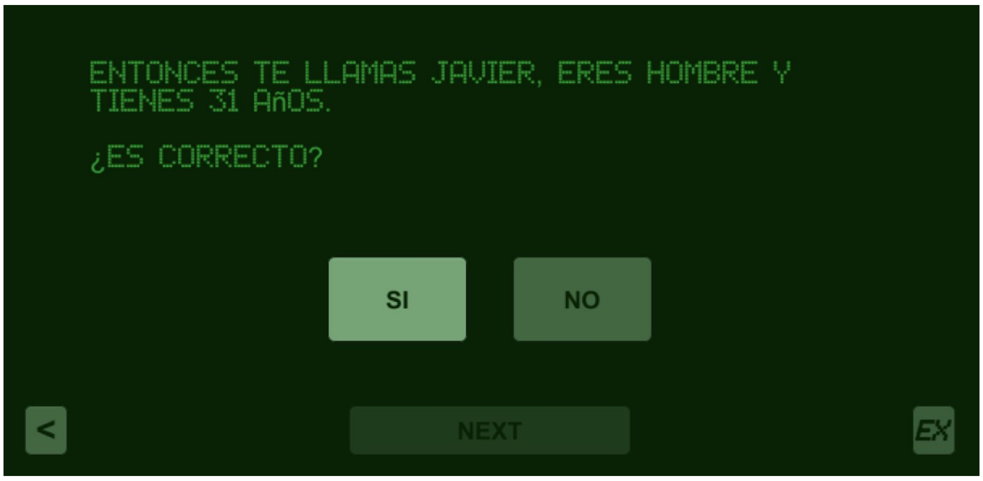
Before starting to build the story, the system ask the user for a succession of answers to certain questions. This information collected beforehand is called *static information* and it serves as the foundations for a totally personalized story, considering the interests, preferences and tastes of the user. All the elements that intervene in the narrative system should have access to that input data. There is also a randomized mode in which the *static data* is randomly calculated so that the user can skip those questions. Some examples of *static questions* can be:

What is your name?

Are you a boy, a girl or something else?

If you could travel in time, would you rather go to the past or to the future?

Do you like happy endings?



ENTONCES TE LLAMAS JAVIER, ERES HOMBRE Y TIENES 31 AñOS.

¿ES CORRECTO?

Fig. 4.8: Image showing an output example of the *static question* process

4.2.3. Story template.

Thanks to the narrative structure based on the hero's journey, which will be adapted according to the provided *static information*, it can be possible to develop a *story template*. This is an exclusive basic structure for the current story, tailored and adjusted to the own preferences of the user.

This template is the framework or skeleton ruling the whole process of story building. Within this template, it is designed the base of the story composed of the FS, narrative genre, characters and villains, places, initial base state, premise, main objective, possible threats that could appear during the route, estimated duration of the story, etc. Hence, this template will store objects containing other objects, values and references of different data structures.

The main set of narrative elements that will fulfil the completion of the auto-generated story are:

- Characters and associated resources.
- Stages.
- Mechanical actions.
- Steps.
- Genetic trees.
- Fundamental nodes graph.

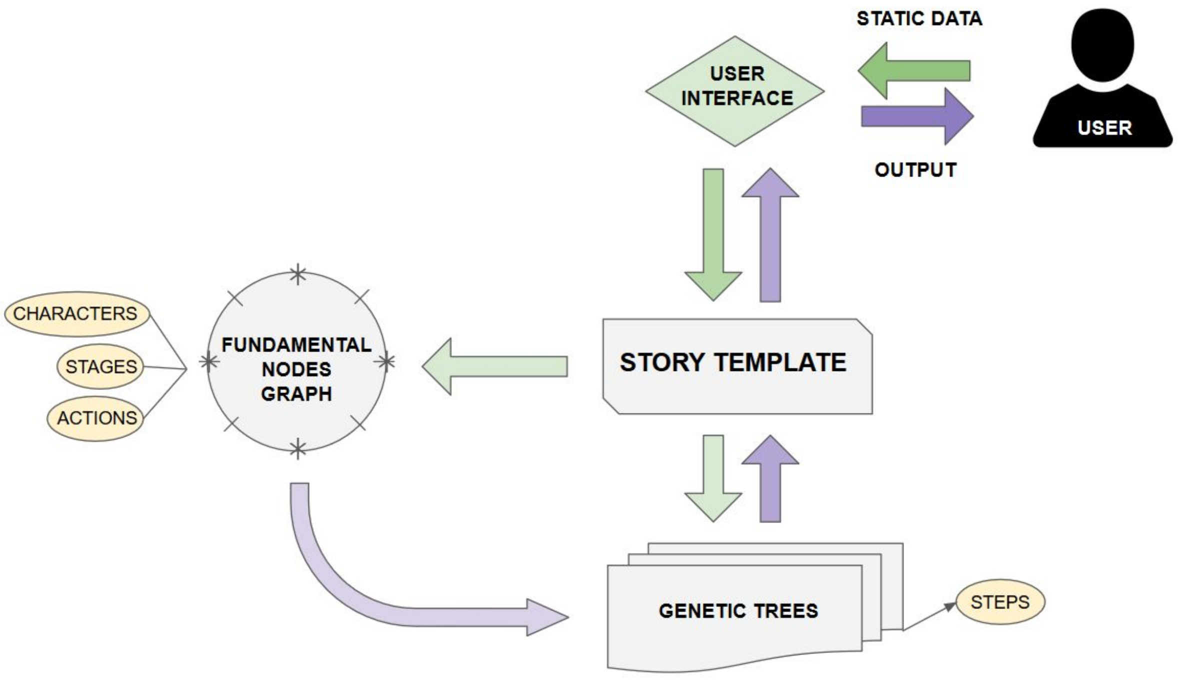


Fig. 4.9: Global system diagram

Firstly, the story template receives the input from the user, *a.k.a static data*, which are stored in a static script. Using this information, characters are created, along with the scenario and *actions* that will be use by the FS of the graph. Each pair of FS establish the initial and final point of a narrative arc, linked through PS using *genetic trees* (a modified version of the genetic algorithm). Finally, the results are transformed into a string to be shown on the screen.

The cornerstone of a story are the characters, stages and *mechanical actions*. These are the basic elements that have the greatest direct influence on the development and resolution of the *steps*. The union of these is what originates the situations and conflicts of the story, thus establishing the *narrative step*. The variety and complexity of narrative elements, whether essential or additional, magnify the space of possibilities for each *step*.

In the next section we will study in detail each narrative element of the system, showing the mission they perform in the story and their interactions with other elements. As done in section 4.1.2, the generated code will not be showed, since there are many different ways to obtain the same result and what is intended is to explain the operations behind this narrative system.

4.3. Algorithmic design.

4.3.1. Characters.

The characters are quite complex objects, since their actions guide the evolution of the whole narration. This is the genome data that hosts each character object:

- Identity
- Physical Statistics
- Nature

▪ Identity

This class is about the personal information of a character. It consists of these elements:

- The *Name* is a string defined either by the user itself, or randomly generated from a list of names according to her gender.

- *Age* is an integer value that can influence the personality. A young child will tend to be more reckless and active, while an older person will tend to be more calm and cautious.

- For the *Gender*, an enumerator was created with the components *Man*, *Woman* and *Queer*. Incidentally, the Queer gender has been added to represent all genres outside the binary spectrum and the provided names for this category can be either boy's or girl's.

The user might choose the identity of the hero or other characters so that to favor identification and immersion.

▪ Physical Statistics

As it is proper in many video games, especially in the role-playing genre, characters have numerical variables that will represent several qualities. These statistics are also used to calculate the fitness of a given solution, that addresses to solve any specific adversity.

These are: *Life*, *Strength*, *Protection*, *Intelligence*, *Courage*, *Speed* and *Luck.*

Statistical values go from zero to 999. They are generated quasi-randomly, favoring certain answers given in the *static questions* by the user.

▪ Nature

Characters must follow the pattern of their own behaviors, which influence their decision making. Their personalities determine the behavior of the individual in different circumstances, which serve to create potential solutions.

Nature is constructed throughout a particular value for each one of next criteria:

➢ *Social*: Address communicative acts and the relationship with other individuals.

➢ *Activity*: Willingness for performing actions that require some physical effort.

➢ *Inspection*: Willingness for performing actions that require some mental effort.

➢ *Confrontation*: Address the contextual framework of a fight or confrontation.

For each previous criteria there is a group of personalities related to them, from which one must be selected. Personalities represent an independent behavior and influence certain actions of a character. For instance, a character with a brave personality will tend to face the enemy during a confrontation, whereas another with a cowardly personality would prefer to run away or hide.

Personalities have a fundamental role in determining the fitness of potential actions a character might unleash. Every personality is attached to a circumstance and a personality value, contained between zero and five, both included. Once we have it, it is intended to extract a personality corresponding as it is showed in the following matrix:

	Social	Activity	Inspection	Confrontation
0	Bad-Tempered	Lazy	Reckless	Coward
1	Apathetic	Calm	Naïve	Pacific
2	Reserved	Carefree	Scatterbrained	Prudent
3	Shy	Fickle	Curious	Confident
4	Tactful	Active	Responsible	Brave
5	Romantic	Hasty	Thoughtful	Aggressive

Fig. 4.10: Personality matrix

To sum up, the class *Nature* provides four different personalities, one for each circumstance, which defines the character's behavior.

To end with the *Character* class, it also includes a stack structure which stores the *actions* that the character has already executed. So, *actions* that have been repeated are less likely to be chosen again, reducing its fitness.

In a possible future version, we could also add some extra features to make our characters narratively more compelling. These are some additions that can be made:

- A relationship matrix where the degree of affectivity between two characters is measured.
- Inventory and carrying resources list, adding items that improve the physical statistics of the bearer character.
- A death rate, which increases or decreases depending on the danger of the situations to which the subject is facing and would add drama, especially in secondary characters.

4.3.2. Stages.

A *stage* is the location where the action takes place. Unlike the characters, *stages* are simpler and they are already created in advance. What does change in every story is the selection of the environment through *static data*, and a random set of places attached to each environment.

Stages do not depend exclusively on a space; they are also directly subjected to the period of time when the events take place. Its mission is to give a spatio-temporal context and ambience to the narrative. Depending on the era, various adjustments can be made to maintain the narrative context with the rest of the elements.

The current project includes twelve different environments:

Cavern, Amazonian, Ancient Egypt, Roman Empire, Viking, Feudalism,
Shogun, Far West, Caribbean, Urban, Cyberpunk, Galactic [2]

Each environment is attached to an array of strings whose signifier acts as places belonged to it. To give an example, the *Caribbean* environment includes:

"Island", "Treasure Lair", "Pirate Ship", "Jungle", "Harbour", "Shore", "Pier",
"Beach", "Waterfall", "Galleon", "Sea", "Tavern", "Cave", "Cliff", "Volcano" [3]

Nevertheless, each environment contains more places than necessary for the development of the story. From this array of places, only a few of them will be selected, and a map including the chosen *stages* will be created.

This *map* is the responsible class for obtaining the set of *stages* that will appear in the story. First, it accesses the array of places in the environment chosen by the user, and randomly selects the *stages* that will belong to the story *map*. Then, the *map* generates a graph where each *stage* corresponds to a node.

The reason our scenario is built under the structure of a graph is that it makes convenient traveling from one place to another within the story. Also, these nodes have a boolean variable to indicate whether the node itself is blocked, so the hero can not go there at that specific moment.

[2] It should be noted that these environments have been created under my own consideration and it is totally fine to delete them or add any other interesting environment to the list.

[3] These places are not meant to be definitive; any place could be introduced instead, providing that it is related to the environment that place is attached to.

4.3.3. Mechanical actions.

Actions are both the basis of the triggering and the resolution of conflicts. They are formed by:

- The name of the *action* such as "attack", "fly" or “talk”.
- The situational context the *action* is held.
- The degree of affinity with a concrete Personality, from zero to five.
- A value which estimates the degree of rhythm such *action* provides.
- A boolean that indicates whether the verb is intransitive, as we will see later.
- A concatenation list of subsequent candidate *actions*.

The concatenation list is the most relevant element, though, as it stores all the possible *actions* capable of being executed in response to the *action* itself. This list is used by the *narrative steps* to create new situations caused by such *action*.

What is expected to achieve with *mechanical actions* is to cover the widest spectrum of possible situations that can take place in the story. The greater the number of defined *actions*, the richer the succession of self-generated events can be. In the current project there are written more than one hundred and twenty different *actions* in total.

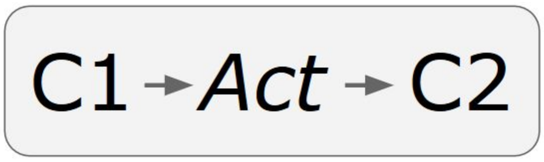
4.3.4. Steps.

The computer itself must be able to create a situation centered on a conflict and look for a solution. A conflict can be considered any circumstance that hinders a goal. Anything can be a conflict: from leaving the keys inside the car, to running into a wall in the middle of a chase. Each of these situations requires unique solutions adapted to the circumstances, since breaking the glass of the car window will not help you overcome the wall from the previous example.

We understand a *step* as the minimal narrative unit that describes a conflict or circumstance. As we have seen in the last paragraph, several situations can be extremely different from each other. Thus, to implement this narrative unit, it has been necessary to abstract what elements a conflict requires to be considered as such.

After much pondering, I came to find a common nexus all possible circumstances share. I found the answer in the syntactic plane, where any circumstance, event or conflict can be divided into three components: an active subject, a passive subject and an *action*.

Both subjects can be any character of the story; whereas the *action* encompass the predicate, which is carried out by the active subject (*a.k.a. executor*) and received by the passive subject (*a.k.a. recipient*). That leave us the following structure.



Where C1 means character 1 (executor), C2 means character 2 (recipient) and Act stands for the action executed by C1 and received by C2

Fig. 4.11: Graphical description of a *step*

This simple structure allows us to represent any situation using two characters, the *executor* and the *recipient*, as well as the *action* that the first performs on the second. But it does not just serve to represent the current situation. If we alter the order of the characters (the *executor* becomes the new *recipient* and the *recipient* becomes the new *executor*) and add a new consequent *action*, we would obtain the next *step*. This process can be seen in the figure below.

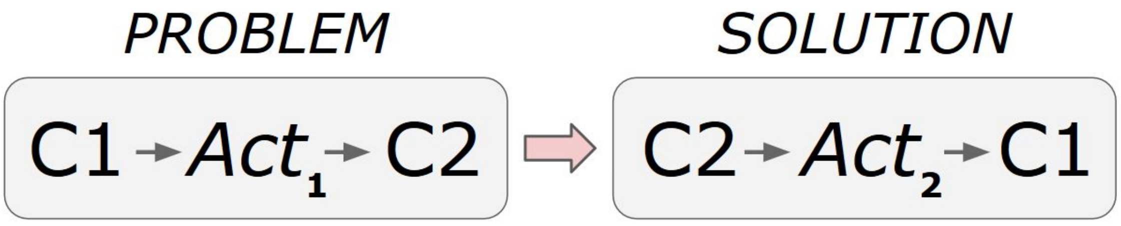


Fig. 4.12: Main structure of consecutive *steps*

We would have created a succession of events, where the passive subject, who receives the *action* that originates the problem, will acquire an active mode in the following *step*, proposing a new *action* as response. With this structure, it is possible to generate with ease new solutions to our conflicts. The negative point of this way to structure *steps* is that it makes it difficult to involve more characters besides these two.

Now we are faced with the question of deciding which *action* to precede the current one. The choice of the *action* is key to guarantee certain coherence between two consecutive *steps*. We have seen that all *actions* have a concatenation list, where all the candidate *actions* to be its consequence are stored. Among all the *actions* in the concatenation list, we must select the one that best suits the actor that intends to execute it.

Therefore, we calculate the fitness value of a each candidate *action* from the comparison of the character nature with the corresponding personality of the executed *action*. The closer they are in the personality matrix, the higher will the fitness be; in case both personalities are the same, the obtained fitness will be maximum. Fitness can also be modified if the character has already repeated that same *action*, giving priority to those that have not been held yet. Once the fitness of all the *actions* is known, the *action* of the *step* will be chosen by biased roulette. The chosen *action* can be eventually replaced in case of mutation. This fitness is relevant in the next section.

There are two ways to build a *step*. The first one receives as argument two characters and the own *action* of the *executor*. This type of *step* is meant to create the FS, since their *actions* are more enclosed due to the hero's journey structure. The other kind of *step* receives the previous *step*, from which the *action* is extracted. This *step* is going to be used for the PS, by requiring the full context of the previous situation to maintain its coherence.

Every *step* is able to create a new following *step*, by exchanging the roles of the actors and choosing a consequent *action*. *Steps* that create a new *step* are going to be called parents, whereas the new ones are considered their children. Links between both parent and child are necessary for the formation of the *genetic tree*.

It should be pointed out that there are cases where the *actions* are reflexive, that is, whose effect is received by the *executor* itself. Some reflexive *actions* are: falling asleep, worrying, getting angry, hiding, and so on. The *action* object indicates with a boolean whether the *action* is reflexive or not.

SUSANA MIRA A ANACLETO.

Fig. 4.13: Visualization of a *step* as output of the narrative system

4.3.5. Genetic trees.

We know that FS are the backbone of the story. The initial and final points of each narrative arc constitute a pair of consecutive FS, extracted from the fundamental nodes graph. However, we are going to need a succession of events that fill in the gap that exists between each pair of key points, tieing them in a narratively coherent way.

The *genetic tree* is a data structure based on the genetic algorithm, to create a ramified population of *step* individuals. Here, *steps* are the nodes of the *genetic tree* structure. These nodes depart from a given initial *step* and generate new descendants coherently towards the situation posed by the final *step*. After the ramification of the *genetic tree* is completed, it will be obtained the best path of *steps*, which closes the narrative arc between two consecutive FS.

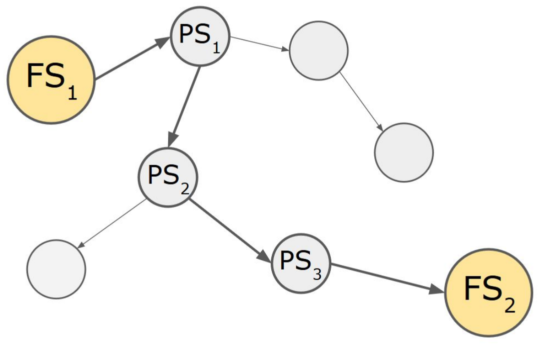


Fig. 4.14: Simplified representation of a *genetic tree*

FS begin disconnected in the great narrative space of possibilities and it will be the mission of the *genetic tree* to unite them.

During the creation of the *genetic tree*, descendants will be produced in each *step* and will evaluate if they are compatible with the final *step*.

To ensure consistency between *steps*, are used two different types of fitness: the previous fitness (*prev-fitness* henceforth) and the posterior fitness (*post-fitness* henceforth). The first one indicates how well the previous context is respected, whereas the second indicates whether the current *step* is compatible with the final FS.

It should be recalled that the process of creating *steps* ensures a positive compatibility between *step* father and child. All candidate *actions* from the concatenation list are valid, but some of them adapt better than others to the new situation, according to the personality of the character. The *prev-fitness* serves to choose a candidate *action* by biased roulette method, which gives priority to the best *actions*. Its value is obtained by evaluating the compatibility of the *action* with the personality of the *executor* character.

During the development of the *genetic tree*, it is essential to find the last-procedural-step (LPS henceforth), which will become the father of the FS that closes the narrative arc. Candidate nodes to become LPS fulfill the following two characteristics: they belong to the evaluating generation of the *genetic tree* and must be compatible with the final node.

The evaluating generation refers to one of the last two node-generations of the tree, according to the order of *executor* and *recipient*. *Steps* are created in a way that the order of characters is required to be opposite between parent and child *steps*. Therefore, LPS can not have the same order of characters than the last node. The evaluating generation of the *genetic tree* is, then, the last generation of nodes where the order of characters is different from the final *step*.

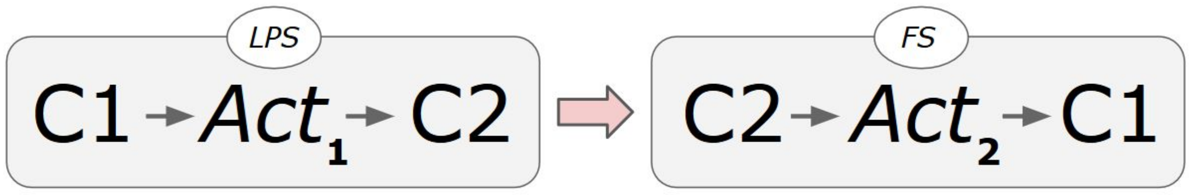


Fig. 4.15: Characters of two successive *steps* can not be placed in the same order

Moreover, it must be ensured that the *action* of LPS is capable of triggering the *action* of the final node. Otherwise, they will not be compatible. In other words, two consecutive *steps* are compatible when their subjects are placed in opposite order and the *action* of the second *step* is included in the concatenation list of the first.

The *post-fitness* starts with a negative initial value like -1 to indicate incompatibility. In case of being a *step* valid, its *post-fitness* value is updated in the same way as the *prev-fitness* (see chapter 4.3.4).

The only elements that the *genetic tree* takes for arguments of the recursive method are the current node, the final node and the current generation number (which is used to determine the number of resulting PS). The tree starts from the root node, which is the first FS. Having these concepts clear, we will proceed with the explanation of the recursive development of the *genetic tree*.

```
Recursive development of the GeneticTree (current step, final step, generation):

    · Evaluate compatibility of LPS with final step and update post-fitness.

    · The exit criteria are the following:
        -  To reach the generation number limit.
        -  The prev-fitness does not reach the minimum value.

    · If it is not affected by the exit criteria, then:
        -  Generate child.
        -  Recursive call where the child is now the current node.
        -  Save the child with best prev-fitness and post-fitness.
        -  Copy post-fitness of best child to current node.
        -  Eliminate unworthy children.
```

Fig. 4.16: Evolutionary process of the *genetic tree* algorithm in pseudo-code

The first part of the code consists in checking if the current node belongs to the evaluating generation of the tree; if that is the case, the *post-fitness* of the current *step* is updated. As it was previously stated, the *post-fitness* value will be positive if the current *step* is compatible with the final *step*.

Next, exit criteria are checked. We understand exit criteria as those conditions to stop the growth of a branch of the tree. Here, the most important exit criterion is to reach the established generation limit. Without this condition, branches of the tree could grow endlessly. If the generation number has reached the maximum value allowed, the current path stops and moves on to the next one. The other criterion establishes a minimum required value of *prev-fitness* as a filter. If the previous context (between the parent *step* and the child) is not good enough, the evolution in that branch stops.

In case the current node has not been affected by these conditions, it must create new descendant *steps*. There is a recursive call for each new child created, being them the following current nodes. The number of the node-generation that is given as an argument is one unit greater than that of the father.

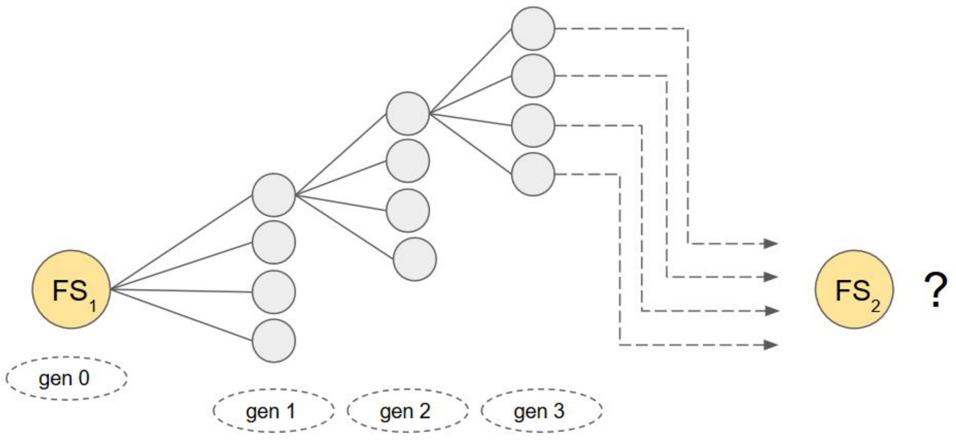


Fig. 4.17: Draft where the third generation of PS is the evaluating generation

Once the recursive call has ended, the child node would have updated its *post-fitness* value, determining how good the best route from that child is. This process is repeated for the rest of the children of the current node. The parent *step* chooses the descendant with better *post-fitness* and *prev-fitness* values. After this, it removes the unworthy ones, that is, those *steps* whose fitness are lower. In this way, the parent will only be linked to the best child and vice versa. Then, it copies the *post-fitness* value of the best child, which determines the quality of the subsequent route.

The *genetic tree* shrinks as the remaining unworthy *steps* are being eliminated. The fact that each parent keeps the child whose fitness is better, guarantees that the best possible path to the final node prevails.

Before invoking the recursive calls of the children *steps*, the *action* of the father must be stored in a list (which should be a stack for efficiency means). Consequently, descendant *steps* will be able to calculate their fitness depending on how many times an *action* has been repeated. Last stored *action* in the list must be deleted before finishing every recursive call, so as to ensure that those repeated *actions* have only been performed on that particular path.

Once the *genetic tree* has finished creating individuals recursively, the whole tree population shrinks, removing all the *steps* except for the best of each branch. In this way, when you get back to the starting FS, nodes that have survived make up the main trunk of the *genetic tree*, which culminates with the addition of the final FS as the child of the LPS. The length of the final tree is given by the established generation limit, as there is one PS per generation. The final structure of the tree would resemble a double-linked list, since each PS has access to its parent and child.

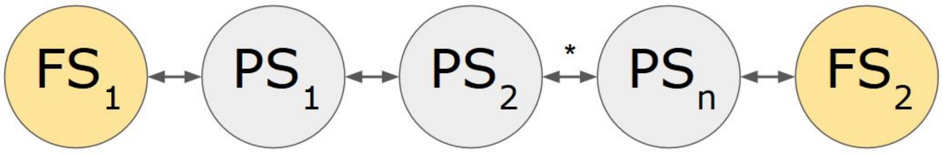


Fig. 4.18: Representation of the final *genetic tree*, where *n* is the generation limit

The objective of the *genetic tree* is to gather a coherent succession of events among a myriad of possible situations. The greater the generation limit number and the number of children created per node, the more solutions there will be to choose from. However, the total amount of *steps* created by the tree (*m*) is a summatory of exponential values, having an impact on the computational cost. It can be calculated with the sumatory of children per node raised to the power of the current generation, as it shows the figure below:

$$m = \sum_{i=1}^{gen\ max} children^{\,i} = children^{\,gen\ 1} + children^{\,gen\ 2} + \ldots + children^{\,gen\ n}$$

Fig. 4.19: Total number of nodes generated during the *genetic tree* algorithm, where *n* is the generation limit and *children* refers to the number of *steps* each node creates

In addition, we need to add the cost of creating each single *step*, which has a quasi-linear order $O(n\ log\ n)$. *Steps* are reproduced with an exponential growth, multiplying its population in each generation. Nonetheless, the efficiency issue only arises when we use high values. It is convenient to set some equilibrium between an acceptable number of solutions and a computational cost that does not compromise the efficiency of the system.

For instance, it is given a system that creates four children per node. In order to generate four PS in the resulting tree, the cost will be *340 * n log n* in the worst scenario. However, to generate eight PS in the same system, the worst-case cost will rise up to *87.380 * n log n*.

In case of wanting an extensive narrative arc, costs can end up being unfeasible. That is why I devised a way to magnify the *genetic tree* length without exponentiating its cost. The idea is based on divide and conquer rule and the process is the following:

Once a *genetic tree* of small magnitude is created, a pair of nodes from the resulting tree is chosen; it is unimportant which pair of nodes we get as long as these *steps* are consecutive. With them, we recursively create a genetic subtree where these nodes act as initial and final points. This subtree is generated in the same as the original tree, being able to modify some parameters such as the generation limit, in order to obtain a different length.

Finally, we update the unions of the new nodes with the main *genetic tree*. Thus, the pair of consecutive nodes that was previously selected in the original tree is now detached from each other. After this process, both *genetic trees* are merged into a larger one. It is possible to add as many subtrees as necessary, until obtaining the desired length for our structure.

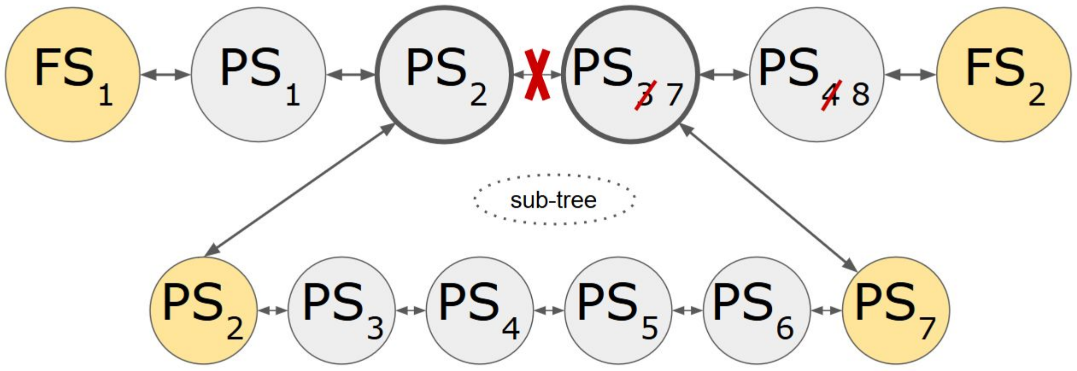


Fig. 4.20: Graphic representation of a genetic subtree and the merging process

In summary, this method allows us to duplicate the base length of the *genetic tree* in a more efficient way. Returning to the previous example, we had that for eight PS, the temporary cost was *87.380 * n log n*. Applying the broadening method, we will obtain an order of *680 * n log n* for eight *steps* instead.

So far, we have only treated the succession of events generated in a single *genetic tree*. A single tree does not cover a complete story, but the corresponding narrative arc between two keypoints. The presented structure of the hero's journey consists of eight FS, so we would need to create a *genetic tree* for each pair.

Accordingly, to join several narrative arcs, it should be connected the *genetic trees* by their opposite ends. In other words, the last node of a tree would become the parent of the next tree first node, in the same way that PS do in the evolutionary process.

4.3.6. Fundamental step graph.

The Fundamental Step Graph or FSG is a directed cyclic graph that constitutes the implementation of the hero's journey. Nodes from this graph correspond to the FS that *genetic trees* will be given in order to create the story.

The data structure chosen is a graph because it corresponds well with the structure of the hero's journey. The graph is directed for obvious reasons given by the narrative requirement of following a concrete unidirectional linear path. Likewise, it could be a cyclic graph because the story will end up, in most cases, with the return to normality, the stability of the base stage.

The main idea of the plot has to be embedded within the nodes of the graph, which should change according to the provided *static data*. Relevant narrative elements such as the *map* and main characters (hero and villain) are key to develop a specific plot and therefore, create the most suitable FSG.

In addition, nodes can be subjected to a level of rhythm. Every FS constitutes an inflection point in the rhythm chart, so that PS created by the *genetic tree* should keep an ascending or descending rhythm. Then, to keep up with the rhythm chart, mechanics with an appropriate degree of *action* are more likely to be chosen.

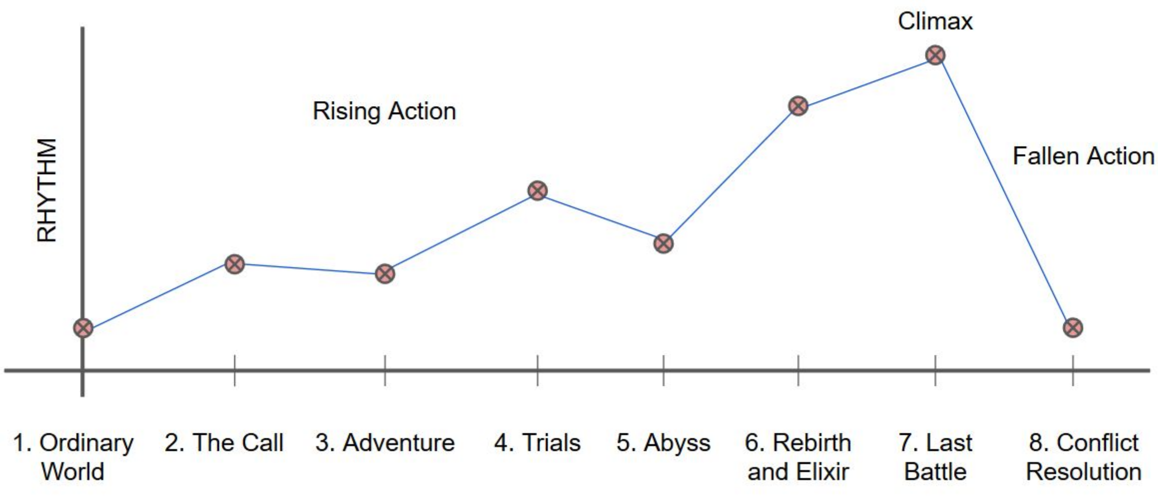


Fig. 4.21: Rhythm Chart showing the inflection points on each FS

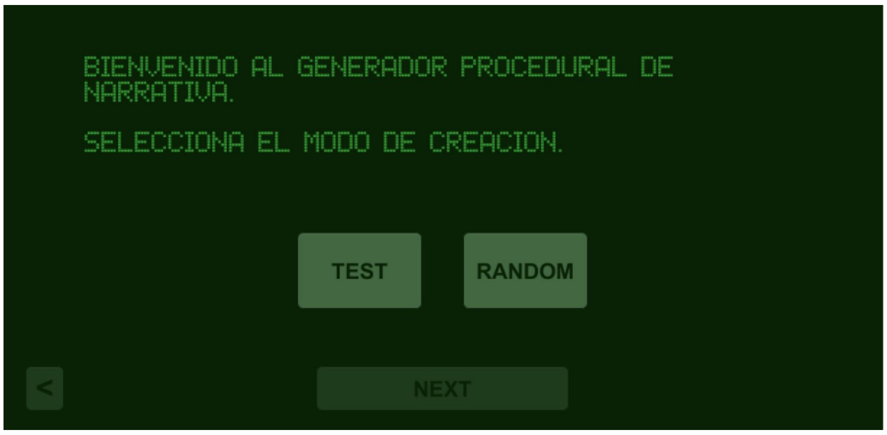
Each pair of FS that conform the graph will mark the initial and final points of each narrative arc and will be used to create the intermediate sequence of PS through *genetic trees*. Once done this, it is possible to guide the protagonists from the initial state towards a final objective.

Nevertheless, there has to be created a great amount of prefabricated FS in order to develop a concrete plot. The corresponding pair of FS attached to a *genetic tree* directly depends on the FS selected in the previous narrative arcs. So, for each key point in the hero's journey structure, there has to be designed a set of specific FS that cover all the possible plots so far.

A single whole story requires a minimum of sixteen different FS (there are eight key points with a pair of FS each). However, the average of FS used to introduce a situation or conflict in each key point is usually higher. The larger the number of FS designed, the more ramified the story will be.

4.4. Results.

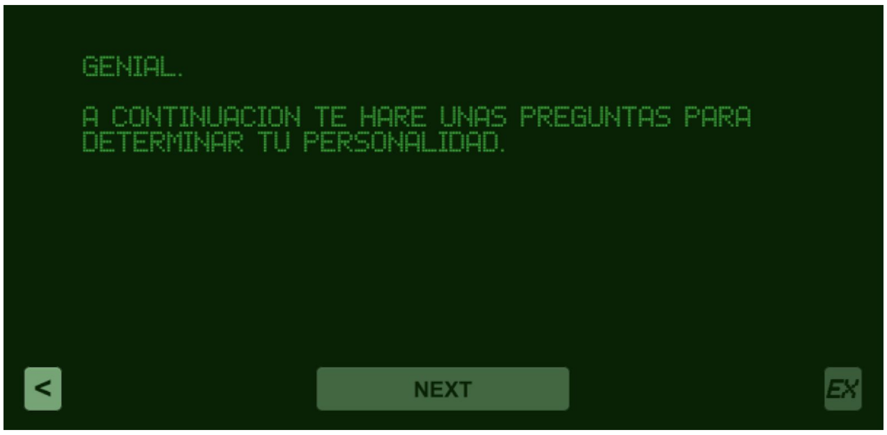
The implementation results of the demonstration have been satisfactory. Firstly, the project allows the user to choose between a completely randomized story (*random* button) and a customized story where the user is asked the *static questions* (*test* button).


BIENVENIDO AL GENERADOR PROCEDURAL DE NARRATIVA.

SELECCIONA EL MODO DE CREACION.

Fig. 4.22: Mode selection menu

By clicking the *test* button, the project proceeds to formulate fifteen different questions, whose response build up some narrative elements such as the main character and the stages for the upcoming story.


GENIAL.

A CONTINUACION TE HARE UNAS PREGUNTAS PARA DETERMINAR TU PERSONALIDAD.

Fig. 4.23: Image extracted from the *static question* mode

In case of selecting the randomized story mode, the user will not be shown these questions, as the *static data* is randomly generated. This mode was initially configured to speed up debugging, but I found it an interesting option for the future user as well.

Once the provided information is stored, the system is capable of creating a simple succession of cohesive *steps*, employing the limited lexical and grammatical resources available. The *story template* generates eight genetic trees, each of them starting from two situations given by the FSG. The reason for having eight *genetic trees* is subject to the predefined key points of the hero's journey structure.

When a new story begins, the most basic elements are introduced first to give us some context. It serves as a presentation of the main characters and the environment set, making way for the actual plot.

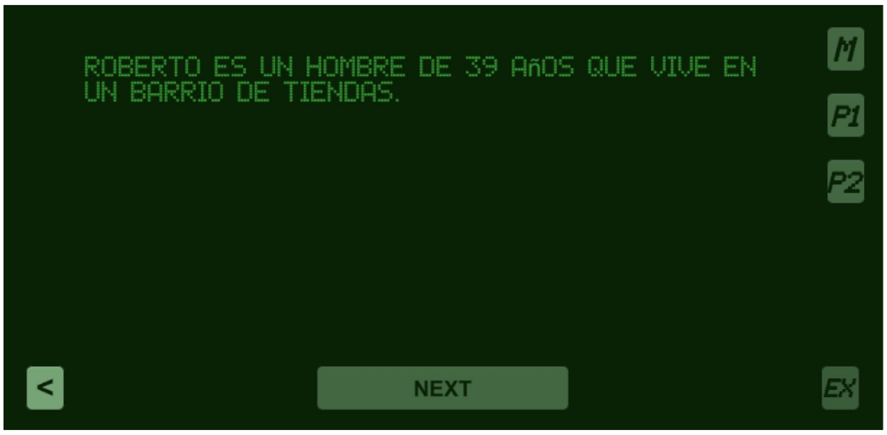

ROBERTO ES UN HOMBRE DE 39 AñOS QUE VIVE EN
UN BARRIO DE TIENDAS.

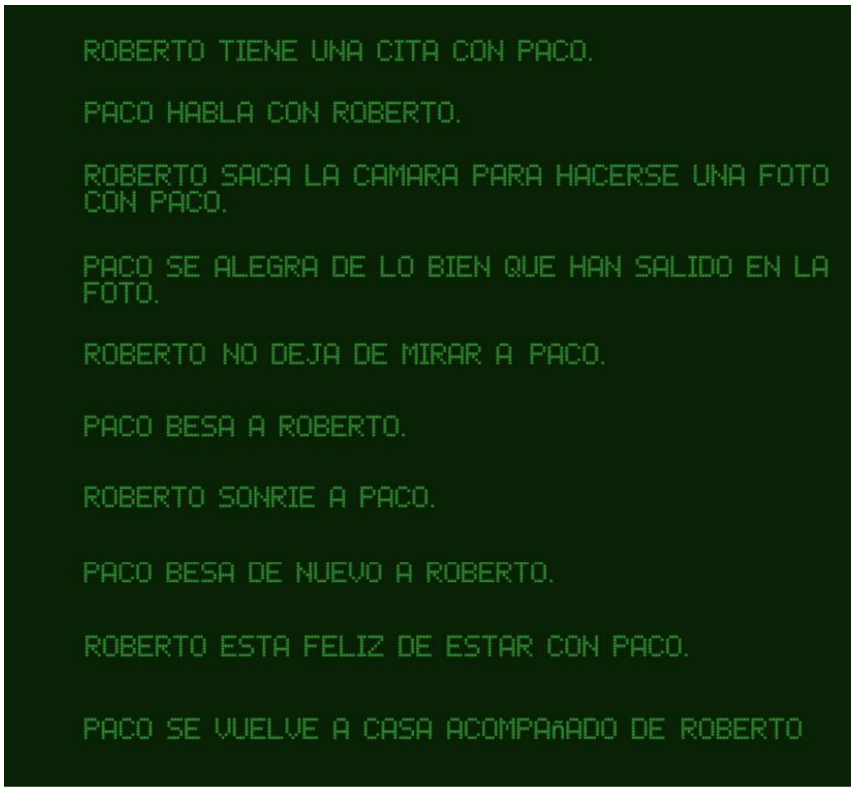
Fig. 4.24: Example of an output text that introduces a story

After this presentation, the resulting narration of *genetic trees* is shown. It is splitted up in a set of concatenated sentences, each of them being a *narrative step*.

The first and last sentences of each *genetic tree* belong to the FS, whose mechanics are chosen from the list of *actions*, that can either give rise to a new plot or come to an end, respectively. Sentences in-between are PS and their *actions* depend on prior *steps* and the nature of characters. PS make the characters go through different situations even if the internal data keeps unchanged.

As we can see in the Figure 4.25, the initial FS is "Roberto tiene una cita con Paco" and the final FS that concludes the narration is "Roberto vuelve a casa acompañado de Paco". The remaining sentences are PS. However, the Figure 4.26 starts under the same exact conditions than Figure 4.25 and it shows a different story.

Even though two *genetic trees* share both FS and the provided *static data* is the same, chances that all their PS coincide are low. With this, it is proven that the narrative generated using *genetic trees* is non-deterministic.



ROBERTO TIENE UNA CITA CON PACO.

PACO HABLA CON ROBERTO.

ROBERTO SACA LA CAMARA PARA HACERSE UNA FOTO CON PACO.

PACO SE ALEGRA DE LO BIEN QUE HAN SALIDO EN LA FOTO.

ROBERTO NO DEJA DE MIRAR A PACO.

PACO BESA A ROBERTO.

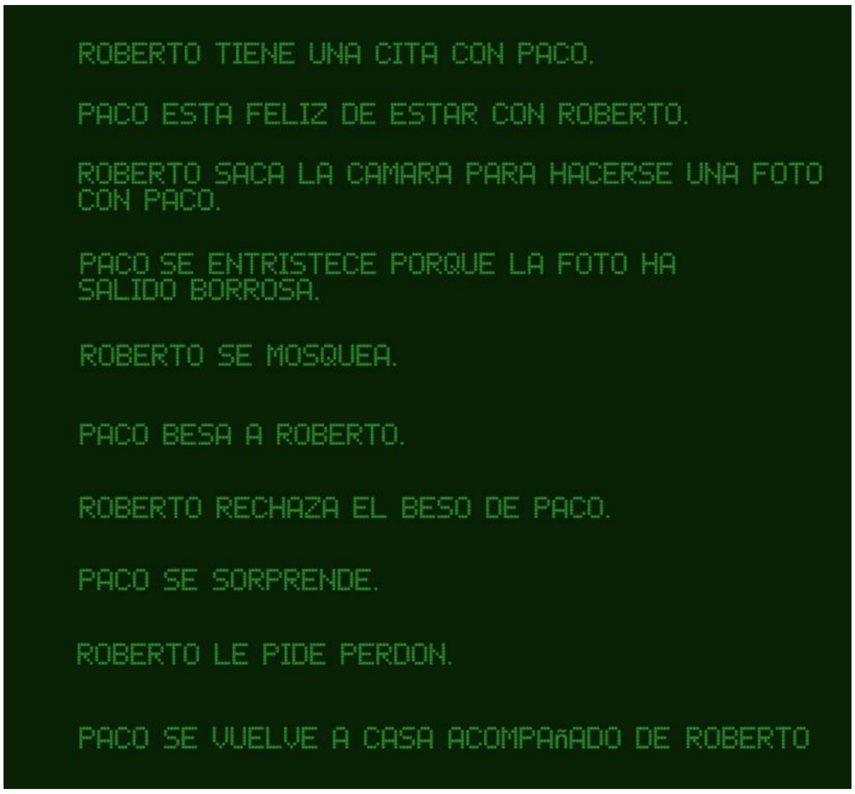
ROBERTO SONRIE A PACO.

PACO BESA DE NUEVO A ROBERTO.

ROBERTO ESTA FELIZ DE ESTAR CON PACO.

PACO SE VUELVE A CASA ACOMPAñADO DE ROBERTO

Fig. 4.25: Screen captures of the successive emerging texts of a story



ROBERTO TIENE UNA CITA CON PACO.

PACO ESTA FELIZ DE ESTAR CON ROBERTO.

ROBERTO SACA LA CAMARA PARA HACERSE UNA FOTO CON PACO.

PACO SE ENTRISTECE PORQUE LA FOTO HA SALIDO BORROSA.

ROBERTO SE MOSQUEA.

PACO BESA A ROBERTO.

ROBERTO RECHAZA EL BESO DE PACO.

PACO SE SORPRENDE.

ROBERTO LE PIDE PERDON.

PACO SE VUELVE A CASA ACOMPAñADO DE ROBERTO

Fig. 4.26: Different story with the same input patterns of the previous figure

The previous figures constitute each a single *genetic tree* narration. Numerous *genetic trees* can be easily used to make up a wider and more complex story. In this regard, every *genetic tree* should comprehend a narrative arc according to the hero's journey structure.

Nevertheless, the current project has some limitations:

1. Only two characters can appear in each narrative arc.
2. It is not possible to repeat the *executor* character in two consecutive *steps*.
3. FS must be previously written to develop a concrete plot and it is needed a considerable amount of them to build a complete narrative-generator.
4. Even though the narrative generated by the genetic tree algorithm is non-deterministic, the selection of FS is not.

These limitations are present in the current demo[4] of this FDP. However, it would be possible to find solutions if more time is spent on the project.

Given the limitation number three, the demo in its current state is not able to build a story with the complete hero's journey structure yet. Instead, independent narrative arcs are created. Narrations held in the program constitute the corresponding narrative arc to "The Call", according to the hero's journey structure. Therefore, it creates eight independent, non-deterministic narrative arcs.

The current demo contains twenty different FS; however, those meant to initialize the story only contemplate the narrative arc of "The Call" and can not be reusable for other arcs. It also includes one hundred and twenty-one different *mechanical actions* in total and twelve environments, with more than fifteen *stages* each. It is estimated that, to create an acceptable narrative-generator with eight narrative arcs, more than eighty FS and over three hundred *mechanical actions* would be required as minimum.

[4] This demo can be found in this link: https://github.com/spaceraptor/GeneticAlgorithmFDP

Conclusions

This FDP has been an exhausting but tremendously rewarding work, and I am very proud of having obtained a functional result, because there were many occasions when I thought that the challenge was superior to me.

The genetic algorithms have been adapted to the needs of a *narrative step* to configure a tree structure capable of linking two concrete points of the space of possibilities, coherently filling the intermediate spaces with procedurally created situations. In this way, it becomes possible to forge a complete non-deterministic narrative with a combination of FS.

It is simple to add lexical and grammatical resources, as well as other variables that enrich the narrative experience and increase the space of possibilities. Of course, there is no need to modify the part of the code that is already written when doing these additions.

Hence, this work can be a great tool to create or enhance narrative video games, especially those based on emergent narratives; to name but a few, The Sims and Animal Crossing franchises.

Furthermore, the applications go beyond the simple fact of building stories. For instance, it can be used to improve interactions with NPC, making them capable of generating more dynamic and natural dialogs. Developing a plug-in that facilitates the use of this system in game engines could also be an interesting option.

Bibliography

[1] *Inteligencia Artificial para desarrolladores - Conceptos e implementación en C#*. Virginie Mathivet. ENI. 2018.

[2] *The Hero with a Thousand Faces*. Joseph Campbell. Collected Works. 1949.

[3] *Artificial Intelligence for Games*. Ian Millington and John Funge. Morgan Kaufmann. 2009.

[4] *Story and Discourse: A Bipartite Model of Narrative Generation in Virtual Worlds*. R. Michael Young. North Carolina State University. 2007.

[5] *Understanding Video Games: The Essential Introduction*. Simon Egenfeldt-Nielsen, Jonas Heide Smith and Susana Pajares Tosca. Routledge. 2016.

[6] *A morphological approach to interactive storytelling*. Dieter Grabson and Norbert Braun. CAST01. 2001.

[7] *Artificial Intelligence and Literary Creativity: Inside the Mind of BRUTUS, a Storytelling Machine*. Selmer Bringsjord and David A. Ferrucci. Rensselaer Polytechnic Institute and IBM T.J. Watson Research Center. 1999.

[8] *Interacting with Virtual Characters in Interactive Storytelling*. Marc Cavazza, Fred Charles and Steven J. Mead. University of Teesside. 2002.

[9] *Macro Structure and Basic Methods in the Integrated Narrative Generation System By Introducing Narratological Knowledge*. Taisuke Akimoto and Takashi Ogata. Iwate Prefectural University. 2012.

[10] *Narrative Models: Narratology Meets Artificial Intelligence.* Pablo Gervás, Birte Lönneker-Rodman, Jan Christoph Meister and Federico Peinado. Universidad Complutense de Madrid, University of Hamburg and Ludwig-Maximilians-University. 2006.

[11] *Search-Based Procedural Content Generation: A Taxonomy and Survey.* Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. IEEE. 2011.

[12] *The Nature of Code: Simulating Natural Systems with Processing.* Daniel Shiffman. 2012.

Glossary

▪ Narrative

Communicative process involved in the act of telling or writing, defined as a succession of events, its timeline and verbal or textual representation. Story and discourse are fundamental elements of the narrative.

▪ Story

The story encompasses the content itself. Account of events or experiences in a certain chronological order, whether true or fictional, which describes a happening, a problem that needs to be solved or a pursued goal.

▪ Discourse

Discourse tells the spectators how a story is being communicated by the discursive agents through written syntax and visual arrangements. To give an example, discourse in novels is shown as text in the form of dialogs and descriptions, whereas in films and videogames, it also involves everything within a scene like the characters, background and even the movement and position of the cameras.

▪ Non-deterministic narrative

Interactive narrative structure based on auto-generated stories, whose plot is expected to be unique every time the user (or the receptor of that narration) tries to repeat the same input patterns. The succession of events has not been established before the beginning of the narration.

▪ Procedural

In programming, it refers to automatic-generated elements, following some kind of internal and common structure.

▪ Artificial Intelligence

Artificial intelligence is used to describe the ability of machines to mimic cognitive functions that humans associate with other human minds, such as learning and problem solving.

▪ Genetic Algorithms

Genetic algorithms are a type of artificial intelligence inspired by diverse researches carried out in the field of biology, which try to simulate the process of evolution of species to solve problems, establishing a relationship between individuals and potential solutions.

From a population of individuals, each one with its own genetic peculiarities and characteristics, they are evaluated according to the quality of the solution they offer to a given problem, generating a value called fitness. The best individuals, that is, those with greater fitness, will have greater probabilities of reproducing with other individuals, creating descendants from the genetic information crossover from their parents and thus obtain a new improved population. During the breeding process, the algorithm also carries out mutations that can randomly change the genetic data of an individual.

▪ Problem

Set of facts or circumstances that hinder the achievement of some purpose.