NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

The Narralive Unity plug-in: Bridging the gap between intuitive branching narrative design and advanced visual novel development

Dimitra S. Kousta

Supervisors: **Dr Maria Roussou,** Associate Professor
Dr Akrivi Katifori, Christos Lougiakis

ATHENS

JULY 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Το Narralive Unity plug-in: Γεφυρώνοντας το κενό ανάμεσα στον σχεδιασμό αφηγήσεων με διακλάδωση και την προηγμένη ανάπτυξη Visual Novels

Δήμητρα Σ. Κούστα

Επιβλέποντες: **Δρ. Μαρία Ρούσσου,** Αναπληρώτρια Καθηγήτρια
Δρ. Ακριβή Κατηφόρη, Χρήστος Λουγιάκης

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2023

BSc THESIS

The Narralive Unity plug-in: Bridging the gap between intuitive branching narrative design and advanced visual novel development

Dimitra S. Kousta

S.N.: 1115201600263

SUPERVISORS: **Dr Maria Roussou,** Associate Professor
Dr Akrivi Katifori, Christos Lougiakis

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Το Narralive Unity plug-in: Γεφυρώνοντας το κενό ανάμεσα στον σχεδιασμό αφηγήσεων με διακλάδωση και την προηγμένη ανάπτυξη Visual Novels

Δήμητρα Σ. Κούστα
Α.Μ.: 1115201600263

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Δρ. Μαρία Ρούσσου,** Αναπληρώτρια Καθηγήτρια
Δρ. Ακριβή Κατηφόρη, Χρήστος Λουγιάκης

ABSTRACT

The topic of this thesis was inspired by the observation of the potential that Visual Novels, a video game genre, hold for applications regarding multiple different domains. Following this observation, it was noticed that a frequent challenge faced by the teams that are interested in developing interactive experiences of this kind is the lack of technological expertise from the individuals that are suitable for the creation of the content, along with the lack of tools that support the smooth collaboration between the experience content designers and the development teams.

Therefore, for this thesis, a plug-in for Unity Engine was designed and developed to support the development of branching narrative experiences with multiple endings through a node-based graph and a set of automation features. The tool was created to link intuitive branching narrative design and the development of interactive Visual Novels by enabling developers to seamlessly import stories created with the web-based authoring tool Narralive Story Maker, catering to authors.

This thesis begins with extensive research on Visual Novels and their different possible narratives, continuing with the design process of the tool before starting its development. After implementing the plug-in, a demo video game was developed to showcase its workflow and functionalities. As a last step, the tool was evaluated with participants, concluding with ideas on its future development.

SUBJECT AREA: Interactive Digital Narratives, Authoring Tools

KEYWORDS: Authoring tool, Unity Engine, Branching Narratives, Visual Novels, Node Based Graph

ΠΕΡΙΛΗΨΗ

Το θέμα αυτής της πτυχιακής ενέπνευσε η παρατήρηση των δυνατοτήτων που έχουν τα Visual Novels, ένα είδος βιντεοπαιχνιδιού, για εφαρμογές σε πολλούς διαφορετικούς τομείς. Ακολουθώντας αυτή την παρατήρηση, διαπιστώθηκε ότι μια συχνή πρόκληση που αντιμετωπίζουν οι ομάδες που ενδιαφέρονται να αναπτύξουν διαδραστικές εμπειρίες τέτοιου είδους είναι η έλλειψη τεχνολογικής γνώσης από τα άτομα που είναι κατάλληλα για τη δημιουργία του περιεχομένου, μαζί με την έλλειψη εργαλείων τα οποία υποστηρίζουν την ομαλή συνεργασία μεταξύ των σχεδιαστών περιεχομένου εμπειρίας και των προγραμματιστών.

Για τους σκοπούς αυτής της εργασίας, σχεδιάστηκε και αναπτύχθηκε ένα plug-in για τη μηχανή Unity με στόχο να υποστηρίξει την ανάπτυξη εμπειριών με αφήγηση με διακλαδώσεις και πολλαπλά τέλη μέσω ενός γράφου με κόμβους και ενός συνόλου λειτουργιών αυτοματοποίησης. Το εργαλείο δημιουργήθηκε με σκοπό να λειτουργήσει ως η σύνδεση μεταξύ της σχεδίασης διηγήσεων με διακλαδώσεις και της ανάπτυξης διαδραστικών Visual Novels, επιτρέποντας στους προγραμματιστές να εισάγουν ομαλά ιστορίες που δημιουργήθηκαν με το διαδικτυακό εργαλείο συγγραφής Narralive Story Maker, κατάλληλο για τους συγγραφείς.

Αυτή η εργασία ξεκινά με εκτεταμένη έρευνα για τα Visual Novels και τις διάφορες δυνατές αφηγηματικές προσεγγίσεις τους και συνεχίζει με τον σχεδιασμό του εργαλείου πριν από την έναρξη της ανάπτυξής του. Αφού ολοκληρωθεί η υλοποίηση αυτού του plug-in, αναπτύχθηκε ένα demo βιντεοπαιχνίδι για να παρουσιαστεί η ροή και οι λειτουργίες του. Ως τελευταίο βήμα, με στόχο την απόκτηση χρήσιμων σχολίων από πιθανούς χρήστες, πραγματοποιήθηκε μια αξιολόγηση με συμμετέχοντες, καταλήγοντας σε ιδέες για τη μελλοντική ανάπτυξή του.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Διαδραστικές Ψηφιακές Αφηγήσεις, Εργαλεία Συγγραφής

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Εργαλείο Συγγραφής, Μηχανή Unity, Διήγηση με Διακλαδώσεις, Οπτικοακουστικά Μυθιστορήματα, Γράφος με Κόμβους

AKNOWLEDGMENTS

I would first like to thank my supervisor, Dr Maria Roussou, for responding to my interest to collaborate with her on my thesis and for helping me choose the topic, as well as introducing me to her colleagues.

I would also like to thank my advisors, Dr Vivi Katifori and Christos Lougiakis, for working together on developing this software. I learned a lot from our time working together, gaining experience in new areas.

I want to thank the people who took part in the evaluation of this software, providing me with valuable feedback.

Last but not least, I would like to thank my loved ones and, especially, my partner, friend, and colleague, Konstantinos K., for supporting me through completing this thesis, and my studies, in general.

CONTENTS

LIST OF IMAGES

PREFACE

This thesis was conducted as my Bachelor’s thesis for the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens.

1. INTRODUCTION

This chapter focuses on introducing the objective of the thesis, the idea behind it as well as the purpose and motivation that led to choosing the specific objective.

1.1 Main Objective

The objective of the thesis is the development of a plug-in for the Unity Engine that works as an extension of a web-based authoring tool supporting the development of Visual Novel video games and interactive experiences in general. At the same time, the solution aims at improving the communication between authors and developers. This thesis involves the process of researching, designing, developing, and evaluating this tool.

The web-based authoring tool that the Unity plug-in is an extension of, is Narralive's Story Maker [1]. This tool is part of the Emotive project, an EU-funded heritage project that uses emotional storytelling to dramatically change how we experience heritage sites. The output of the Emotive project is a number of prototype tools and applications for heritage professionals and visitors that produce interactive digital stories for museums and cultural sites [2]. Since the Unity plug-in that was developed for this thesis is explicitly linked to the Narralive Story Maker tool, it was given the name Narralive Unity plug-in.

The implementation resulted in a tool that imports the Story Maker's data in Unity and creates a node-based graph displaying the said data. Also, it provides the user with a set of automation features that assist with the process of scene generation and the transformation of the data into ready-to-use code. This way, the developers can work on their project faster and easier, skipping the steps of transferring and managing the authors' data and creating Unity objects displaying them while keeping the authors' process of the story's development within the web tool.

1.2 Purpose

The main purpose of the thesis is to bridge the gap between the design of interactive stories with branching narratives and the development of Visual Novels with multiple game mechanics. This tool proposes a replacement for the manual transfer of data and designs, as well as tools that combine the processes of design and development, which often result in authoring environments that are hard to comprehend for authors without technical expertise.

Additionally, since the tricky aspect of video games with complex narratives is keeping track of the storyline, the tool provides a visual graph that can assist the developer with having a clear idea of the story on which they are working. Lastly, since the thesis tool is based on a preexisting web tool, another outcome of its implementation is that it gives the user the ability to transfer their online data on a platform where they can work offline.

1.3 Motivation

What gave the motivation for the creation of the tool was the observation of the challenges faced by teams and institutions interested in the creation of interactive experiences for multiple purposes. The primary challenge lies in the fact that the individuals that intend to produce content of this kind often lack the necessary technological proficiency, while the available authoring tools target users with some level of programming abilities. To successfully succeed in such projects, a team must involve diverse roles, including domain experts, storytellers, and digital asset designers [3], often leading to the need for collaboration and communication with external professionals.

With the motivation to equip a team or institution interested in creating interactive experiences with a solution to connect diverse professionals without compromise, the focus of this thesis involves the integration of the story model generated by the Narralive Story Maker into a game development tool within the Unity Game Engine.

2. THEORETICAL BACKGROUND

This chapter is dedicated to the theoretical research that was conducted prior to moving to the technical aspect and the development of the Unity plug-in. This research concerns the Visual Novel genre, its potential for multiple different uses, as well as the Narralive Story Maker that the plug-in is based on.

2.1 Visual Novels

Visual novels have witnessed a remarkable rise in popularity as a distinctive medium of interactive storytelling, captivating audiences through their compelling narratives, intricate character development, and engaging visual elements. This sub-chapter aims to introduce visual novels and their potential besides entertainment, explore different narrative styles within the genre, present their origins, and showcase some of the best examples that have contributed to their success.

2.1.1 Genre Description

In video games, a visual novel is a genre that describes a story-based game that focuses its gameplay on an interactive story. Usually, visual novels have simple game mechanics which can often be limited to a simple choice of text [4].

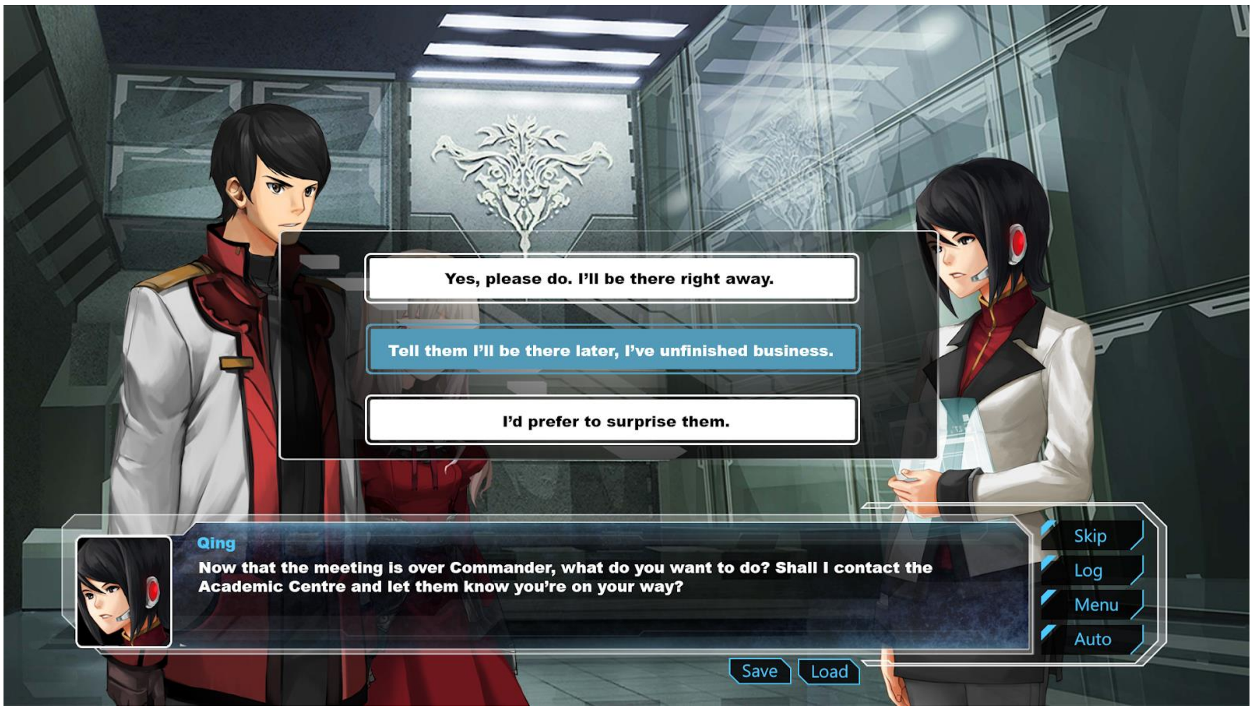

Yes, please do. I'll be there right away.
Tell them I'll be there later, I've unfinished business.
I'd prefer to surprise them.
Qing
Now that the meeting is over Commander, what do you want to do? Shall I contact the Academic Centre and let them know you're on your way?

Figure 1: Visual Novel styling example from "Cross Fate"

Visual novels can be easily compared to a novel book since their main purpose is to tell a story, with the exception that in a visual novel, the story can be manipulated by the player. Most of the time, in a visual novel, there are multiple different endings, with the outcome being based on the choices that were made by the player through the gameplay. Other times, a visual novel can be developed so that the ending is predetermined, and the player's choices only affect the path that will lead to that particular ending. The simplest visual novel that can be developed does not give the player any choices and focuses the gameplay solely on storytelling. In the last case, the visual novel gives an experience closer to a book or a movie than a game.

2.1.2 Branching, Linear, and Non-linear Plot

When it comes to storytelling, there are three types of stories that game developers, or storytellers in general, can choose from. These types of narrative concepts are linear, non-linear, and branching. In this section, we will examine and detect their differences [5].

Linear

A linear narrative mostly relates to the chronological order that the events are shown in. In this type of story, it is not likely to find flashbacks or jumps forward in time. More specifically, linear stories are stories that progress in a straight line, from beginning to end, with no branching paths or alternate endings. In other words, the events in a linear story unfold in a predetermined order, and the outcome is fixed.

Linear stories are more suitable when the target is a rich story since the developers can focus on one storyline. Also, it gives the player a more solid experience of a series of events. Additionally, linear stories tend to be more relaxing regarding the gameplay since the player does not struggle with choices and the consequences of bad ones while also spending their time being more invested in the plot. Also, they can be easier to follow and understand, as there is a clear progression from beginning to end. This can make them more accessible to a wider audience, as there is less complexity to track and remember.

Non-linear

In contrast to linear storytelling, a non-linear story is a type of narrative that doesn't follow a traditional chronological order, meaning that events are not presented in the order in which they occurred. Instead, a non-linear story may jump back and forth in time, or it may present events out of order, creating a more complex and potentially confusing narrative structure for the reader or viewer.

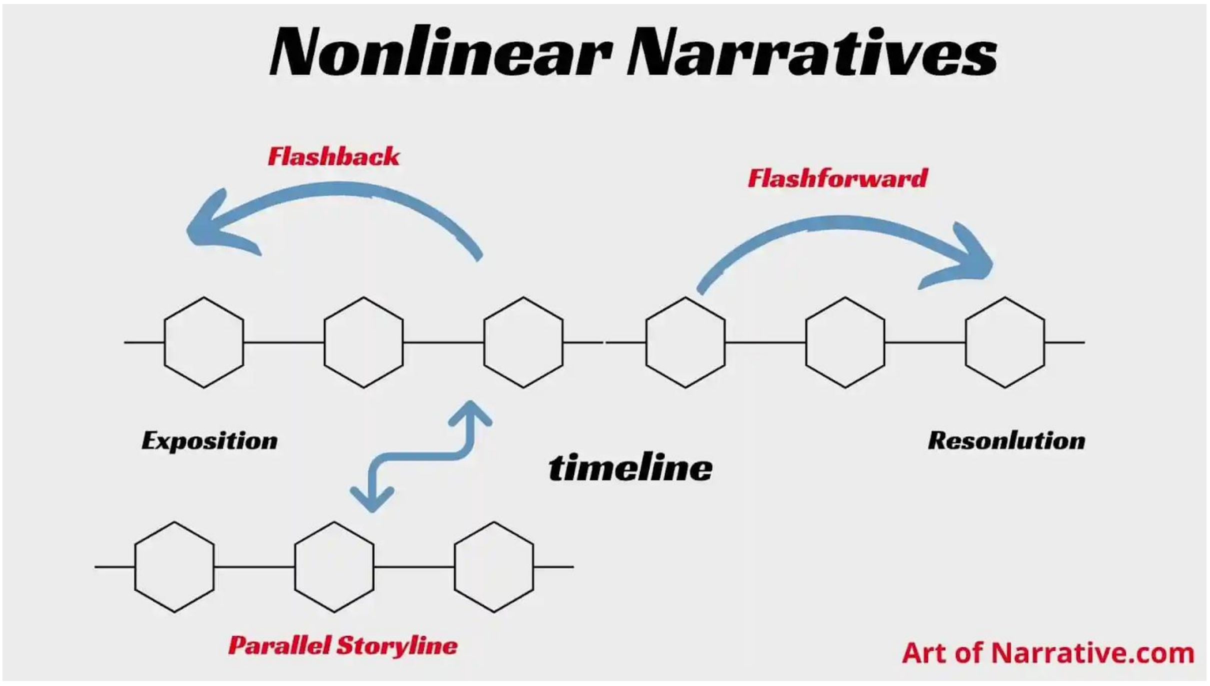


Figure 2: Example of Nonlinear Narrative

There are several reasons why a story might be told non-linearly. One reason is to create suspense or tension by withholding information from the reader or viewer until later in the narrative. For example, a non-linear story might present the resolution of a mystery before revealing the events that led up to it, forcing the reader or viewer to piece together the story's events to understand what happened.

Another reason for using a non-linear narrative is to explore the complexity of human experience and how events and memories can intertwine and overlap. Non-linear stories can convey how different events and memories can influence and shape one another, creating a more layered and nuanced understanding of characters and their motivations.

Non-linear stories can be challenging to write and follow, as they require the reader or viewer to pay close attention to the order in which events are presented and to make connections between events that may not be immediately obvious. However, when done well, non-linear stories can create a rich and rewarding experience for the reader or viewer, as they encourage a more active and engaged interpretation of the narrative.

Branching

A branching story is a type of narrative that allows the reader or viewer to choose which direction the story will take at various points in the narrative. Branching stories are often found in interactive media such as video games, but they can also be found in other forms such as choose-your-own-adventure books or branching storylines in television shows or films.

Branching stories allow the reader or viewer to actively participate in the narrative as they make choices that determine the course of the story. This can create a more immersive and engaging experience for the reader or viewer, as they feel that their choices impact the story.

There are several ways in which branching stories can be structured. One approach is to present the reader or viewer with a series of choices at various points in the story, each of which leads to a different outcome. For example, in a branching story set in a medieval fantasy world, the reader might be presented with the choice to join a group of rebels or to side with the ruling government, each of which leads to a different set of events and outcomes.

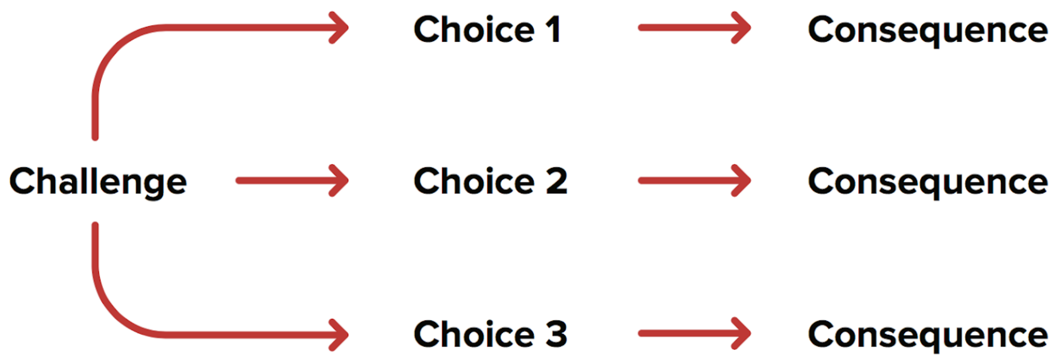


Figure 3: Example of Branching Narrative

Another approach is to use a branching story to present multiple perspectives on a single event or series of events. For example, a branching story might present the same event from the perspective of several different characters, each with their motivations and experiences. This can create a more complex and nuanced understanding of the story and its characters. An example of a video game based on the perspective of different characters is The Last of Us Part 2 where the player gets to act as both the protagonist and their enemy.

Branching stories can be challenging to write, as they require the creation of multiple possible paths through the narrative and the ability to anticipate and account for the choices that the reader or viewer might make. However, when done well, branching stories can create a highly engaging and immersive experience for the reader or viewer as they become an active participant in the narrative.

2.1.3 Origins of Visual Novels

Visual novels trace their roots back to Japan, where they emerged in the 1980s as text-based adventure games with accompanying visuals. The early examples featured minimal graphics and emphasized their storytelling through text and sound effects. Over time, technological advancements allowed for the integration of artwork, voice acting, and branching narratives, leading to the evolution of visual novels as a distinct genre.

The portopia serial murder case

The first visual novel was created in 1983 with the title “The portopia serial murder case” [6]. Considering that “Computer Space,” the first commercial video game in general, was created in 1971, the genre of visual novels began in early gaming history. It was created by Yuji Horii and published by the company Enix, which was later merged with the company Square to create the well-known video game company Square Enix.

The game was originally released on the NEC PC-6001 and was later available for PCs, Nintendo consoles, and mobiles. For those who are interested in trying the first visual novel out, the game is available online [7].

As the title implies, the aim of the game is the solving of a murder mystery, and it gave the player multiple ways to do so, meaning it was non-linear. The main gameplay consists of static images, interaction with non-player characters, and moving through the game's world and it was the first point-and-click game to be developed, serving as an inspiration to today's well-known names, such as Hideo Kojima, creator of the Metal Gear franchise and Death Stranding.

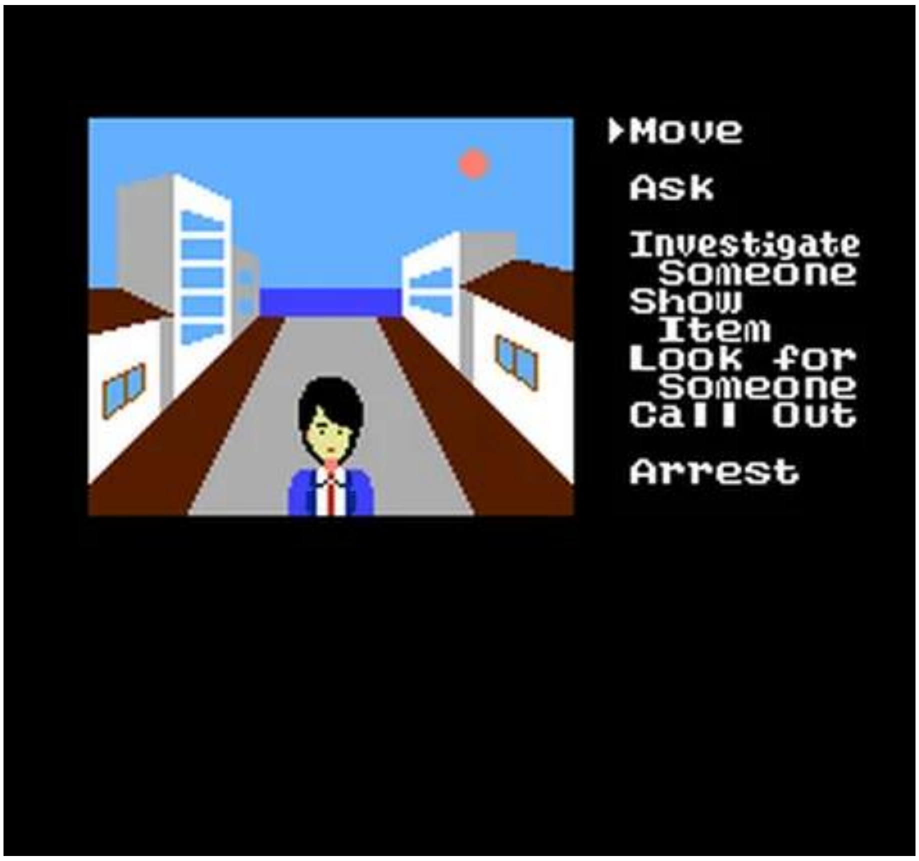


Figure 4: Scene from "The portopia serial murder case"

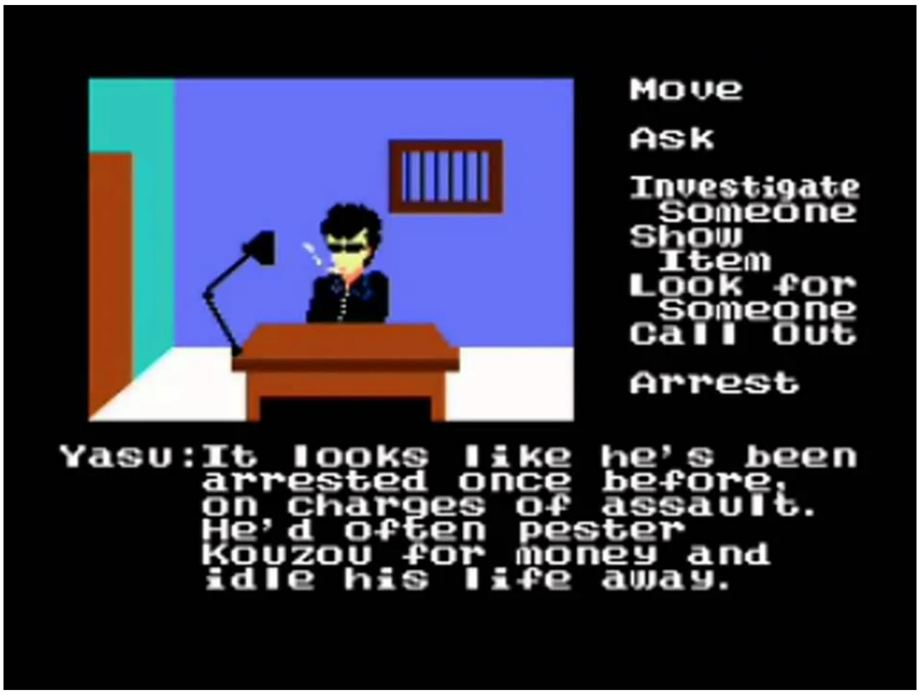
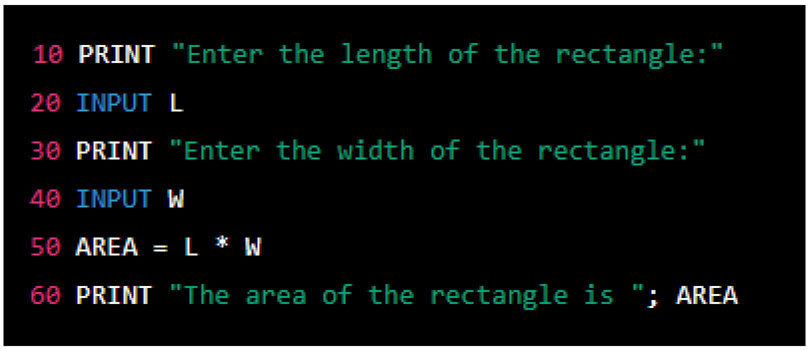

Move
Ask
Investigate Someone
Show Item
Look for Someone
Call Out
Arrest
Yasu:It looks like he's been arrested once before, on charges of assault. He'd often pester Kouzou for money and idle his life away.

Figure 5: Scene from "The portopia serial murder case"

The game was created using the programming language BASIC and ran on the MSX-DOS operating system. BASIC (Beginners' All-purpose Symbolic Instruction Code) is a high-level programming language that was developed in the 1960s, designed to be easy to learn and use, with a simple syntax that closely resembles English. It is often used as a first programming language for beginners, as it can be used to create simple programs quickly and easily. It is a procedural programming language, which means that it follows a step-by-step process to execute a program. Also, it uses commands and statements to control the flow of execution and perform various operations such as input/output, arithmetic and logic operations, and loops. BASIC programs are typically composed of one or more procedures or subroutines, groups of statements that perform a specific task. Variables can be used to store and manipulate data within a program.

This programming language has been widely used on various platforms like personal computers, microcomputers, and video game consoles. However, as technology progressed, more advanced languages like C, C++ and Java became more popular among developers, and the use of BASIC has decreased. A simple program written in BASIC would look like the following.



```
10 PRINT "Enter the length of the rectangle:"
20 INPUT L
30 PRINT "Enter the width of the rectangle:"
40 INPUT W
50 AREA = L * W
60 PRINT "The area of the rectangle is "; AREA
```

Figure 6: Example of BASIC code

This program prompts the user to enter the length and width of a rectangle and then calculates the area by multiplying the length and width. The result is then printed on the screen.

Crystal Dragon

A few years later, in 1986, the visual novel “Crystal Dragon” was released by Square, the second company that later became a part of Square Enix, as mentioned above. Similar to Enix’s “The portopia of serial murder case”, Square’s “Crystal Dragon” is a point-and-click adventure based on static visuals and text [8].

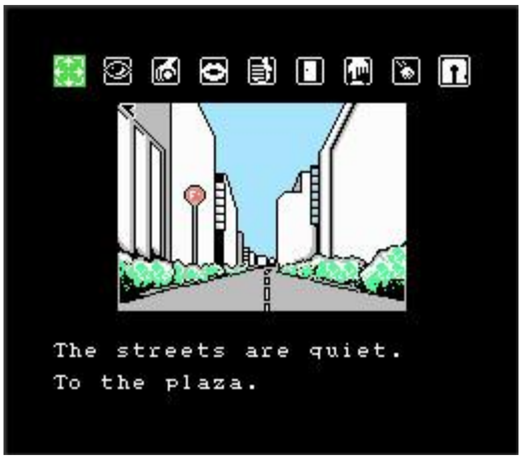


Figure 7: Scene from "Crystal Dragon"

The main plot of the game concerns the main character traveling in space and facing a dragon. It was released for the Famicon Disk System, a peripheral for Nintendo’s Family Computer console.

Snatcher

Just a couple of years later, in 1988, Hideo Kojima, one of the most famous game creators who, as mentioned, was inspired by the first visual novels, developed “Snatcher,” a visual novel of his own. Snatcher was a far more modern work compared to its ancestors, as it consisted of detailed graphics, voice lines, and animated cutscenes [9].

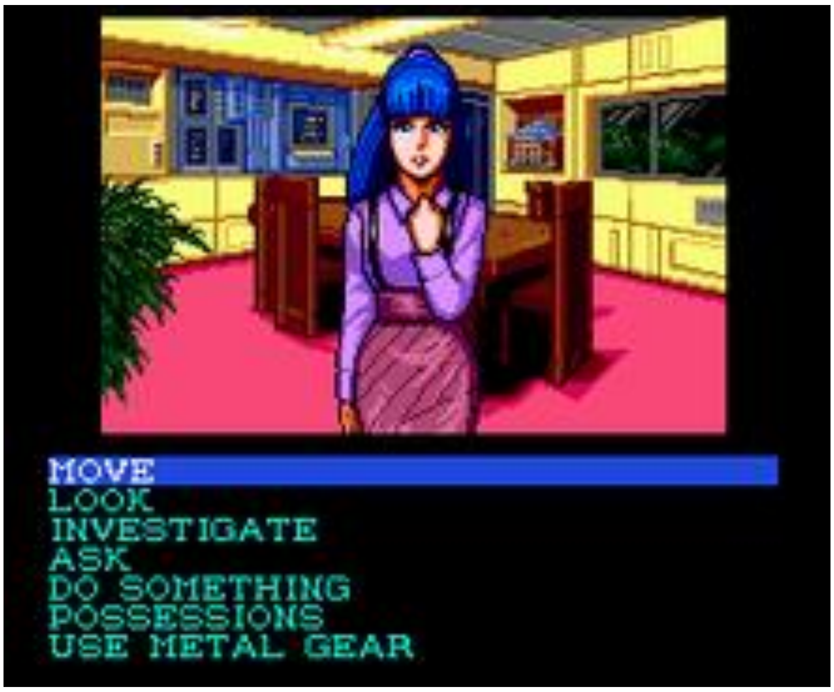
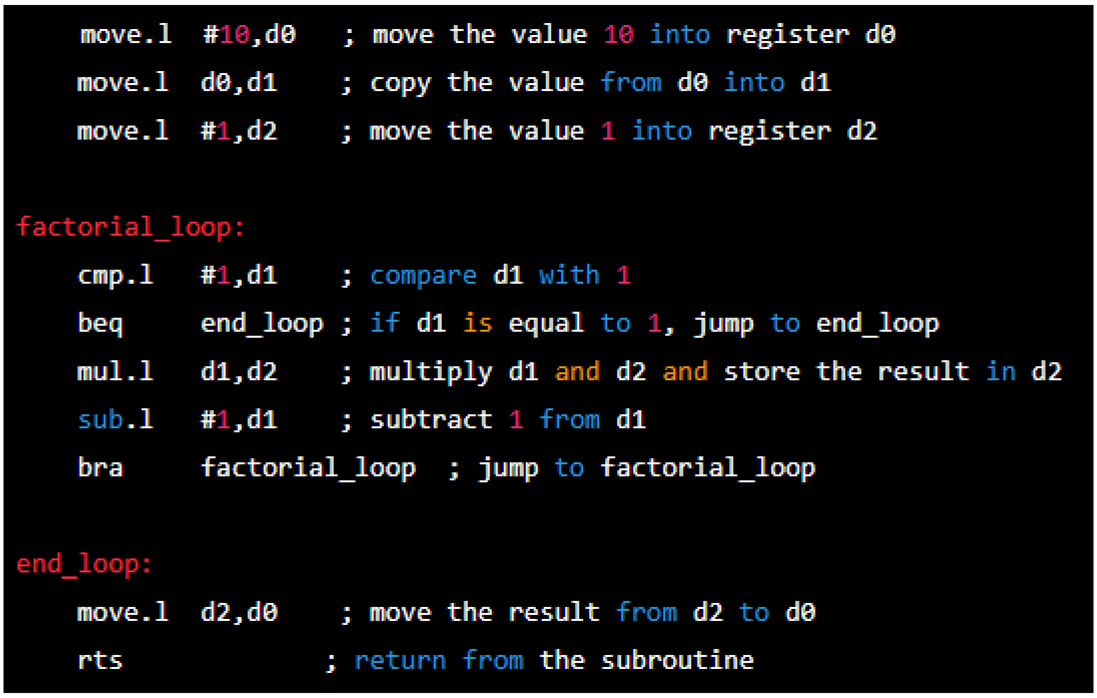


Figure 8: Scene from "Snatcher"

Snatcher was developed using a combination of programming languages and tools. The game's engine was written in 68000 assembly language, a low-level programming language that allows for efficient use of the hardware resources of the platforms it was developed for. The following code is an example of a program written in 68000 assembly that calculates the factorial of a number.



```
    move.l  #10,d0     ; move the value 10 into register d0
    move.l  d0,d1      ; copy the value from d0 into d1
    move.l  #1,d2      ; move the value 1 into register d2

factorial_loop:
    cmp.l   #1,d1      ; compare d1 with 1
    beq     end_loop ; if d1 is equal to 1, jump to end_loop
    mul.l   d1,d2      ; multiply d1 and d2 and store the result in d2
    sub.l   #1,d1      ; subtract 1 from d1
    bra     factorial_loop  ; jump to factorial_loop

end_loop:
    move.l  d2,d0      ; move the result from d2 to d0
    rts              ; return from the subroutine
```

Figure 9: Example of 68000 assembly code

The game's graphics were created using a combination of pixel art and 3D graphics, and the sound was composed using the sound hardware of the platforms for which it was developed. Snatcher was one of the first games to use fully-voiced cutscenes, which were achieved by using the CD-ROM format of the Sega CD, which allowed for the storage of large amounts of audio data. As a result, it was considered a technical achievement at the time of its release due to its advanced graphics, use of full-motion video and CD-ROM technology, and a branching narrative.

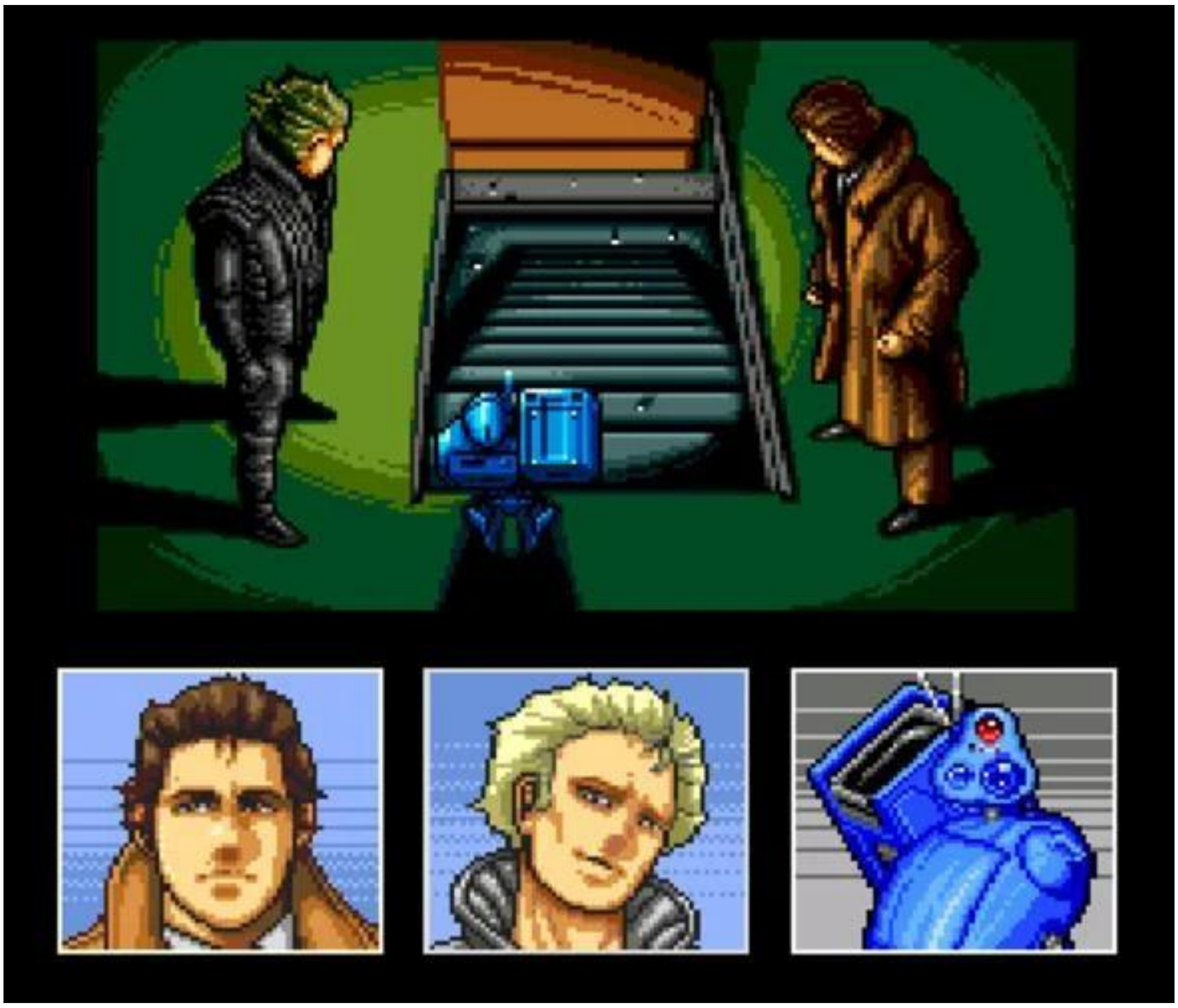


Figure 10: Scene from "Snatcher"

The game was published by Konami, and it was one of the first visual novels that were released in America. Following this game, Hideo Kojima continued working on and releasing visual novels.

YU-NO: A Girl Who Chants Love at the Bound of this World

For the next few years, multiple games of the genre appeared but there is one particular title that became a milestone for visual novels. The game, in particular, is “YU-NO: A Girl Who Chants Love at the Bound of this World” [10]. YU-NO introduced a new system called Automatic Diverge Mapping System (A.D.M.S) [11]. This system contained a screen that indicated which direction the player was heading in the story, making the replaying more interesting. The direction is presented by branches and marks showing the position of the player as well as the position of possible endings.

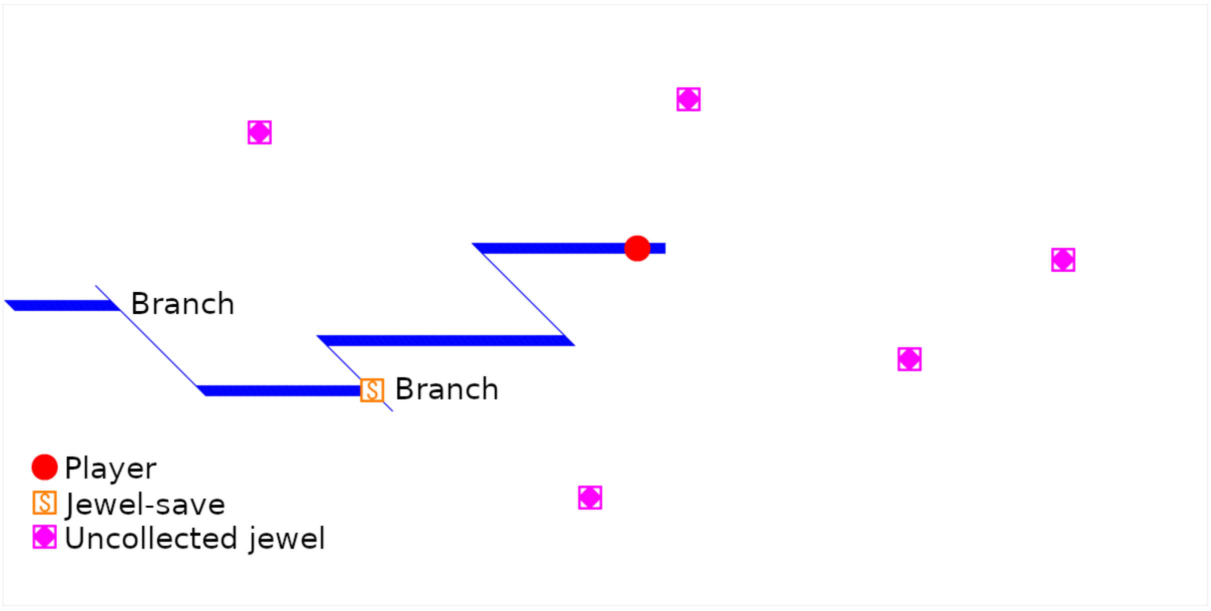


Figure 11: Example of A.D.M.S.

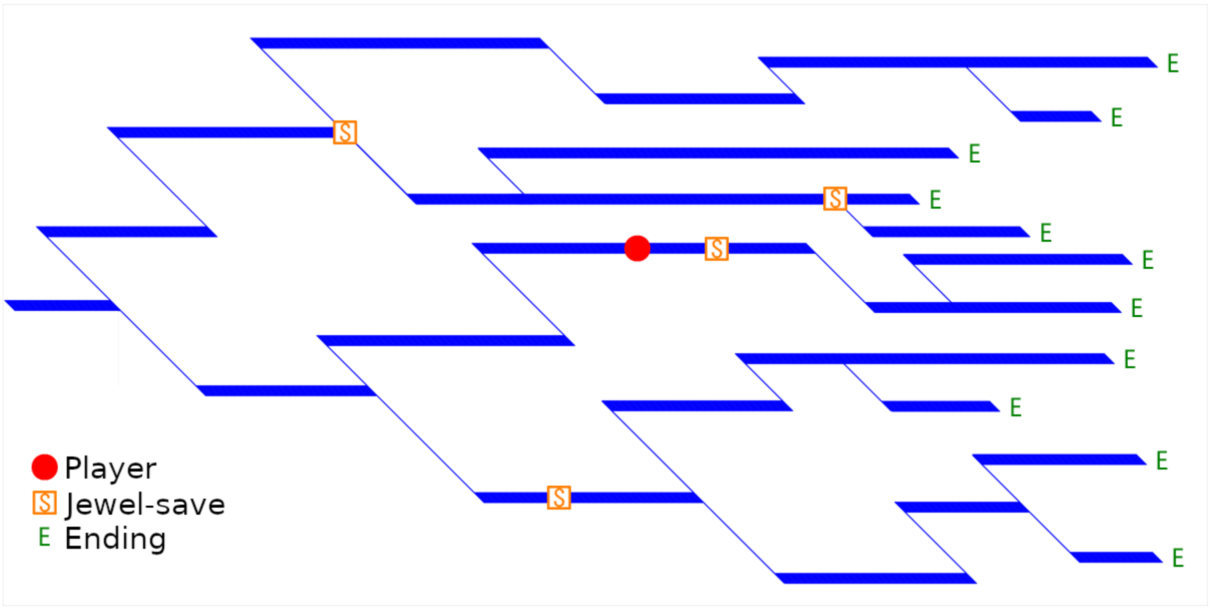


Figure 12: Example of A.D.M.S.

2.1.4 Top rated Visual Novels

As in every video game genre, there are multiple visual novels that have made it to the top of gamers' lists. In this section, some notable examples will be mentioned on account of getting a deeper understanding of the genre.

Doki Doki Literature Club!

"Doki Doki Literature Club!" was released in 2017 and is one of the most famous visual novels, also known outside of the genre. The basic plot is about a high school literature club where the protagonist is invited to join [12].

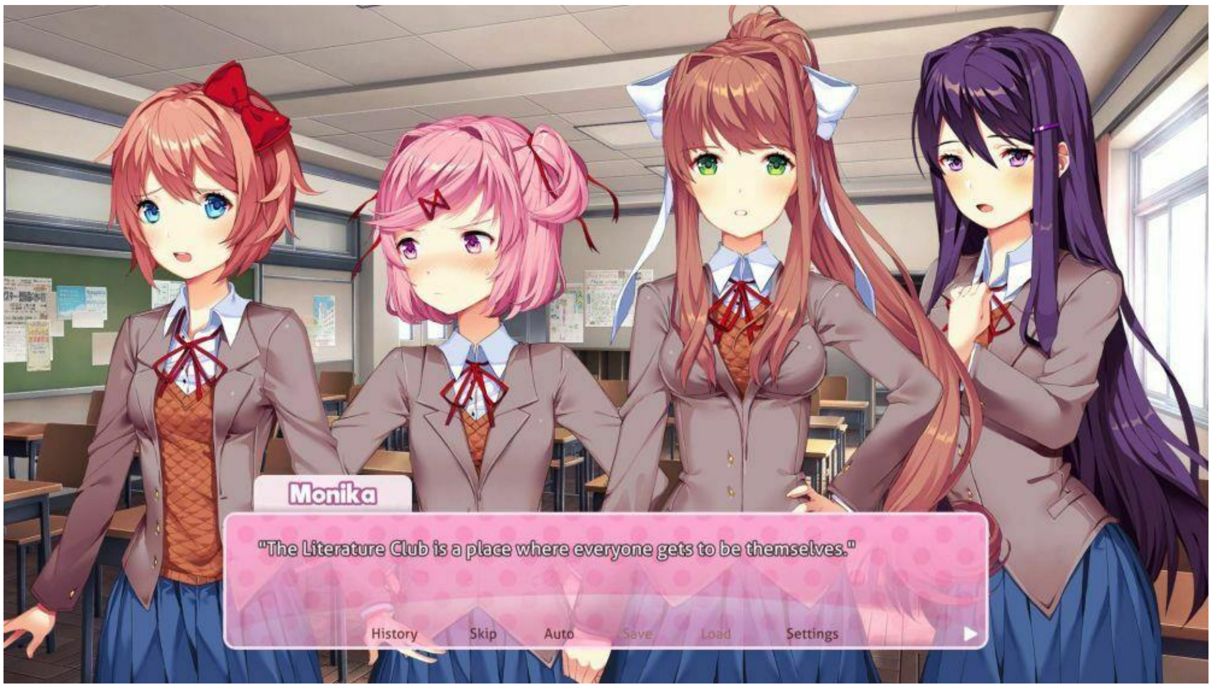

Monika
"The Literature Club is a place where everyone gets to be themselves."
History
Skip
Auto
Save
Load
Settings

Figure 13: Scene from "YU-NO: A Girl Who Chants Love at the Bound of this World"

The narration is provided by the main character, controlled by the player. Depending on the player's choices, the protagonist's relationships with other characters are affected. The main mechanic that is used is a minigame where the player has to choose words from a random set in order to compose a poem. Based on the words they choose, the other characters will respond in a positive or a negative way. More specifically, in the minigame, the concerning characters, other members of the literature club, are displayed at the bottom of the screen. In this way, the player can view if each character responds positively or negatively through their reactions.

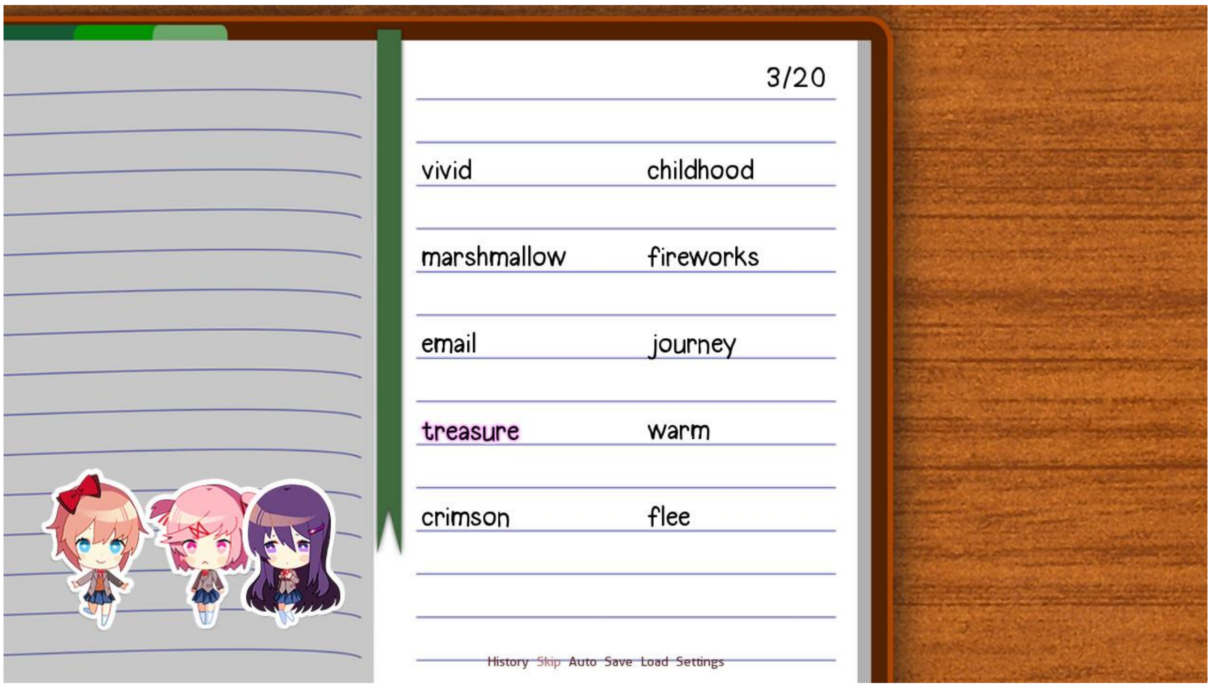
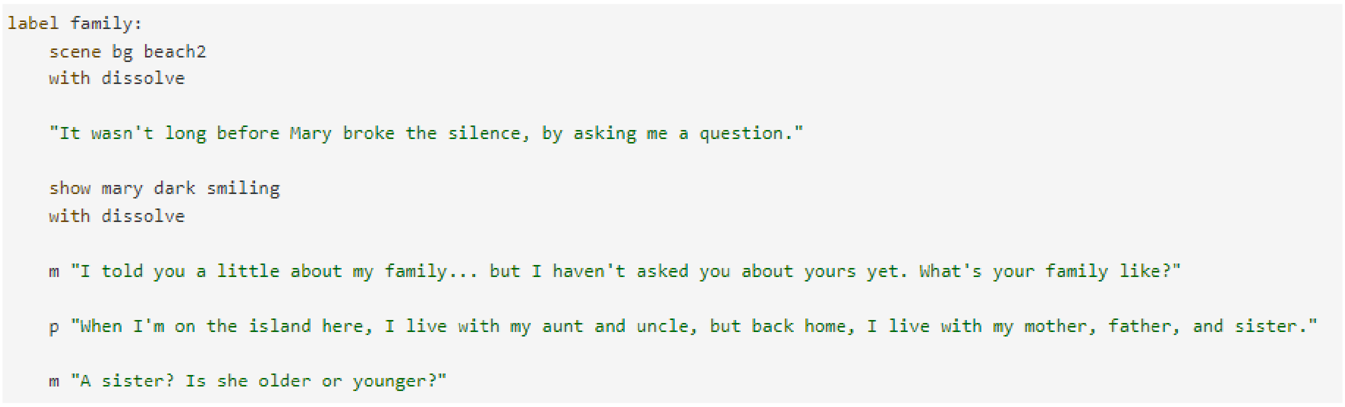


Figure 14: Scene from "YU-NO: A Girl Who Chants Love at the Bound of this World"

The game has two possible endings, one being a sad one and the other being a happy one. The positive ending occurs if the player views all the optional cutscenes while playing the game.

Doki Doki Literature Club! scored 78% to 89% on Metacritic, with the range being due to scores for different consoles. It has also received IGN's People's Choice award for "Best PC game of 2017".

Dan Salvato, the developer of the game, used Ren'Py, a visual novel-oriented game engine. It is based on Python, and uses a simple scripting language [13].



```
label family:
    scene bg beach2
    with dissolve

    "It wasn't long before Mary broke the silence, by asking me a question."

    show mary dark smiling
    with dissolve

    m "I told you a little about my family... but I haven't asked you about yours yet. What's your family like?"

    p "When I'm on the island here, I live with my aunt and uncle, but back home, I live with my mother, father, and sister."

    m "A sister? Is she older or younger?"
```

Figure 15: Example of Ren'Py code

Ren'Py is a free, open-source engine that allows the developers the opportunity to distribute their games without a fee. Due to the simplicity of the scripting language, it does not require the developers to have significant coding knowledge.

Ace Attorney

"Ace Attorney" is a popular series of visual novel games, with its first entry being "Phoenix Wright: Ace Attorney," which was released in 2001. It consists of six main series games and five spin-offs [14].

Figure 16: Cover from "Ace Attorney"

Through this game, the player controls defense attorneys while interacting with their clients and witnesses. In the first section of a total of two sections of the game, the player gathers information through interaction with the other characters. After they complete the first section, they move on to the next section where they defend their clients in a courtroom trial.

Figure 17: Scene from "Ace Attorney"

The basic trilogy of the Ace Attorney franchise was developed in Unity, and it had a major impact on the gaming world, leading to the creation of a number of fan-made, open-source software for the creation of more gameplay. Typically, such software implements a case maker where a user can create a new case that can be played on an Ace Attorney clone. The following screenshot showcases an example of Ace Attorney Online, an online case maker [15].

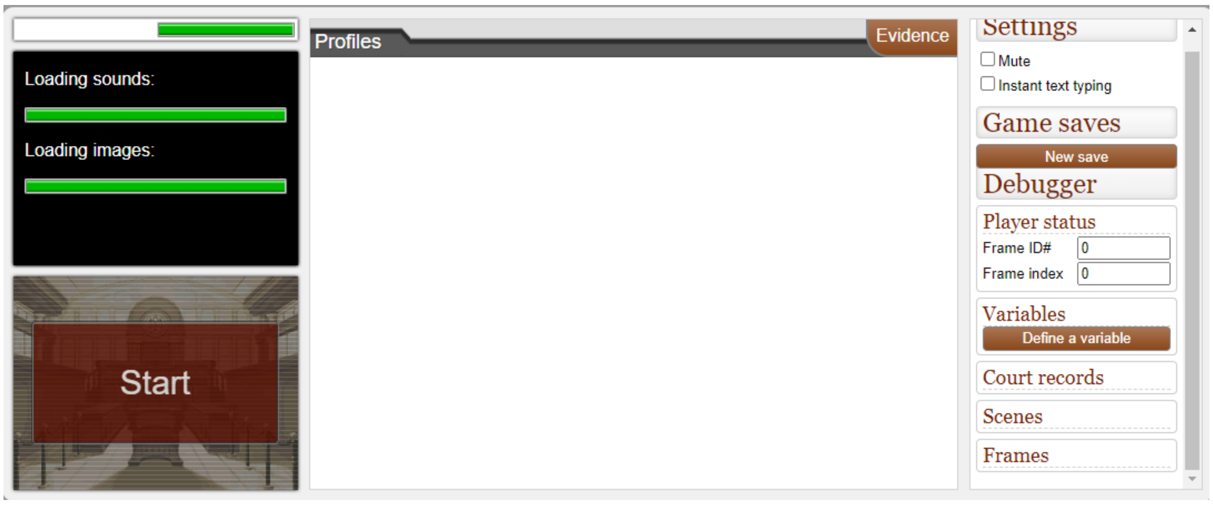
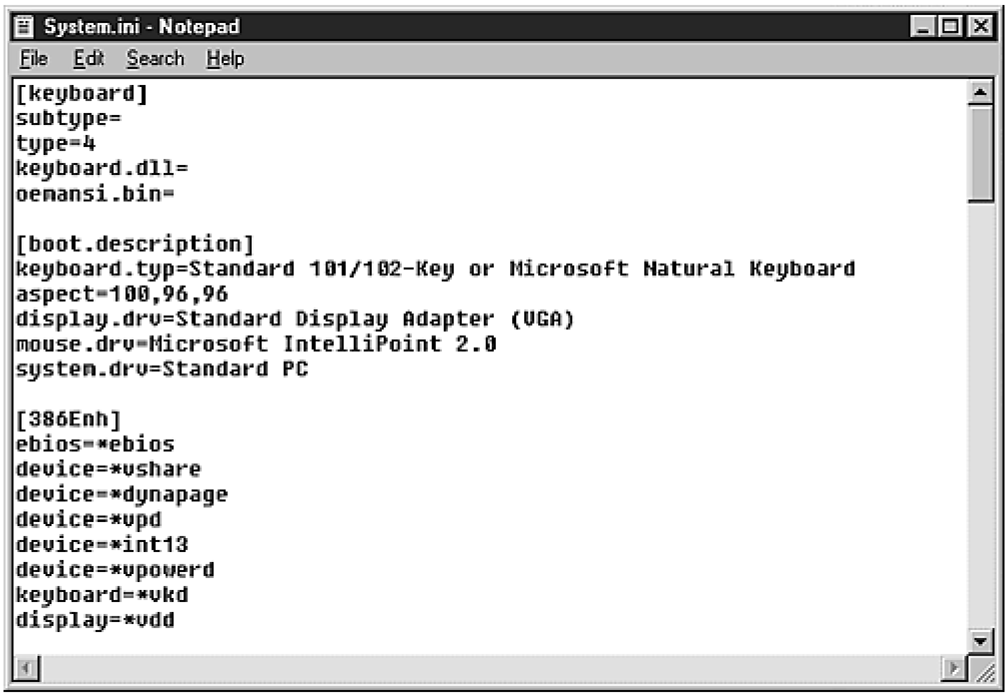


Figure 18: Screenshot of "Ace Attorney Online"

Another case maker is Ace Attorney DS [16]. This software is a Visual Novel Engine that creates scenes while supporting the emulation of Phoenix Wright: Ace Attorney, the first game of the series. Ace Attorney DS uses .ini files making the usage of the engine easier as it does not require coding knowledge. This type of file is a configuration file for software that consists of text-based content with key-value pairs.



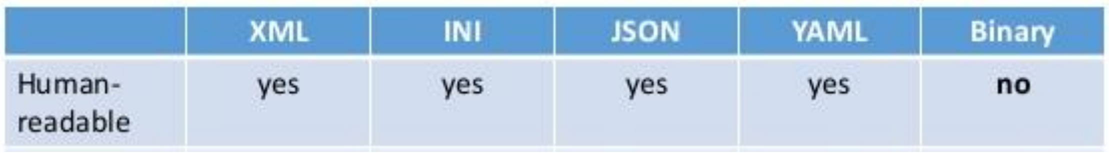
```
[keyboard]
subtype=
type=4
keyboard.dll=
oemansi.bin=

[boot.description]
keyboard.typ=Standard 101/102-Key or Microsoft Natural Keyboard
aspect=100,96,96
display.drv=Standard Display Adapter (VGA)
mouse.drv=Microsoft IntelliPoint 2.0
system.drv=Standard PC

[386Enh]
ebios=*ebios
device=*vshare
device=*dynapage
device=*vpd
device=*int13
device=*vpowerd
keyboard=*vkd
display=*vdd
```

Figure 19: Example of .ini file

Other known formats that are similar to .ini are JSON and XML, as well as YAML and Binary [17].



	XML	INI	JSON	YAML	Binary
Human-readable	yes	yes	yes	yes	**no**

Figure 20: Human-readable data formats

Clannad

Clannad is a Japanese visual novel which was released in 2004 for Windows from the company Key and it was developed using the software Ren'Py. Due to its publicity, it was later ported to other consoles such as PlayStation 2, Xbox 360, PlayStation 3, PlayStation Vita, PlayStation 4, and Nintendo Switch [18].

Figure 21: Cover of "Clannad"

Clannad is a visual novel that follows a branching narrative structure, meaning that the story can unfold in different ways depending on the choices made by the player, with a significant amount of the gameplay being spent on dialogue and reading the narrative. However, the game does have a "true" ending that can be achieved by making certain choices and progressing through the story in a specific way.

The story of Clannad follows Tomoya Okazaki, a high school student struggling to find meaning in life. He meets a girl named Nagisa Furukawa, who is also struggling with personal issues, and they become friends. Together, they try to help each other overcome their problems and find a reason to live.

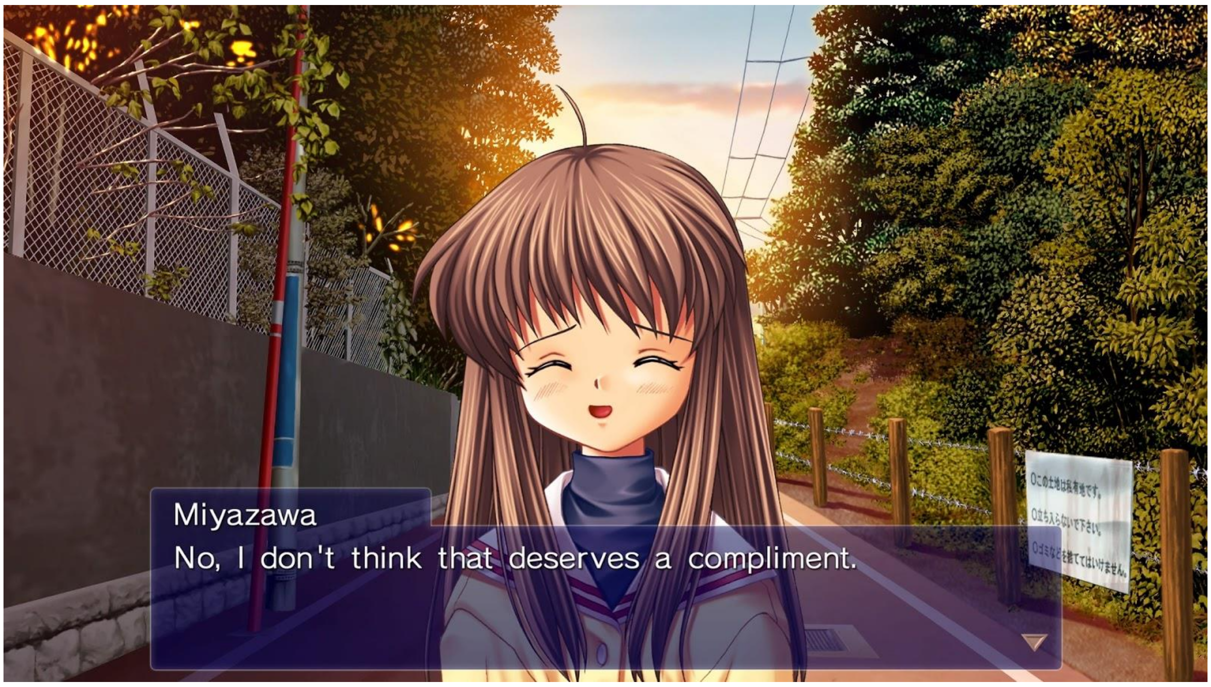

Miyazawa
No, I don't think that deserves a compliment.

Figure 22: Scene from “Clannad”

In order to achieve the "true" ending of Clannad, the player must make choices that lead Tomoya and Nagisa to overcome their personal struggles and find happiness together. This requires the player to progress through the story in a specific way, following a linear plot that ultimately leads to the true ending.

Overall, while Clannad does have a branching narrative structure that allows for multiple endings, the true ending of the game follows a linear plot that ultimately leads to the resolution of the main characters' personal struggles and the achievement of happiness.

Steins;Gate

Steins;Gate is a science fiction visual novel video game developed by 5pb. and Nitroplus. It was released in Japan in 2009 and has since been translated into multiple languages and released worldwide [19].

Steins;Gate
シュタインズ・ゲート
There is no end though there is a start in space. — Infinity.
It has own power, it ruins, and it goes though there is a start also in the star. — Finite.
Only the person who was wisdom can read the most foolish one from the history.
The fish that lives in the sea doesn't know the world in the land. It also ruins and goes if they have wisdom.
It is funnier that man exceeds the speed of light than fish start living in the land.
It can be said that this is an final ultimatum from the god to the people who can fight.

Figure 23: Cover of "Steins;Gate"

The game follows the story of Rintaro Okabe, a self-proclaimed "mad scientist," and his group of friends as they discover and develop the ability to send messages back in time using a modified microwave. The story branches out based on the decisions made by the player, leading to multiple endings depending on the choices made throughout the game.

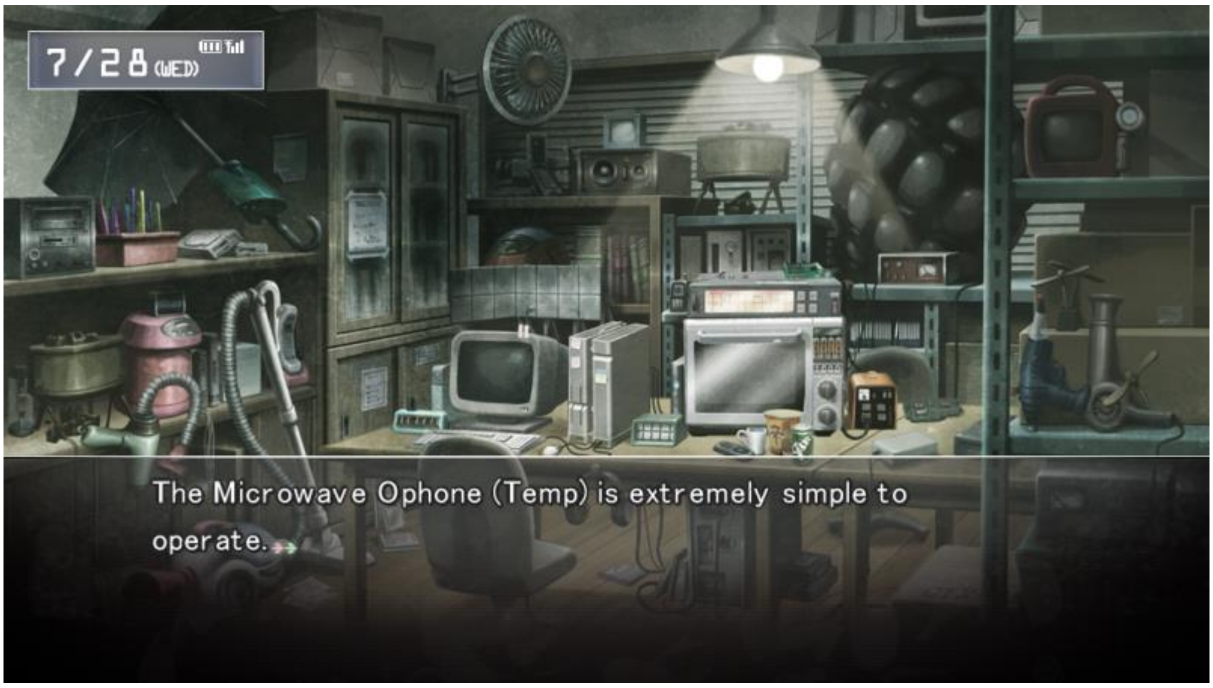


Figure 24: Scene from "Steins;Gate"

Steins;Gate was developed using the visual novel engine "Ren'Py," which allowed for the branching narrative structure and multiple endings. This means that each playthrough of the game can be unique, as the player's choices will determine the course of the story, allowing for a high level of replayability and encouraging multiple playthroughs to see all the different endings. It has been highly acclaimed for its complex and engaging storyline, well-developed characters, and branching narrative structure, making it a standout in the visual novel genre.

2.1.5 Visual Novels' Value

As described in this chapter, Visual Novels can vary from simple storytelling experiences to software with multiple and complex mechanics for interaction and a plethora of multimedia to display information. Consequently, this genre offers qualities that make them an ideal game genre for applications beyond mere entertainment. They hold immense potential in realms such as education [20], where they can serve as valuable tools for imparting information and enriching cultural heritage initiatives [21]. The combination of immersive storytelling, enriched with multimedia elements, and interactive yet low-demand gameplay, positions Visual Novels as a versatile medium for diverse purposes. They can provide captivating learning experiences and serve as dynamic platforms for presenting information on any subject, such as awareness campaigns or interactive tutorials [22].

2.2 Story Maker

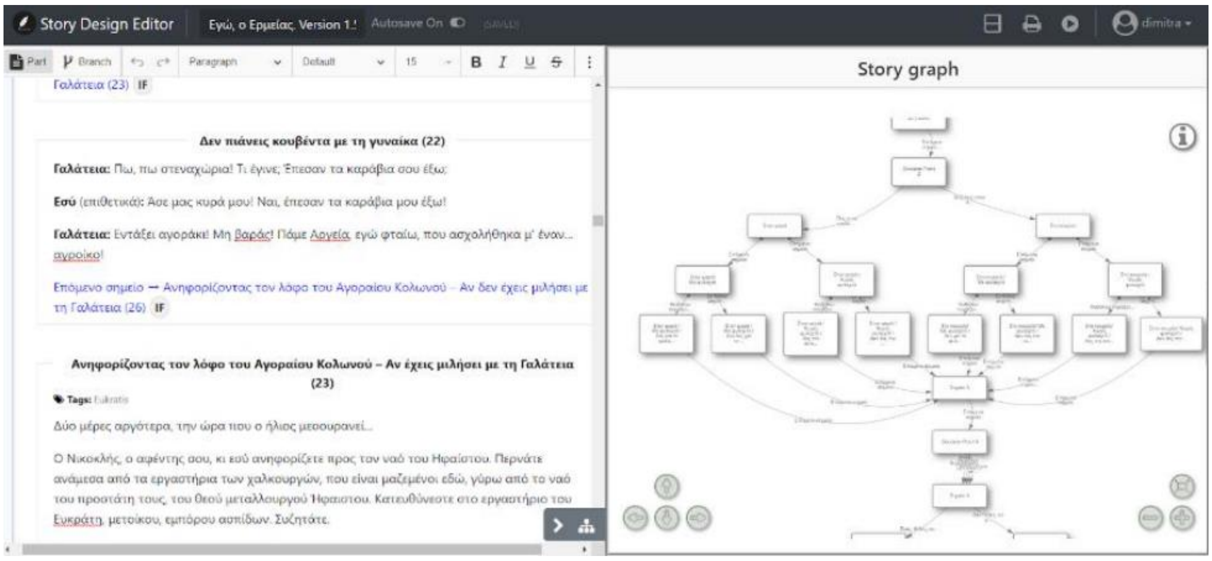
As mentioned in the introduction, the tool that was developed for the purpose of the thesis is an extension of the Narralive Story Maker. The Story Maker is an online platform designed to aid in the creation of engaging digital stories, particularly for cultural heritage purposes. It is user-friendly and intended for authors who may not have prior experience with interactive narratives or coding. The tool provides guidance and resources for designing the narrative structure and enables authors to publish their work as web-based experiences or as an Android app for mobile access by museum visitors, whether on-site or virtual. It is written in JavaScript on top of the Angular framework and backed by a document store. Any communication between the app and the backing store is made through a RESTful interface.

The app consists of two main components, the Story Design Editor and the Storyboard Editor. The Story Design Editor serves as a first step to define the structure and concept of the branching narrative in a text-editor-like format. The Storyboard Editor can be used as a next step, to transform the concept into the final experience offering built-in template support to create different types of multimedia activities.

2.2.1 The Story Maker Editor

As mentioned above, the Story Design Editor is a tool that assists in crafting the concept and structure of a story using a simple text editor, similar to popular word processors like Microsoft Office and Google Docs.

The editor also provides advanced features for crafting branching narratives. The narrative can be organized into coherent sections called Parts and connected through Branches. The tool provides a visual representation of the narrative structure, generated automatically as the author writes and edits the story. The Parts and Branches can be used to create various types of interactive stories, from simple linear narratives to complex branching ones, by utilizing tags and conditions.



Γαλάτεια (23) IF

Δεν πιάνεις κουβέντα με τη γυναίκα (22)

Γαλάτεια: Πω, πω στεναχώρια! Τι έγινε; Έπεσαν τα καράβια σου έξω;

Εσύ (επιθετικά): Άσε μας κυρά μου! Ναι, έπεσαν τα καράβια μου έξω!

Γαλάτεια: Εντάξει αγοράκι! Μη βαράς! Πάμε Αργεία, εγώ φταίω, που ασχολήθηκα μ' έναν... αγροίκο!

Επόμενο σημείο → Ανηφορίζοντας τον λόφο του Αγοραίου Κολωνού – Αν δεν έχεις μιλήσει με τη Γαλάτεια (26) IF

Ανηφορίζοντας τον λόφο του Αγοραίου Κολωνού – Αν έχεις μιλήσει με τη Γαλάτεια (23)

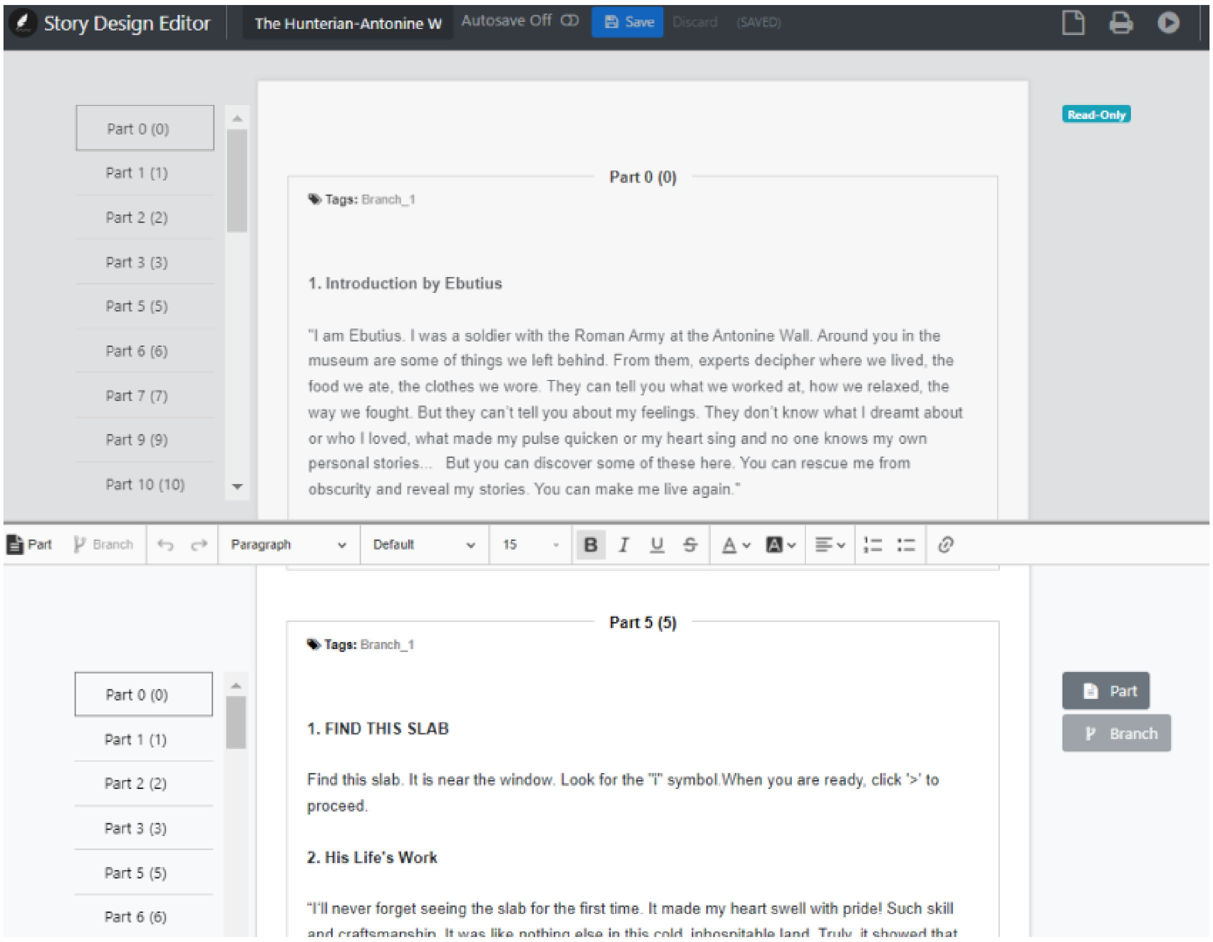
Tags: Eukratis

Δύο μέρες αργότερα, την ώρα που ο ήλιος μεσουρανεί...

Ο Νικοκλής, ο αφέντης σου, κι εσύ ανηφορίζετε προς τον ναό του Ηφαίστου. Περνάτε ανάμεσα από τα εργαστήρια των χαλκουργών, που είναι μαζεμένοι εδώ, γύρω από το ναό του προστάτη τους, του θεού μεταλλουργού Ηφαίστου. Κατευθύνεστε στο εργαστήριο του Ευκράτη, μετοίκου, εμπόρου ασπίδων. Συζητάτε.

Figure 25: Screenshot from the Story Maker Editor

Tags can be added to Parts or Branches and can be used in a Branch condition to determine whether to show or hide a specific Branch. The conditions are determined by evaluating boolean constraints based on tags encountered in the experience.



Part 0 (0)

Tags: Branch_1

1. Introduction by Ebutius

"I am Ebutius. I was a soldier with the Roman Army at the Antonine Wall. Around you in the museum are some of things we left behind. From them, experts decipher where we lived, the food we ate, the clothes we wore. They can tell you what we worked at, how we relaxed, the way we fought. But they can't tell you about my feelings. They don't know what I dreamt about or who I loved, what made my pulse quicken or my heart sing and no one knows my own personal stories... But you can discover some of these here. You can rescue me from obscurity and reveal my stories. You can make me live again."

Part 5 (5)

Tags: Branch_1

1. FIND THIS SLAB

Find this slab. It is near the window. Look for the "i" symbol.When you are ready, click '>' to proceed.

2. His Life's Work

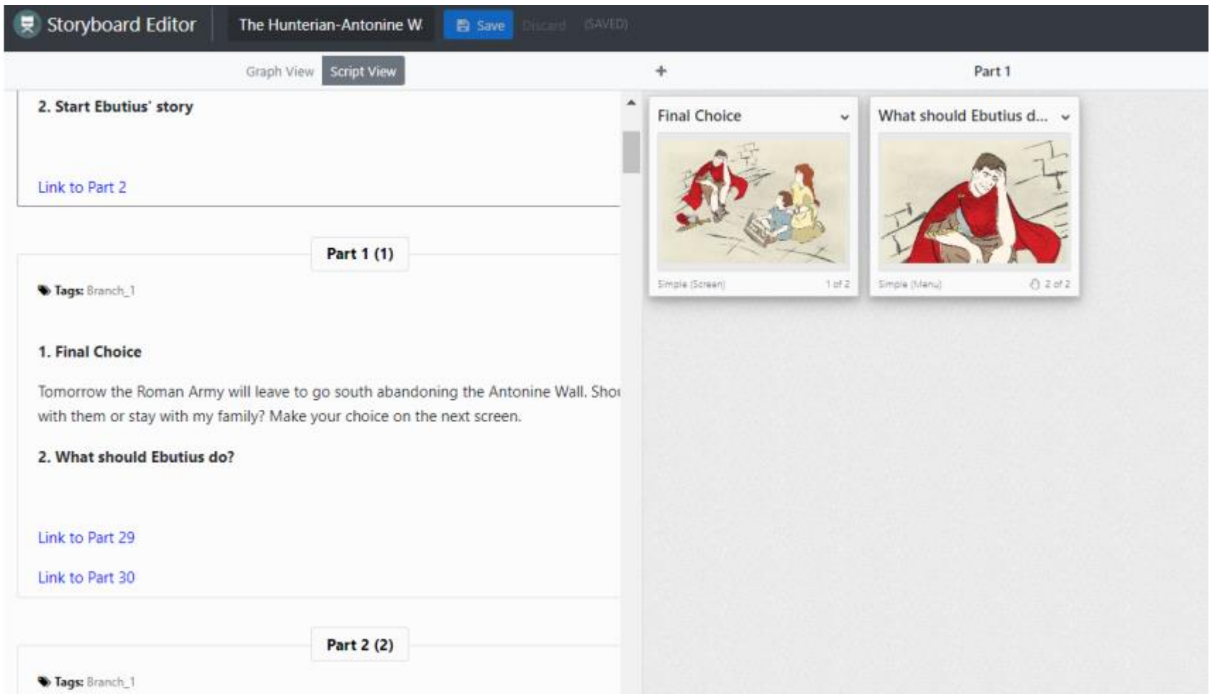
"I'll never forget seeing the slab for the first time. It made my heart swell with pride! Such skill and craftsmanship. It was like nothing else in this cold, inhospitable land. Truly, it showed that

Figure 26: Screenshot from the Story Maker Editor

2.2.2 The Storyboard Editor

The Storyboard Editor covers the production of the final experience. During the production phase, all Parts of the script are materialized through the creation of a set of Screens.

The authors have access to a selection of templates when creating Screens to implement a Part of their story. These templates simplify the creation process by eliminating the need for designing and programming user interface elements. The templates offered range from basic ones for combining text, images, and audio to more advanced templates catering to the common needs of cultural heritage experience creators.



2. Start Ebutius' story

Link to Part 2

Part 1 (1)

Tags: Branch_1

1. Final Choice

Tomorrow the Roman Army will leave to go south abandoning the Antonine Wall. Shou
with them or stay with my family? Make your choice on the next screen.

2. What should Ebutius do?

Link to Part 29

Link to Part 30

Part 2 (2)

Tags: Branch_1

Figure 27: Screenshot from the Storyboard Editor

For the development of the Unity tool, the main focus was on the Story Design Editor, since this is the component responsible for the structure of a story.

3. RELATED WORK

To develop a useful tool for other developers, an important step is to search for similar software that already exists. This way, we can get inspiration and focus on the helpful characteristics of this software while avoiding the aspects that may not be user-friendly and can be improved. In this chapter, we will present the tools Ink, Twine, and Yarn.

3.1 Ink

Ink was developed by Inkle Studios and is designed to create branching narratives and dynamic dialogue systems in games. It allows users to write interactive scripts using a text-based markup language. The scripts are then parsed and executed within the game to provide players with different story paths and dialogue options based on their choices. In order to use it in Unity, a Unity package must be downloaded and installed. This package includes a visual editor to write and organize one's Ink scripts, a runtime component to parse and execute the scripts, and a scripting API to communicate between Unity and Ink.

Some of the key features of Ink is that it allows the user to create stories with multiple branching paths, where the player's choices determine the direction of the narrative and the ability to track and manipulate data through provided variables, enabling dynamic storytelling and conditional branching based on the game state.

There are several difficulties with using Ink. Firstly, it lacks visual editing within Unity. While Ink offers a graph-based visual editor called Inky, it is a separate tool from Unity. This means that the user needs to switch between the Inky editor and Unity for editing and testing, which can disrupt workflow and create additional overhead. Also, although Ink provides basic programming capabilities such as variables, conditions, and loops, it may not be suitable for designing complex game mechanics or logic outside of narrative and dialogue systems [23].

3.2 Twine

Twine is another popular tool for creating interactive storytelling experiences, similar to Ink but with some notable differences. It is an open-source and user-friendly tool that allows users to create branching narrative games without requiring programming knowledge.

A major positive aspect of Twine is that it provides a visual, node-based editor that allows users to create and connect passages representing different sections of their story. To build an interactive narrative, the user can add text, links, and choices within each passage. Also, it is suitable for users with little or no programming experience, as its interface is designed to be user-friendly.

On the other hand, while Twine is primarily designed for interactive storytelling and branching narratives, it may not offer the same level of programming flexibility or integration with game engines as tools like Ink, limiting the resulting product without complex gameplay mechanics or interactions [24].

3.3 Yarn

Yarn is a lightweight and flexible open-source tool for creating interactive dialogue systems and narrative branching in games. It is often used as an alternative to Ink or Twine, offering its own unique set of features.

Such features include the possibility of being integrated into various game engines, including Unity and Unreal Engine. It also comes with visual node-based editors that provide a graphical interface to create and organize dialogue trees. Additionally, Yarn supports variables, expressions, and conditions, enabling dynamic dialogue and narrative based on game state and player choices. The developer can store and manipulate data within Yarn, making it versatile for creating complex and adaptive dialogue systems.

On the negative aspects, while Yarn has visual node-based editors, the visual editing capabilities may not be as advanced or user-friendly as dedicated visual scripting tools like Twine. Also, depending on the size and complexity of a Yarn project, performance may be a concern as handling large dialogue trees with numerous variables and expressions can impact runtime performance [25].

3.4 Conclusion

Researching related software helps identify the features that are important to focus on and the aspects that must be avoided or improved. More specifically, the Unity tool should be implemented to allow data manipulation by storing and providing the story's data from StoryMaker. Ideally, it should contain a node-based visual graph that presents paths and choices that is readable and user-friendly. Also, it should be well-integrated and compatible with the Unity engine so that it follows Unity's logic and interface. This way, the tool will not limit Unity's features for complex game mechanics. Additionally, it would be positive if the tool was designed to be easy for beginners but also offers features for advanced developers, such as code for further development. Lastly, to achieve good performance, the graph's data should be stored after the graph is generated while the tool is open and not have the graph rerendered when the user switches between windows.

4. SPECIFICATIONS AND DESIGN

When it comes to the development of every software, a good practice is to start with describing the specifications of it and providing designs through mockups to assist with the implementation of the project. After investigating similar software and pointing out good features and weaknesses, the designing of the tool, which is presented in this chapter, can be made with more clear targets.

4.1 Specifications

To determine how this tool should be implemented, it is first essential to specify the target group of the users it addresses. Since this thesis aims to provide a game development tool, the first characteristic of this target group is the occupation or interest in game development. Also, since it is made with the purpose to bridge the gap between authors and game developers in order to provide a solution that separates the two professions while allowing the developer to use complex game mechanics, this tool is mainly addressed to experienced developers that may already collaborate with teams that develop interactive experiences.

4.1.1 User Story

After describing the user groups that the tool targets, we can now present a user story that will determine the tool's specifications. This scenario considers that it is conducted by a user with coding experience and is occupied with game development.

- The user enters their credentials to get access to StoryMaker's stories.
- After the user chooses the desired story, the tool imports data into C# objects.
- The user gets a visual of the story's layout through a graph.
- The user navigates through the story in a user-friendly environment.
- Having access to the graph, the user can view detailed data of the story, such as choices, paths, and texts.
- The user can choose to automate certain features of Unity.
- The user can access both premade code and code tailored for the specific story.
- The user can use the Unity editor while having the tool open for guidance to develop a video game.
- The user may choose to include the features provided by the tool in their own code and game while using other features of the Unity Engine.

4.2 Design

When designing software, having well-defined specifications plays a crucial role in the entire development process. Specifications serve as a blueprint that outlines the desired functionality, features, and behavior of the software. Thus, as a next step towards the start of the development is to design the features of the tool, based on the specifications.

Firstly, there needs to be a window where the user will enter their credentials and choose a story to view on the graph from a list of stories. When it comes to the graph, it has to consist of nodes where each node should have a list of details. In order to provide the user with a set of functionalities that can assist with the development of their video game in Unity, there can be a generation of Unity scenes that represent the nodes from a button inside the graph window and/or a button in every node so that there can be separate scene generation for each node. Lastly, another button for scripting generation can create a C# file with a class that represents the specific story that is opened in the graph, giving access to the story's data in the form of C# classes.

Since the tool is now designed with features that create files automatically, it is important to design the folder structure where these files will be. This structure should be clear and organized. Firstly, a folder named “Narralive” will be created. Inside this folder, there will be a dedicated folder for every story with scenes and/or scripts generated by the tool, with each folder having the story's title. Inside every story's folder, there will be two other folders, a “Scenes” and a “Scripts” and each of these folders will be created after the user chooses to use the tool's scene and/or script generation separately.

When it comes to creating mockups for the tool, the main focus is on the way a graph would present the data of the story. The basic idea is to base the Unity graph on the one produced in the Story Maker.

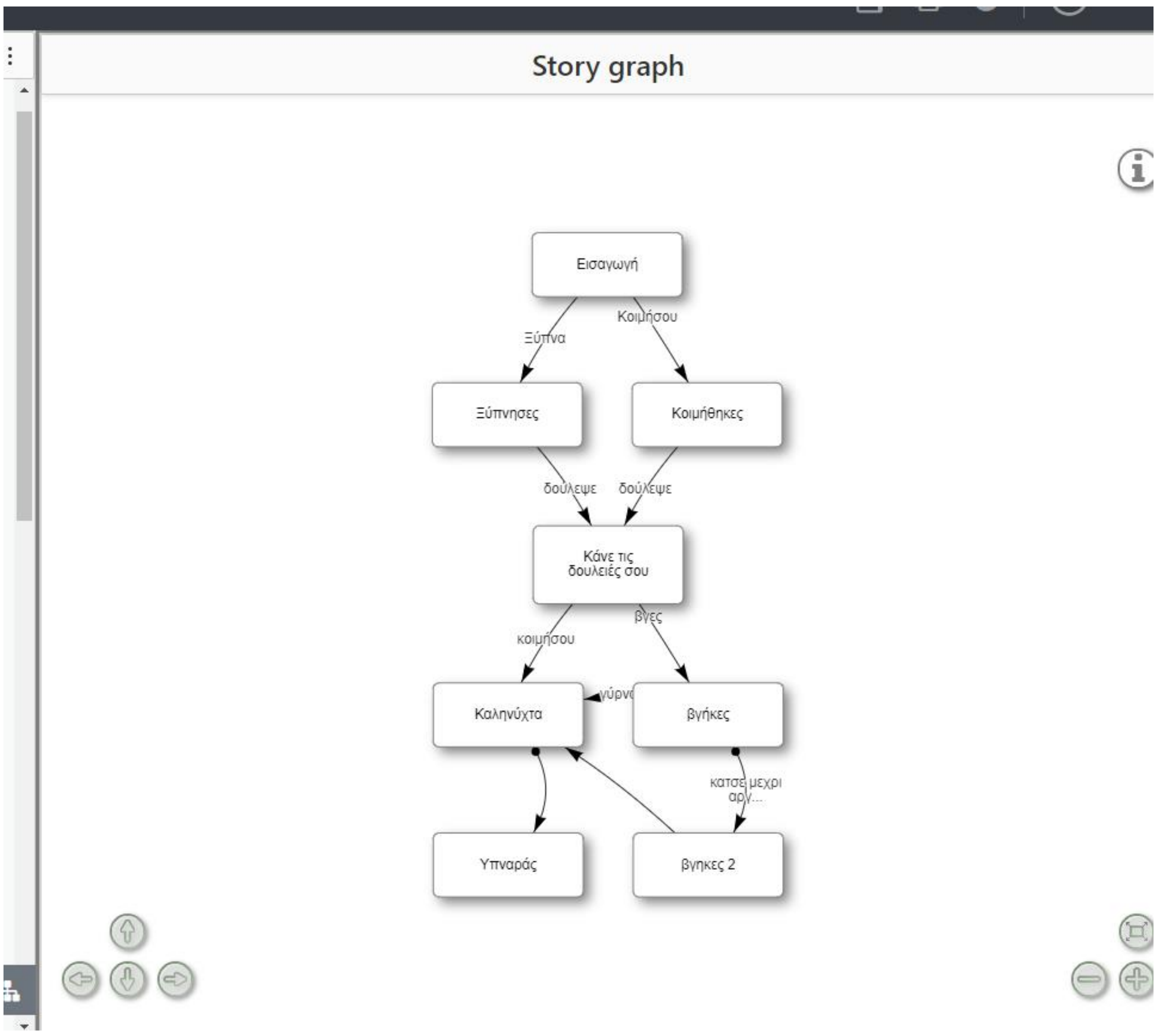


Figure 28: Example of a graph in Story Maker

Since the user will need to have adequate visual access to Unity's interface while working on their project, the creation of the graph was designed to appear in a different window after importing the story. In this way, the user can open the graph window whenever needed, and after doing so, they can go back to the main Unity window. The initial design for the graph interface is a window featuring a grid.

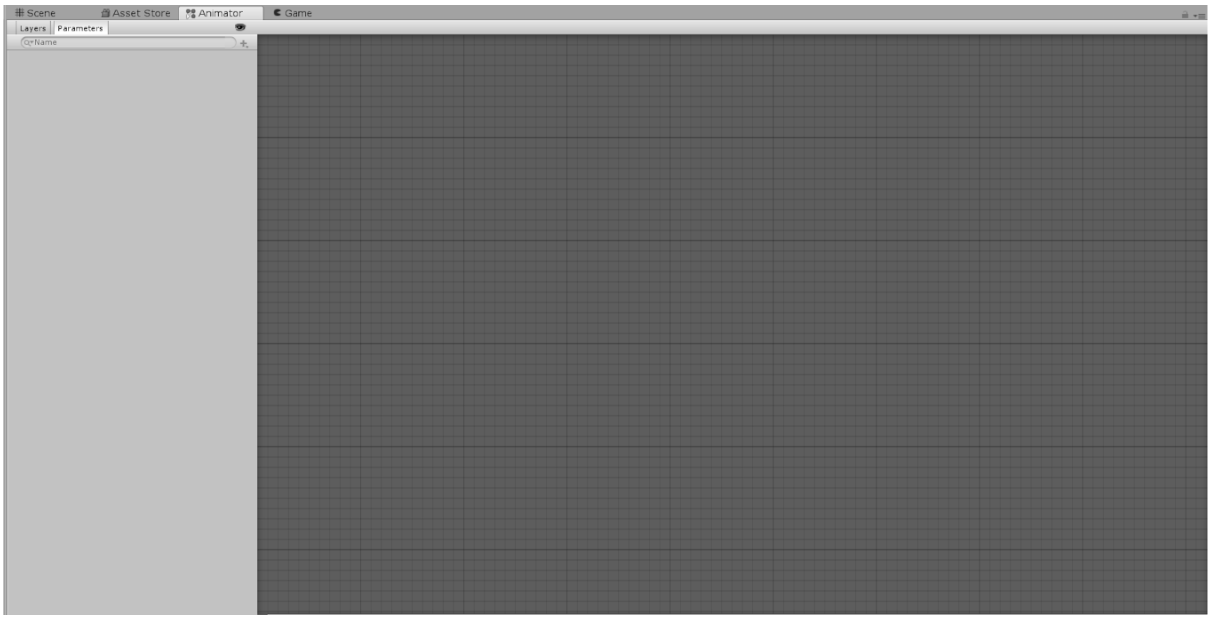


Figure 29: Example of a grid in Unity

Next step is to visualize how the story will be viewed on the graph. Since the story is made of parts, with each part consisting of a text and a number of choices, the most suited visual for the graph is a set of nodes connected with each other through the choices.

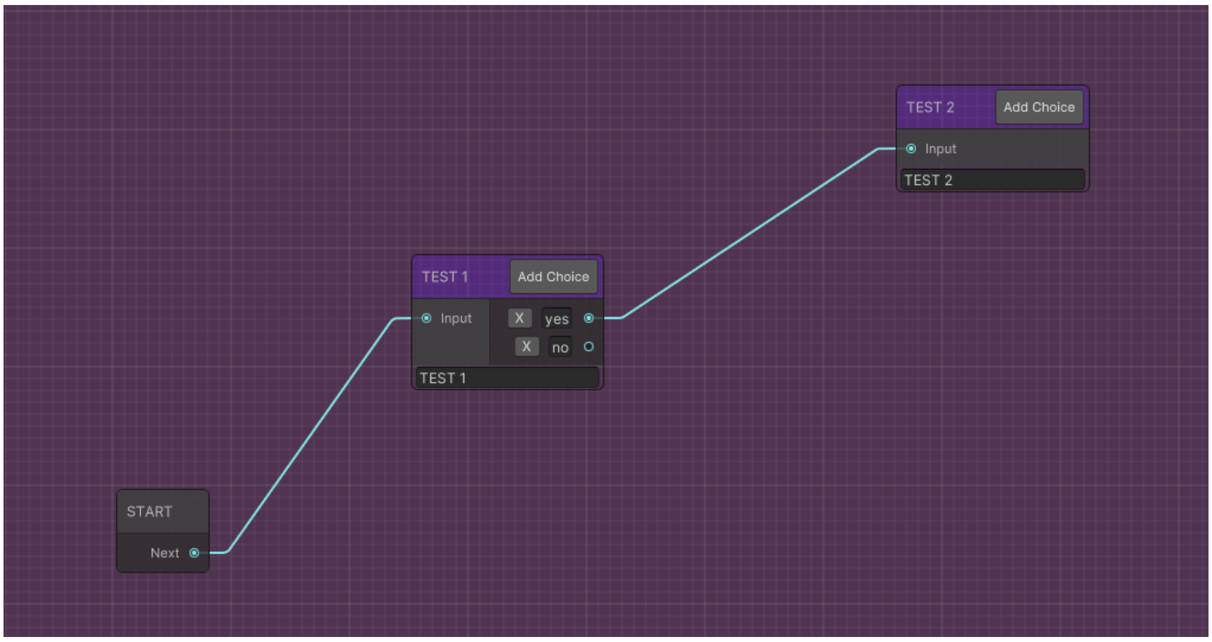
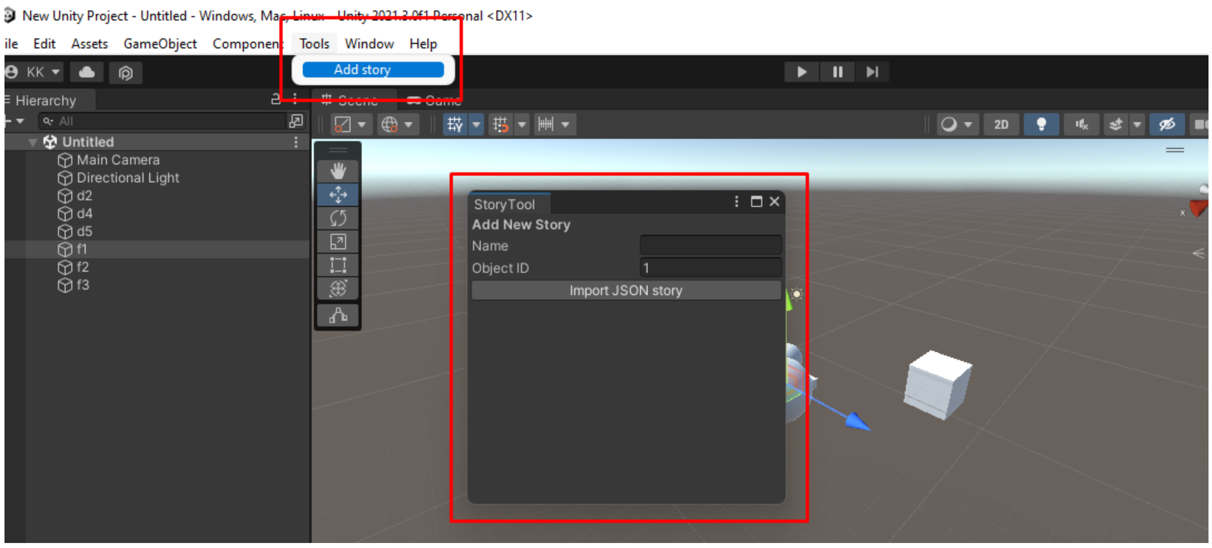


Figure 30: Screenshot of the graph in Unity

Another type of window essential for the tool is a window where the user can enter the needed credentials at the beginning of their project to import data from Story Maker. A basic mockup that was created in the process of designing the tool depicts a simple window, opening from Unity’s tabs on the upper left of the software. In the development process, this window will be enriched with the specified features of credential entering and a dropdown list of stories to choose from.



StoryTool

Add New Story

Name

Object ID 1

Import JSON story

Figure 31: Screenshot of the "Add story" window in Unity

5. TECHNOLOGIES USED

After having a solid idea about the genre of Visual Novels as well as the specification of the tool, to continue with the implementation, the next step is to consider the technologies that will be used for it. This chapter is separated into two parts, one regarding the Unity Engine and one regarding the JSON format, which is the format that contains the story's data from Story Maker.

5.1 The Unity Engine

This portion of the chapter aims to introduce the Unity Engine, one of the most popular and influential game engines in the industry, while also providing a broader understanding of game engines in general.

5.1.1 A brief introduction on Game Engines

A game engine is a software development environment designed for creating video games. It provides a set of tools and technologies that developers can use to build games more efficiently. These tools and technologies often include game mechanics, game logic, graphics rendering, physics, audio, networking, and more.

There are many different game engines available, each with its own unique set of features and capabilities. Some popular game engines include Unity, Unreal Engine, CryEngine, and GameMaker. They are an essential tool for game developers, providing tools and technologies that allow them to build games more efficiently and create more immersive gaming experiences.

5.1.2 The popularity of Unity

Unity is a cross-platform game engine that is widely used in the development of video games. It was first released in 2005 and has since become one of the most popular game engines in the industry [26].

Figure 32: Unity's logo

An important aspect of Unity is its ability to support the development of games for a wide range of platforms, including PC, console, mobile, and virtual reality (VR). It has a powerful set of tools and technologies that allow developers to build games more efficiently, including a game engine, graphics rendering, physics, audio, networking, and more.

In terms of graphics, Unity supports the development of both 2D and 3D games. It includes a variety of rendering options, including support for hardware acceleration, lighting, shadows, and post-processing effects.

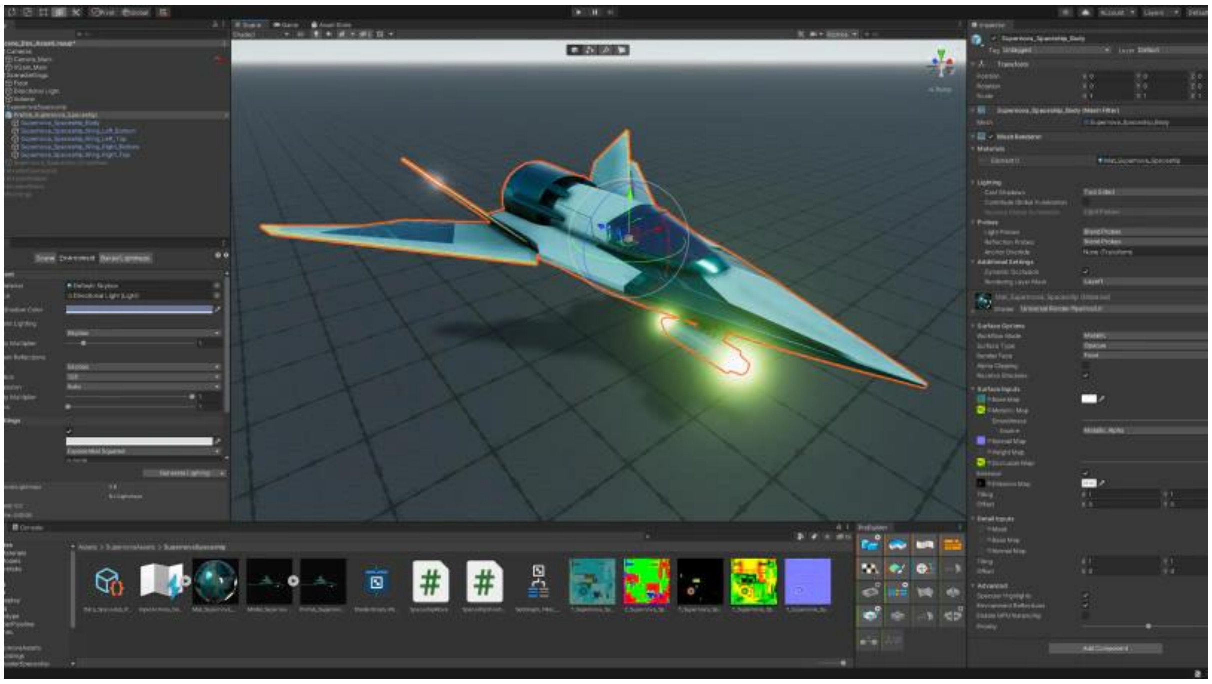


Figure 33: Example of Unity

Unity also includes a robust physics engine that allows developers to create realistic and immersive gameplay. It includes support for various types of physics, including rigid body, soft body, and cloth simulation.

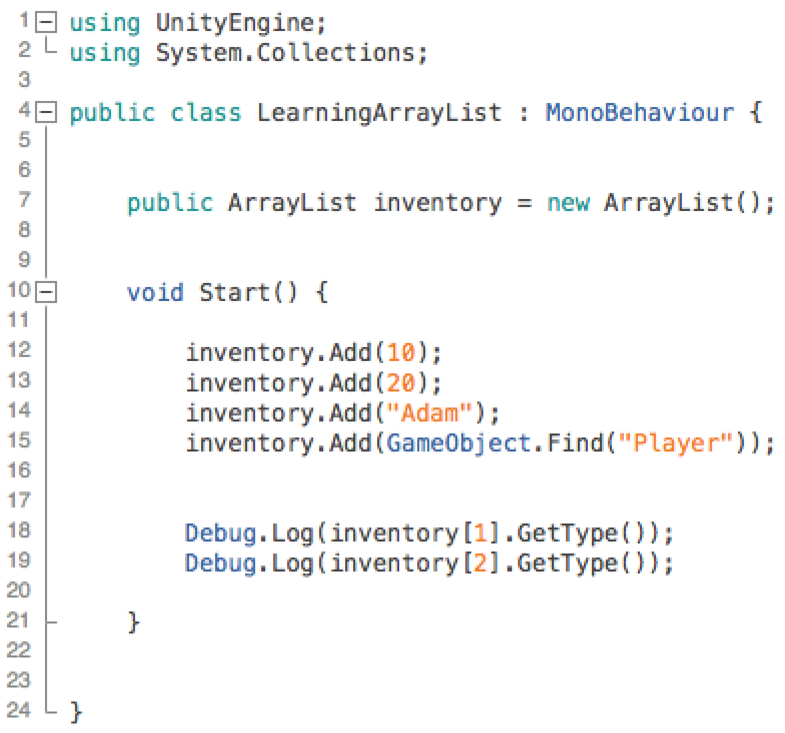
One of the key strengths of Unity is its ease of use. It has a user-friendly interface and a large community of developers who contribute to its development. This makes it a popular choice for beginners and small indie studios.

Overall, Unity is a powerful and flexible game engine that is widely used in the development of games for a variety of platforms. Its ease of use, wide range of features, and large community of developers make it a popular choice for game developers of all skill levels. Based on this, Unity ended up being the most fitted choice for the development of the tool that the thesis regards to, especially due to the usage of the asset store.

5.1.3 How Unity Works

The Unity engine uses a component-based architecture, which means that objects in a Unity project are made up of smaller pieces called components. These components can be added to objects to give them specific behaviors or abilities, such as the ability to move or interact with other objects. It also includes a visual scripting system called Unity Script, which allows developers to create behaviors and interactions without writing code. This makes it easier for developers with little programming experience to create interactive content using the Unity engine.

The programming language used in the engine is C#, a general-purpose programming language developed by Microsoft that is widely used for building various applications, including games. In the Unity engine, C# is used to create scripts that define the behavior of game objects and the overall gameplay logic of a project. These scripts can be attached to game objects in the Unity editor, and they can be used to control the behavior of those objects at runtime.



```
using UnityEngine;
using System.Collections;

public class LearningArrayList : MonoBehaviour {


    public ArrayList inventory = new ArrayList();


    void Start() {

        inventory.Add(10);
        inventory.Add(20);
        inventory.Add("Adam");
        inventory.Add(GameObject.Find("Player"));


        Debug.Log(inventory[1].GetType());
        Debug.Log(inventory[2].GetType());

    }


}
```

Figure 34: Example of C# code in Unity

C# scripts in Unity can access a wide range of features and functionality provided by the engine, including input, physics, rendering, and networking. In addition to the standard features of the C# language, the Unity engine also provides a range of specialized classes and functions that are specific to game development. These include classes for managing game objects, handling input, and accessing engine features such as audio and animation.

5.1.4 Unity Asset Store

The Unity Asset Store is a platform where developers and artists can share and sell their assets, tools, and plugins to a vast community of Unity developers. The Asset Store offers a variety of assets, ranging from 2D and 3D models, animations, audio and sound effects, to complete game templates and source code.

The Asset Store provides access to a large library of pre-made assets that can save developers a considerable amount of time and effort. For example, a developer can purchase a high-quality 3D model to use in their game instead of spending time creating one from scratch. The store also offers a wide range of tools, such as environment creation tools, character animation systems, and physics engines, to help developers quickly create and prototype their projects.

To access the Asset Store, Unity users simply need to open the Unity Editor and go to the Asset Store tab. From there, users can browse the available assets, download a free sample, or purchase the full asset. Once an asset is purchased, it is immediately available for use in the current Unity project.

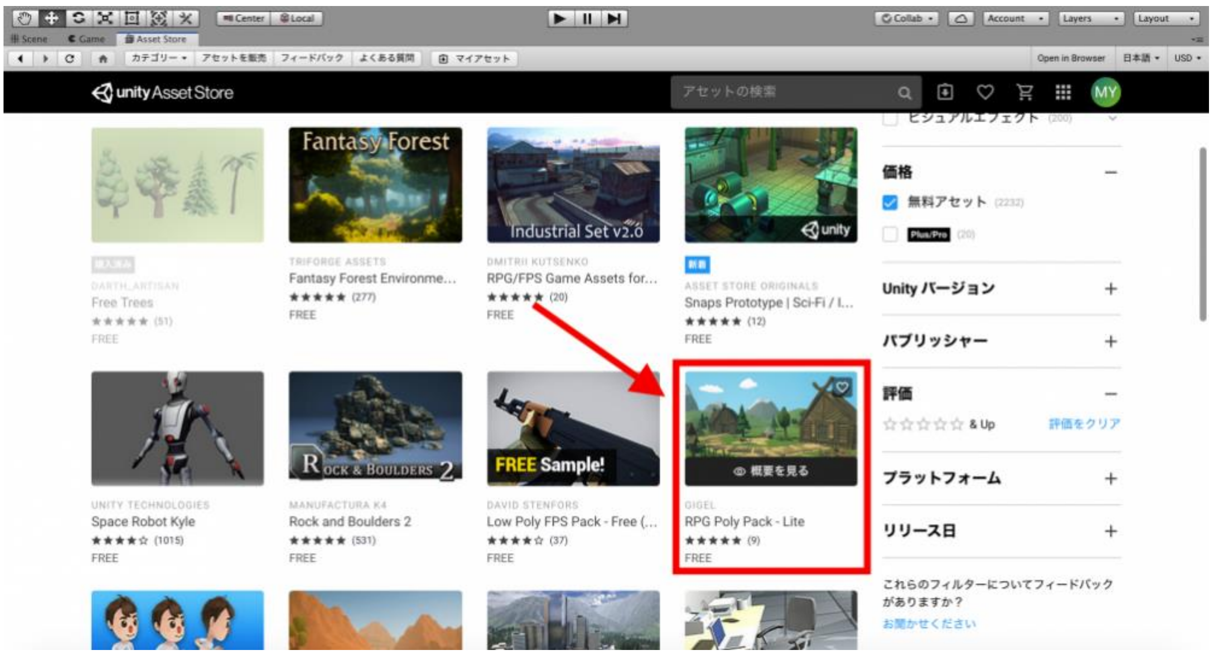


Figure 35: Screenshot of Unity's Asset store

5.2 The JSON format

Since the data of each story created with the StoryMaker is stored in a JSON which will then be used from the Unity tool to extract the story's data, it is important to get a good idea about this format. Also, since through this project we will try to manipulate and understand the JSONs that will be received from the StoryMaker, we ended up using Postman, a tool regarding web requests which will be discussed in this chapter's part as well.

5.2.1 What a JSON is

JSON, or JavaScript Object Notation, is a lightweight and widely used data interchange format. It is easy for both humans and machines to read and write, making it a popular choice for data exchange over the internet.

At its core, JSON is simply a text format that uses a specific set of rules for representing data structures. It was first introduced in 2001 and has since gained widespread adoption due to its simplicity, ease of use, and broad compatibility with different programming languages and platforms [27].

JSON is based on two basic structures: objects and arrays. An object is a collection of key-value pairs, where each key represents a property name and each value represents the property's value. For example, an object representing a person's details could have properties such as "name", "age", and "email".

An example of a JSON would be:

```
{
  "name": "John Smith",
  "age": 35,
  "email": "john.smith@example.com"
}
```

JSON data can also be nested, meaning that objects and arrays can contain other objects and arrays. For example, an object representing a shopping cart could have an array of items, where each item is an object with its own properties such as name, price, and quantity.

An example of a nested JSON is:

```
{
  "name": "John Smith",
  "age": 35,
  "email": "john.smith@example.com",
  "shoppingCart": [
    {
      "name": "Product 1",
      "price": 10.99,
      "quantity": 2
    },
    {
      "name": "Product 2",
      "price": 7.99,
      "quantity": 1,
      "details": {
        "weight": 1.5,
        "dimensions": {
          "height": 10,
          "width": 5,
          "depth": 2
        }
      }
    },
    {
      "name": "Product 3",
      "price": 5.99,
      "quantity": 3
    }
  ]
}
```

In this example, we have an object that represents a person's details, including their name, age, and email address. This object also contains a nested array called "shoppingCart", which contains three objects representing items in the person's shopping cart.

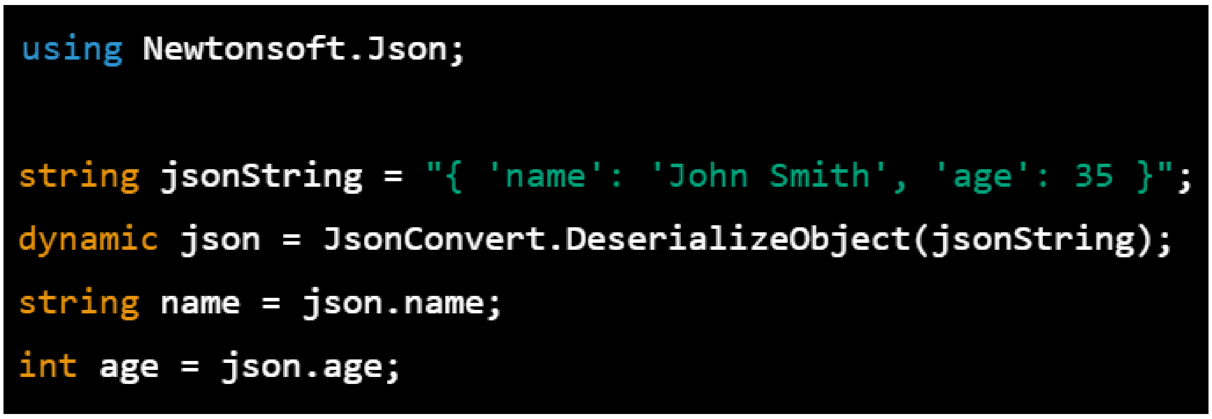
The first item in the shopping cart is a simple object with properties for name, price, and quantity. The second item is a more complex object with additional nested properties under the "details" key. These nested properties include the weight of the item and its dimensions, which are represented as a nested object.

5.2.2 Unity and JSON

Due to JSON’s trait of carrying data with ease, while remaining lightweight, it is more popular for use in web development, where the need of transporting data efficiently is important. Especially for APIs (Application Programming Interfaces), which allow different systems to communicate with each other over the internet, this particular format is widely used.

However, even though Unity is centered in game development, the engine does work with JSON, with several libraries available to parse and generate JSON data. For example, Newtonsoft.Json is a popular JSON framework for .NET, and it can be used to work with JSON in the game engine. Using Newtonsoft.Json in Unity is as simple as downloading the framework from the Newtonsoft website and adding it to a Unity project as a package. One of the advantages of using Newtonsoft.Json in Unity is its flexibility and ease of use. The framework provides a wide range of methods for working with JSON data, including parsing and generating JSON strings, querying and manipulating JSON data, and serializing and deserializing JSON to and from C# objects.

An example of how to parse a JSON string in Unity using Newtonsoft's JsonConvert.DeserializeObject method is shown in Fig. 36.



```
using Newtonsoft.Json;

string jsonString = "{ 'name': 'John Smith', 'age': 35 }";
dynamic json = JsonConvert.DeserializeObject(jsonString);
string name = json.name;
int age = json.age;
```

Figure 36: Example of code that parses a JSON string using Newtonsoft

Another advantage of using Newtonsoft.Json in Unity is its performance. The framework is highly optimized and can handle large JSON datasets quickly and efficiently.

Based on the above, the Newtonsoft framework is an ideal choice for the needs of the thesis as it can assist with the import of the story's data from the Story Maker.

5.2.3 Postman

A useful software used to simplify the JSON management for the project is Postman. Postman is a popular API development tool that simplifies the process of building, testing, and documenting APIs. It allows developers to make HTTP requests, test responses, and inspect API data. The tool supports a variety of HTTP methods while supporting various authentication methods such as OAuth 2.0, Basic Auth, and API keys [28].

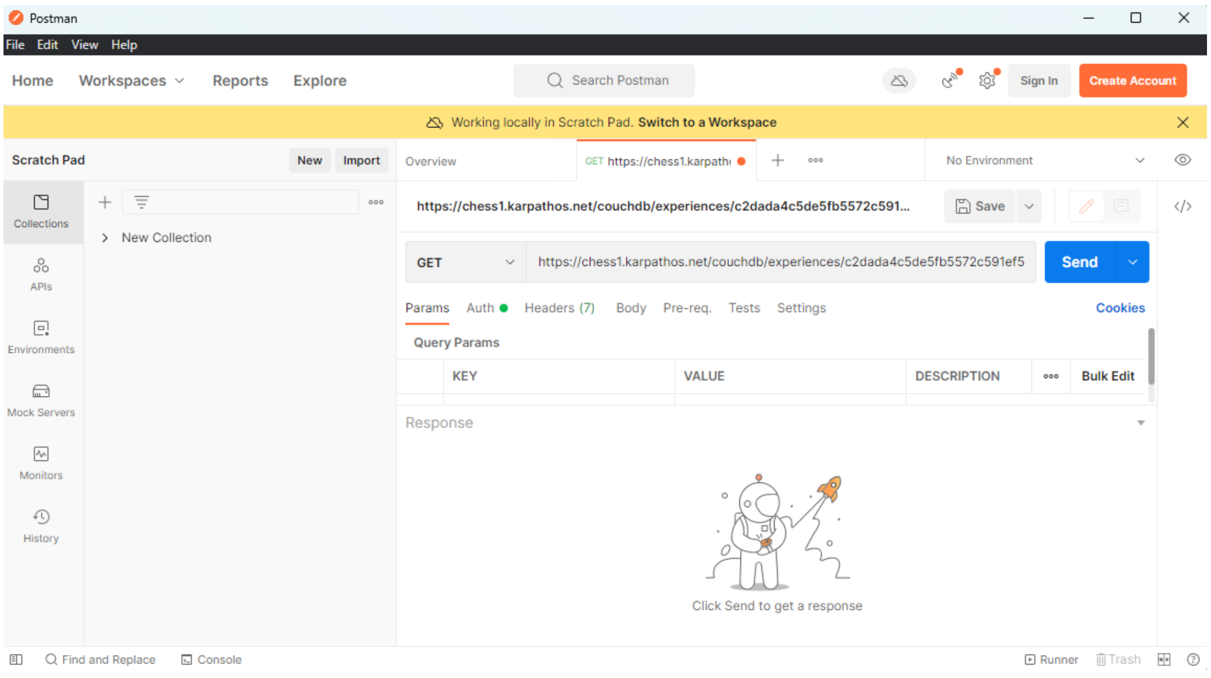
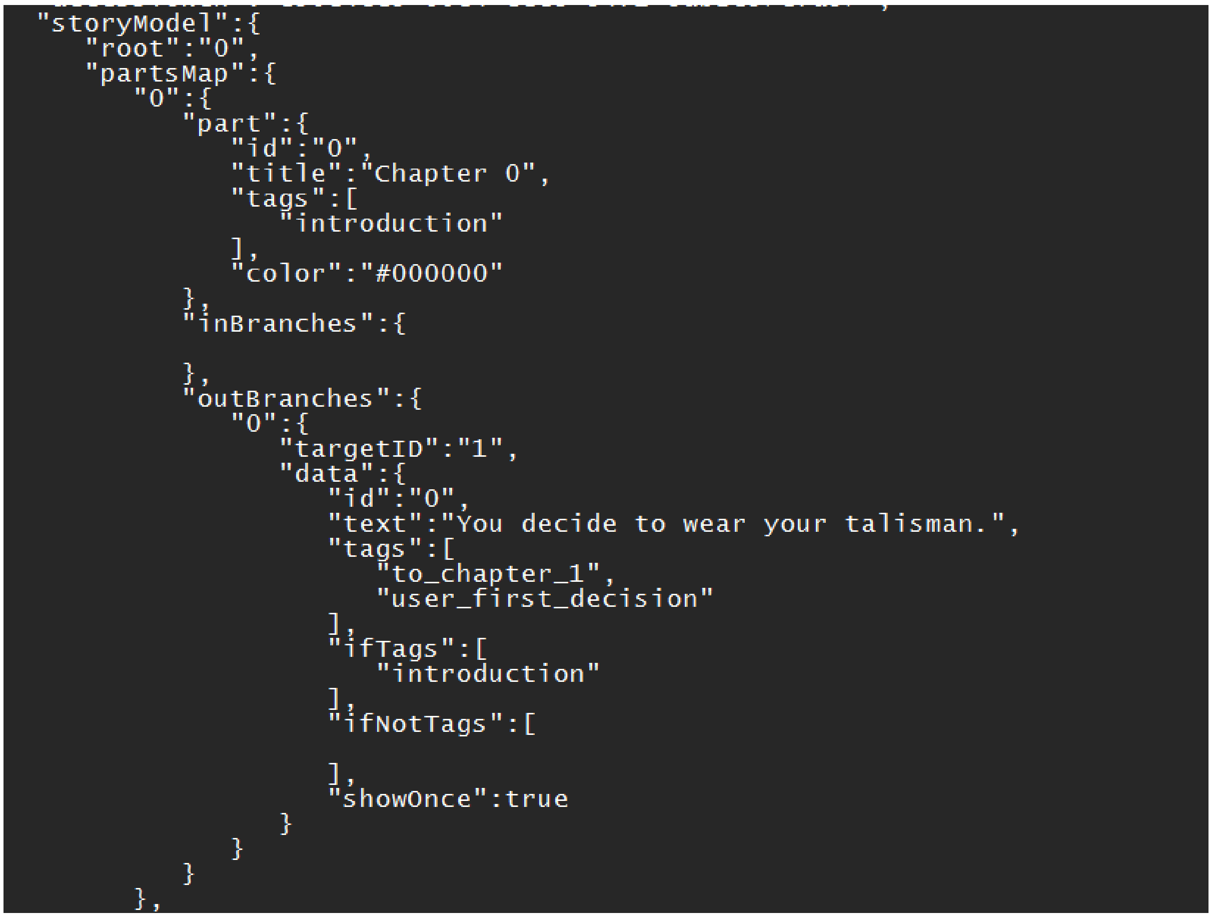

Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Sign In
Create Account
Working locally in Scratch Pad. Switch to a Workspace
Scratch Pad
New
Import
Overview
GET https://chess1.karpath
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
New Collection
https://chess1.karpathos.net/couchdb/experiences/c2dada4c5de5fb5572c591...
Save
GET
https://chess1.karpathos.net/couchdb/experiences/c2dada4c5de5fb5572c591ef5
Send
Params Auth Headers (7) Body Pre-req. Tests Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Response
Click Send to get a response
Find and Replace
Console
Runner
Trash

Figure 37: Screenshot of Postman

One of the main benefits of using Postman is its ability to streamline the API development process. With the tool, developers can quickly create API requests and test them in a user-friendly interface. Postman also offers a powerful testing framework that can be used to create automated tests for APIs. This feature is particularly useful for ensuring that APIs function as expected before they are deployed to production environments.

Postman was useful for this project by making web requests to the Story Maker from a user-friendly environment. The responses were then used both as dummy data to test the code and to validate that the web requests made through the code were correct in getting the desired data. An example of such JSON, produced by the Story Design Editor, would look like the following.



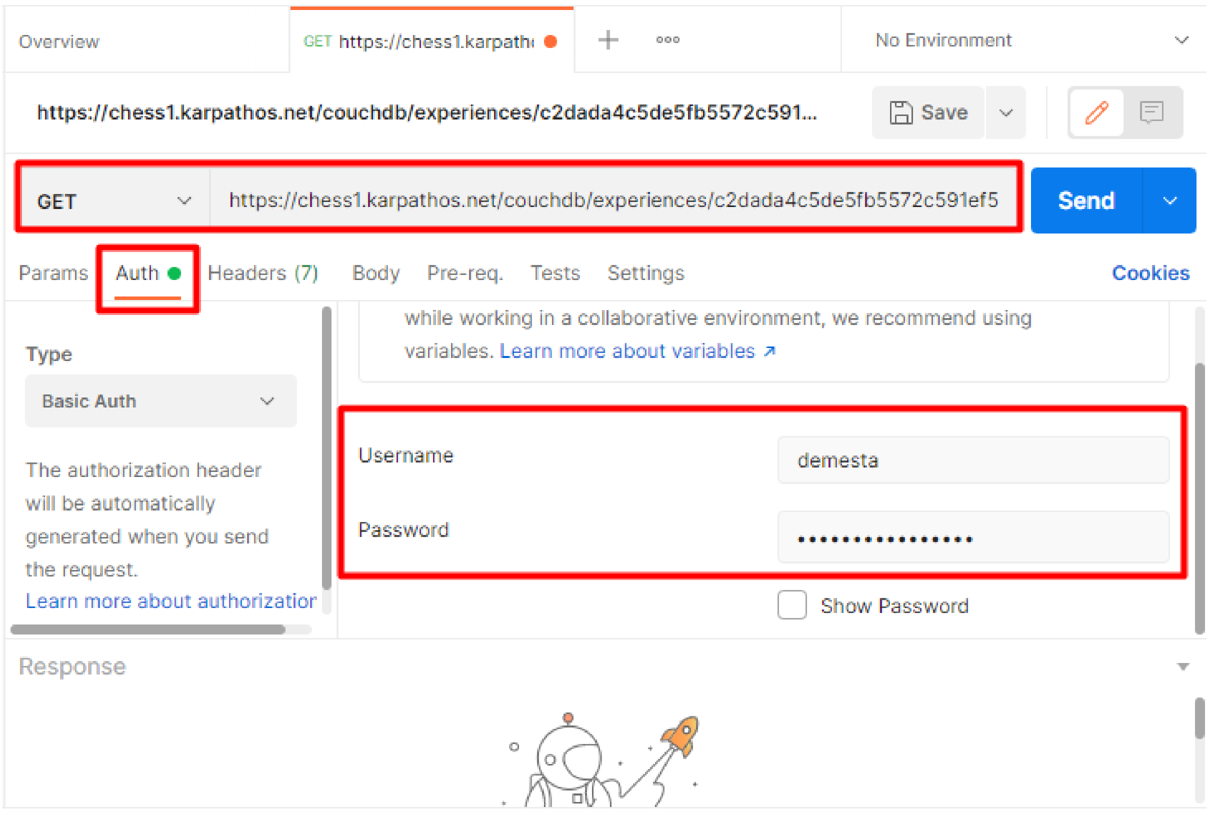
```
"storyModel":{
   "root":"0",
   "partsMap":{
      "0":{
         "part":{
            "id":"0",
            "title":"Chapter 0",
            "tags":[
               "introduction"
            ],
            "color":"#000000"
         },
         "inBranches":{

         },
         "outBranches":{
            "0":{
               "targetID":"1",
               "data":{
                  "id":"0",
                  "text":"You decide to wear your talisman.",
                  "tags":[
                     "to_chapter_1",
                     "user_first_decision"
                  ],
                  "ifTags":[
                     "introduction"
                  ],
                  "ifNotTags":[

                  ],
                  "showOnce":true
               }
            }
         }
      },
```

Figure 38: Example of StoryMaker's JSON

By providing the link to the Story Maker’s database as well as a user’s credentials, Postman would return the needed JSON.


Overview | GET https://chess1.karpath | No Environment

https://chess1.karpathos.net/couchdb/experiences/c2dada4c5de5fb5572c591...

GET https://chess1.karpathos.net/couchdb/experiences/c2dada4c5de5fb5572c591ef5

Params | Auth | Headers (7) | Body | Pre-req. | Tests | Settings | Cookies

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

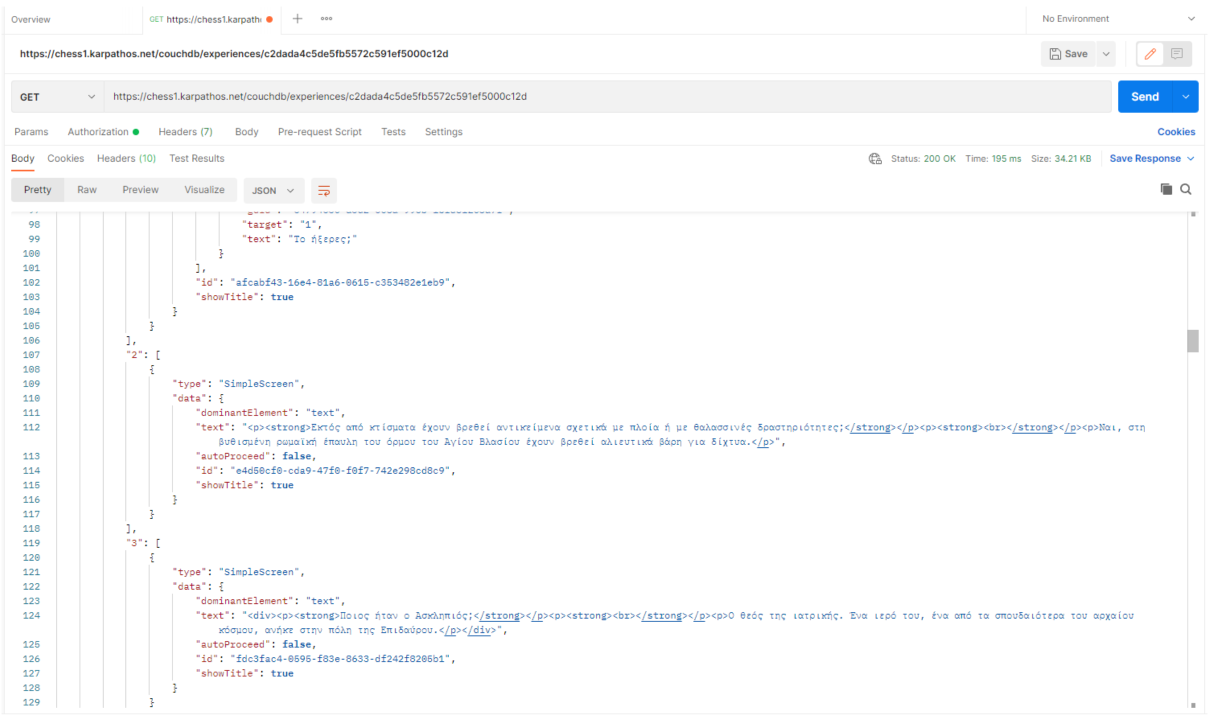
Username: demesta

Password: ••••••••••••••••

Show Password

Response

Figure 39: Screenshot of Postman's verification


Overview | GET https://chess1.karpath | No Environment

https://chess1.karpathos.net/couchdb/experiences/c2dada4c5de5fb5572c591ef5000c12d

GET https://chess1.karpathos.net/couchdb/experiences/c2dada4c5de5fb5572c591ef5000c12d

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings | Cookies

Body | Cookies | Headers (10) | Test Results — Status: 200 OK Time: 195 ms Size: 34.21 KB — Save Response

Pretty | Raw | Preview | Visualize | JSON

```
                        "target": "1",
                        "text": "Το ήξερες;"
                    }
                ],
                "id": "afcabf43-16e4-81a6-0615-c353482e1eb9",
                "showTitle": true
            }
        }
    ],
    "2": [
        {
            "type": "SimpleScreen",
            "data": {
                "dominantElement": "text",
                "text": "<p><strong>Εκτός από κτίσματα έχουν βρεθεί αντικείμενα σχετικά με πλοία ή με θαλασσινές δραστηριότητες;</strong></p><p><strong><br></strong></p><p>Ναι, στη
                    βυθισμένη ρωμαϊκή έπαυλη του όρμου του Αγίου Βλασίου έχουν βρεθεί αλιευτικά βάρη για δίχτυα.</p>",
                "autoProceed": false,
                "id": "e4d50cf0-cda9-47f0-f0f7-742e298cd8c9",
                "showTitle": true
            }
        }
    ],
    "3": [
        {
            "type": "SimpleScreen",
            "data": {
                "dominantElement": "text",
                "text": "<div><p><strong>Ποιος ήταν ο Ασκληπιός;</strong></p><p><strong><br></strong></p><p>Ο θεός της ιατρικής. Ένα ιερό του, ένα από τα σπουδαιότερα του αρχαίου
                    κόσμου, ανήκε στην πόλη της Επιδαύρου.</p></div>",
                "autoProceed": false,
                "id": "fdc3fac4-0595-f83e-8633-df242f8205b1",
                "showTitle": true
            }
        }
```

Figure 40: Screenshot of Postman after requesting and receiving a story from StoryMaker

6. THE DEVELOPMENT OF THE PLUG-IN

After designing the tool and doing extended research on the Visual Novels and how Unity and JSON work, the focal point can be shifted to the development of the tool on which the current thesis is based on. Therefore, this chapter focuses on describing the steps and process of developing and, finally, completing the tool.

6.1 Setting up

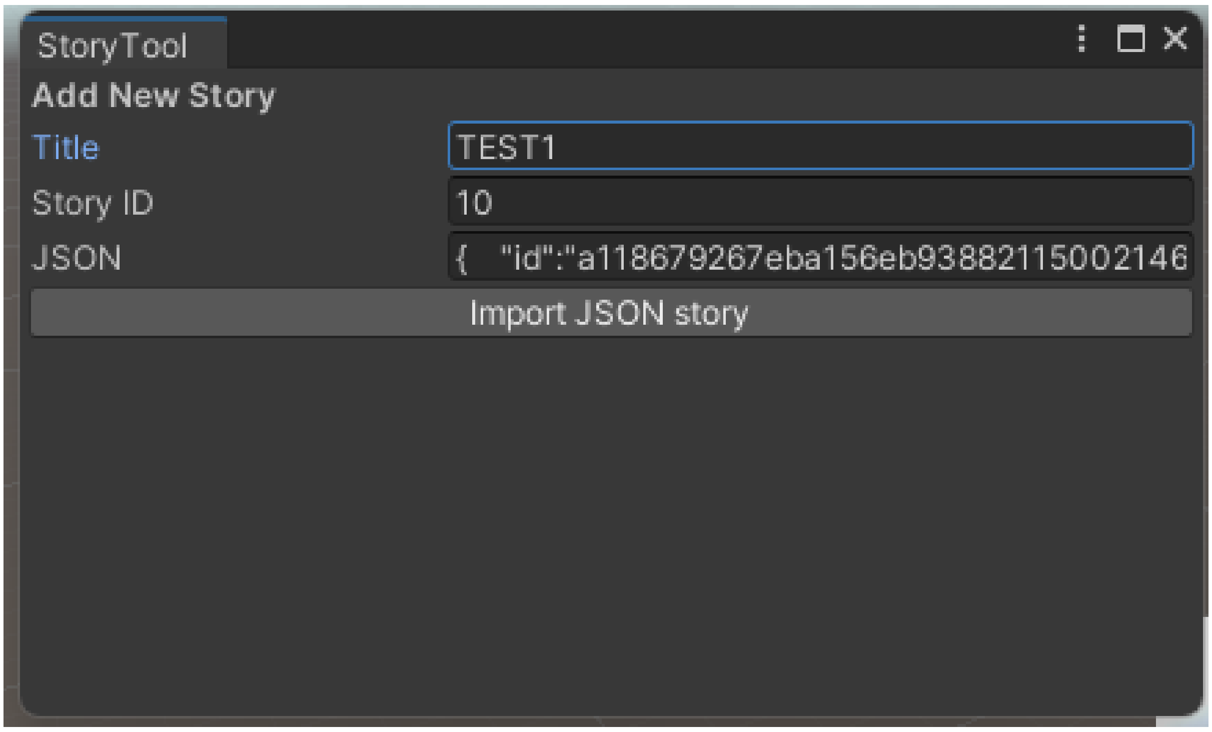
Before starting the development, it was first necessary to figure out the technical aspects and decide on the different possible ways some parts would be implemented. The most important step to begin working on the code was to import the JSON of a story into the Unity project.

Another critical point was to analyze the structure of the JSON generated by the Story Maker. Specifically, it was important to understand what part of the JSON was useful for the tool and what type of data were to work with. After analyzing the JSON it was concluded that the sub-JSON that contains the structure of the story is the “storyModel”. This sub-JSON works as a map for the different parts of the story. The first part is the beginning node, the root. Every part contains an id, a title, tags, a color code, inBranches and outBranches. The last two properties indicate the node linking from and to the node. Since the outBranches were proven enough to determine the connection between two nodes, it was decided that the inBranches would not be used in the code. An example of what type of data appear in the storyModel sub-JSON is the following:

```
"storyModel": {

          "root": "15",
          "partsMap": {
            "15": {
              "part": {
                "id": "15",
                "title": "Εισαγωγή",
                "tags": [],
                "color": "#000000"
              },
              "inBranches": {},
              "outBranches": {
                "19": {
                  "targetID": "16",
                  "data": {
                    "id": "19",
                    "text": "Ξύπνα",
                    "tags": [],
                    "ifTags": [],
                    "ifNotTags": [],
                    "showOnce": false
                  }
                },
                "20": {
                  "targetID": "17",
                  "data": {
                    "id": "20",
                              "text": "Κοιμήσου",
                              "tags": [],
                              "ifTags": [],
                              "ifNotTags": [],
                              "showOnce": false
                          }
                      }
                  }
              }
      }
```

Also, before reaching the point of turning the story’s data into C# classes, the implementation started with the creation of the window where the user can enter the needed credentials. As a first trial to test the behavior of the window, it was implemented by providing the JSON. Therefore, the window consisted of the title “StoryTool”, the description “Add new story”, a “Title” named text insertion, a “Story ID” number insertion” and the “JSON” text insertion. Finally, a button named “Import JSON story” would trigger, after being clicked, the code to read the input.


StoryTool

Add New Story

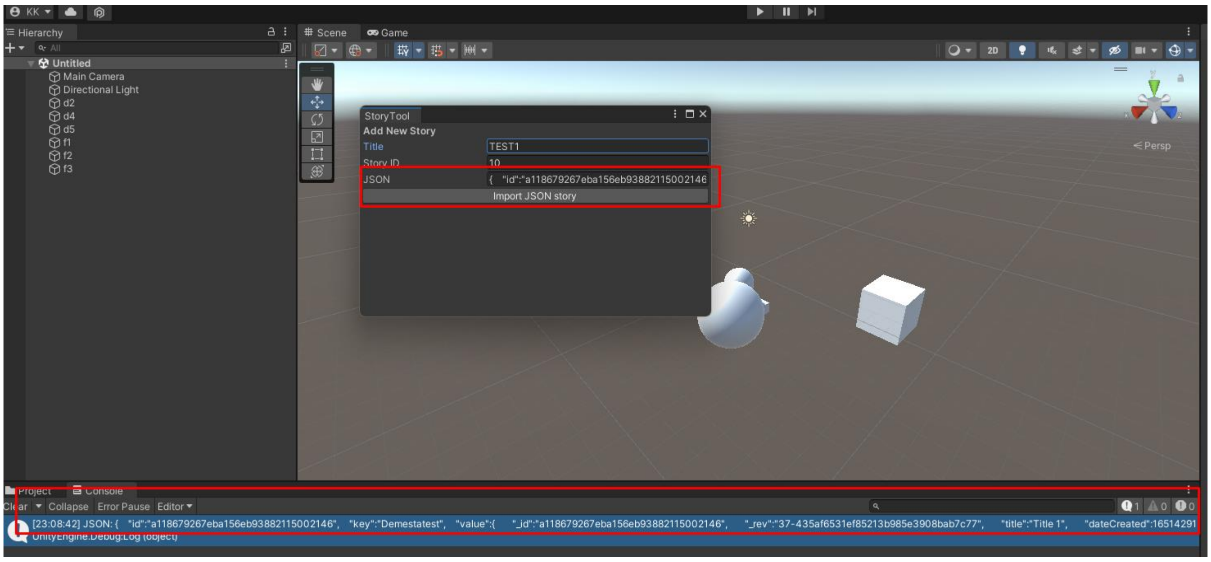
Title: TEST1

Story ID: 10

JSON: {   "id":"a118679267eba156eb93882115002146

Figure 41: Screenshot of a window in Unity that tests the JSON import

As a basic way to test if the triggering of the button and the reading of the user’s input worked as expected, a simple console message writing the JSON input was added in the code.

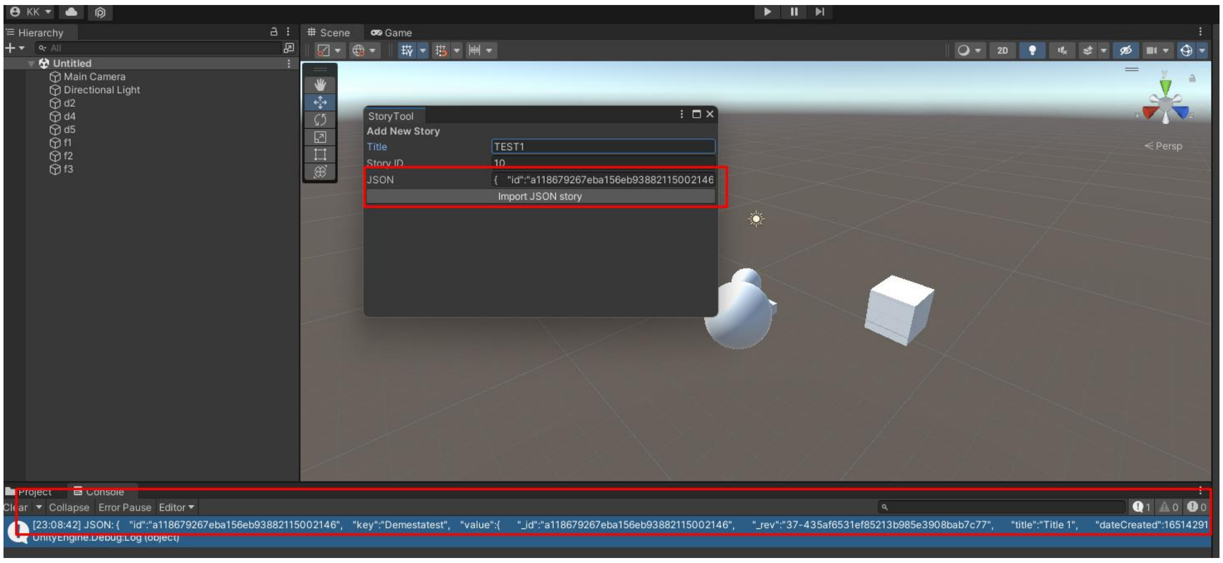


StoryTool
Add New Story
Title: TEST1
Story ID: 10
JSON: {   "id":"a118679267eba156eb93882115002146
Import JSON story

```
[23:08:42] JSON: {   "id":"a118679267eba156eb93882115002146",   "key":"Demestatest",   "value":{    "_id":"a118679267eba156eb93882115002146",    "_rev":"37-435af6531ef85213b985e3908bab7c77",    "title":"Title 1",    "dateCreated":16514291
UnityEngine.Debug:Log (object)
```

Figure 42: Screenshot of a window in Unity that tests the JSON import

To check that the results were correct, it was only needed to compare the JSON given on the “Add Story” window with the message displayed on Unity’s console.

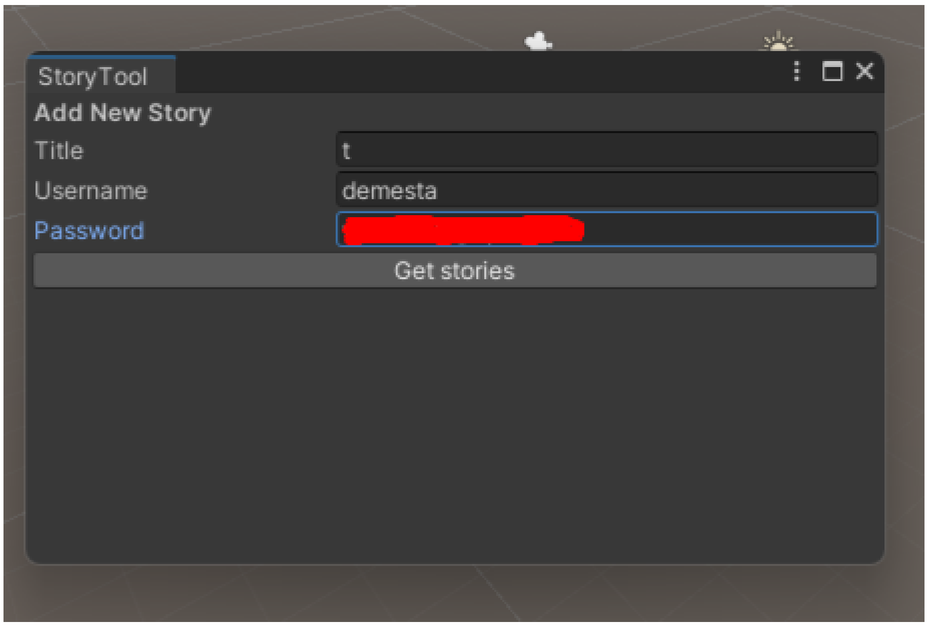


StoryTool
Add New Story
Title: TEST1
Story ID: 10
JSON: {   "id":"a118679267eba156eb93882115002146
Import JSON story

```
[23:08:42] JSON: {   "id":"a118679267eba156eb93882115002146",   "key":"Demestatest",   "value":{    "_id":"a118679267eba156eb93882115002146",    "_rev":"37-435af6531ef85213b985e3908bab7c77",    "title":"Title 1",    "dateCreated":16514291
UnityEngine.Debug:Log (object)
```

Figure 43: Screenshot of a window in Unity that tests the JSON import

Additionally, since it is mandatory, as designed, to provide username and password in order to get a successful response from the web request, after testing the web request functionality of C#, the “Add Story” window was changed to get the user’s credentials.



StoryTool

Add New Story

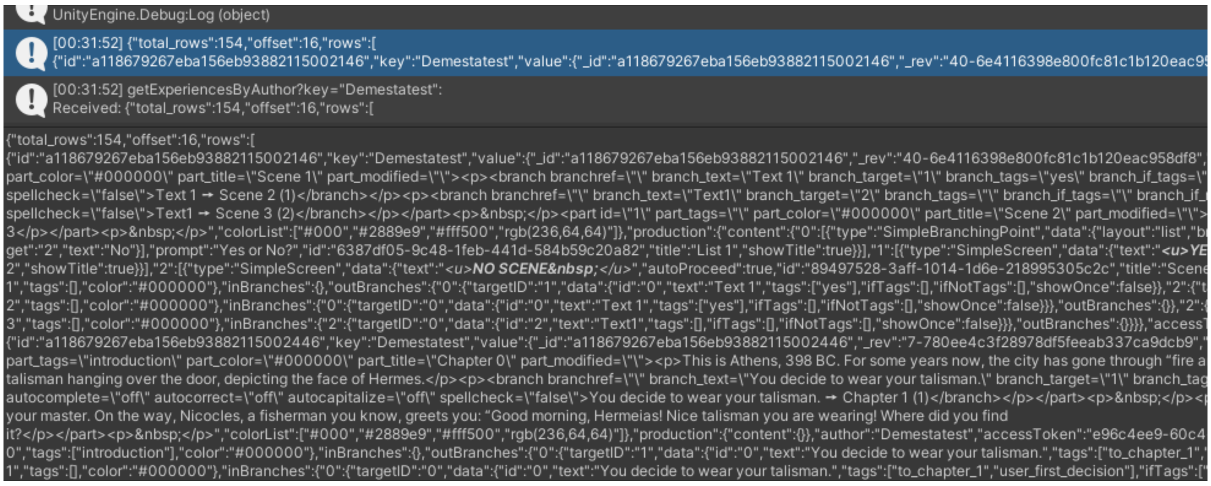
Title: t

Username: demesta

Password: [redacted]

Figure 44: Screenshot of the window that adds a story in Unity

After testing, it was proved that the code for authenticating the user through a web request was correct. To test that, a simple code would write a console message containing the response from the request with the user's Story Maker data.



```
UnityEngine.Debug:Log (object)
[00:31:52] {"total_rows":154,"offset":16,"rows":[
{"id":"a118679267eba156eb93882115002146","key":"Demestatest","value":{"_id":"a118679267eba156eb93882115002146","_rev":"40-6e4116398e800fc81c1b120eac95
[00:31:52] getExperiencesByAuthor?key="Demestatest":
Received: {"total_rows":154,"offset":16,"rows":[

{"total_rows":154,"offset":16,"rows":[
{"id":"a118679267eba156eb93882115002146","key":"Demestatest","value":{"_id":"a118679267eba156eb93882115002146","_rev":"40-6e4116398e800fc81c1b120eac958df8",
part_color=\"#000000\" part_title=\"Scene 1\" part_modified=\"\"><p><branch branchref=\"\" branch_text=\"Text 1\" branch_target=\"1\" branch_tags=\"yes\" branch_if_tags=\"
spellcheck=\"false\">Text 1 → Scene 2 (1)</branch></p><p><branch branchref=\"\" branch_text=\"Text1\" branch_target=\"2\" branch_tags=\"\" branch_if_tags=\"\" branch_if_
spellcheck=\"false\">Text1 → Scene 3 (2)</branch></p></part><p>&nbsp;</p><part id=\"1\" part_tags=\"\" part_color=\"#000000\" part_title=\"Scene 2\" part_modified=\"\">
3</p></part><p>&nbsp;</p>","colorList":["#000","#2889e9","#fff500","rgb(236,64,64)"]},"production":{"content":{"0":[{"type":"SimpleBranchingPoint","data":{"layout":"list","br
get":"2","text":"No"}],"prompt":"Yes or No?","id":"6387df05-9c48-1feb-441d-584b59c20a82","title":"List 1","showTitle":true}}],"1":[{"type":"SimpleScreen","data":{"text":"<u>YE
2","showTitle":true}}],"2":[{"type":"SimpleScreen","data":{"text":"<u>NO SCENE&nbsp;</u>","autoProceed":true,"id":"89497528-3aff-1014-1d6e-218995305c2c","title":"Scene
1","tags":[],"color":"#000000"},"inBranches":{},"outBranches":{"0":{"targetID":"1","data":{"id":"0","text":"Text 1","tags":["yes"],"ifTags":[],"ifNotTags":[],"showOnce":false}},"2":{"t
2","tags":[],"color":"#000000"},"inBranches":{"0":{"targetID":"0","data":{"id":"0","text":"Text 1","tags":["yes"],"ifTags":[],"ifNotTags":[],"showOnce":false}}},"outBranches":{}},"2":
3","tags":[],"color":"#000000"},"inBranches":{"2":{"targetID":"0","data":{"id":"2","text":"Text1","tags":[],"ifTags":[],"ifNotTags":[],"showOnce":false}}},"outBranches":{}}}},"access
{"id":"a118679267eba156eb93882115002446","key":"Demestatest","value":{"_id":"a118679267eba156eb93882115002446","_rev":"7-780ee4c3f28978df5feeab337ca9dcb9","
part_tags=\"introduction\" part_color=\"#000000\" part_title=\"Chapter 0\" part_modified=\"\"><p>This is Athens, 398 BC. For some years now, the city has gone through "fire a
talisman hanging over the door, depicting the face of Hermes.</p><p><branch branchref=\"\" branch_text=\"You decide to wear your talisman.\" branch_target=\"1\" branch_tag
autocomplete=\"off\" autocorrect=\"off\" autocapitalize=\"off\" spellcheck=\"false\">You decide to wear your talisman. → Chapter 1 (1)</branch></p></part><p>&nbsp;</p><
your master. On the way, Nicocles, a fisherman you know, greets you: "Good morning, Hermeias! Nice talisman you are wearing! Where did you find
it?</p></part><p>&nbsp;</p>","colorList":["#000","#2889e9","#fff500","rgb(236,64,64)"]},"production":{"content":{}},"author":"Demestatest","accessToken":"e96c4ee9-60c4
0","tags":["introduction"],"color":"#000000"},"inBranches":{},"outBranches":{"0":{"targetID":"1","data":{"id":"0","text":"You decide to wear your talisman.","tags":["to_chapter_1","
1","tags":[],"color":"#000000"},"inBranches":{"0":{"targetID":"0","data":{"id":"0","text":"You decide to wear your talisman.","tags":["to_chapter_1","user_first_decision"],"ifTags":["
```

Figure 45: Screenshot of the printed string that is received from requesting stories from StoryMaker in Unity

6.2 Challenges and obstacles

Arguably, the biggest challenge of the project was to transform the JSON data into C# classes and data structures. These components should have the element of being easily readable through the code in order to be imported into methods that will create the nodes of the graph. After analyzing the JSON, the classes representing the JSON data of the core sub-JSON that was needed to start the visualization of the story's layout in a graph ended up as the following:

```
public partial class StoryModel
   {
      public long Root { get; set; }
```

```
        public System.Collections.Generic.Dictionary<string, PartsMap> PartsMap { get; set; }
    }

    public partial class PartsMap
    {
        public Part Part { get; set; }
        public System.Collections.Generic.Dictionary<string, Branch> InBranches { get; set; }
        public System.Collections.Generic.Dictionary<string, Branch> OutBranches { get; set; }
    }

    public partial class Branch
    {
        public long TargetId { get; set; }
        public Data Data { get; set; }
    }

    public partial class Data
    {
        public long Id { get; set; }
        public string Text { get; set; }
        public System.Collections.Generic.List<object> Tags { get; set; }
        public System.Collections.Generic.List<string> IfTags { get; set; }
        public System.Collections.Generic.List<string> IfNotTags { get; set; }
        public bool ShowOnce { get; set; }
    }

    public partial class Part
    {
        public long Id { get; set; }
        public string Title { get; set; }
        public System.Collections.Generic.List<string> Tags { get; set; }
        public string Color { get; set; }
    }
```

Another challenge was that the nodes needed to be positioned through the graph in a way that they do not overlap. An easy fix for that would be to calculate their position dynamically, instead of giving all the nodes the same position. This could be achieved by the following code:

```
foreach (var perNodeKey in storyModel.PartsMap.Keys )
            {
                Vector2 v = new Vector2(100 * i + 300, i * 100 + 300);
                i++;
                var tempNode = _graphView.CreateNode(storyModel.PartsMap[perNodeKey].Part.Title, v);
            }
```

This logic would result in each node being generated on the bottom right from the previous one.

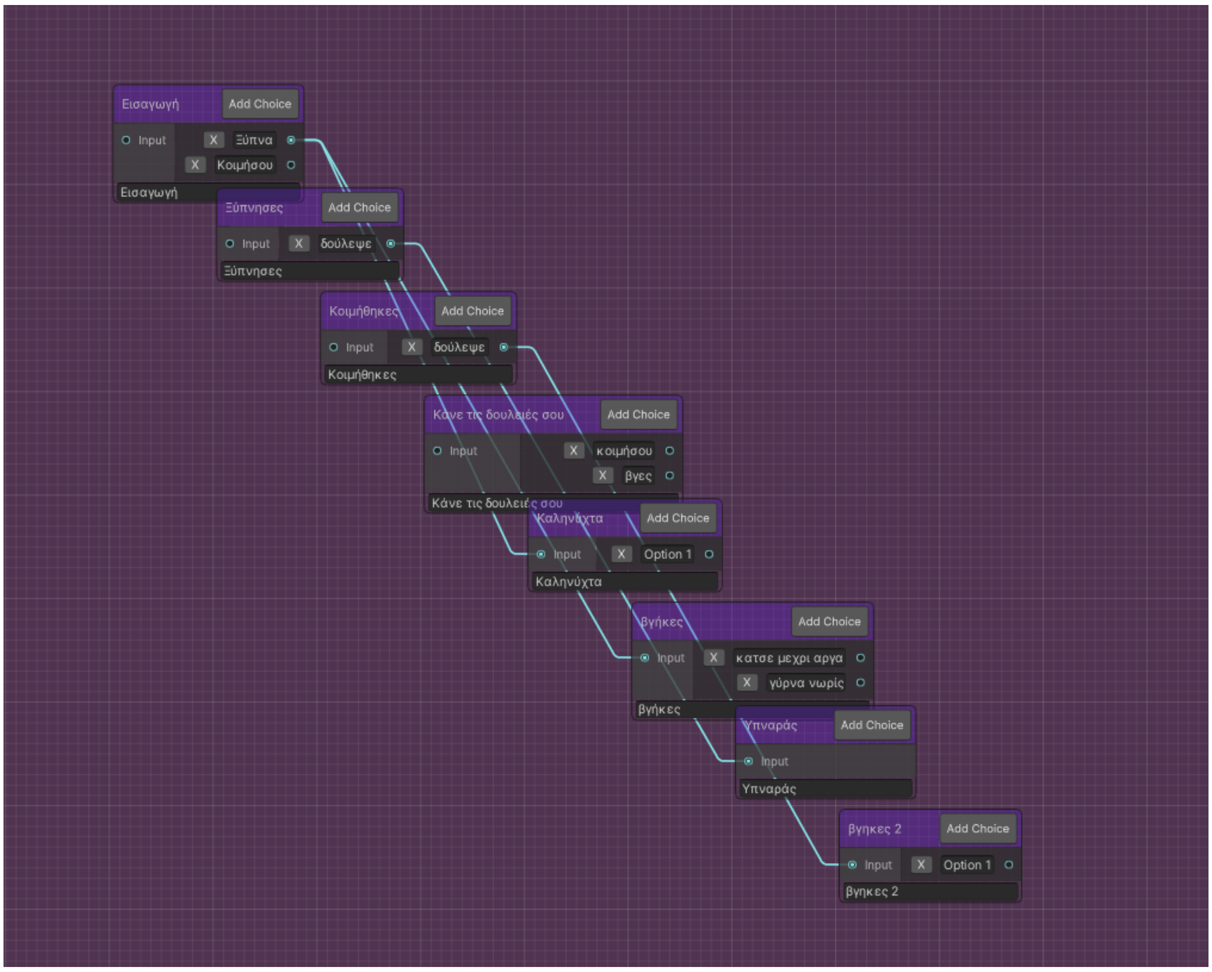


Figure 46: Screenshot of an attempt regarding the graph

Although this implementation gives a solution for the overlapping of the nodes, it does not display the graph in a user-friendly way that depicts the flow of the story. Especially in a more complex story, with a big number of branching narratives and loops, it would be impossible for the user to understand it. Thus, it became important to find an algorithm that takes into consideration the relation between the nodes for the calculation of their positions.

The algorithm we chose and implemented was BFS (Breadth-First Search). The BFS algorithm is a graph traversal algorithm that explores all the vertices of a graph in breadth-first order. It starts at a starting vertex and visits its neighbors first before moving on to the next level of neighbors. Using a queue, BFS ensures that vertices are visited in the order they were discovered. The steps that this algorithm follows are:

1. Choose a starting vertex and mark it as visited.
2. Create a queue and enqueue the starting vertex.
3. Initialize an empty array or set to store the visited vertices.
4. While the queue is not empty, repeat steps 5-7.
5. Dequeue a vertex from the queue and visit it.
6. Enqueue all unvisited neighbors of the current vertex and mark them as visited.
7. Repeat steps 5-6 until the queue becomes empty.

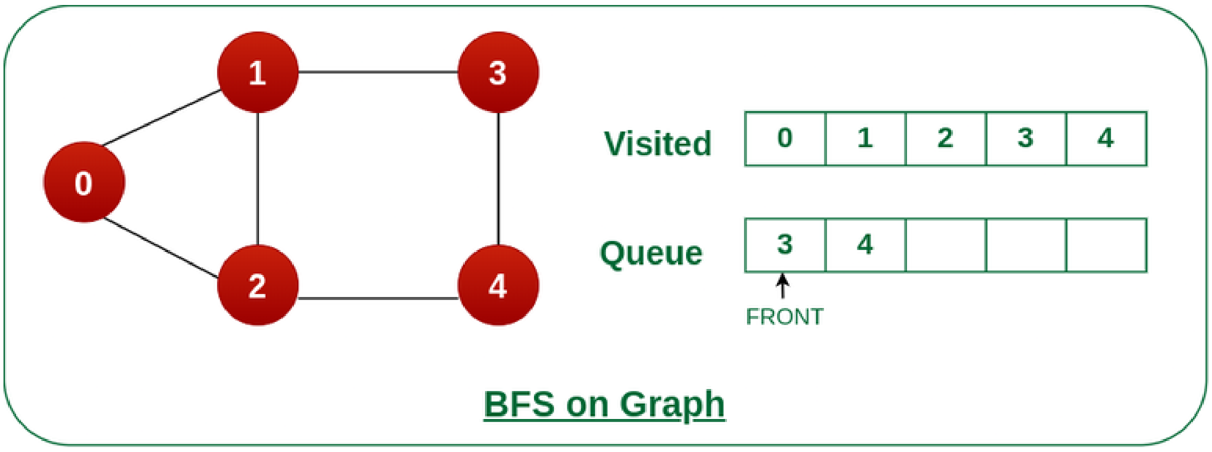


Figure 47: Example of the BFS algorithm's process

Thus, this algorithm was incorporated in the part of the code where the graph’s nodes are generated.

```
  int level = 1;

List<string> visited = new List<string>();
List<string> successorNodes = new List<string>();
Queue<string> fringe = new Queue<string>();
Dictionary<string, int> levelDictionary = new Dictionary<string, int>();

fringe.Enqueue(storyModel.Root.ToString());

levelDictionary.Add(storyModel.Root.ToString(), level);

while (true)
{
   var currentNode = fringe.Dequeue();

   visited.Add(currentNode);

   if(storyModel.PartsMap.Keys.Count == levelDictionary.Keys.Count)
   {
      break;
   }

   foreach (var outBranchId in
storyModel.PartsMap[currentNode].OutBranches.Keys)
   {
      var targetId =
storyModel.PartsMap[currentNode].OutBranches[outBranchId].TargetId.ToString();
      if (!levelDictionary.ContainsKey(targetId))
      {
         levelDictionary.Add(targetId, levelDictionary[currentNode]+1);
      }

      successorNodes.Add(targetId);
   }

   foreach(var successorNode in successorNodes)
```

```
        {
            if (fringe.Contains(successorNode) || visited.Contains(successorNode))
            {
                continue;
            }
            fringe.Enqueue(successorNode);
        }

        successorNodes.Clear();

    }
```

The implementation of the BFS algorithm gave the following result in the layout of the nodes.

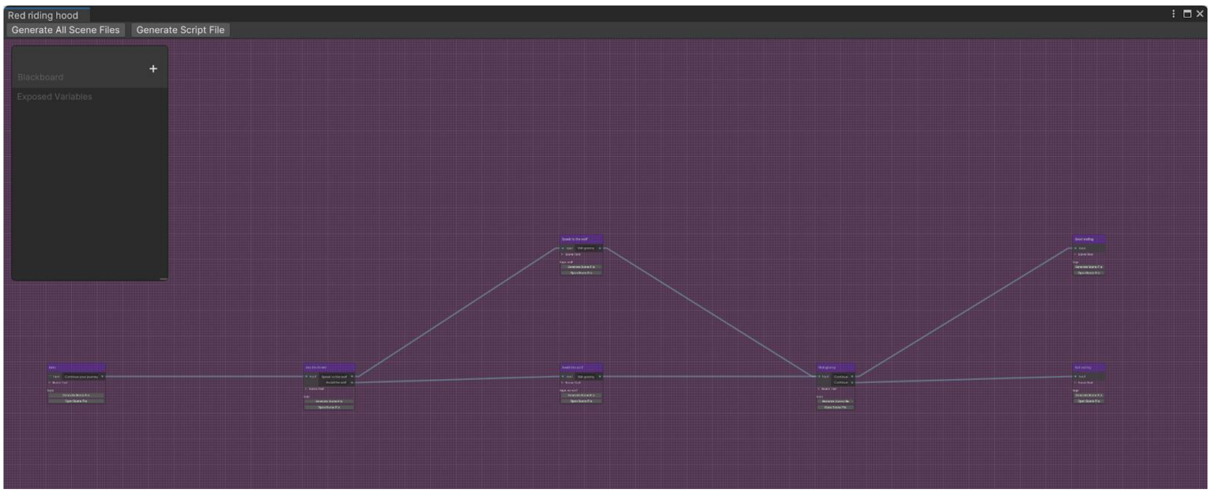


Figure 48: Screenshot of the graph in Unity after implementing the BFS algorithm

6.3 Completed implementation

After a general design, writing code for test purposes, facing challenges, and making a number of changes after a “trial and error” process, the tool was completed.

6.3.1 Code analysis

The basic code file is responsible for the creation of the input window, the web request, the conversion of the JSON data into objects and the graph window. It begins with the import of basic Unity libraries as well as the Newtonsoft library.

```
using System.Collections;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using CustomEditorWindow.Editor;
using Subtegral.DialogueSystem.Editor;
using Unity.EditorCoroutines.Editor;
```

Next, a set of empty strings is declared with the first two strings being one for a username and one for a password. Their purpose is to read the input of the user. Lastly, a fifth string named “state” will serve as an indication of whether the credentials are correct.

```
public class StoryTool : EditorWindow
{
   string username = "";
   string password = "";
   string state = "";
```

Next, one empty array for the story titles and one for the story ids are initialized. An integer named “selectedStory” is declared with a starting value of 0.

```
  private int selectedStory = 0;
  private string[] titles = new string[] { };
  private string[] ids = new string[] { };
```

A new Menu Item named “Add story” is being created under the tab “Tools”.

```
   [MenuItem("Tools/Add story")]
```

A method named “ShowWindow” serves for the opening of a new window where the user can enter their credentials. It is labeled “Enter credentials” and it contains the text fields for the user’s inputs.

```
   public static void ShowWindow()
   {
      GetWindow(typeof(StoryTool));
   }

   private void OnGUI()
   {
         GUILayout.Label("Enter credentials", EditorStyles.boldLabel);

      this.username = EditorGUILayout.TextField("Username", this.username);
      this.password = EditorGUILayout.PasswordField("Password", this.password);
```

In contrast with the username field, the password is a PasswordField, instead of a TextField. This way, the security of the password is achieved, as this method replaces the characters entered by the user with asterisks.

Next, a button named “Get stories” triggers, when clicked, a function named “searchStories”.

```
      if (GUILayout.Button("Get stories"))
      {
         searchStories();
      }

      this.state = EditorGUILayout.TextField("", this.state);

   }
```

After some space between the components and a “Select a story:” label, the code for the dropdown list containing the story titles is placed. It uses the number of the selectedStory to fetch the story id from the ids array and add it to a link for the web request.

```
          GUILayout.Space(20);

        GUILayout.Label("Select a story:");

        selectedStory = EditorGUILayout.Popup(selectedStory, titles);

        if (GUILayout.Button("Get Story"))
        {
            string link = "https://chess1.karpathos.net/couchdb/experiences/" +
this.ids[selectedStory];
            EditorCoroutineUtility.StartCoroutine(getStory(link), this);
        }
```

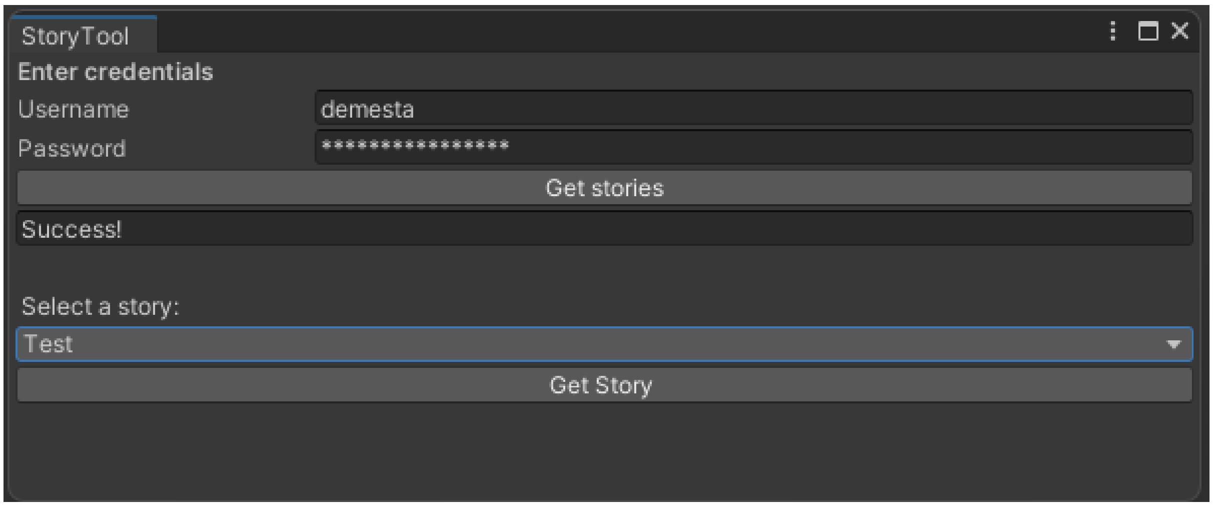

StoryTool
Enter credentials
Username demesta
Password ****************
Get stories
Success!
Select a story:
Test
Get Story

Figure 49: The complete implementation of the window that verifies the user and imports stories in Unity

The "searchStories" function starts with checking if the username and/or password is empty. If either of these fields is empty a message is displayed on the window asking the user to enter the missing field.

```
    private void searchStories()
    {
        if (this.username == string.Empty) {
            this.state = "Error: Please enter username.";
            return;
          if (this.password == string.Empty) {
            this.state = "Error: Please enter password.";
            return;
          if (this.username == string.Empty && this.password == string.Empty) {
            this.state = "Error: Please enter username and password.";
            return;
        }
```

Having the necessary information given by the user, a string with the request link can now be formed.

```
        string link = "https://chess1.karpathos.net/couchdb/experiences/" + this.storyId;

        EditorCoroutineUtility.StartCoroutine(GetUserData(link), this);
```

```
    }
```

In order to make a successful request from Story Maker, it is necessary to authenticate the user. A function named “authenticate” uses the username and password strings to encode them into an authentication string supported by Basic Auth, an HTTP method required for the request.

```
    string authenticate()
    {
        string auth = this.username + ":" + this.password;
        auth = System.Convert.ToBase64String(System.Text.Encoding.GetEncoding("ISO-
8859-1").GetBytes(auth));
        auth = "Basic " + auth;
        return auth;
    }
```

Finally, the web request can now be made.

```
    IEnumerator GetUserData(string uri)
    {
        using (UnityWebRequest webRequest = UnityWebRequest.Get(uri))
        {
            string authorization = authenticate();
            webRequest.SetRequestHeader("AUTHORIZATION", authorization);

            yield return webRequest.SendWebRequest();

            checkResponse(uri, webRequest);
```

To check if the web request is successful, the checkResponse function uses a switch method to check the possible results. These results are a general error response that may indicate that the server is down, an authentication error that indicates wrong credentials or a successful response. Apart from changing the string “state” in order to display on the window the result, the switch method creates console messages with details on the request’s outcome.

```
void checkResponse(string uri, UnityWebRequest webRequest)
    {
        string[] pages = uri.Split('/');
        int page = pages.Length - 1;

        switch (webRequest.result)
        {
                case UnityWebRequest.Result.ConnectionError:
                case UnityWebRequest.Result.DataProcessingError:
                    Debug.LogError(pages[page] + ": Error: " + webRequest.error);
                    this.state = "Error!";
                    break;
                case UnityWebRequest.Result.ProtocolError:
                    Debug.LogError(pages[page] + ": HTTP Error: " + webRequest.error);
                    this.state = "Error! Check if username and password are correct.";
                    break;
                case UnityWebRequest.Result.Success:
```

```
                Debug.Log(pages[page] + ":\nReceived: " +
webRequest.downloadHandler.text);
                this.state = "Success!";
                break;
        }
}
```

Going back to the GetUserData function, after calling the checkResponse function, having the JSON with the user's data, it continues with parsing the JSON. It extracts the number of available stories from the JSON value "total_rows". Using this value, the "titles" and "ids" arrays can fill with the JSON data in a loop for every row.

```
        JObject obj = JObject.Parse(webRequest.downloadHandler.text);

        int totalRows = obj["total_rows"].ToObject<int>();
        var rows = obj["rows"];

        string[] ids = new string[totalRows];
        string[] titles = new string[totalRows];

        for (int rowCount = 0; rowCount < totalRows; rowCount++)
        {
           var thisRow = rows[rowCount];
           ids[rowCount] = (string)thisRow["id"];
           titles[rowCount] = (string)thisRow["value"]["title"];
        }

        this.titles = titles;
        this.ids = ids;
```

The "getStory" function that is called when the user clicks the "Get Story" button under the dropdown list starts, similarly to the GetUserData function, by authenticating and requesting with the link that contains the story id.

```
IEnumerator getStory(string uri)
   {
      using (UnityWebRequest webRequest = UnityWebRequest.Get(uri))
      {
         string authorization = authenticate();
         webRequest.SetRequestHeader("AUTHORIZATION", authorization);

         yield return webRequest.SendWebRequest();

         checkResponse(uri, webRequest);
```

After receiving and parsing the JSON, an object can be created. This object will contain all the necessary data.

```
        JObject obj = JObject.Parse(webRequest.downloadHandler.text);
        var jsonString = obj["storyModel"];

        StoryModel node = jsonString.ToObject<StoryModel>();
```

Having the required data, a new window named "Narrative Graph" opens and a function that is responsible for the creation of the graph is called.

```
            var window = GetWindow<StoryGraph>();
            window.titleContent = new GUIContent("Narrative Graph");

            StoryGraphView storyGraphView = new StoryGraphView(window);
            window.FileName = "myStory";
            window.RequestDataOperation(false,node);

        }
    }

}
```

A second file contains most of the functions that create the nodes, the connections and, finally, the graph.

First, a function for every Node that needs to be created loops through the keys of the story's parts. For each key, it calculates a vector that will give the node a position on the graph so that the nodes do not overlap. It also gives the node a title and the choices it may contain.

```
private void GenerateDialogueNodes()
        {
            int i = 1;
            foreach (var perNodeKey in storyModel.PartsMap.Keys )
            {
                var numOfLevelNodes = levelDictionary.Where(x => x.Value ==
levelDictionary[perNodeKey]).Select(x => x.Key).ToList().Count;
                var yLevel = levelDictionary.Where(x => x.Value ==
levelDictionary[perNodeKey]).Select(x => x.Key).ToList().IndexOf(perNodeKey);

                var offset = (numOfLevelNodes / 2) * (-500);
                Vector2 v = new Vector2(1000 * levelDictionary[perNodeKey], yLevel * 500 +
offset + 400);

                string sceneText = "";
                if (obj["production"]["content"][perNodeKey][0]["type"].ToString() ==
"SimpleScreen")
                {
                    sceneText =
obj["production"]["content"][perNodeKey][0]["data"]["text"].ToString();
                }

                var tempNode =
_graphView.CreateNode(storyModel.PartsMap[perNodeKey].Part.Title, v, sceneText,
storyModel.PartsMap[perNodeKey].Part.Color);

                tempNode.GUID = perNodeKey;
                _graphView.AddElement(tempNode);
                DialogueNode currentNode = Nodes.Find(x => x.GUID == perNodeKey);

                foreach (var outBranchId in
storyModel.PartsMap[perNodeKey].OutBranches.Keys)
                {
```

```
                _graphView.AddChoicePort(currentNode,
storyModel.PartsMap[perNodeKey].OutBranches[outBranchId].Data.Text,
storyModel.PartsMap[perNodeKey].OutBranches[outBranchId].Data);
            }
}
```

A different function is responsible for the calculation of the connection between the Nodes, through the choices that were created in the previous steps.

```
private void ConnectDialogueNodes()
{
        foreach (var perNodeKey in storyModel.PartsMap.Keys )
         {
            DialogueNode currentNode = Nodes.Find(x => x.GUID == perNodeKey);

            int index = 0;
            foreach (var outBranchId in storyModel.PartsMap[perNodeKey].OutBranches)
            {
               var targetNodeGUID = outBranchId.Value.TargetId.ToString();
               var targetNode = Nodes.First(x => x.GUID == targetNodeGUID);
               LinkNodesTogether(currentNode.outputContainer[index].Q<Port>(), (Port)
targetNode.inputContainer[0]);
               index++;
            }
         }

}
```

Also, a function creates the needed links for the connections.

```
private void LinkNodesTogether(Port outputSocket, Port inputSocket)
      {
         var tempEdge = new Edge()
         {
            output = outputSocket,
            input = inputSocket
         };
         tempEdge?.input.Connect(tempEdge);
         tempEdge?.output.Connect(tempEdge);
         _graphView.Add(tempEdge);
      }
```

When it comes to the generation of scenes, since it requires a specific path, first, there is a need to check if the folders exist and, if not, create them.

```
if(!Directory.Exists("Assets/Narralive"))

       Directory.CreateDirectory("Assets/Narralive");
if(!Directory.Exists("Assets/Narralive/StoryMaker"))

       Directory.CreateDirectory("Assets/Narralive/StoryMaker");
if(!Directory.Exists("Assets/Narralive/StoryMaker/" + storyTitle))

       Directory.CreateDirectory("Assets/Narralive/StoryMaker/" + storyTitle);

if (!Directory.Exists(folderPath))

       Directory.CreateDirectory(folderPath);
```

```
if (!File.Exists(scenePath)) { // Create a new scene
        Scene newScene = EditorSceneManager.NewScene(NewSceneSetup.EmptyScene);
// Save the new scene
EditorSceneManager.SaveScene(newScene, scenePath);
AssetDatabase.Refresh();
}
```

To create each node, a function called CreateNode is responsible for creating them, setting their position and adding their title and text.

```
public DialogueNode CreateNode(string nodeName, Vector2 position, string text = null)
        {
            var tempDialogueNode = new DialogueNode()
            {
                title = nodeName,
                DialogueText = text,
                GUID = Guid.NewGuid().ToString()
            };
            tempDialogueNode.styleSheets.Add(Resources.Load<StyleSheet>("Node"));
            var inputPort = GetPortInstance(tempDialogueNode, Direction.Input,
Port.Capacity.Multi);
            inputPort.portName = "Input";
            tempDialogueNode.inputContainer.Add(inputPort);
            tempDialogueNode.RefreshExpandedState();
            tempDialogueNode.RefreshPorts();
            tempDialogueNode.SetPosition(new Rect(position,
                DefaultNodeSize));
            var foldout = new Foldout
            {
                text = "Scene Text",
                value = false
            };
            tempDialogueNode.mainContainer.Add(foldout);
            var label = new Label(text);
            label.style.whiteSpace = WhiteSpace.Normal; // enable text wrapping
            label.style.maxWidth = 300f;
            foldout.contentContainer.Add(label);

            return tempDialogueNode;
        }
```

Lastly, the tags are added in each node with the following code.

```
var textField = new TextField()
          {
             name = string.Empty,
             value = outputPortName,
             tooltip = "tags: " + tags + "\nifTags: " + iftags + "\nifNotTags: " + ifnottags + "\nshowOnce: " + showOnce
          };
          textField.RegisterValueChangedCallback(evt => generatedPort.portName = evt.newValue);
          generatedPort.contentContainer.Add(new Label("  "));
          generatedPort.contentContainer.Add(textField);
```

6.3.2 Testing and verification

In order to determine the validity of the implementation after having the graph fully automated, the way of checking the results was to compare the tool’s graph with the one in the Story Maker.

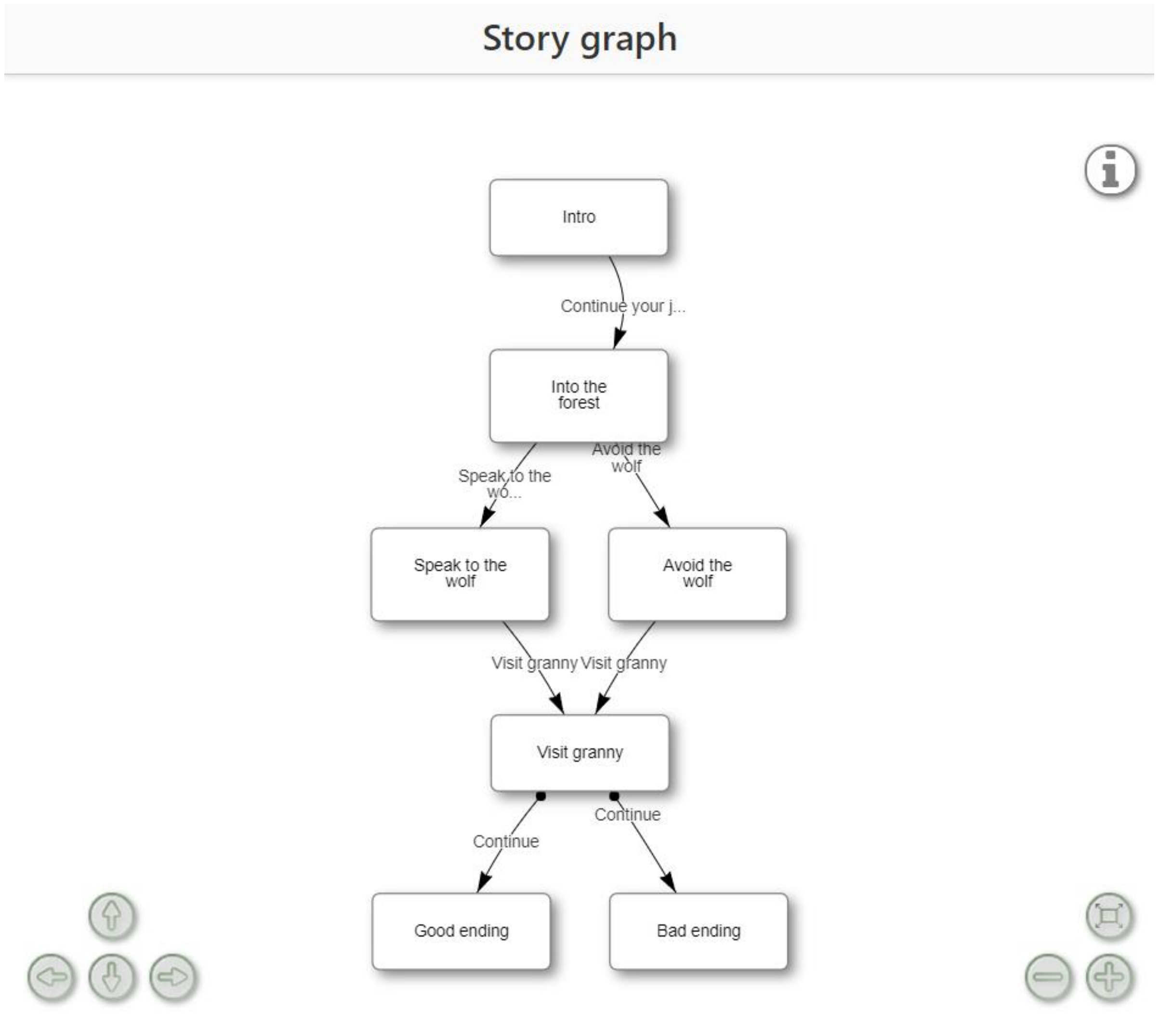


Figure 50: The graph for the "Red riding hood" story in StoryMaker

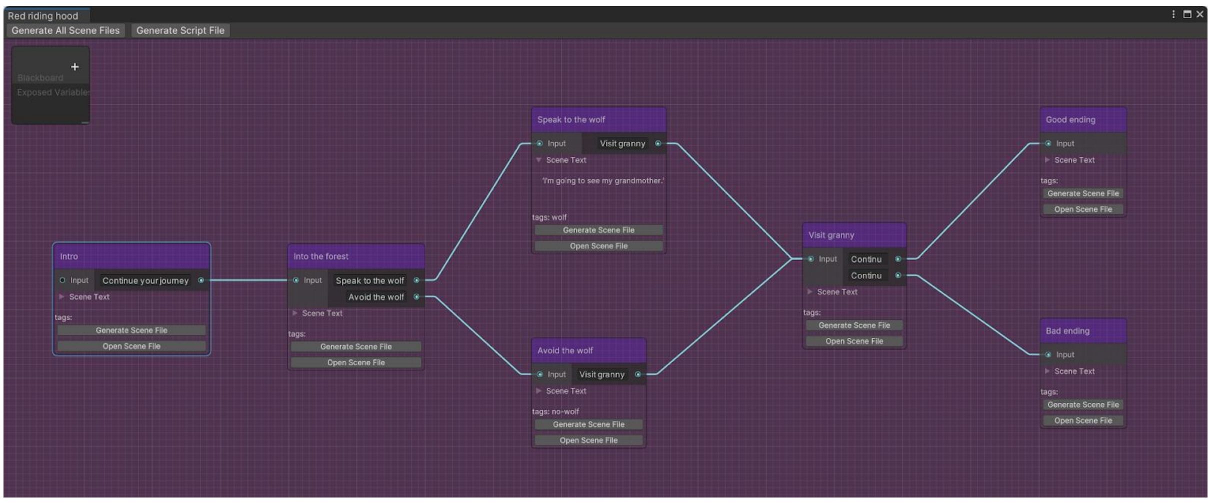


Figure 51: The graph for the "Red riding hood" story produced by the Unity plug-in

After ensuring that the two graphs were identical, the last phase of testing, regarding the graph, was testing the functionality of the tool with different, and more complicated, stories.

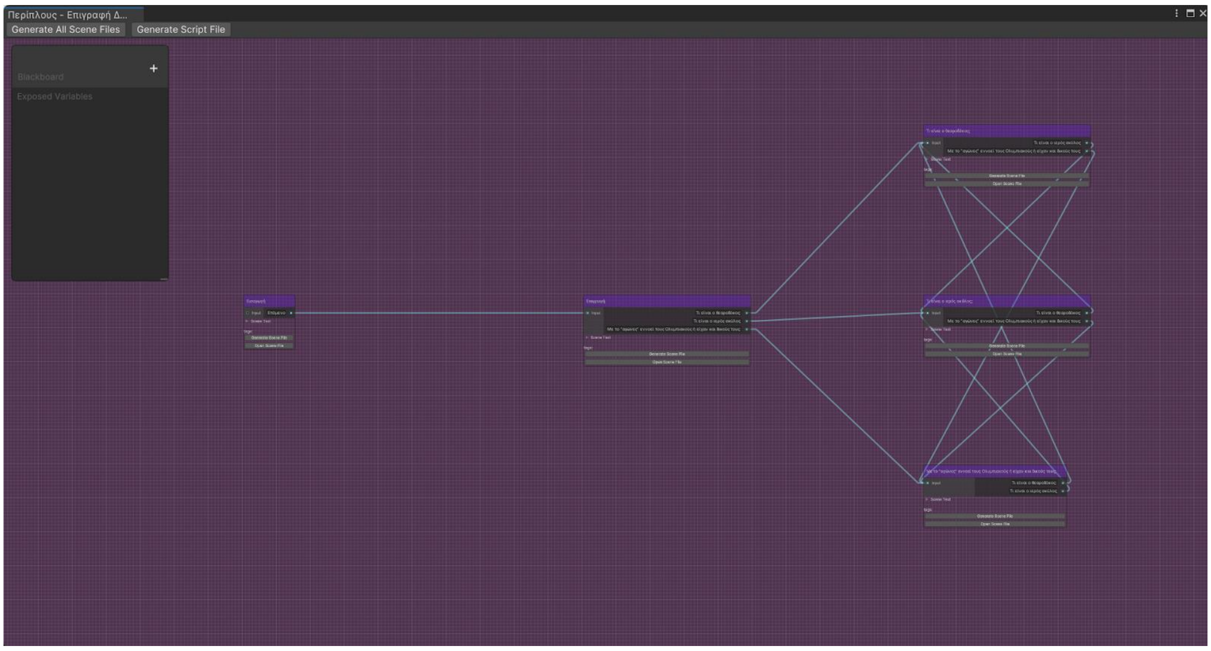


Figure 52: Example of a story's graph produced by the Unity plug-in

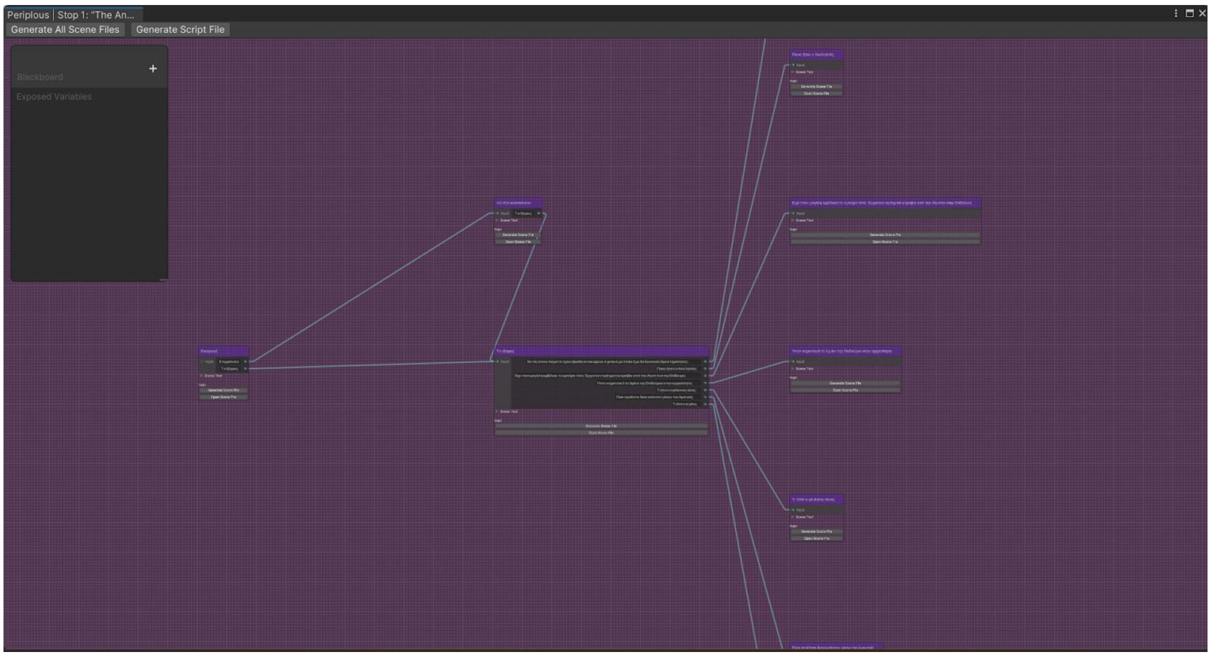


Figure 53: Example of a story's graph produced by the Unity plug-in

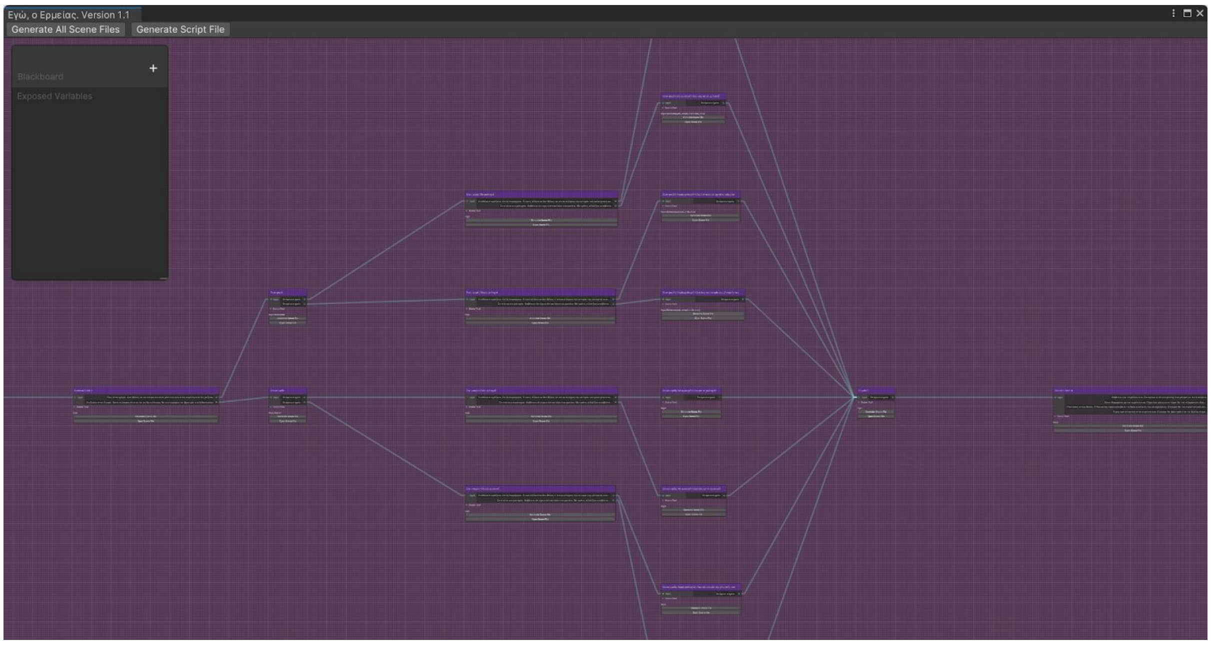


Figure 54: Example of a story's graph produced by the Unity plug-in

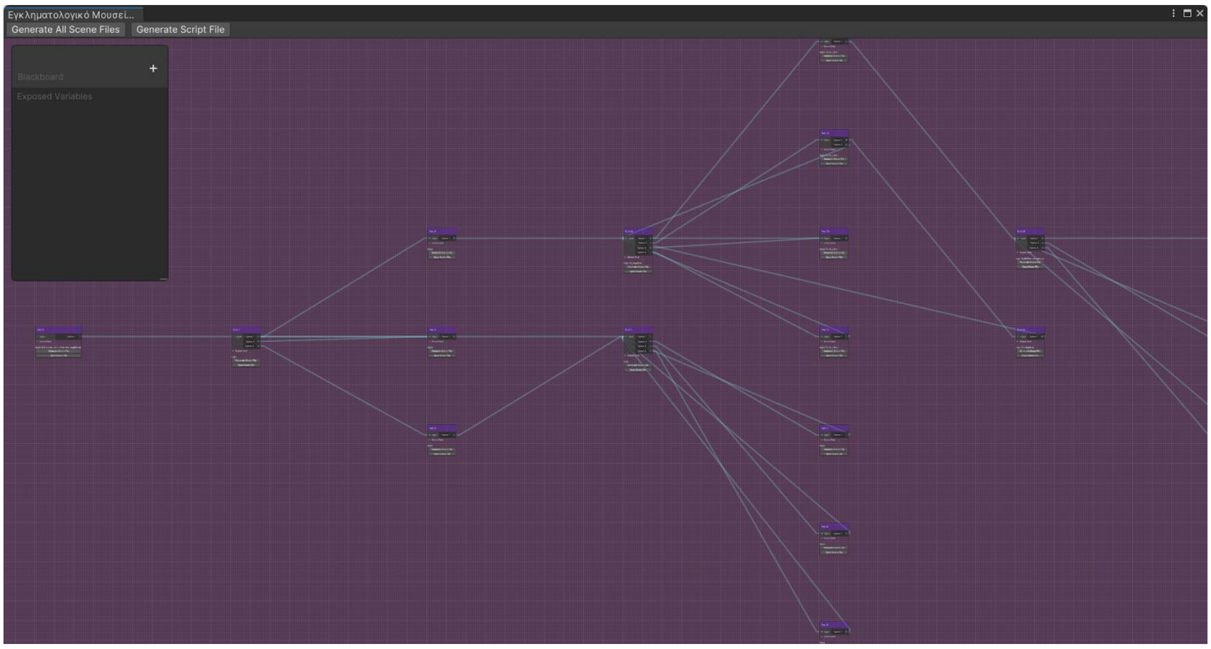


Figure 55: Example of a story's graph produced by the Unity plug-in

By testing the different stories, it was concluded that the tool works correctly and the BFS algorithm for the graph’s layout is displaying each story’s structure clearly.

6.3.3 The workflow of the tool

Having the code completed and tested, the tool can now be presented from the point of view of a basic usage. This section can serve as a documentation of the tool, viewed by a new user that intends to import and view their story in Unity.

First, the user opens the “Add New Story” and enters their credentials.

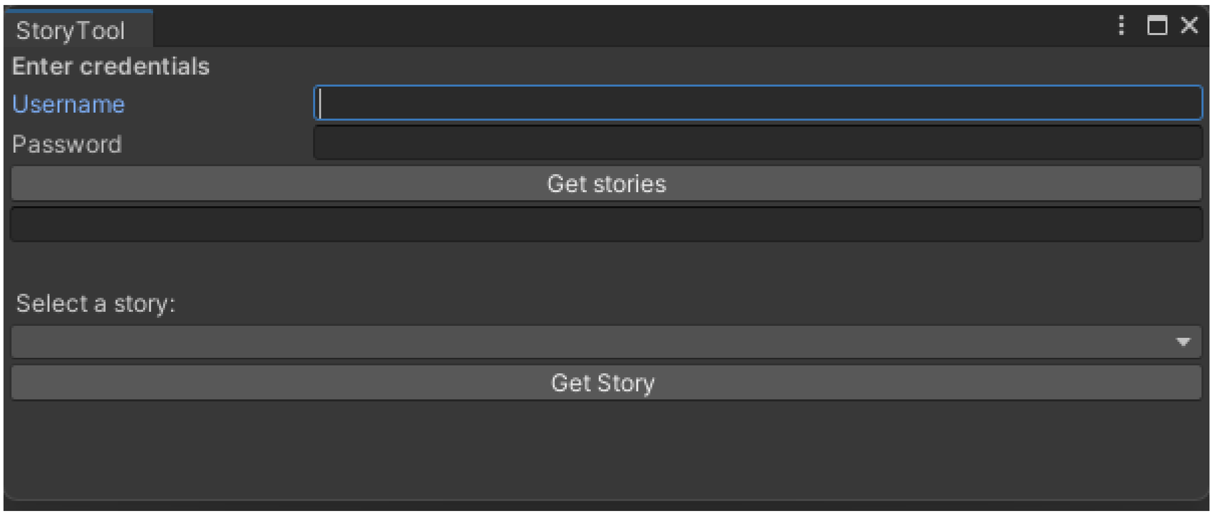


Figure 56: The window where the user enters their credentials and selects which story to import

If the credentials are correct and the request is successful a success message is displayed and the dropdown list fills with story titles. The user can open the list and select a story.

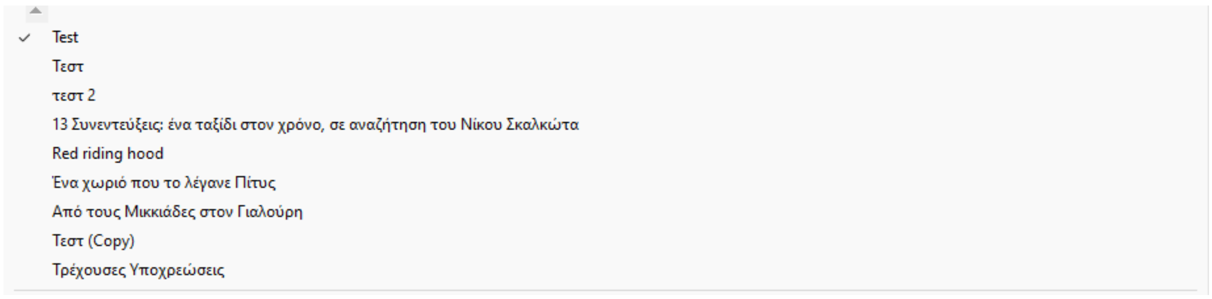

✓ Test
Τεστ
τεστ 2
13 Συνεντεύξεις: ένα ταξίδι στον χρόνο, σε αναζήτηση του Νίκου Σκαλκώτα
Red riding hood
Ένα χωριό που το λέγανε Πίτυς
Από τους Μικκιάδες στον Γιαλούρη
Τεστ (Copy)
Τρέχουσες Υποχρεώσεις

Figure 57: The dropdown menu with the available stories

After selecting the desired story, the user can press the “Get Story” button.

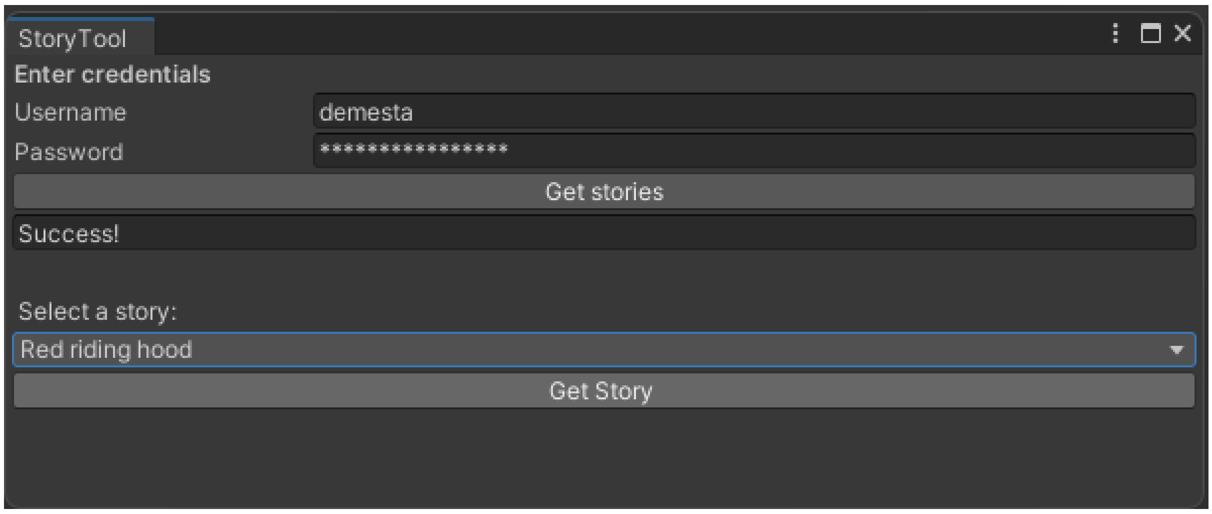

StoryTool
Enter credentials
Username: demesta
Password: ****************
Get stories
Success!
Select a story:
Red riding hood
Get Story

Figure 58: The window after the user gets verified and chooses a story to import

A new window of the graph appears on the screen.

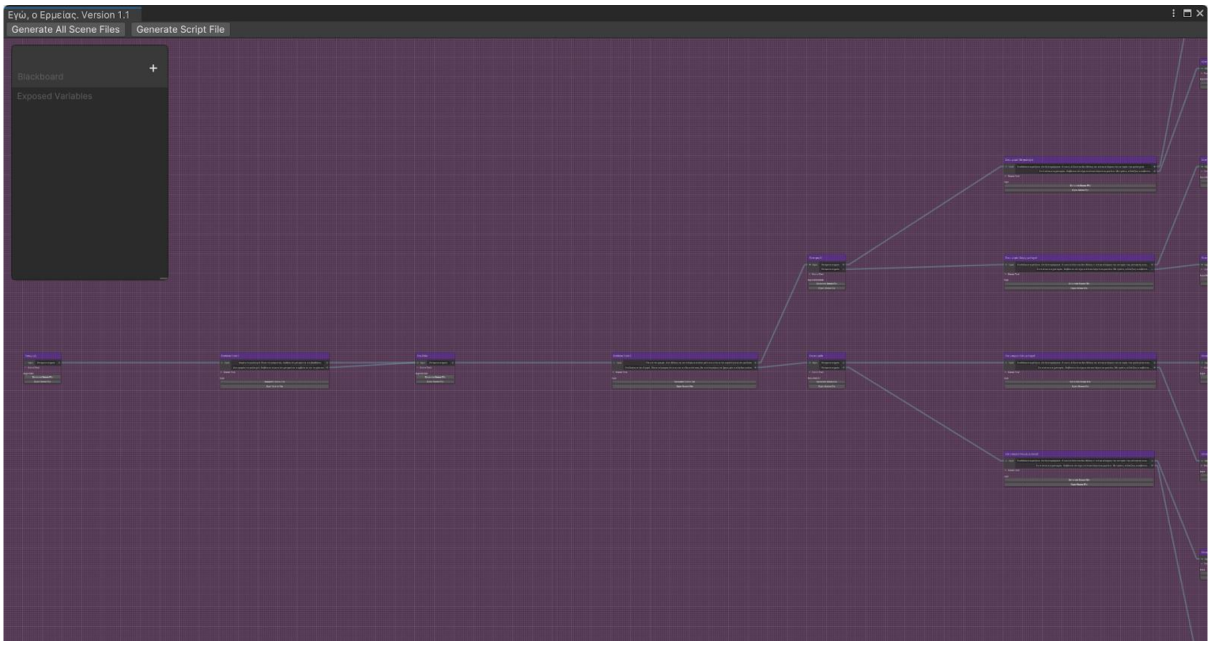
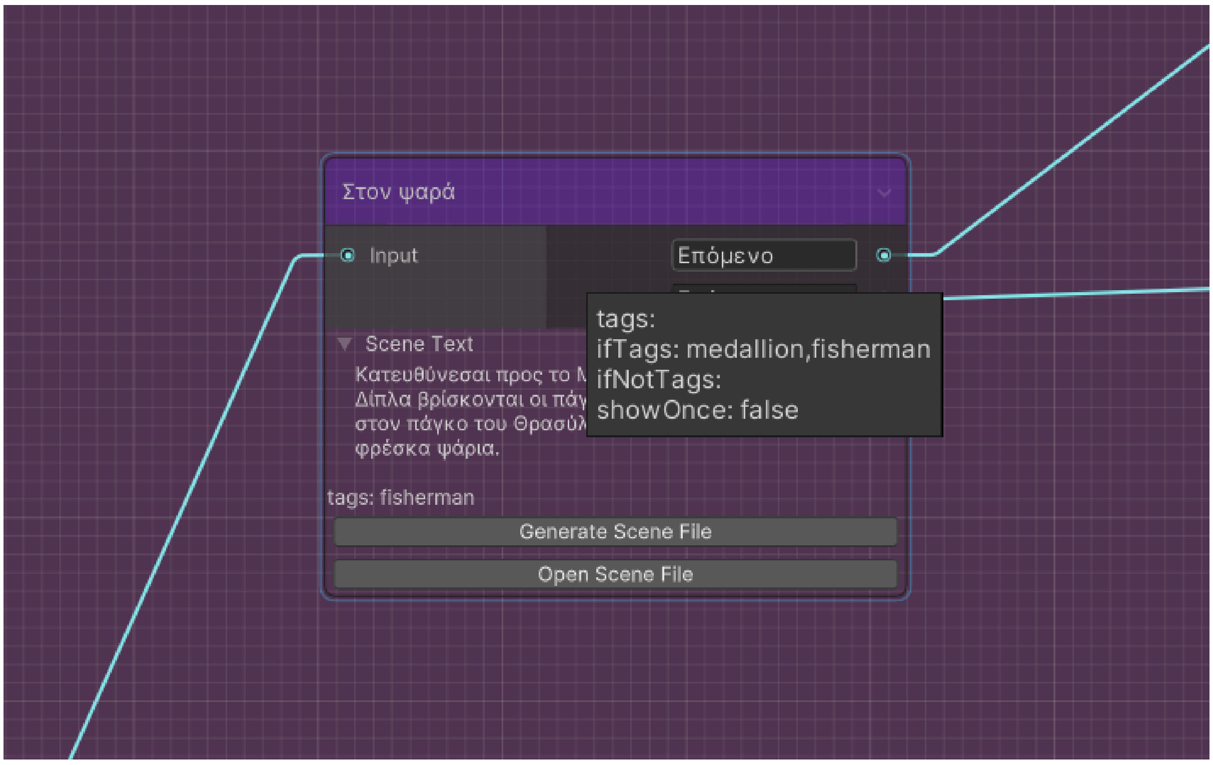


Figure 59: The generated graph displaying the story that the user chose

The user may examine the content of the nodes by clicking on the dropdown of the texts, hover over the choices to view the tags, and navigate through the graph to view the connections of the nodes and the possible different paths and endings of the story.



tags:
ifTags: medallion,fisherman
ifNotTags:
showOnce: false

Κατευθύνεσαι προς το Μ
Δίπλα βρίσκονται οι πάγ
στον πάγκο του Θρασύλ
φρέσκα ψάρια.

tags: fisherman

Figure 60: Example of the data provided in a node

The user can find a “Generate all Scene files” and a “Generate Script file” button in the top of the graph window while each node contains a “Generate Scene file” as well. The generation of scene files can be made either in bulk, by generating one scene for each node, or the user may decide to generate the scenes of specific nodes. The result is empty scenes files with the names of the nodes’ titles that can be opened in Unity through another type of button that can be found in each node with the name “Open Scene”. The

“Generate Script file” generates a C# file with a class that represents the specific story that is opened in the graph and gives access to the story’s data in the form of C# objects.

The above files are organized in a specific folder structure. Firstly, a folder with the tool’s name is created and inside this folder there is a dedicated folder for every story that has scenes and/or scripts generated by the tool, with each folder having the title of the story it refers to. Inside every story’s folder there may be two other folders, a “Scenes” and a “Scripts” and each of these folders will be created after the user chooses to use the tool’s scene and/or script generation separately.

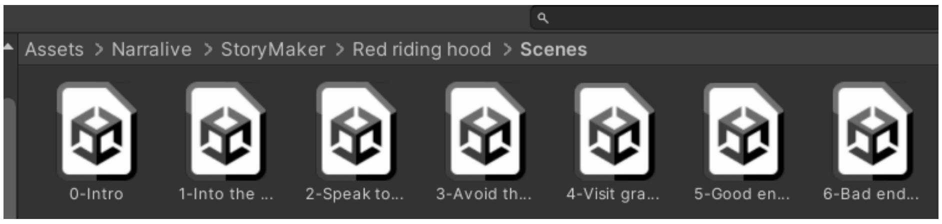


Figure 61: The auto-generated scenes

6.4 Demo

After completing the development of the tool, a demo that showcases a potential usage of the software was made. The demo demonstrates the tool's technical prowess but also emphasizes its real-world value, showing its potential as an asset for game development. In order to do that, a small game was made using both the generated scenes and the generated code of the tool as well as the scene narration through the graph.

6.4.1 Demo implementation

For the purposes of the demo game, “Red riding hood”, a small story from the StoryMaker’s database, was chosen. The basic product of the demo is a quick visual novel game with two possible endings, based on the player’s choices. The game is divided into multiple scenes, one for every node of the graph and with each scene matching the corresponding node in terms of text and choices. The scenes’ UI holds a simple layout of a background image, a text and one or more buttons, with each button representing one choice that, when clicked, leads to another scene.

First step in implementing the demo was to open the “Add story” window of the tool, fetch the story’s data by choosing it from the stories’ list and generate the graph.

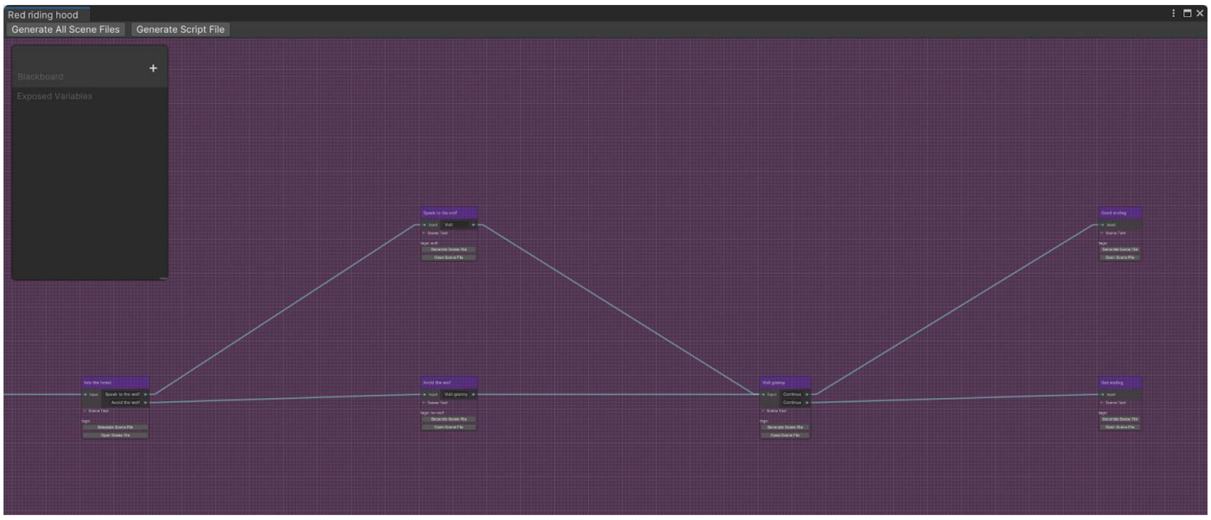


Figure 62: The generated graph that represents the "Red riding hood" story

After taking a look at the graph to get an idea of how this specific story is structured, the next step was to use the “Generate All Scene Files”.

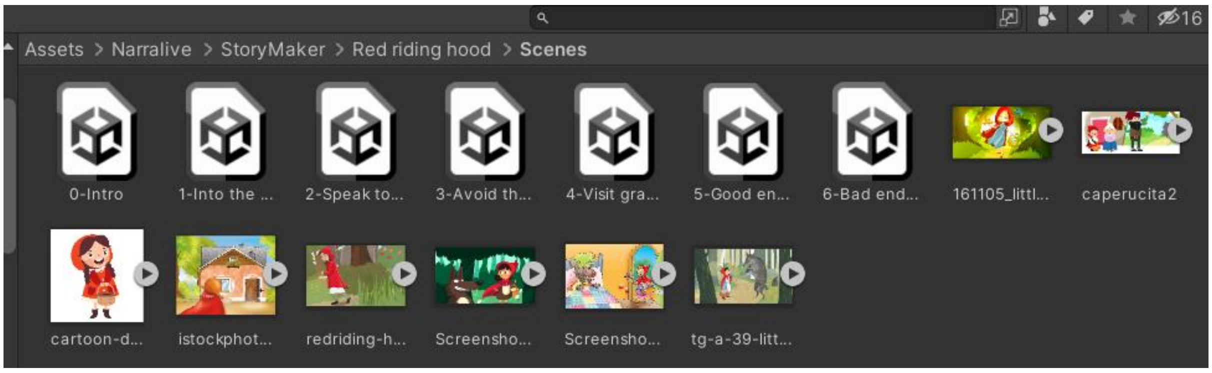
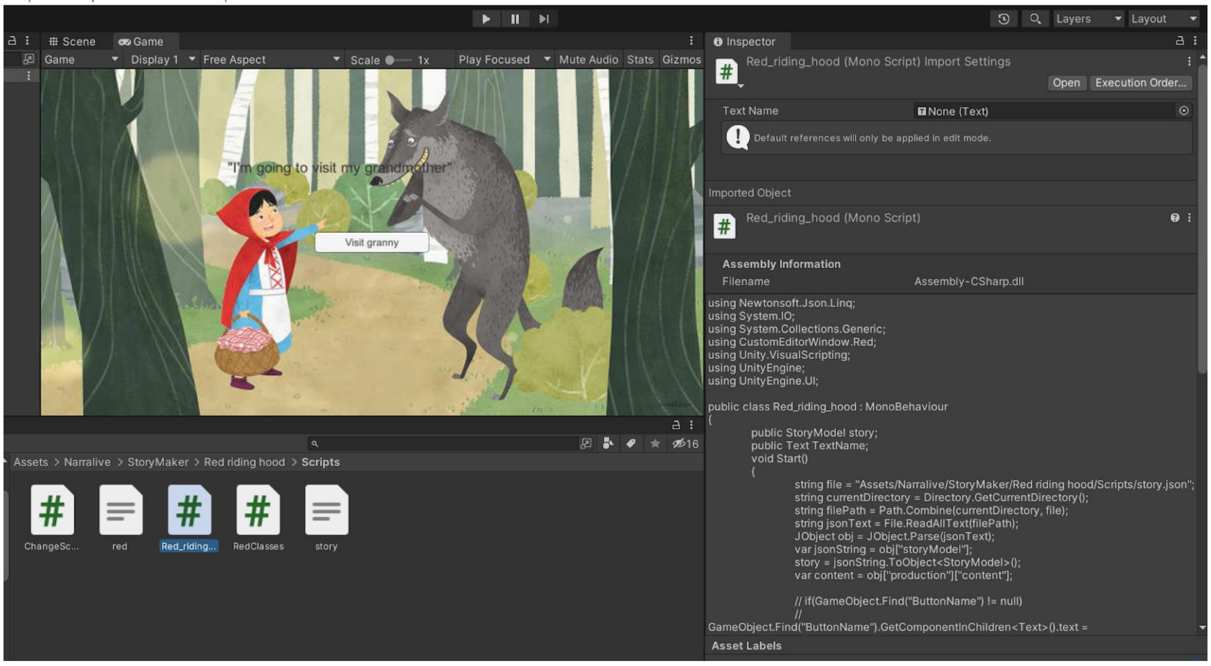


Figure 63: The generated scenes for "Red riding hood" with some images used for the demo

Also, the “Generate Script File” was used to get the starting code of the game with the main class that was used as well as the game’s data such as texts and tags.



Red_riding_hood (Mono Script) Import Settings

Text Name: None (Text)

Default references will only be applied in edit mode.

Imported Object

Red_riding_hood (Mono Script)

Assembly Information
Filename: Assembly-CSharp.dll

```
using Newtonsoft.Json.Linq;
using System.IO;
using System.Collections.Generic;
using CustomEditorWindow.Red;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

public class Red_riding_hood : MonoBehaviour
{
        public StoryModel story;
        public Text TextName;
        void Start()
        {
                string file = "Assets/Narralive/StoryMaker/Red riding hood/Scripts/story.json";
                string currentDirectory = Directory.GetCurrentDirectory();
                string filePath = Path.Combine(currentDirectory, file);
                string jsonText = File.ReadAllText(filePath);
                JObject obj = JObject.Parse(jsonText);
                var jsonString = obj["storyModel"];
                story = jsonString.ToObject<StoryModel>();
                var content = obj["production"]["content"];

                // if(GameObject.Find("ButtonName") != null)
                //
GameObject.Find("ButtonName").GetComponentInChildren<Text>().text =
```

Figure 64: Screenshot of the development of the demo, displaying the generated code

Having the main structure of the game being set up it was time to start working on each scene, with the process being similar for every scene. An image was added as the background of the scene, then a text and then, after consulting the node in the graph that matched the scene, a number of buttons.

Code-wise, the sub-classes that represent the different types of data of the game, such as the array of “outBranches”, were used in order to load the correct texts in the scene’s and buttons’ texts.

6.4.2 Demo Result

To get a complete result of this demo, it was built and saved as a .exe file so that it can be playable on a computer.

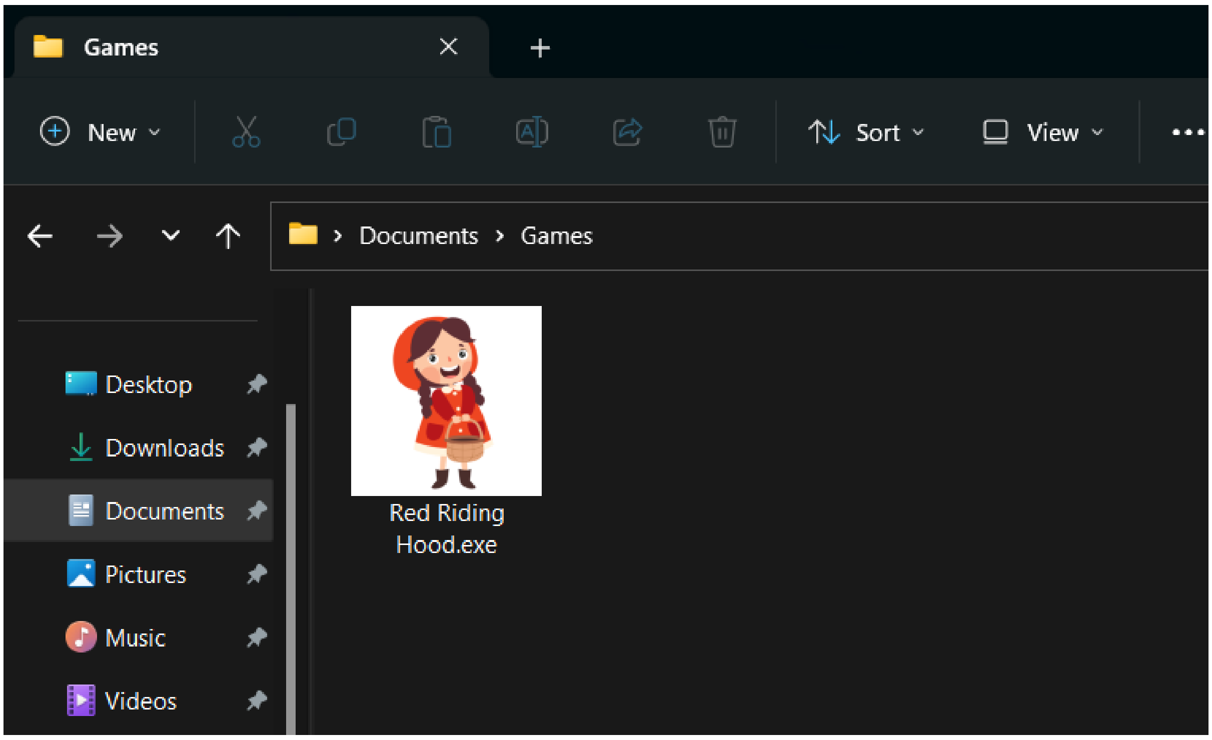
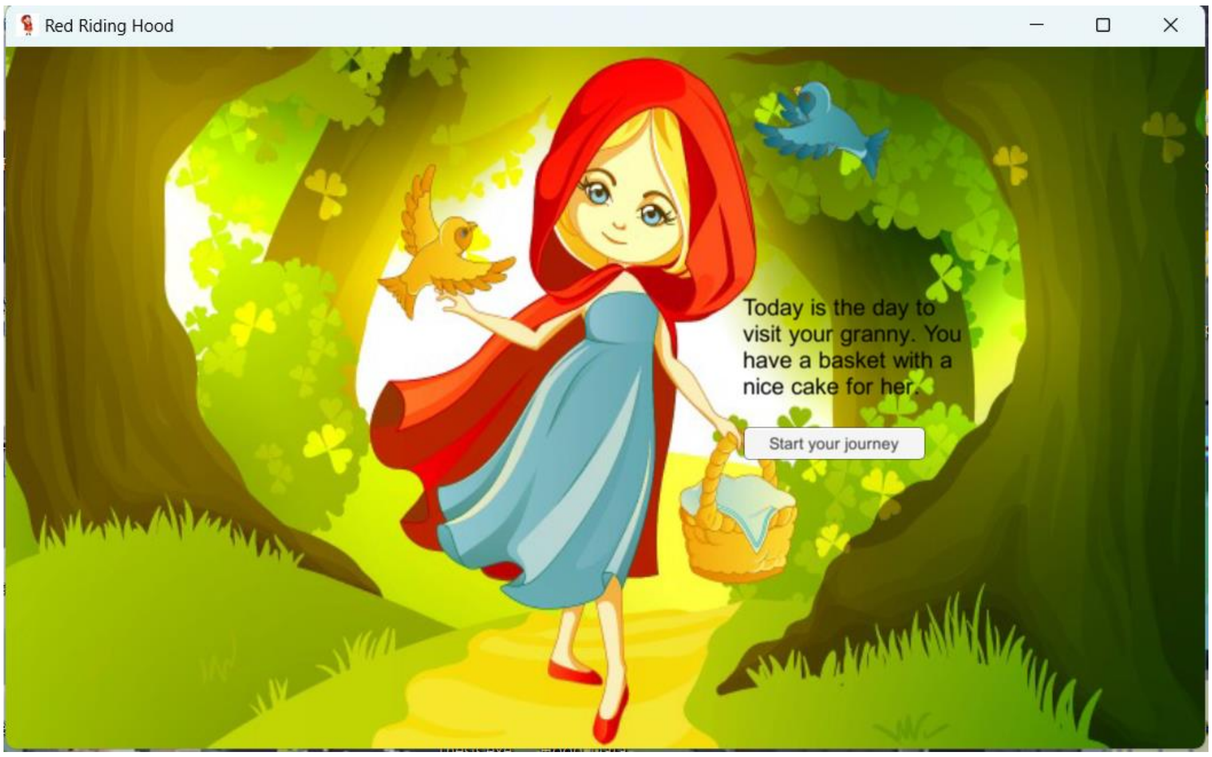


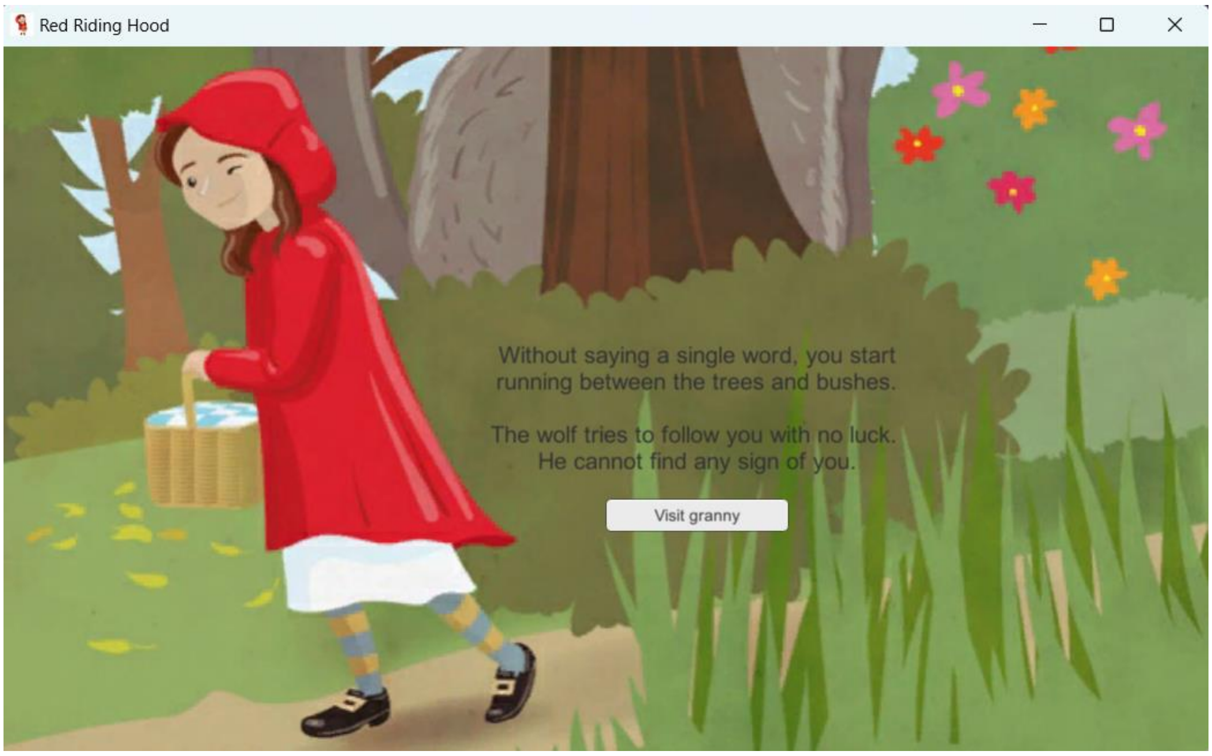
Figure 65: Screenshot of the .exe of the demo

For simplicity, after opening the .exe file, the demo game starts immediately without having a loading screen. For a complete game, a “Main Menu” screen with basic options such as “Start Game”, “Continue Game”, “Settings” and “Exit” is needed. Also, while playing the game, there should be at all times an option to “Save the game”, “Return to Main Menu”, etc., by clicking on the “Escape” button of the computer as it is usual for most video-games.



Today is the day to visit your granny. You have a basket with a nice cake for her.

Figure 66: The first scene of the demo



Without saying a single word, you start running between the trees and bushes.

The wolf tries to follow you with no luck. He cannot find any sign of you.

Figure 67: A scene of the demo

For the purposes of the demo, after reaching the ending of the game, an “Exit” button closes the program. In a complete game, this button should be replaced with one that returns to the Main Menu.

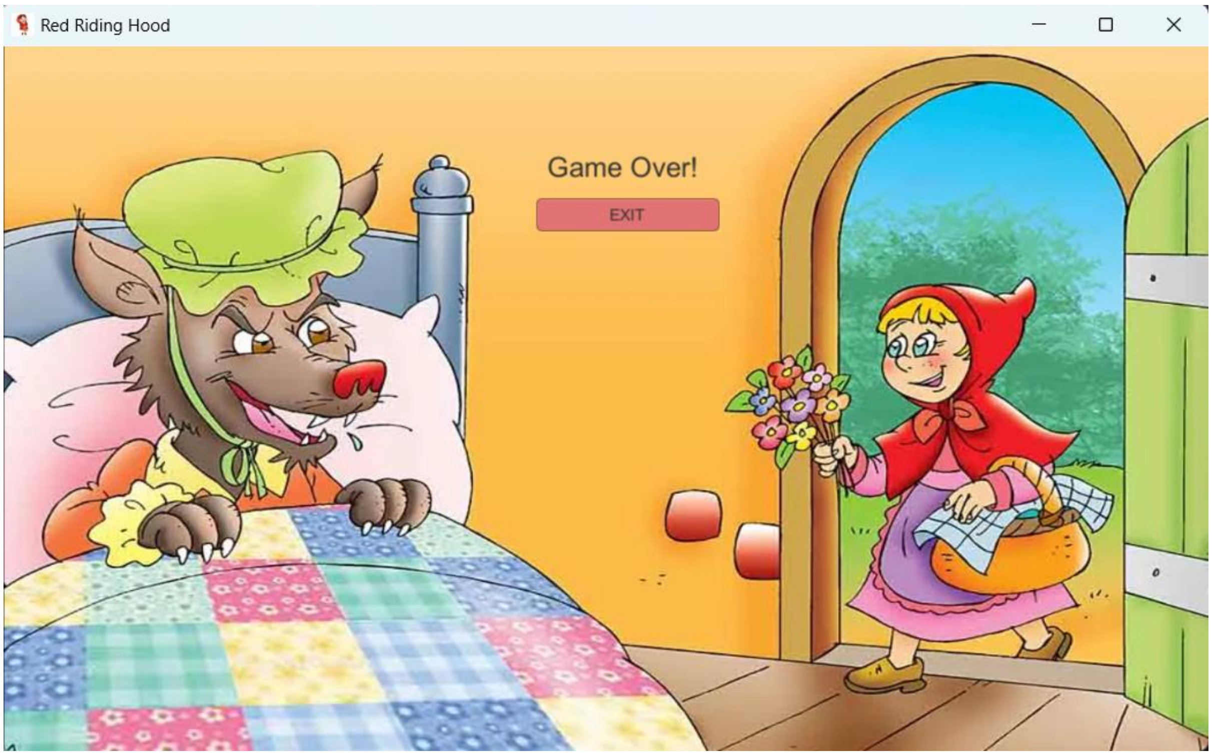


Figure 68: The bad ending of the demo

7. EVALUATION

The tool’s usability and effectiveness were assessed using a small yet skilled group of Unity programmers. A carefully chosen set of participants was recruited to encompass a diverse array of backgrounds and expertise levels. The evaluation procedure entailed introducing the software to the participants and granting them sufficient time to thoroughly investigate its functionalities and features. Participants were encouraged to engage in a variety of tasks, employing the software as they would in a real-world setting. Detailed guidelines and instructions were furnished to promote consistency in their evaluations.

Following the testing phase, the participants were asked to provide feedback and rate their experience using a standardized evaluation questionnaire. This questionnaire encompassed aspects such as usability, performance, reliability, and overall satisfaction with the software. The collected data and feedback from the participants played a crucial role in analyzing the tool's strengths, weaknesses, and areas for improvement, yet further studies would be required to obtain a more comprehensive understanding and validate the software for use.

7.1 Evaluation Procedure

The evaluation took place at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens in May 2023. The procedure has been approved by the Ethics Committee of the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens with approval number DIT_230712_01. A convenience sampling method was employed and a total of three participants were recruited for the evaluation. Each participant was briefed by e-mail about the study and a mutually convenient date and time was scheduled. Upon arrival, the researcher welcomed each participant and guided them to a designated table where a laptop with the tool pre-installed awaited their use (Fig. 69). Then, a consent form that outlines the nature and objectives of the study was given to the participant. The consent form ensures that individuals fully comprehend what their involvement entails, containing details about data privacy and confidentiality to reassure participants that their personal information will be protected. After having the participant’s consent, the researcher made an introduction regarding the Narralive and the Unity plug-in. The participant was asked to use the Unity plug-in by following a set of tasks, written in a document that was open on the laptop, along with Unity. When the tasks were completed and the participant was given time to test the plug-in, the researcher asked questions regarding their experience. Apart from the participant’s answers to the standard questions, the researcher was taking notes from observing the participant while using the tool. Each evaluation session lasted for about 20 minutes.

Figure 69: Observing a participant during the evaluation of the tool

To collect reliable data from the evaluation, the process was divided into 5 parts. These parts were:

1. Introduction of the StoryMaker and the Unity tool to the participant.
2. General questions regarding the participant’s technical background.
3. The participant was given time to try the tool by following a set of tasks.
4. The participant was asked to evaluate the tool by filling out a scale questionnaire.
5. The participant was asked open questions.

The questionnaire used is the SUS model (System Usability Scale), a widely used questionnaire-based tool used to evaluate the usability of a system, product, or interface. It was developed by John Brooke in 1986 and has since become one of the most commonly used usability evaluation methods. The SUS consists of a 10-item questionnaire that aims to measure the perceived usability of a system. Each item is rated on a 1-5 scale ranging from "Strongly Disagree" to "Strongly Agree".

In order to calculate the score from the SUS questionnaire, the following values must be calculated:

- X = Sum of the points for all odd-numbered questions – 5
- Y = 25 – Sum of the points for all even-numbered questions
- SUS Score = (X + Y) x 2.5

The separation between the odd-numbered and even-numbered questions is based on the fact that the odd-numbered questions have a positive wording while the even-numbered ones have a negative wording [29].

The questionnaire covers aspects such as the system's complexity, ease of use, user confidence, and overall satisfaction and the questions are the following:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.

9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Indicative open questions included:

1. What did you find more useful?
2. What did you find more difficult?
3. What kind of improvements, changes, and future work would you suggest?

The open questions were made in a discussion environment while observing the participant's use of the tool. After addressing all questions by the participant and covering the researcher's agenda, the session was concluded.

7.2 Evaluation Results

In this chapter the results of the evaluation are presented, divided for each individual reviewer. First, we gather the comments and thoughts that occurred from the open questions and the overall evaluation process, and then we calculate the results of the SUS questionnaire.

User 1

User 1 is a developer who has some basic experience with the concept of the *Story Maker*. They also have experience with Visual Novels from the player's point of view. As a developer, this user has knowledge mostly around JavaScript, Python and C#. Using Unity Engine they are developing a Matchmaking System.

Open questions:

1. *I found the graph's layout helpful and detailed, especially since it gives the feeling of a timeline. I also like the fact that I can navigate inside the graph and open Scenes in Unity from the nodes.*
2. *At first, I faced a small difficulty with understanding the classes that represent the story's data and I needed some time to study them.*
3. *I think that a future step that would have a good impact would be the ability to edit the graph.*

Other comments:

I would not change something from the current functionalities.

SUS score: 87.5/100

User 2

The second user who tested the tool has a significant amount of experience with Story Maker and they have knowledge of how it works. They are a developer who mostly codes in C# and C++ and, using Unity, they work with Virtual Reality. They have, also, created a first-person 2D video game. Lastly, they have played numerous video games of the Visual Novel genre.

Open questions:

1. *I found helpful that the way the graph is made provides the user with supervision on the story. Especially in a large, complex story, the user would definitely get lost without this feature. Also, the automation of the Scene generation where each*

scene file gets the title of the node has the same logical flow of how a developer would work.

2. *In principle, I did not find something difficult while using the tool, but I think that Documentation is necessary.*
3. *I can imagine having the ability to duplicate scene content and add scene objects in bulk. Also, I would like some more actions inside the nodes, like a "Clear Scene" button.*

Other comments:

I really like the ability to make massive scene generations and folder organization, since this feature would save me, as a Unity developer, a lot of time.

SUS score: 90/100

User 3

User 3 is an experienced Unity developer who, in contrast with the previous participants, has never seen the Story Maker and was introduced to it for the first time within the Unity Tool's evaluation. As a developer, this user mostly uses C# and JavaScript. With Unity they have developed 2D and 3D applications. Furthermore, using the Unreal Engine, they have created a mockup of a branching narrative system. Lastly, they have a significant amount of experience with playing Visual Novels.

Open questions:

1. *I think that it is positive that this tool was developed in the GUI environment since it makes what you are viewing easy to understand as it keeps a UI/UX consistency inside Unity Engine. I also think that the auto-generation features are a plus. Finally, I think that the structure of the C# classes is logical, clear and understandable.*
2. *I needed some time to view the code and start working on it.*
3. *I would suggest extending the features of auto-generation with the focus being on enriching the scenes.*

Other comments:

As a Unity developer I prefer using single scenes and changing their contents using code rather than having one scene file for every game's scene. Because of that, I would not use the "Generate all Scene files" feature.

SUS score: 85/100

8. CONCLUSION AND FUTURE WORK

While working on this thesis, research on the genre of Visual Novels and related authoring software was conducted. After this essential research, both in theoretical and technical aspects, the Unity tool was designed, developed, and tested. This work resulted in some conclusions and thoughts on future implementations.

8.1 Conclusion

The research conducted as part of this thesis confirmed that Visual Novels, a seemingly simple category of video games, have important potential for multiple domains. A common challenge encountered by institutions interested in creating interactive experiences for education, information, and showcasing cultural heritage is the disparity between professionals skilled in content design and technical experts capable of transforming data interactive experiences. To bridge this gap, this thesis focused on extending a web-based authoring tool into a plug-in for the Unity Engine. Despite the small sample size, the evaluation confirmed the potential of the tool and its helpful features.

8.2 Improvements and Future Work

Based on the evaluation results, an important improvement for the plug-in is the creation of documentation to be included in the Unity package. Another useful addition would be a tutorial in the form of a video, which describes the process of using the plug-in, showcasing its workflow and features.

When it comes to future work on the tool, there is a plethora of ideas that could be implemented to increase the features and utilities provided to game developers. Inspired by the suggestions made in the evaluation process, a future addition includes actions related to scene generation, such as “Clear Scene” and “Duplicate Scene.” Another addition would be the development of an algorithm that matches the scene’s data, such as text and choices, to the scene’s objects in order to automate the developer’s work even further. Lastly, an idea is to expand the usage of the plug-in for more web-based authoring tools that export their data in JSON format after investigating and analyzing their structure.

ABBREVIATIONS - ACRONYMS

JSON	JavaScript Object Notation
API	Application Programming Interface
PC	Personal Computer
BASIC	Beginner's All-Purpose Symbolic Instruction Code
EU	European Union
CD	Compact Disc
CD-ROM	Compact Disc Read-Only Memory
2D	Two Dimensional
3D	Three Dimensional
REST	REpresentational State Transfer
VR	Virtual Reality
HTTP	Hypertext Transfer Protocol
UI	User Interface
ID	Identification
EXE	Executive
SUS	System Usability Scale

REFERENCES

[1] Vrettakis, E., Lougiakis, C., Katifori, A., Kourtis, V., Christoforidis, S., Karvounis, M., and Ioanidis, Y. (2020). The Story Maker - An Authoring Tool for Multimedia-Rich Interactive Narratives. In AG. Bosser, D. E. Millard, & C. Hargood (Eds.), Interactive Storytelling. ICIDS 2020. Lecture Notes in Computer Science, vol 12497 (pp. 349–352). Springer, Cham.; https://doi.org/10.1007/978-3-030-62516-0_33

[2] Emotive project, Available: https://emotiveproject.eu/

[3] Katifori, A., Karvounis, M., Kourtis, V., Perry, S., Roussou, M., and Ioannidis, Y.: Applying Interactive Storytelling in Cultural Heritage: Opportunities, Challenges and Lessons Learned. In 11th International Conference on Interactive Digital Storytell-ing, ICIDS 2018, Dublin, Ireland, December 5–8, 2018

[4] Wikipedia, “Visual Novels”, Available: https://en.wikipedia.org/wiki/Visual_novel

[5] Linear, Non-linear and Branching Narrative Concepts, Available: https://krisnextgen.wordpress.com/linear-non-linear-and-branching-narrative-concepts/

[6] Wikipedia, “The Portopia Serial Murder Case”, Available: https://en.wikipedia.org/wiki/The_Portopia_Serial_Murder_Case

[7] Inverse, “39 YEARS AGO, THE PORTOPIA SERIAL MURDER CASE CHANGED VIDEO GAMES FOREVER” , Available: https://www.inverse.com/gaming/portopia-serial-murder-case

[8] Wikipedia, “Crystal Dragon”, Available: https://en.wikipedia.org/wiki/Crystal_Dragon

[9] Wikipedia, “Snatcher (video game)”, Available: https://en.wikipedia.org/wiki/Snatcher_(video_game)

[10] Wikipedia, “YU-NO: A girl Who Chants Love at the Bound of this World”, Available: https://en.wikipedia.org/wiki/YU-NO:_A_Girl_Who_Chants_Love_at_the_Bound_of_this_World

[11] RPGFan, “YU-NO: A girl Who Chants Love at the Bound of this World”, Available: https://www.rpgfan.com/review/yu-no-a-girl-who-chants-love-at-the-bound-of-this-world/

[12] Wikipedia, “Doki Doki Literature Club!”, Available: https://en.wikipedia.org/wiki/Doki_Doki_Literature_Club!

[13] Ren’Py, Available: https://www.renpy.org/

[14] Wikipedia, “Ace Attorney”, Available: https://en.wikipedia.org/wiki/Ace_Attorney

[15] Ace Attorney Online, Available: https://www.aaonline.fr/index.php

[16] GameBrew, “Ace Attorney DS”, Available: https://www.gamebrew.org/wiki/Ace_Attorney_DS

[17] DevOps School, “Comparison between XML vs JSON vs YAML”, Available: https://www.devopsschool.com/blog/comparison-between-xml-vs-json-vs-yaml/

[18] Wikipedia, “Clannad (video game)”, Available: https://en.wikipedia.org/wiki/Clannad_(video_game)

[19] Wikipedia, “Steins;Gate”, Available: https://en.wikipedia.org/wiki/Steins;Gate

[20] Oygardslia, K., Lærke Weitze, C., and Shin, J.: The Educational Potential of Visual Novel Games: Principles for Design. Replaying Japan, Vol. 2., 2020

[21] Katifori, A., Karvounis, M.,Kourtis, V., Perry, S., Roussou, M., and Ioannidis, Y.: Applying Interactive Storytelling in Cultural Heritage: Opportunities, Challenges and Lessons Learned. In 11th International Conference on Interactive Digital Storytell-ing, ICIDS 2018, Dublin, Ireland, December 5–8, 2018

[22] Parra, G., Klerkx, J., and Duval, E.: Understanding Engagement with Interactive Public Displays: an Awareness Campaign in the Wild, 2014

[23] Ink, Available: https://www.inklestudios.com/ink/

[24] Twine, Available: https://twinery.org/

[25] Yarn, Available: https://yarnspinner.dev/

[26] Unity, Available: https://unity.com/

[27] Wikipedia, “JSON”, Available: https://en.wikipedia.org/wiki/JSON

[28] Postman, Available: https://www.postman.com/

[29] John Brooke.: SUS: A quick and dirty usability scale (1995)