TÉCNICO LISBOA

Validating the plot for Interactive Narrative games

Carolina Maria Godinho Veloso

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Rui Filipe Fernandes Prada

Examination Committee

January 2021

Acknowledgments

First of all, I would like to thank all of my friends who helped me accomplish this work, particularly André and Nádia for their valuable support and prompt guidance during the development of this thesis and for being there when I needed most.

A big thank you to everyone that participated in the user tests made throughout this project and devoted their time to help me. Without all of you, this dissertation would not have been possible.

I would also like to thank my thesis advisor, Professor Rui Prada of Instituto Superior Técnico, for giving me the opportunity to explore the subject of this dissertation and for his guidance.

Finally, I want to thank my family for their support and patience. I love you.

Abstract

The authoring of an interactive dialog in video games is an overwhelming and complex task for game writers. Developing an Interactive Narrative that balances authorial intent and players' agency requires frequent in-depth testing, and the limited range of tools to assist authors in verifying their story can limit the creation of more complex narratives.

Through reviews of the existing literature, we discuss the challenges of Interactive Story design and provide a model consisting of a set of metrics for testing interactive dialogs. Using this model, we developed a prototype for the Story Validator. This debugging tool allows game writers to experiment with different hypotheses and narrative properties in order to identify inconsistencies in the authored narrative and predict the output of different playthroughs, with visual representation support.

Using the Story Validator, we conducted a series of user tests to investigate whether the tool adequately helps users identify problems in the game's story. The results showed that the tool enables content creators to easily test their stories, setting our model as an essential step towards automated authoring assistance for interactive narratives.

Keywords

Interactive Narrative, Authoring tools, Game Writing, Twine

Resumo

A autoria de diálogos interativos para videojogos é uma tarefa árdua e complexa para os criadores de jogos. O desenvolvimento de uma Narrativa Interativa que harmoniza a intenção autoral e a agência dos jogadores requer testes frequentes e aprofundados, e o número limitado de ferramentas para auxiliar os autores na verificação de suas histórias pode restringir a criação de narrativas mais complexas.

Por meio de revisões da literatura existente, analisámos os desafios de design de histórias interativas e construímos um modelo que consiste num conjunto de métricas para testar diálogos interativos. Aplicando este modelo, desenvolvemos um protótipo para o Story Validator, uma ferramenta de debbuging que permite aos escritores de jogos experimentem diferentes hipóteses e propriedades narrativas para identificar inconsistências na narrativa criada e prever o resultado de diferentes playthroughs, com suporte gráfico visual.

Utilizando o Story Validator, conduzimos uma série de testes de utilizadores para investigar se a ferramenta ajuda adequadamente os usuários a identificar problemas na história do jogo. Os resultados mostraram que a ferramenta permite aos criadores de histórias testarem facilmente as suas histórias, distinguindo o nosso modelo como uma etapa essencial para a assistência automatizada à autoria de narrativas interativas.

Palavras Chave

Narrativa Interativa, Ferramentas de Autoria, Escrita de Jogos, Twine

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ABL	A Behavior Language
AI	Artificial Intelligent
BFS	Breadth-First Search
CYOA	Choose Your Own Adventure
DFS	Depth-First Search
DM	Drama Manager
DODM	Declarative Optimisation-based Drama Manager
FAtiMA	Fearnot AffecTIve Mind Architecture
GAIPS	Group of Artificial Intelligence for People and Society
GUI	Graphical User Interface
IN	Interactive Narrative
JSON	JavaScript Object Notation
MCTS	Monte Carlo Tree Search
PACE	Player Appraisal Controlling Emotions
PaSSAGE	Player-Specific Stories via Automatically Generated Events
PDDL3.0	Planning Domain Definition Language version 3.0
SBDM	Search-Based Drama Manager
SEQ	Single Ease Question
SUS	System Usability Scale
UCB1	Upper Confidence Bound

1 Introduction

Contents

1.1 Motivation and Problem

The presence of an immersive story or narrative in video games is often a central part of game design [3, 4]. As an attempt to combine interactivity and storytelling, Interactive Narrative (IN) is a form of nonlinear narrative that gives the game an element of choice and, consequently, allows the player's actions to alter the course of the story [5, 6].

Although IN has enormous potential, enabling the creation of interactive systems that combine player interaction and dynamic plots, its main limitation lies in the challenges that authors face when writing this type of stories [7]. Developing an IN where players can feel immersed and engaged involves making the player's actions and choices have a powerful influence on the direction of the narrative, which makes it challenging for the author to guarantee a well-formed story.

Most traditional approaches rely on extensive and rigorous playtesting to obtain information on how the players experience the narrative and what might need to be improved. Playtesting provides insightful information that is not anticipated during development, helping authors ensure that both the game's narrative and the player's behaviour are well balanced [8, 9]. However, obtaining quality feedback for a detailed analysis can be challenging, expensive, and time-consuming [10].

Various works have offered different solutions to handle narrative conflicts caused by user behaviour in-game. However, hardly any works have focused on letting the author simulate and question their narrative during development. Instead, they opt for online Artificial Intelligent (AI) approaches (such as drama managers [11]) that, during gameplay, provide ways to dynamically adapt the narrative and resolve conflicts created by unintended player's actions. These approaches create changes to the narrative that the author might not have intended, making them lose control over their story.

On the other hand, most authoring tools, such as Twine [12], Tinderbox and RenPy [13], while facilitating the creation of IN by providing a bird-eye view of the overall story, they lack the proper tools to identify possible continuity errors, to keep track of specific narrative properties and to envision the output of playthroughs prior to human playtesting. Instead, these tools rely heavily on the author's intuition to foresee these challenges or on an exhaustive number of later playtests.

1.2 Objectives

This work outlines the development of a tool that supports game writers in the creation of IN, whilst maintaining the human author's directorial control. Essentially, we strive to maintain the idea of authorial intent [14] and develop a system that allows human authors to express their artistic intentions without feeling constrained, by providing ways for the author to test and question their narrative through different story metrics, and to find possible design issues.

While online AI approaches are a popular solution, this dissertation focus instead on an offline approach, working as a validating tool, that attempts to pre-determine possible narrative problems during the game's developmental stage, by letting the author explicitly test different hypotheses and narrative properties. Although both online and offline are viable solutions and, ultimately, the most effective results could be achieved by using them together, we decided to focus on the latter solution, which is indispensable even in the presence of the former.

1.3 Contribution

As a product of this work, we have designed, developed, and tested a prototype for the **Story Validator**.

Based on other studies and existing authoring tools, we have underlined the current challenges concerning the authoring process of Interactive Narratives. With this information, we worked on conceptualizing a model for testing interactive dialogues for video games.

Using the model proposed, we developed a prototype for the **Story Validator**. This tool takes a JavaScript Object Notation (JSON) file outputted from Twine and, by treating narrative as a branching tree, uses a Depth-First Search (DFS) based algorithm (presented in Section 4.3) to traverse all possible narrative paths and provide insightful data on different story metrics, and design issues encountered, via visual representations. This work has been made publicly available on the Github repository https://github.com/carolveloso97/StoryValidator .

Through user feedback and statistical analysis, we have managed to improve less intuitive aspects of our first prototype and develop an improved version of the tool, which allows game writers to create more complex interactive narratives without feeling limited by a variety of design challenges.

1.4 Outline of the Dissertation

This dissertation is organised as follows: In Chapter 2, we present some of the concepts necessary to understand the work developed. Then, in Chapter 3 we thoroughly analyse the different approaches developed so far in the field of IN, initially focusing on AI-based interactive stories, as well as exploring some of the existing authoring tools for creating IN and how they tackle (or not) some of the problems that we have attempted to solve in Chapter 4. In this chapter, we also present our conceptual model and define the structure of the **Story Validator** and its functionality. The user testing is presented in Chapter 5, while in Chapter 6, we analyse the results and draw conclusions. Finally, in Chapter 7, we reflect on the work developed and present some suggestions for improvements that may be implemented in the future.

2 Concepts

Contents

This chapter explains some of the concepts needed to understand this dissertation, in particular, the definition of Interactive Narratives (IN) and common authoring challenges. Additionally, we will also give a brief explanation of Tree Data Structures and their traversal algorithms: Breadth-First Search (BFS) and Depth-First Search (DFS).

2.1 The Concept of Interactive Narratives

Interactive narratives (IN) are a form of nonlinear storytelling, where the player's actions and choices have a direct influence on the unfolding of the story. They function similarly to the Choose Your Own Adventure (CYOA) storybooks of the 1970s, popularised by Bantam Books [15], where the reader is faced with various decision points at which they must make a choice that alters the course of the story. At each decision point, the story branches in different directions, often leading to different outcomes.

Games with nonlinear narratives often have higher levels of interactivity, which facilitates the sense of *causal-agency* [16]. *Causal-agency* is the perception of being in control, meaning the player feels immersed in the virtual world as they believe that their actions have significant effects on the development of the narrative. In IN, instead of being tied up to one linear story, the player feels free to take their own path. This “freedom”, however, is often an illusion. Each branch in the story has to be carefully tailored by the game's writer, who consequentially has to predict the different possible player behaviours that can affect the story at various states so that they can present those choices to the player.

Nevertheless, narrative choice needs to be thought through carefully and authoring an IN is not done without a few challenges.

2.2 Authoring Interactive Narratives

The authoring of interactive stories in video games represent is an overwhelming and complex task for game writers, both in narrative design and implementation. The authoring process for IN is extremely different from that of linear narratives [17]. First, in linear stories the author is not required to create extra material to cover for different player's actions, therefore, less content needs to be authored. Furthermore, linear stories are less prone to bugs and inconsistencies since there is only one possible course of events. Because in nonlinear stories (such as IN) the player can change how the narrative unfolds, the author needs to guarantee that the plot remains coherent or it might hinder the players' experience.

2.2.1 Authoring challenges in Interactive Narratives

The following section presents a non-exhaustive list of challenges pertaining to the development of interactive narratives.

Authorial intent vs Player agency

The most challenging problem with interactive narratives is the necessity to balance authorial intent and player agency in the context of storytelling [18]. Authorial intent is the trajectory that the game writer wishes the player follows, regardless of how the player acts during the game. It is often referred to as the author's goals [14]. Because interactive narratives allow for the user to interact freely within the story world, users often have the power to behave in ways that are inconsistent with the plot. This can either prevent the plot from advancing or make the player experience the story in a way that was not intended by the author.

Narrative exponential growth

Additionally, as the plot grows in complexity and the number of decision points increases, the authoring experience will often require substantial changes to keep the narrative coherent and can become overwhelming for the author [19]. For example, if a narrative has twenty decision points, where each branch off into two new different paths, then the author will have to write and monitor of 2^{20} = 1048576 different branches. Not to mention that the impact a choice has may end up only revealing itself in future states of the story, which is not always easy to predict. In worst cases, the game's writer might end up with branches that lead nowhere and hinder the player's experience.

Monitoring other game variables

Besides what choices the player makes, the development of the story can be based on different variables and status, that are updated throughout to game. A common example are the karma systems like in *Fallout 3* [20], which consider the player's good and evil deeds, and shape the world around them as they progress. In more complex narratives, with multiple branches, game variables can become difficult to keep track and, like the previous point, their consequences may only unravel in later states in the game, which is not easy for the author to foresee during development.

Measuring the impact in user's experience

The player play-styles are reflected not only in how they interact with the narrative but also on how they expect the story to unfold [21]. Due to the complex nature of interactive narratives, it is difficult for the author to predict the player's behaviour and their overall emotional experience. For example, the author might want to ensure that player follows a formerly defined emotional trajectory (e.g. [22]), as a result of the choices they made during gameplay. This is not only challenging to predict without prior human playtesting, as it is hard to ensure during development.

2.3 Branching Trees

A common strategy for implementing interactive stories is to use branching trees [23], in which each node represents a decision point. In games that use branching narratives, the author needs to predict what actions the player might want to take and outline the proper response to them, scripting each decision point into the system.

The main drawback of this system is that, due to its structure, branching trees grow exponentially as the number of decision points increases, which leads to a high workload on the part of the author to compute such complex systems.

2.3.1 Trees Structure and Traversal

In computer science, a tree [24] is a non-linear data structure used to store and organize data of hierarchical nature in a graphical form. A tree is a graph *G* composed by a set of vertices *V*, called "nodes", and a set of edges *E* connecting pairs of vertices, called "branches", which should not form a cycle. The top node is the "root", and every edge joints a "parent" to a "child" node. Nodes without children are referred to as "leaves".

Tree traversal (also known as tree search) refers to the process of visiting every node in a tree data structure. Unlike linear data structures (e.g., Arrays), a tree has multiple ways of being traversed.

Tree traversal algorithms can be classified in the following two categories: Uninformed (or Exhaustive) methods, where the search is carried out without any additional information, such as Breadth-First Search (BFS) Algorithm and Depth-First Search (DFS) Algorithm (see Figure 2.1); and Informed (or Heuristic) methods, where the search is carried out by using additional information to determine the next step towards finding the solution, such as Best First Search algorithms and Monte Carlo Tree Search (MCTS).

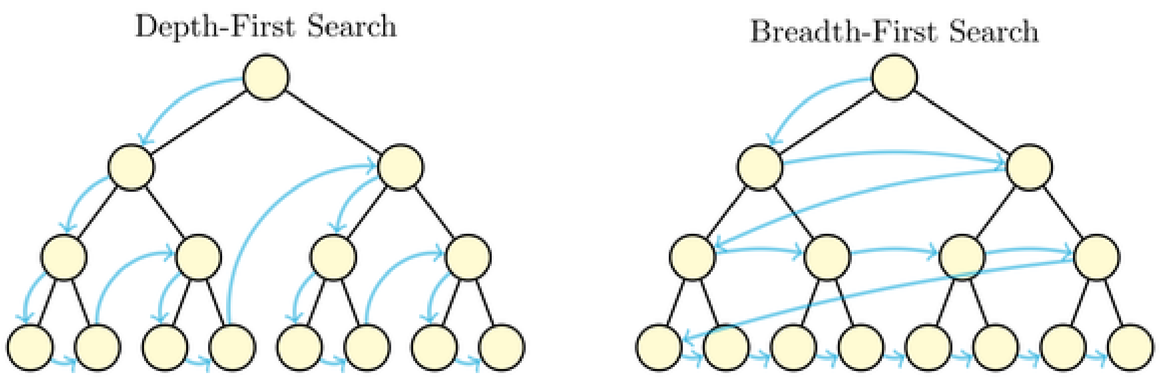


Figure 2.1: Differences between Depth-First Search and Breadth-First Search [1]

2.3.2 Breadth-First Search (BFS) Algorithm

BFS is a recursive algorithm that traverses all of the reachable nodes of a graph or tree data structure, by following the “go wide, bird’s eye-view” philosophy. Invented in the late 1950s by Edward F. Moore, who used it to find the shortest path out of a maze [25, 26], BFS starts by visiting all nodes of current depth before moving to the next depth in the tree. Algorithmically, this means that all reachable vertices at a distance *k*, are visited before any vertex reachable at *k + 1*. The algorithm repeats this process until all nodes that are reachable from the root vertex have been discovered.

To find the nodes that are at distance *d* from the root node, BFS has a Time Complexity of $O(b^d)$, where *b* represents the branching factor. To keep track of the vertices visited, BFS often uses a queue (FIFO) data structure.

2.3.3 Depth-First Search (DFS) Algorithm

Depth-First Search (DFS), created in the 19th century by French mathematician Charles Pierre Trémaux [27] as a strategy for solving mazes [28, 29], is a recursive algorithm that traverses all of the reachable nodes of a graph or tree data structure. As implied by its name, DFS follows the “go deep, head first” philosophy, starting at the root and exploring as deep as possible along a given branch, until it reaches a leaf. It then starts backtracking until it finds an unexplored branch to traverse. In other words, after reaching an end-node, DFS proceeds to visit the unexplored edges that come out of the most recently discovered node. The algorithm repeats this process until all nodes that are reachable from the root vertex have been discovered.

The time complexity for DFS search is $O(b^m)$, where *b* is the branching factor and *m* represents the maximum depth. Each of the nodes generated that need to be visited, are added to a stack (LIFO) data structure. Once the algorithm reaches a leaf node, the nodes start to be popped off from the top of the stack.

2.3.4 Best First Search

An example of a informed search algorithm, is the Best-first search, also known as Greedy search. While the BFS and DFS algorithms blindly explore paths without considering any cost function, the Best First Search uses an evaluation function, or heuristic, to decide which among the various available nodes is most promising (i.e., the ”best”) before exploring. Some cases of Best-first search include Dijkstra’s algorithm [30] and the A* algorithm [31].

2.3.5 Monte Carlo Tree Search

The MCTS is a heuristic search technique that has been applied to simulate human decision-making [32]. MCTS allows the agent to explore the most promising paths and eventually reach the best outcome by following the steps:

1. Selection: Start at the root node and recursively select the most promising child-node, until an unexpanded node is reached.

   Traditionally, the MCTS algorithm uses the Upper Confidence Bound (UCB1) formula given bellow:

   $$UCB1 = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N_i}{n_i}} \tag{2.1}$$

   Where w_i is the number of wins the node considers after the i-th move, n_i is the number of simulations for the node considered after the i-th move was visited, c is the exploration constant and N_i stands for the total number of simulations after the i-th move run by the parent node.

2. Expansion: Unless the unexpanded node is a terminal state (e.g., win/loss/draw), expand the node and choose a new child-node, to increase the number of options available in the game.

3. Simulation: Until reaching the end of the game, simulate a playout (i.e. rollout) of the game. This playout can be reached by performing a set of random moves or by following a fixed strategy.

4. Backpropagation: After reaching the terminal state, traverse the tree upwards, updating the information of each node, using the reward of the playout.

These four steps are repeated multiple times until the resources given to the agent are exhausted, and the agent selects the next move that has the best outcome.

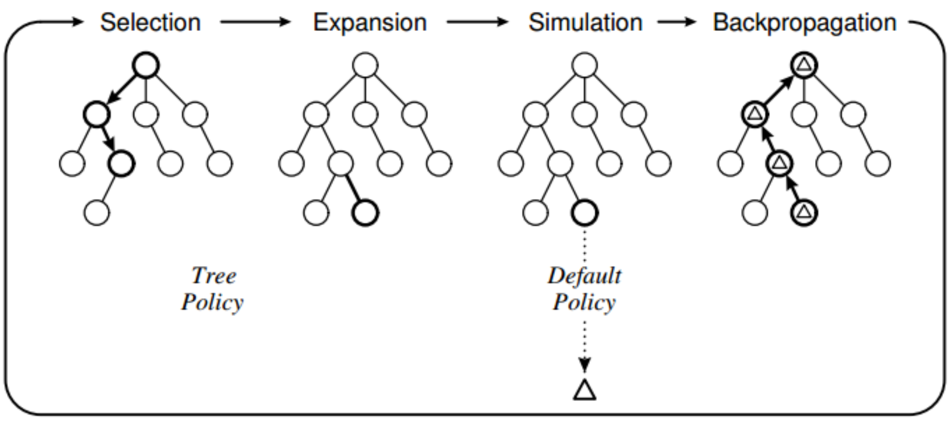


Figure 2.2: Scheme of the Mont Carlo Tree Search [2]

3

State of The Art

Contents

In this chapter, we will analyse several existing works regarding the authoring of IN. First, in Section 3.1, we explore the idea of online Artificial Intelligent approaches, particularly drama managers. Afterwards, we explore some of the existing authoring tools currently available for creating interactive narratives and their limitations.

3.1 Interactive Drama

In order to maintain narrative coherency even when the player has freedom of choice, some past approaches have focused on the idea of integrating real-time Artificial Intelligence methods to shape the narrative, allowing players to freely interact the game but, at the same time, making sure the narrative still follows a coherent structure. A popular example, first proposed by Bates [11], is the use of a *drama manager*.

A Drama Manager (DM) is an omniscient agent that monitors the game world and guides the player's experience through the story, by searching for possible future plot points based on the evaluation of the current game world, while still allowing the players to interact freely. The two main approaches for drama managers are Search-Based Drama Manager (SBDM) [33] and Declarative Optimisation-based Drama Manager (DODM) [34]. In the case of SBDM, the DM monitors the player's behavior, watching out for possible conflicting actions, and utilizes a set of DM actions to guide the player towards a set of plot points modeled by the author. By contrast, the DODM attempts to incorporate player actions into the narrative by dynamically reconfiguring the story world, so it still achieves all story objectives pre-defined by the author.

Figure 3.1: The interactive drama *Façade*

For instance, one of the most famous examples of the implementation of a DM is the Mateas and

Stern interactive drama *Façade* [35–37]. *Façade* uses its DM to monitor and update the simulation in real-time in response to text the user inputs by selecting the next story *beat*. A story *beat* corresponds to a dramatic point in the narrative, annotated by the author. Each *beat* also consists in a set of *preconditions* that need to be fulfilled for the beat to run; *effects* that determine the consequences the beat has on the story, upon completion; and *goals* that correspond to the actions the agents need to carry out.

In the game, the player is invited to Grace and Trip's house, the two AI characters, and it involves the player having to type text on the keyboard to speak and communicate with them and try to save their marriage. The narrative and dialog are written using a language called A Behavior Language (ABL) [38], that coordinates the behaviour of both characters.

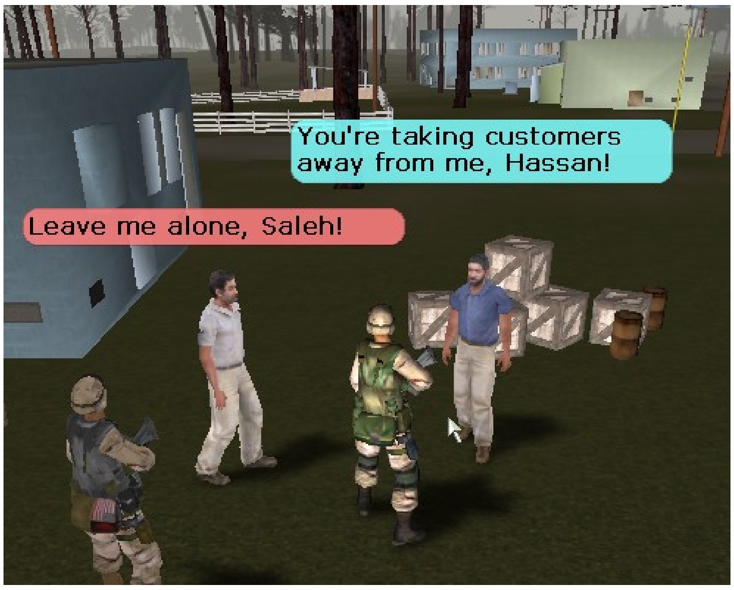


Figure 3.2: The Automated Story Director operating in *IN-TALE*

Similar to Façade, Riedl et al. presented the prototype for *IN-TALE* (the Interactive Narrative Tacit Adaptive Leader Experience) [39], where the agents were implemented using the ABL language. When the user interacts with an agent, using either dialog or physical interaction, the agent handles that interaction according to a collection of behaviours directed by the DM called the *Automated Story Director*. The way the *Automated Story Director* handles user interactions is by maintaining a script of expected events that it then uses to provide directions to the character agents. On the case the user performs an action that is not predicted by the *Automated Story Director*, and creates an inconsistency between the expected narrative and a current state of the simulation, the DM is able to detect those discrepancies and adapt the story, so it still achieves all story goals. The way the DM adapts the story is by first attempting to disconnect all the events that were directly linked to the expected (but not accomplished) narrative state and then provide new narrative events that restore the player to the state he was prior to their actions.

Later on, Riedl used the same *Automated Story Director* [40], this time creating an interactive version of the original *Little Red Riding Hood* story, where even if the player decides to detour from the original story and, for example, kills the Wolf prematurely, the story can allow for this action to occur and still be able to reach all the following plot points specified by the author (e.g., Wolf eats Granny).

Once again, to do this, Automated Story Director automatically plans out new narrative follow-ups to respond to the player's actions and achieve all concrete narrative goals pre-defined by the author. One proposed solution to the previous action, given by the DM, is to have a fairy resurrect the Wolf. The Automated Story Director guides the narrative by following what they call the *exemplar narrative*. The *exemplar narrative* is defined by the author and encodes the decisions he expects the player to perform. Even if the player does not follow said decision, the DM must be able to adjust the narrative in order to meet the main plot points specified in the *exemplar narrative*.

3.1.1 Author's Goals

As an extension of these works, Riedl went on to propose the inclusion of *author goals* to the planner [14], to force *complexity* in the narrative generation and to allow authors to inject their preferences and intents into the narrative. According to him, the *author's goals* describe the states in the narrative that must be achieved before story completion. If the author desires, the goals can be specified to occur in a particular order. To accomplish this, the *author goals* are defined in the form of sets $(set_1 \ldots set_n)$, that need to occur for the path taken by the planner to be considered valid. In other words, the planner is constrained to produce a particular narrative path by adapting the narrative, so it meets all author goals.

Similarly, Porteous and Cavazza [41] propose that the previously mentioned Riedl's narrative *complexification* can be achieved by using the Planning Domain Definition Language version 3.0 (PDDL3.0) state trajectory constraints. In other words, the author is able to stipulate a set of events (or goals) that must be achieved in the narrative, using different operators, like for example, *sometime* and *sometime-before*. Still, without the need to specify the exact timing, those events occur in the narrative. Each event goal is independent of the overall narrative goal, limiting the time-specific connections between events to specific cases, where a particular event needs to occur before another.

In order to control the narrative, the algorithm is given constraints — a set of facts about the story and overall goal. Then the same algorithm generates a plan, i.e., a sequence of operators that contribute to achieving the goal and that, at the same time, can satisfy all the constraints. By using these constraints, the algorithm can then generate an IN where events can be manipulated to respond to different player's actions without it conflicting with the rest of the narrative. It also means that the author can individually build and change any narrative's event at any point in time, without it altering the rest of the story.

3.2 Player Modelling for Interactive Narratives

In the previous Section, we have emphasized the importance of the author's intention in relation to how the narrative should be directed in response to the users' actions. In this next Section, we will focus on how the player play-styles are reflected not only in how they interact with the narrative but also on how they expect the narrative to unfold.

Regarding this work, in order to more accurately pre-determine possible narrative problems and account for various possible user actions at different points in the narrative, it is essential to learn different user characteristics and define player models. Player Modelling is the study of computational models of players in games based on the observation and capture of behavioural patterns.

An example of this is the Player Appraisal Controlling Emotions (PACE), an AI experience manager that shapes the narrative of the story in response to the player's actions, to ensure that players follow the emotional trajectory formerly defined by the author [22]. PACE achieves this, by monitoring the player's play-style during the gameplay and, then, creating a player model, which they use to predict player's emotions during the game and thus, use it to modify the narrative, so it elicits the response most similar to what the author specified beforehand. To do this, PACE allows the author to specify at key points in the narrative what emotional state they want their players to follow. The AI system then decides which narrative events are best to possibly make players experience those emotions.

Another important contribution to the study of how player models influence the unfolding of an interactive story is the system Player-Specific Stories via Automatically Generated Events (PaSSAGE) [42]. PaSSAGE is an interactive storytelling system that, by observing the player actions in the game, automatically learns the model of the player's preferred playstyle. The system then uses this model to dynamically select story events that match the player play style. PaSSAGE uses Robin's Laws five game player types [43] — *Fighters*, *Power Gamers*, *Tacticians*, *Storytellers*, and *Method Actors* — to create a personalized model for each player. Every player model is defined as a vector with different values, one for each of the five styles of play (e.g., F = 2, M = 1, S = 0, T = 1, P = 0). As the player performs actions during the gameplay, the weight of the values changes and updates accordingly.

Unfortunately, in the context of interactive narratives, the application of Player Modelling to decision-making remains mostly unexplored. In the next section, we explore the use and study of player models in another area.

3.2.1 Player Modelling in Automated Playtesting

Player Modelling has also been explored in other areas such as Automated Playtesting, where agents driven by artificial intelligence are used to emulate human behaviour during gameplay.

Currently, the main approach that game developers take to analyse their game is through human playtesting [44]. Playtesting is often conducted during the video game design process to help developers improve their games, by providing insightful information about the player's behaviour, preferences, and how they interact with the game overall. In addition, playtesting is also essential to access the flaws in the gameplay and to identify unclear or uninteresting sections, adding fun elements, or balancing existing ones. However, recruiting and testing users is expensive and time-consuming. One proposed solution to reduce the costs of playtesting is through Automated Playtesting [45], where intelligent agents play through the game, simulating testing sessions, allowing the developers to gather valuable information regarding the current game design and find possible flaws.

More recently, several approaches have explored the application of Player Modelling to Automated Playtesting. For instance, Holmgård et al. [46] applied archetypal generative player models called Procedural Personas to enhance Automated Playtesting. By capturing different players' characteristics based on player models or from observed data, these Procedural Personas define player archetypes and play through the game, acting as critics and evaluating the game content. To play their test game, Minidungeons 2, four Procedural Personas were defined — Runner, Monster Killer, Treasure Collector and Completionist — each with a different primary objective. These personas are implemented using a variation of the MCTS. By using persona-specific evaluation formulas in the MCTS, and replacing the standard UCB1 equation of MCTS, Holmgård et al. [46] were able to create agents with differentiated goals and behaviours.

Another example of using Procedural Personas to evolve the utility function for the MCTS is the work of Mugrai et al. [47] of Match-3 games. Similarly to the previous work mentioned, they defined four Procedural Personas — Agent MaxS, Agent MinS, Agent MaxM and Agent MinM — each modeling a different type of playstyle.

Although they did not apply it to Automated Playtesting, Devlin et al. [48] created agents whose behaviour reflected players playstyle, to play a game of Spades, by biasing the MCTS with player models data. The data used for biasing the MCTS was based on the information gathered from a random sample of human players who had played AIFactory Spades, a free mobile game.

Other works have designed human-like agents for Automated Playtesting by building them based on already established player models and without the use of the MCTS. For instance, Stahlke et al. developed a framework, named PathOS, that simulates human testing sessions using intelligent agents [49]. These agents mimic the behaviour of human players by following a set of heuristics (e.g., curiosity, aggression, and completion), that are configured to reflect different playstyles. These heuristics were designed based on player typology, namely, the player satisfaction model BrainHex.

The BrainHex model [50, 51] was developed with the goal of better understanding what experiences attract some players to play a video game but repel others, based on patterns in their gameplay behaviour. The BrainHex model was developed based on the two previous *Demographic Game Design* models of the same authors — (DGD1[1] and DGD2[2]) — as well as the findings from different neurobiological research papers, such as the work of Biederman and Vessel related to the release of endorphins in the brain, which they relate to human preference for certain experiences [53]. The model is composed of seven player archetypes: Seeker, Survivor, Daredevil, Mastermind, Conqueror, Socializer and Achiever; each associated with different parts of the human brain.

3.3 Interactive storytelling authoring tools

The authoring process remains one of the most significant challenges in the creation of interactive stories, and there is a need for authoring tools that both enable and assist authors in the creation of their content. Authoring tools for interactive narratives provide different visual representations and structural elements that allow authors to write creative works.

3.3.1 Tinderbox

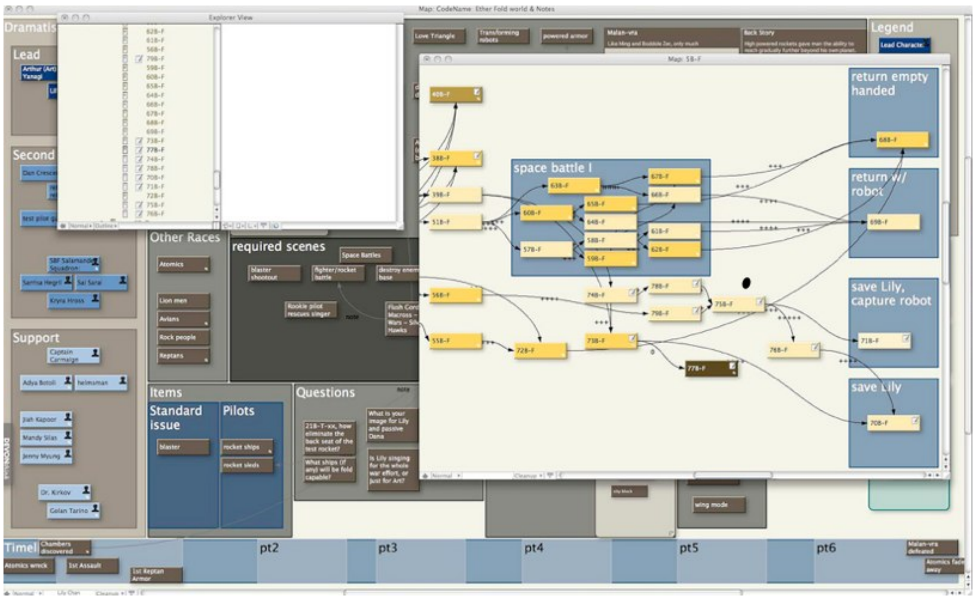


Figure 3.3: Tinderbox authoring environment

Tinderbox[3], by Mark Bernstein [54] is a commercial spacial hypertext editor. The tool is mainly targeted at supporting authors' writing processes by creating and associating notes and ideas in the form

[1]Demographic Game Design 1 — Complete research of the model published in [52]
[2]Demographic Game Design 2 — Mentioned in [51]
[3]http://www.eastgate.com/Tinderbox/

of maps and charts. As they create their notes, authors can link and arrange them, building relationships between different text-based items.

Tinderbox also includes intelligent agents who continuously scan the author's notes, looking for and highlighting those that meet specific criteria, such as overdue tasks and key terms. The agents' behaviour is very simplified, or, as the author puts it, "users may regard [these agents] as a simple user interface shortcut."

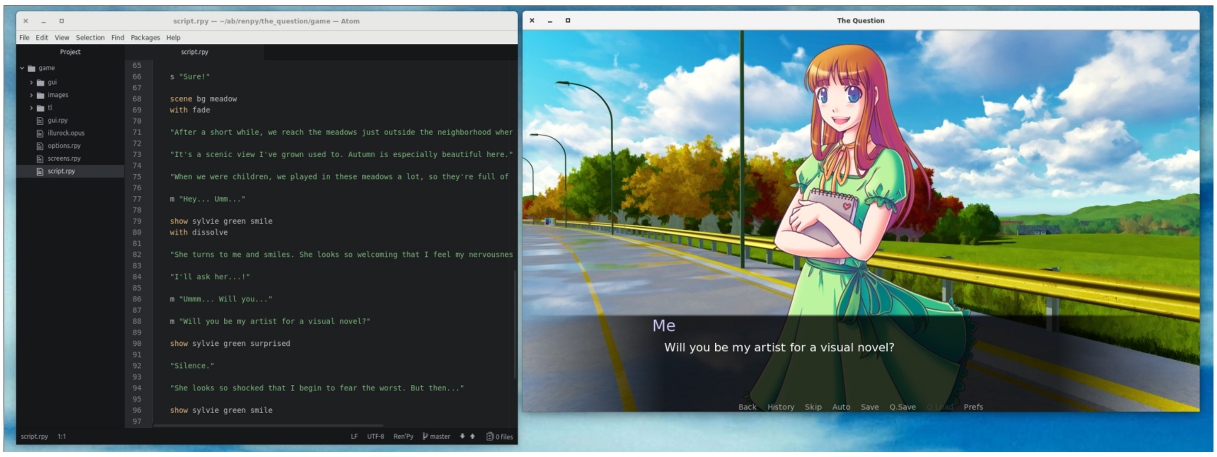
Ultimately, while this authoring tool provides many visual signifiers to utilize on individual nodes (such as colour and shape), it lacks metrics that inform the author on the possible outcomes and errors.

3.3.2 Ren'Py

The Ren'Py[4] engine, by Tom 'PyTom' Rothamel, is a free, open-source software for creating visual novels — computer-based stories that combine images, sound and nonlinear storytelling.

Ren'Py is based on the Python programming language and is used to create kinetic stories that integrate the player's choices and multiple endings and track different variables and other game stats. The tool, however, does not provide a visual representation of the story, such a node graph, and since a story has to be written directly on the text editor, if a project reaches a larger scale, the author might become overwhelmed by the amount of paths they have to monitor.

Ren'Py also includes a check script, called Lint, that runs through the project checking for errors and unusual behaviour. However, this system only includes errors in the project's python code and, according to the Ren'Py documentation page, "Lint is not a substitute for thorough testing" [55].


script.rpy — ~/ab/renpy/the_question/game — Atom
File Edit View Selection Find Packages Help
Project
script.rpy
game
gui
images
tl
gui.rpy
illurock.opus
options.rpy
screens.rpy
script.rpy
script.rpy 1:1
LF UTF-8 Ren'Py master 0 files
The Question
Back History Skip Auto Save Q.Save Prefs

```

    s "Sure!"

    scene bg meadow
    with fade

    "After a short while, we reach the meadows just outside the neighborhood wher

    "It's a scenic view I've grown used to. Autumn is especially beautiful here."

    "When we were children, we played in these meadows a lot, so they're full of

    m "Hey... Umm..."

    show sylvie green smile
    with dissolve

    "She turns to me and smiles. She looks so welcoming that I feel my nervousnes

    "I'll ask her...!"

    m "Ummm... Will you..."

    m "Will you be my artist for a visual novel?"

    show sylvie green surprised

    "Silence."

    "She looks so shocked that I begin to fear the worst. But then..."

    show sylvie green smile

```

Me

Will you be my artist for a visual novel?

Figure 3.4: The Ren'Py development environment

[4]https://www.renpy.org/

3.3.3 FAtiMA Toolkit

Another example of an authoring tool is the FAtiMA Toolkit[5], created by GAIPS (Group of Artificial Intelligence for People and Society) [56].

Fearnot AffecTIve Mind Architecture (FAtiMA) Toolkit is a collection of tools and assets designed for the creation and use of cognitive and reactive agents with socio-emotional competences [57]. This tool was initially developed in 2005 to guide the emotional behaviour of agents in the application FearNot! (Fun with Empathic Agents Reaching Novel Outcomes in Teaching), a serious game about bullying [58] [59]. Along with the development of this authoring tool, several applications that integrate FAtiMA Toolkit system have been created, such as SUECA in 2016 [60] and Space Modules Inc. in 2018 [61].

Figure 3.5: FearNot! (2007)

When designing the toolkit, one of the main applications was for the creation of interactive storytelling scenarios with emotionally responsive agents that interact socially with human players in a variety of contexts. To achieve this, FAtiMA uses a character-based approach, meaning that the narrative is shaped by the way players interact with the agents and vice-versa. This differs from other popular game authoring tools, such as Twine, that are designed towards a plot-centric approach, which tends to restrict player involvement with the narrative to pre-defined key points [62].

In FAtiMA, agents are called Role Play Characters (RPC), and the agents' emotional model is derived from the OCC cognitive theory of emotions [63], named after the creators Ortony, Clore, and Collins. By using the *Emotional Appraisal editor* within the authoring tool, the author can define which emotional response they aspect the agent to have to a particular event.

In Addition, the *Emotional Decision Making* module is used to define the actions the agents can

[5]http://fatima-toolkit.eu

perform (including the possible targets and conditions) and helps them decide their next action.

Additionally, the toolkit provides a *Dialogue Editor* for the author to define which dialogue actions will be made available for the agents and the players to choose from. Regarding all utterances, each is coded with a set of *properties* that define the current and next state of the narrative, as well as other particular characteristics and effects of those dialogue choices. To verify if all dialogue paths defined by the author can be reached and to detect any possible errors, the editor offers a validation mechanism.

Finally, using the *World Model*, the author can define what are the effects of a specific action when performed by an agent. These effects can be applied either to the RPCs or to the environment they are in and the author is able to monitor them in the Simulator.

3.3.4 Twine

Created by game designer Chris Klimas in 2009, Twine[6] is a free to use, open-source tool designed to facilitate the creation of interactive stories with branching narratives. At its core, Twine is a hypertext-based authoring tool for developing interactive games that function similar to the CYOA book series where the reader is faced with various decision points at which they must make a choice that leads to different outcomes. Twine's UI offers a bird-eye view storyboard, using a directed graph layout to create and visualize the narrative structure.

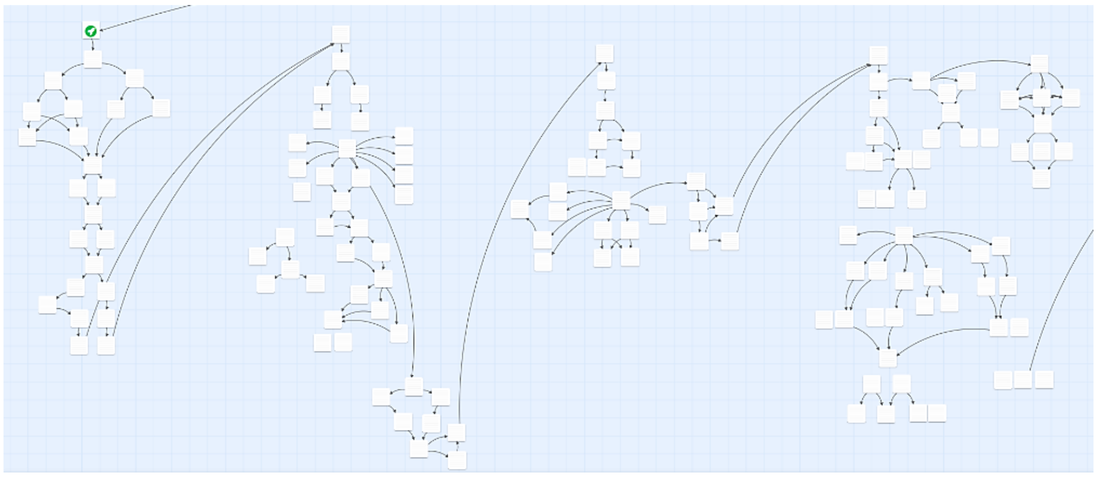
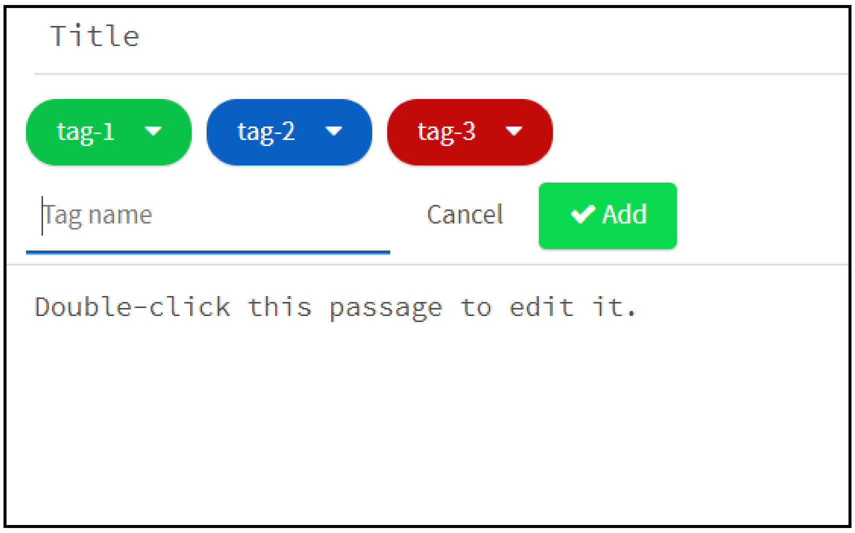


Figure 3.6: A bird's-eye view of a storyboard in Twine 2.0.

The creation of games with Twine, due to its simple design, requires only two elements: *Passages* and *Links*. *Passage* is the Twine's terminology for the nodes on the story graph that players navigate through. Each *Passage* contains a block of text (known as lexia in hypertext theory) that is shown to the player when they reach that *Passage* during gameplay. In Twine, *Passages* are distinguished by their title, which is not shown to the player.

[6]https://twinery.org/

Passages can also possess one or more tags (see Figure 3.7). In Twine, tags function as labels that add information to the *Passage* but are not visible on the published version of the story.



Double-click this passage to edit it.

Figure 3.7: An example of the *tag* system in Twine 2.0.

Similar to branching narratives, where arcs connect nodes to each other, *Passages* are connected through *Links*, represented by an arrow on Twine's storyboard. To create a *Link*, the author needs to write the title of the *Passage* they want to link to surrounded by double square brackets: `[[Open the door]]` .

To the player, this is displayed as a point of interaction, or a decision point, which they must click to advance in the story. It is also important to note that if the author wants to conceal the name of the destination from the player, they can use any of the following syntaxes:

TYPED	DISPLAYED
You [[open the door—>Surprise Party]] with some reluctance.	You open the door with some reluctance.
You [[Surprise Party<—open the door]] with some reluctance.	You open the door with some reluctance.
You [[open the door|Surprise Party]] with some reluctance.	You open the door with some reluctance.

Table 3.1: Twine's syntax to define *Links*

In all these examples, the text shown on the page is different from the *Passage* title. The clickable text will say "open the door," but the link will lead to a *Passage* named "Surprise Party". Often in CYOA books, as they read along, the reader is asked to make a choice. These choices appear at the bottom of the page, like so:

> *If you want to confront the dragon, turn to page 15.*
>
> *If you want to run away, turn to page 29.*

Depending on which one they pick, the reader is lead to different paths, with new choices, and eventually they reach either a successful or disastrous ending. Twine's design allows for a similar principle, where the author can write different *Links* on separate lines inside a *Passage*, as follows:

```
[[Confront the dragon]]

[[Run Away]]
```

This creates both a *Link* to the *Passage* named "Confront the dragon" and a *Link* to the *Passage* "Run Away". Because the player can only pick one at a time, adding multiple *Links* to a *Passage* creates a branching point, meaning the story splits into different paths. "Path" refers to the "route" the player has taken through the game's narrative. In Twine, the story branches at each decision point, so the choices the player makes throughout the game determine the path. At the end of the game, the narrative path represents all the *Passages* visited by the player during gameplay.

Besides which dialog options are chosen by the user, the path in which the narrative develops can also be dependent on different story's variables. These variables persist between *Passages* and store data that can be updated throughout the story. All variables must have a "$" prefix, and we create these variables using a <<set>> macro and following a (set: variable to value) principle: (set: $var to 2). We can later change the value of the variables with another <<set>> and setter operators, such as "to", "+=", "-=". "*=" and "/=".

To test the values stored in a story variable at a certain point, Twine uses conditional statements. In Twine, conditional statements come in three forms: if, else-if and else, known as the <<if>> macros. Similar to other programming environments, the <<if>> macro works on a true or false logic. If the contents in the statement are true, the macro runs its hook. Otherwise, it moves on to test the next macro. Here is an example:

> (if: $var is 10) [[Option 1]]
>
> (else-if: $var > 10) [[Option 2]]
>
> (else:) [[Option 3]]

Using the previous example, if the value stored in $var is 10, then only the *Passage* "Option 1" will be displayed as a clickable text, and the others will be omitted. This means that certain *Links* will only appear to the player and/or can be selected if the conditions of the <<if>> macro are true.

Lastly, Twine also allows the inclusion of images and audio through the use of specific HTML tags and supports Cascading Style Sheets (CSS) and JavaScript, which enable high-skilled programmers to create more complex games.

3.3.4.A Debugging and Troubleshooting

At the bottom of the Twine's storyboard page, there is a toolbar with both a "Test" and "Play" buttons. Both will open a playable version of the created story in a browser window, that the author can play from beginning to end. However, the "Test" button will also enable debugging facilities that help the author identify possible errors in the story. These include:

- A "Debug View" option, which shows the *Passage*'s current state and lists the contents of data maps, variables and macros in a different colour swatch.
- A "Source" toggleable pane which displays the current *Passage*'s source code in a small window for users to compare it to the *Passage*'s state.
- A "Errors" toggleable pane that displays the number of errors found in the story (up to 500). This only includes "syntax" mistakes, such as calling a non-existing variable.
- A "Variables" toggleable pane that keeps a record of all the story variables created and their value at the current narrative state.

It is important to note, however, that because of the way that Twine is designed, the author is obligated to play through the whole story in debugging mode to find every possible error or to consult all the metrics, since they can only view one *Passage*'s state at a time. This means that testing a more complex interactive narrative, with multiple possible paths and endings, can prove to be a cumbersome and time-consuming task. Not to mention that the debugging mode lacks certain metrics that are essential for providing ways for the author to adjust their stories to their own narrative goals, such as: number of paths, number of endings, non-visitable *Passages*, endings' reachability imbalance, and others.

3.4 Concluding Remarks

The above-mentioned (in Section 3.1) studies show that online Artificial Intelligent approaches, particularly drama managers, are a popular solution to guarantee narrative coherence while allowing the player to freely interact with the game. Unfortunately, there has been hardly any work done regarding off-line approaches that attempt to identify possible narrative problems during development and allow the author to validate their narrative with different story metrics and predict the output of different playthroughs.

Additionally, we observed that many authoring tools, such as Twine or Tinderbox, designed help writers create interactive stories lacked the proper tools that helped them analyse important narrative metrics and identify possible continuity errors. While these tools are designed to facilitate the authoring process, they nonetheless rely heavily on the author's diligent and methodical eye to predict all gameplay scenarios and possible complication that might arise, which makes story creation a wearing, expensive, and time-consuming process.

Nonetheless, due to its wide adoption and easy to work architecture, we concluded that Twine was the most promising candidate for the incorporation of our model, which we will describe in the next chapter.

4
Implementation and Development

Contents

This chapter presents an overview of the approach for the development of the model for the **Story Validator**. In Section 4.1, we start by providing the overall structure for our model. In Section 4.2, we describe how we created the IN used for testing. In Section 4.3, we describe the functionality of our model and in Section 4.4, we present the GUI conceptual representation of the **Story Validator** tool and describe in detail each of its components. Finally, in Section 4.5, we explain in detail how the developed tool is able to identify common challenges pertaining to the development of interactive narratives.

4.1 Overall Approach

Based on the knowledge acquired on the previous section, we sought out to develop a tool that supports game writers in the creation of IN in stages before human playtesting, by letting the author explicitly test different hypotheses and narrative properties to identify possible design mistakes.

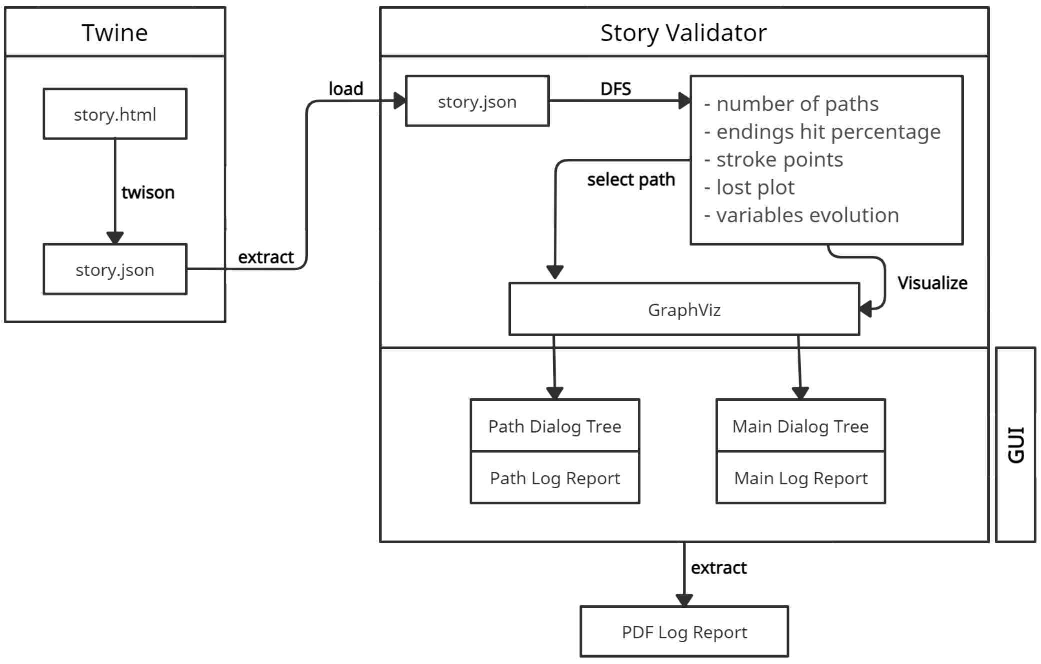


Figure 4.1: Conceptual model of our solution

Figure 4.1 represents the conceptual model for our solution. The model is composed of two main components: an **interactive story** (in JSON format) created using Twine's authoring environment and the **Story Validator** which takes the file outputted from Twine and, by treating the branching narrative as a branching tree, uses a Depth-First Search (DFS) based algorithm to traverse all possible narrative paths and provide insightful data. The information is then presented to the user through the use of a graphical interface (GUI).

4.2 Creating an interactive narrative with Twine

For the creation of the interactive narrative that will be used for testing, we chose to use Twine 2, an open-source hypertext-based authoring tool for developing interactive, non-linear stories, as described in chapter Section 3.3.4.

In this story, which we named ”The Case”, the player plays as the detective tasked with solving the murder of Miss M. (the victim). The detective's main suspect is Mr. S, who was seen arguing with the victim prior to her murder. The evidence is not sufficient and, therefore, they will need a confession. The suspect is brought to the station for questioning, but there is a catch: Mr. S is a very wealthy man and upsetting him during questioning will lead him to ask for his lawyer and for the detective to lose the case. Depending on the choices in dialog made when talking with the suspect, the player is led to different endings, that reflect their crime solving skills. To stipulate what are the effect in the unfolding of the narrative, we attach (and update) variables to each option. These variables represent the emotional state of the suspect NPC: anger, happiness, and fear.

Because Twine is designed to develop and publish interactive stories in the Web, all its data is encoded in a single HTML file. But while HTML is the default hypertext output for Twine, we found that exporting Twine's internal XML data in a JSON format would be an added value, for reasons of simplicity and easier processing. Therefore, we opted to use the story format Twison by lazerwalker that converts the Twine 2 story into a JSON document. In its current state, the **Story Validator** tool only reads Twine files in JSON format. This means that the story needs to be manually saved into a JSON file before it can be read by the tool, but hopefully in the future it will be able to convert HTML files to JSON automatically.

This JSON file contains the data of each *Passage*, including the *Passages*' text, name, pid, *Links* and position on the Twine's storyboard. After loading a JSON file into the **Story Validator** tool, if we parse it into a variable (e.g. story), we can then access the data inside. For example:

```
story.get("passages")[0].get("links")
```

gives us access to all the *Links* of the first *Passage* (i.e. starting node).

4.3 Functionality

As mentioned previously, we extract the story's data from the JSON file that is obtained from Twine. By extracting the information about the *Passages* and their *Links* from the JSON data and passing them as the tree's nodes and edges, we can create a tree-like structure, similar to the branching dialogue tree that we built in Twine.

In order to reach our goal, our solution needed to provide insightful data on different story metrics and identify design issues that could be encountered by player during gameplay. With this is mind, we defined the following narratives metrics:

- **Number of paths:** this metric enumerates all possible traversals of the story tree, including which *Passages* the player visits on each narrative path.
- **Endings Hit Percentage:** calculates the distribution of story's endings, to understand which ones are more likely to be reached (i.e., which endings have more paths leading to them).
- **Stroke Points:** we also needed to identify which plot points were common in all narrative paths. These refers to *Passages*, that regardless of the choices the player makes, are always reached.
- **Lost Plot:** this metric identifies narrative sections that, while plotted by the author, were never reached in an actual playthrough due to some design error. It should also notify the author of paths that end abruptly and do not reach an ending.
- **Variables Evolution:** a branching narrative might contain different variables that the author needs to monitor. This metric keeps track of those variables throughout the different story paths.

The above mentioned metrics were defined based both on our personal experience as developers of interactive stories using Twine, and to address the common authoring problems we discussed in Section 2.2.1. The same challenges were reported by the user study's participants in Section 6.1, which corroborated our initial decision.

As mentioned previously, we used an DFS algorithm (adapted to work with Twine's framework) to traverse the story's branching tree, since we wanted to explore the different *Passages* by following the same trajectory as a human player (i.e., starting at the root *Passage* and traversing through several *Passages* until reaching an ending point). While traversing through the tree, the algorithm gathers information about the story, based on the defined metrics.

Next, we present our solution's algorithm (Algorithm 4.1), as well as a detailed explanation of how the algorithm was adapted to collect each of the metrics defined above.

Algorithm 4.1: Adapted Depth-First Algorithm

```
begin
    // Initialize variables
    $sequence \longleftarrow [\,]$
    $numberPaths \longleftarrow 0$
    $endingCount \longleftarrow \{\}$
    $strokePoints \longleftarrow [\,]$
    $lostPlot \longleftarrow story.get("passages")$
    $deadEnds \longleftarrow [\,]$
    $varsDict \longleftarrow \{\}$
    // Get root passage (pid 0), add to sequence and remove from lostPlot
    $startnode \longleftarrow story.get("passages")[0]$
    $sequence.push(startnode)$
    $lostPlot.remove(startnode)$
    // Traverse story and show results
    $visitPassages(story, startnode)$
    $showResults()$
    Function visitPassages ($story, passage$):
        // Set up and update variable values
        $updateVariables(passage)$
        // Assess "if" statements when existent and creates a list of
        // unvisitable links, that can not be visited due to "if" constrictions
        if $passage$ has $"if"$ then
            $unvisitable \longleftarrow ifTreatment(passage)$
        $links \longleftarrow passage.get("links")$
        if $links$ is not $NULL$ then
            foreach $link$ in $links$ do
                if $link$ not in $unvisitable$ then
                    $sequence.push(link)$
                    $lostPlot.remove(link)$
                    $visitPassages(story, link)$
            // Remove last element from stack sequence
            $sequence.pop()$
        else
            // Update number of paths
            $numberPaths \longleftarrow numberPaths + 1$
            $endingTreatment(passage)$
            // Compare sequence array with current strokePoints array
            if $strokePoints$ is not $NULL$ then
                $strokePoints \longleftarrow sequence$
            else
                $tmp \longleftarrow []$
                foreach $elem$ in $sequence$ do
                    if $elem$ in $strokePoints$ then
                        $tmp.push(elem)$
                $strokePoints \longleftarrow tmp$
            $sequence.pop()$
```

```
// Method to set up and update story variables
Function updateVariables (passage):
    $passageText \longleftarrow passage.get("text")$
    // From passage extract each set macro
    $sets \longleftarrow passageText.split("(set")[1]$
    foreach s in sets do
        $var, cond, value \longleftarrow s.split(" ")$
        switch (cond)
            case "to" :
                $varsDict(var) \longleftarrow value$
            case " += " :
                $varsDict(var) += value$
            case " -= " :
                $varsDict(var) -= value$
            ... // verify all cases

// Method to get story variables values
Function getVariableValue (variable):
    return $varsDict.get(variable)$

// Method that returns an array with all the passages that cannot be visited due
// to "if" constrictions
Function ifTreatment (passage):
    $passageText \longleftarrow passage.get("text")$
    // From passage extract each if statement
    $statements \longleftarrow passageText.split("(if")[1]$
    foreach s in statements do
        $sect1, sect2 \longleftarrow sect1.split('[', 1)$
        // Add the passage to the unvisitable array
        $unvisitable.add(sect2)$
        // From sect1 extract variables, conditional operators and values
        $var, cond, value \longleftarrow sect1.split(" ")$
        // Get the current value of the variable in question
        $realValue \longleftarrow getVariableValue(var)$
        switch (cond)
            case "is" :
                if realValue "is" value then
                    $unvisitable.remove(sect2)$
            case " > " :
                if realValue ">" value then
                    $unvisitable.remove(sect2)$
                ... // verify all cases
    return unvisitable

// Method that saves all paths that reach a dead-end
Function lostPath (path):
    $deadEnds.add("PATH\#" + str(numberPaths) + " : " + str(path))$
```

```
// Method that verifies if passage is an ending point and updates ending count
Function endingTreatment (passage):
    // Check if passage has tag ENDING-POINT
    if $passage.get("tags")$ is not $ENDING-POINT$ then
        $lostPath(sequence)$
    else
        // Updates the number of times the ending passage is reached
        $endingCount(passage.get("name")) += 1$

// Method that outputs the results of the story analysis
Function showResults ():
    $results \longleftarrow$ " "
    // Number of paths
    $results.add("Number\ of\ paths\ is:\ " + numberPaths)$
    // Ending Hit Percentage
    foreach end in endingCount do
        $results.add("There\ are\ " + endingCount(end) + "paths\ that\ reach\ " + end)$
        $results.add("Ending\ Hit\ Percentage\ is:\ " + round(endingCount[end]$ x
         $100\ /\ numberPaths))$
    // Stroke Points
    $results.add("Passages\ " + strokePoints + "\ are\ visited\ in\ all\ paths")$
    // Lost Plot
    if $deadEnds$ is not Empty then
        $results.add("The\ following\ paths\ never\ reach\ an\ ending\ point:" +\ deadEnds)$
    else
        $results.add("All\ paths\ reach\ an\ ending\ point")$
    if $lostPlot$ is not Empty then
        $results.add("The\ following\ passages\ are\ never\ visited:\ " +\ lostPlot)$
    else
        $results.add("All\ passages\ are\ visited")$
    // Variables Evolution
    foreach var in varsCount do
        $results.add("Variable" + var + "\ end\ value\ is\ " + varsCount(var))$
    print$(results)$
```

Number of Paths

When searching our tree, the DFS begins at the starting node (which is obtained from the JSON data) and explores one *Link* at a time, adding each *Passage* visited to a stack. When it reaches an [Ending Passage], the total number of paths is increased by one, the elements in the stack are saved as a path sequence and we begin popping nodes from the stack until we find a node that has an unvisited textitLink.

However, is important to point out that, unlike a regular DFS algorithm, these nodes created in Twine can include a <<*if*>> macro, which influences the flow of the story. These <<*if*>> macros are used to test the current value stored in a story variable. This means that certain *Links* will only appear to the player and/or can be selected if the conditions of the <<*if*>> macro are true.

In the end, using the DFS, we are able to calculate the total amount of valid narrative paths (i.e. paths that reach an [Ending Passage]) that the player can take in the submitted story file.

Ending Hit Percentage

At the end of the game, the story reaches different endings, depending on the players' alignment and choices. As the DFS algorithm traverses all the paths, we keep count of how many paths reach each [Ending Passage]. We then calculate the corresponding Hit Percentage using the following formula:

$$Ending_i\ Hit\ Percentage\ (\%) = \frac{np_i}{\sum_{j\,=\,1}^{ne} np_j} \times 100$$

where i refers to index for each [Ending Passage]; np is the total number of paths for each [Ending Passage] i; j is the index of summation and ne is the total number of [Ending Passages]. j starts out equal to 1 and is incremented by one for each [Ending Passage], stopping when j = ne. In other words, if the JSON file has five [Ending Passages], then the interval of values of j is [1, 5], one for each [Ending Passage].

Stroke Points

Whenever the DFS algorithm reaches an [Ending Passages], we compare the current path sequence with the previous one, intersecting their values to find common nodes in all paths. Eventually, we can gather which *Passages* are visited in all narrative paths. We named these *Passages*, Stroke Points.

Lost plot

Sometimes, certain design errors in the development of the story can hinder the player's gameplay. Ideally, in a well-constructed interactive story, all *Passages* should be visited at least once, and all paths should reach an [Ending Passage].

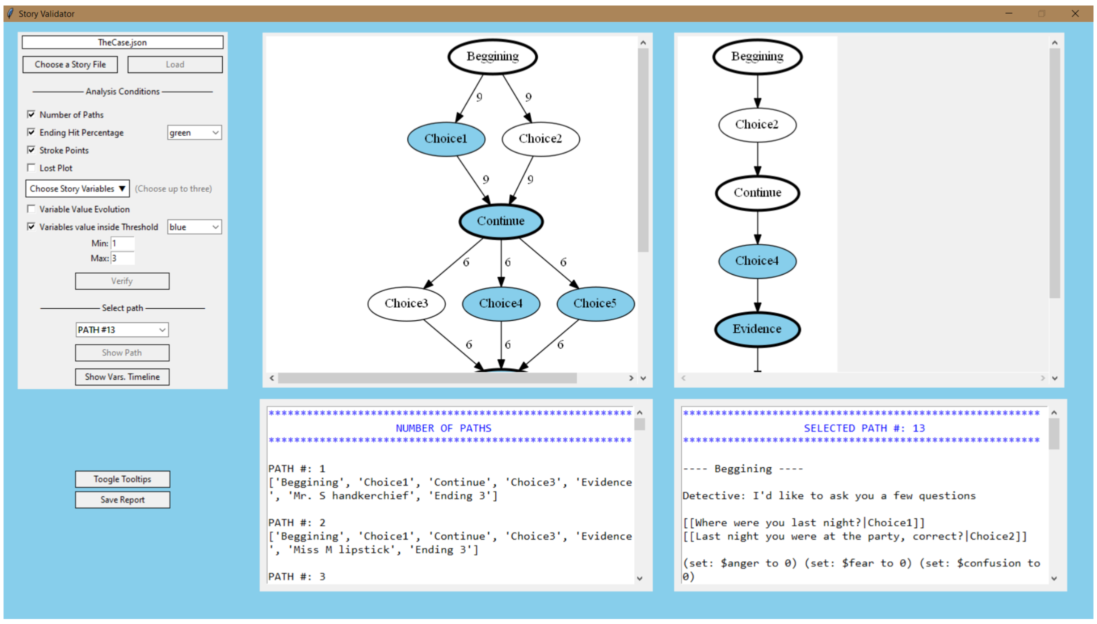
From the JSON data we extract the name of all the *Passages* included in the story and save them in an array. Then, as the DFS algorithm traverses the story tree, we remove the *Passages* that are visited from the array. Eventually, the array will only include *Passages* that are never reached by any possible narrative path. If the length of this array is bigger than 0, the contents of the array are then printed. Otherwise, the message "All *Passages* are visited." is displayed instead.

Moreover, as explained previously, each iteration of the DFS ends once it reaches a vertex without any edges, in other words, a *Passage* without any *Links*. While this vertex is expected to be a [Ending Passage], it can also be a Dead-End, this is a *Passage* that, once reached, stops the player from progressing. In order to identify a Dead-End, we take advantage of the tag system from Twine and add an ENDING-POINT tag to each of the *Passages* that we want to define as an ending.

Therefore, if the algorithm reaches a *Passage* without any *Links* but this *Passage* does not have an ENDING-POINT tag, then we have identified a Dead-End and the path sequence in question is printed. Otherwise, the message "All paths reach an ending." is displayed instead.

Select Story Variables

From the JSON data, we can extract all the [Story Variables] present in the story. As the DFS algorithm visits each *Passage*, it keeps track and updates all [Story Variables], storing the values in a nested list.



```
*************************************************************
                      NUMBER OF PATHS
*************************************************************

PATH #: 1
['Beggining', 'Choice1', 'Continue', 'Choice3', 'Evidence
', 'Mr. S handkerchief', 'Ending 3']

PATH #: 2
['Beggining', 'Choice1', 'Continue', 'Choice3', 'Evidence
', 'Miss M lipstick', 'Ending 3']

PATH #: 3
```

```
*************************************************************
                   SELECTED PATH #: 13
*************************************************************

---- Beggining ----

Detective: I'd like to ask you a few questions

[[Where were you last night?|Choice1]]
[[Last night you were at the party, correct?|Choice2]]

(set: $anger to 0) (set: $fear to 0) (set: $confusion to
0)
```

Figure 4.2: Story Validator tool interface

Furthermore, it was important that our solution provided a way for the author to analyse any path in detail. Therefore, we made sure that the **Story Validator** would provide reports (according to the selected metrics) both of the overall story and of each chosen path, through visual representations.

4.4 GUI application

Based on previous experience with the programming language, we decided to use Python 3 to build a Graphical User Interface (GUI) using tkinter (see Figure 4.2). The tkinter package is a standard GUI library for Python that allows users to create Graphical User Interfaces.

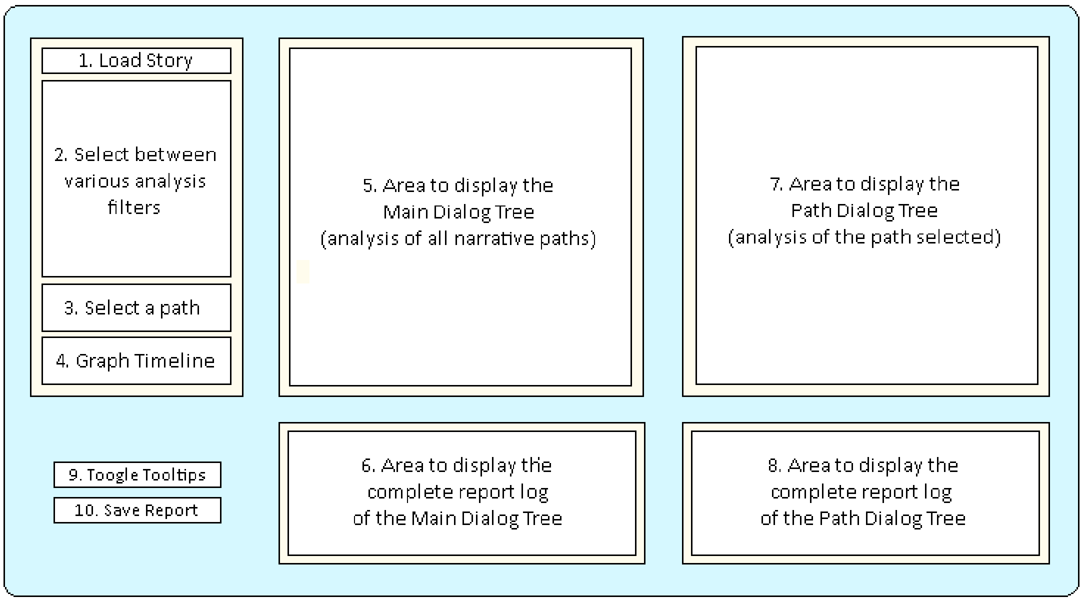

1. Load Story
2. Select between various analysis filters
3. Select a path
4. Graph Timeline
5. Area to display the Main Dialog Tree (analysis of all narrative paths)
6. Area to display the complete report log of the Main Dialog Tree
7. Area to display the Path Dialog Tree (analysis of the path selected)
8. Area to display the complete report log of the Path Dialog Tree
9. Toogle Tooltips
10. Save Report

Figure 4.3: The Story Validator GUI conceptual structure

Figure 4.3 presents the GUI conceptual representation that was created for the Story Validator of which we describe each of its components as follows:

1. **Load Story —** Opens a file dialog window in tkinter that asks the user to select a story file (in JSON format) to be analysed by the tool.

2. **Analysis Conditions —** The user can select one or more options from a selection of analysis conditions (see Figure 4.5), based on the metrics defined in Section 4.3. Each one of these conditions will influence different visual aspects of the Dialog Trees (5 and 7) and what information is shown on the Main Log Report (6).

3. **Select Path —** From a drop-down menu, the user can select which story path they want to analyse in detail. The path selected will appear on the Path Dialog Tree (7) area.

4. **Graph Timeline —** We use Python's Pyplot library to plot a dotted chart with the changes of the values of the story variables selected (throughout the path selected (3)), where the y-axis corresponds to the variable's values, and the x-axis corresponds to the *Passages* visited in the path selected (see Figure 4.4).

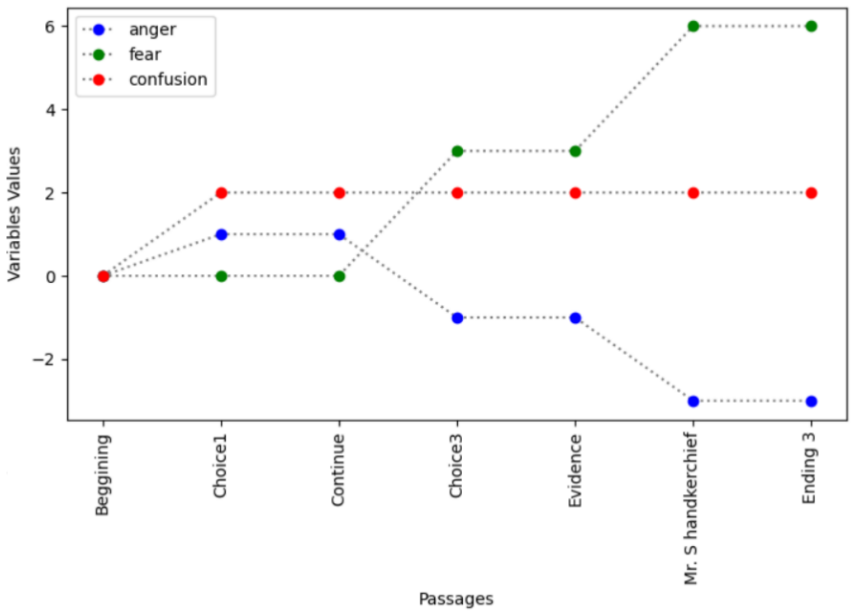


Figure 4.4: Graph Timeline of the story variables $anger, $fear and $confusion

5. **Main Dialog Tree —** We use GraphViz's DOT language to draw a dialog tree where each node represents a *Passage* in the story. If a *Passage* has *Links* to other *Passages*, they are represented in the tree as edges. This tree shows all possible narrative paths in the story selected and updates according to the Analysis Conditions (2) selected.

6. **Main Log Report —** Displays textual information regarding the overall narrative, according to the Analysis Conditions (2) selected.

7. **Path Dialog Tree —** We use GraphViz's DOT language to draw a second dialog tree where each node represents a *Passage* in the path selected (3). This tree also updates according to the Analysis Conditions (2) selected.

8. **Path Log Report —** Indicates what path was selected in (3) and prints the complete interactive story text into the log as well.

9. **Toggle Tooltips —** We use a tooltip widget that appears when the mouse is above the widget and explains what each Analysis Condition does for first time users. Then, we added the possibility for the user to switch these tooltips on-and-off by pressing the Toggle Tooltips button.

10. **Save Report —** We create and print a report that contains all the tests performed regarding the current visualisation, including what Analysis Conditions (2) were checked, the story variables and story path (3) selected, both dialog trees (5) and (7) and corresponding logs (6) and (8) and the Graph Timeline. This is accomplished by generating a PDF file using the PyFPDF library for

Python. These PDFs can be used for a clearer reading of the results and for comparing validations with different conditions selected.

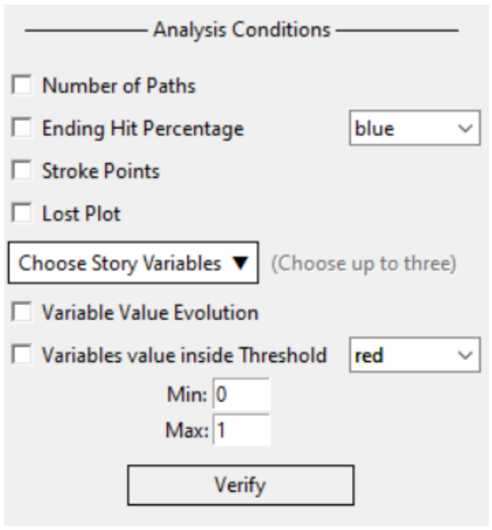


Figure 4.5: Analysis Conditions as it appears on the Story Validator tool.

4.5 Identifying common branching narrative problems

Often during the creation of interactive stories there is the need to add, change and remove *Passages* and this process can quickly impact the dynamic of the story. This is not always easy to predict, however, as the impact of a *Passage* can reveal itself only in future states.

Not to mention that, in interactive narratives, the path in which the narrative develops can be dependent on different game's variables: what items the player picks ups, which dialogs choices they make, what actions they pursue, player's success or failure at a specific challenge, etc. Consequently, as the story grows in complexity and the number of decision points increases, the authoring experience can become overwhelming for the author, often requiring extensive modifications to maintain the narrative coherent.

As stated previously, the main objective of this work is to support game writers by evaluating their game's narrative and working as a debugging tool. Therefore, the tool performs a series of tests and reports on possible narrative problems.

For the rest of this section, we will present a collection of various challenges that authors of interactive narratives usually face, and how our tool is able to help identify and work on those problems.

Keeping track of Passages visited

A branching narrative will often have several narrative paths that the player can take, resulting from the choices they make. For that reason, as the number of possible narrative paths grows, it becomes tougher for the author to keep track of the *Passages* that are visited.

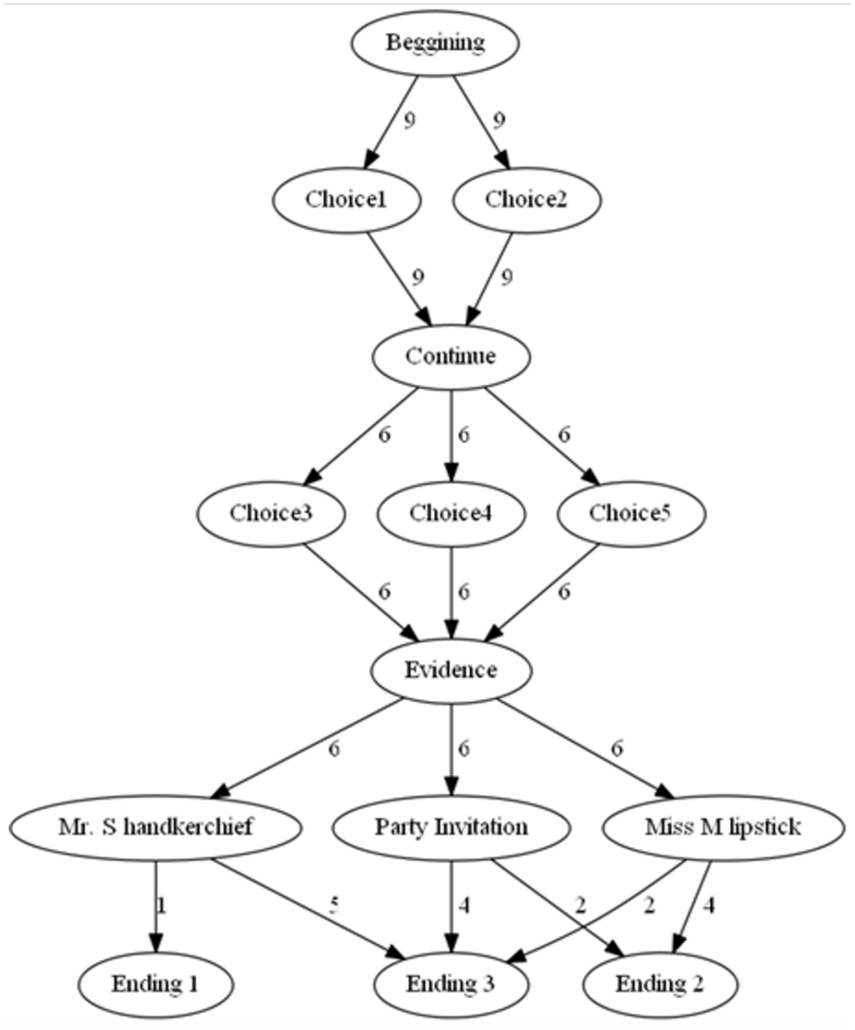


Figure 4.6: Main Dialog Tree for "TheCase" story with the Analysis Condition "Number of Paths" selected.

The tool's algorithm runs through the story by selecting different options until it reaches an [Ending Passage]. Then it starts a new playthrough, and repeats, choosing other options. At the end, the tool is able to tell the user how many different paths the player can take, along with which *Passages* are visited on each path. The Main Dialog Tree (5) displays a tree with all the narrative paths that the player can possibly take, giving the author a general idea of how the story flows. In order to make a more in-depth assessment, the Main Report Log (6) displays which *Passages* are visited in each path. Furthermore, the user can pick a [Story Path] to analyse in detail (3), and the path will be displayed on the Path Dialog Tree (7).

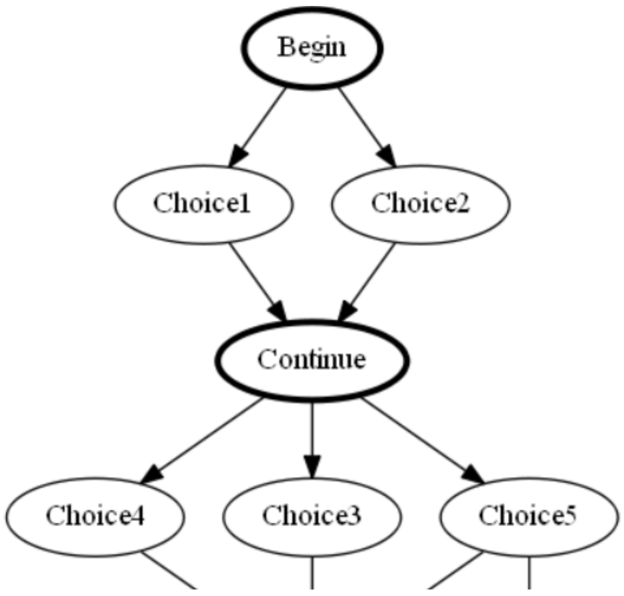


Figure 4.7: Main Dialog Tree for ”TheCase” story with the Analysis Condition ”Stroke Points” selected.

Moreover, the tool is also able to identify Stroke Points (Figure 4.7). We define Stroke Points as *Passages* that are visited in all possible narrative paths, meaning that regardless of the choices the player makes, they always end up reaching these *Passages*. This can be part of the designer goals, as it can be useful to ensure some parts of the story is always conveyed to the player. However, it may happen that the author does not want to withdraw the player's ability to choose, in which case Stroke Points become a problem.

While travelling through all narrative paths, the tool's algorithm keeps track of the *Passages* visited. At the end, *Passages* that are visited in all narrative paths will be marked as Stroke Points. If the user has the analysis condition Stroke Points selected, the Main Report Log (6) will print out which *Passages* are visited in all paths. Moreover, on the Dialog Trees (5 and 7), Stroke Point nodes will have a bolded outline for easier identification.

Keep track of endings' reachability

Depending on the choices in dialog made, the player is led to different endings, which we call [Ending Passages]. However, it is difficult for the author to predict and monitor the distribution of those endings, without it being a laborious and time-consuming task.

As a design objective, the author may want to create certain restrictions on the distribution of the endings, such as having an ending that is more common to obtain (i.e. has more paths that reach it than other endings), or even a ending with only one possible path.

If the analysis condition Endings Hit Percentage is selected, the Main Report Log (6) then provides the author with percentages on the likelihood of reaching each of the [Ending Passages], as well as how

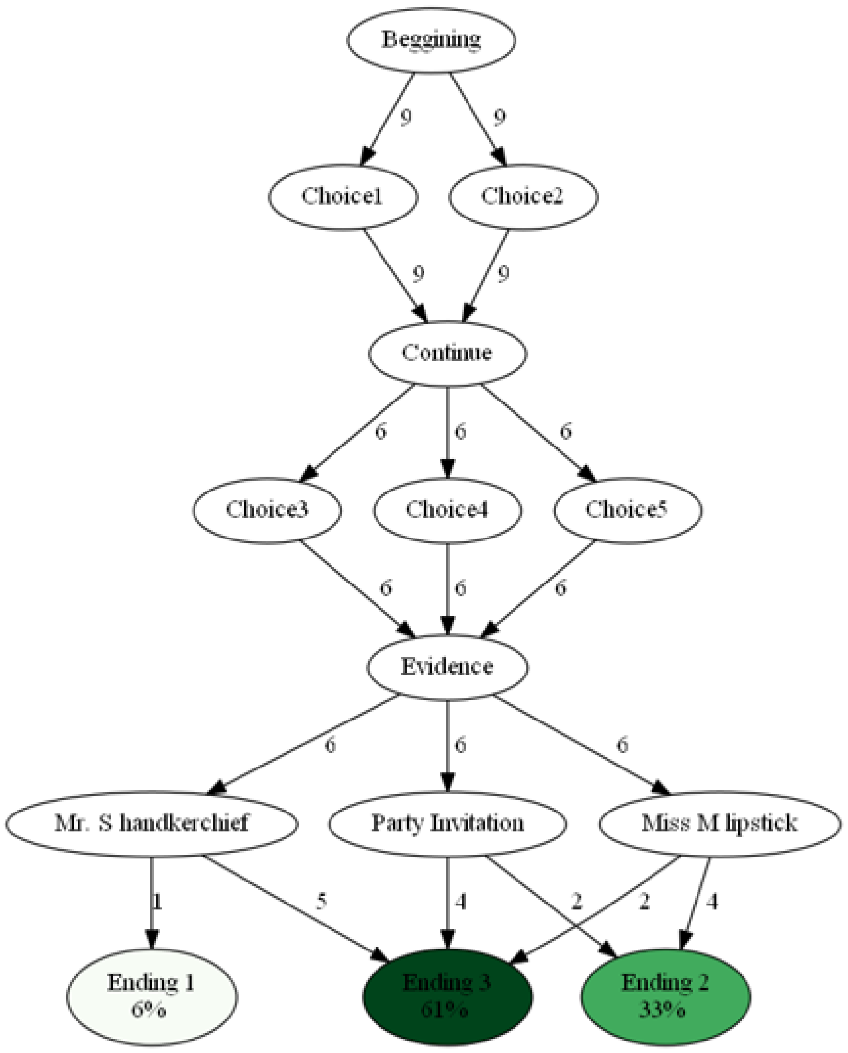


Figure 4.8: Main Dialog Tree for "TheCase" story with the Analysis Condition "Ending Hit Percentage" selected.

many paths can reach each ending. Besides, on the Main Dialog Tree (5) the user can observe the distribution of the paths that enter each of the [Ending Passages] if the analysis condition Number of Paths is selected. In conclusion, the user can easily explore each of the paths that reach an ending, and check if they are in accordance with their design goals.

Keep track of story's variables

As mention previously, in IN, the path in which the narrative develops can be dependent on different game's variables. For simplicity purposes, this work focus solely on numerical variables whose values changes depending on the choices in dialog that the player makes. Having variables updating depending on the characters or plot state can make the interactive story more interesting for the player, however, it is difficult for the author to keep track and control those values over multiple possible interactions.

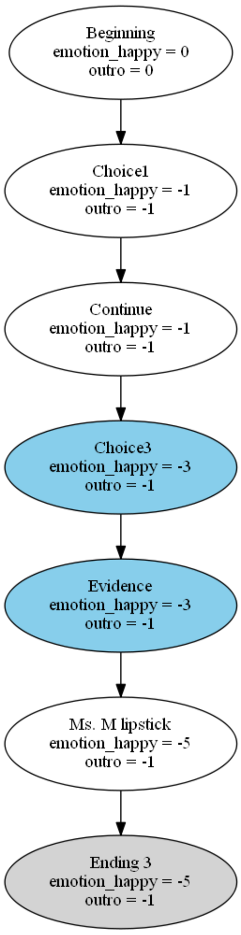


Figure 4.9: Path Dialog Tree for ”TheCase” story with the Analysis Condition ”Variables Evolution” selected.

The tool is able to identify and keep track of all [Story Variables] defined by the author and their values throughout each path. The user can also select which [Story Variables] they wish to analyse (up to three). By selecting the analysis condition Variables Value Evolution, the user can observe the value changes of each [Story Variable] selected, as well as the numeric value of each variable when an [Ending Passage] is reached.

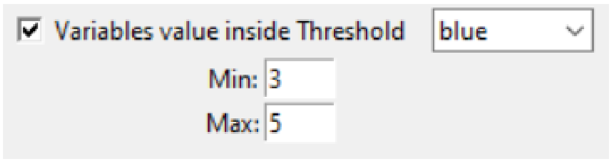

[✓] Variables value inside Threshold: blue
Min: 3
Max: 5

Figure 4.10: Variable Value inside Threshold

By selecting the analysis condition Variables value inside Threshold, the tool highlights the *Passages* where the selected [Story Variables] have a value between the MIN and MAX values, both defined by the user (see Figure 4.10). The user is also able to choose the colour with which to fill those *Passages*.

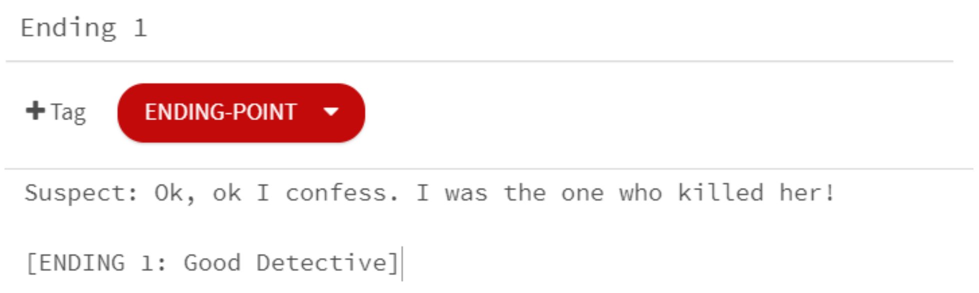
The tool can also draw a graph with the value evolution of each [Story Variables] selected, so the user can better keep track of the variables throughout the story (see Figure 4.4).

Avoiding Dead-Ends and losing plot

A Dead-End is a *Passage* in the story that, once reached, prevents the player from continuing to play. This problem typically happens due to an error in defining the *Passages* that should come afterwards, meaning they have “entering” conditions that is impossible to meet, or if the path is an ”endless” path. It is crucial for the designer to identify these cases, as they abruptly stop the player from continuing playing, however, doing so is difficult, due to the combinatorial nature of the exploration of the story.

A Dead-End is different from an [Ending Passage] since the latter corresponds to the end of the story. For that reason, having a Dead-End means that the story has one or more paths that cannot reach an [Ending Passage].

As mentioned previously, Twine 2 supports the addition of tags in *Passages*. By taking advantage of this system, we defined that the author must attach the tag ENDING-POINT to a *Passage* to denote that *Passage* as a [Ending Passage]. Therefore, while traversing through the story, if it reaches a *Passage* where it cannot go any further, and that *Passage* does not have an ENDING-POINT tag, then the tool knows it has reached a Dead-End.


Ending 1

Suspect: Ok, ok I confess. I was the one who killed her!

[ENDING 1: Good Detective]

Figure 4.11: Example of an ENDING-POINT tag in Twine 2.

Furthermore, it is also important to identify if there are sections in the story that are never visited, regardless of how many times the player plays through the game and what choices are made. This problem may happen again due to a Dead-End or if a *Passage* is isolated. The tool’s algorithm travels through all possible paths, keeping track of which *Passages* are visited and which ones are not. At the end, *Passages* that are never visited will be marked as Lost Plot.

If the user selects the analysis condition Lost Plot, the Main Report Log (6) will print out the *Passages* that were never visited as well as display which paths were not able to reach an [Ending Passage]. Additionally, on the Main Dialog Tree (5), *Passages* that are never visited will appear as single nodes with no edges connected to them, and on the Dialog Trees (5 and 7), Dead-End nodes will take on a rectangular shape in oppose to its regular round shape, so they are easier to identify for the user.

5

User Tests

Contents

The following chapter presents the methods that were used to perform the user study. Section 5.1 introduces the aims of the research. Section 5.2 provides an overview of the participants in the usability tests. Section 5.3 describes the materials that were necessary to perform the experiments. Section 5.4 presents how Phase 1 was performed. Finally, Section Section 5.5 illustrates Phase 2 of user testing.

5.1 Objectives

This controlled experiment was conducted with the intent of

- Finding out if the tool adequately helps the users identify problems in the game story.
- Determining whether users can operate the tool with ease and identify usability issues.

For this purpose, we conducted two phases of user testing, with a total of 25 participants. The data collected to uncover the usability problems in these studies were a variety of qualitative and quantitative data.

During Phase 1, we examined how the users feel about the tool's design and if it is easy and intuitive to use. During Phase 2, users were asked to use the **Story Validator** to identify problems in a branching story with various design issues and suggest possible solutions.

5.2 Participants

The ideal sample of participants for these studies should have a high percentage of game developers or people with experience in story-writing for games. They are the most valuable testers since the tool is designed for them. Moreover, their feedback can help optimise the tool's design and improve usability by identifying missing features or design errors. Regardless, the participants of phase 2 were required to have no prior experience with the tool, meaning they could not have participated in Phase 1.

For phase 1 of user testing, the number of expected participants were of four to five users. For Phase 2, at least twenty subjects were expected.

5.3 Material Requirements

We conducted both phases through remote usability testing, where the participants performed the tasks from their home or office and shared their screens with the facilitator.

For this purpose, the following resources and materials were required:

- Two computers (one for the participants and another for the facilitator) with internet connection.
- Microphones (either built-in on the computer or not)

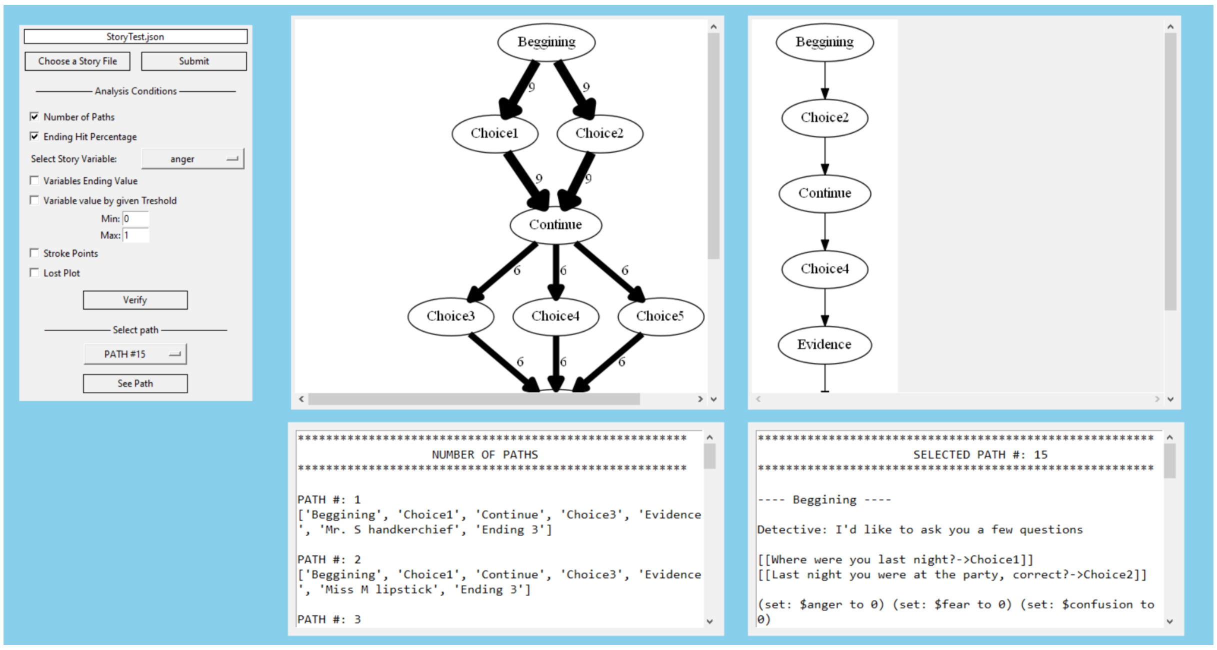
- Computer mouse
- Zoom conferencing software
- A copy of the tool sent to the participants via email

5.4 Phase 1 methodology

During Phase 1, we performed a user test of the first prototype, to assess the usability of the tool.

5.4.1 Prototype

Figure 5.1 shows the first prototype of the **Story Validator** that was used during Phase 1 of user testing.


StoryTest.json

Choose a Story File | Submit

Analysis Conditions

[✓] Number of Paths

[✓] Ending Hit Percentage

Select Story Variable: anger

[ ] Variables Ending Value

[ ] Variable value by given Treshold

Min: 0

Max: 1

[ ] Stroke Points

[ ] Lost Plot

Verify

Select path

PATH #15

See Path

```
*******************************************************
                  NUMBER OF PATHS
*******************************************************

PATH #: 1
['Beggining', 'Choice1', 'Continue', 'Choice3', 'Evidence
', 'Mr. S handkerchief', 'Ending 3']

PATH #: 2
['Beggining', 'Choice1', 'Continue', 'Choice3', 'Evidence
', 'Miss M lipstick', 'Ending 3']

PATH #: 3
```

```
*******************************************************
                  SELECTED PATH #: 15
*******************************************************

---- Beggining ----

Detective: I'd like to ask you a few questions

[[Where were you last night?->Choice1]]
[[Last night you were at the party, correct?->Choice2]]

(set: $anger to 0) (set: $fear to 0) (set: $confusion to
0)
```

Figure 5.1: Story Validator first prototype

This prototype was an earlier version of the final application that was presented in Chapter 4 (see Section 4.4 for details), designed to evaluate the concept, and collect feedback from users in order to improve the tool.

This prototype consisted in a GUI application created using Python's tkinter library and included the same analysis conditions as presented in **??**. It differed from the final model in the fact that it did not include toggleable tooltips, the option to print a pdf report of the results, the story variables' values graphical timeline and it only allowed to test one story variable at a time. In addition, with the exception of the background, the prototype had a black and white wireframe.

5.4.2 Procedure

To perform the test, the users received a branching story (created using Twine) with no design problems to get familiarized with the tool and were then asked to perform 10 tasks (see Section 5.4.3), using the **Story Validator** first prototype. While the participants completed the tasks, we observed their performance and took notes. The sessions were also recorded to allow further analysis. The recordings were anonymous and were only used to review and gather data that we might have missed or did not have time to annotate during the test. All the participant were informed about the video recordings prior to each test and gave their consent.

During this phase we used the think-aloud method, meaning that we asked the test participants to use the system while continuously verbalizing their thoughts. This helped us gather possible properties and design changes that the users might want to see in the updated version of the tool.

After completing the tasks, the participants were asked to rate how easy or difficult it was to solve each task through a Likert-scale based questionnaire. Additionally, participants were given a demographic questionnaire (see Appendix B) to help us identify the profile of our sample population and were asked to score the usability of the tool using the System Usability Scale (SUS).

5.4.3 Scenarios and expected performance

In this section, we present the sequence of tasks that users were asked to perform during Phase 1, as well as the ideals steps the users should take to complete each task.

First, the user must open "The Case" story file (see Appendix A for transcript) on their browser and play through the different story paths, to get familiarized with the story. Afterwards, they were asked to complete the following tasks using the **Story Validator** tool:

Task 1: Tell us what is the total number of paths.
Ideal steps:

1. Under the section "Analysis Conditions" select the option "Number of Paths".
2. Press the "Verify" button.
3. On the log, at the bottom of the "Number of Paths" section, there will be a line that says: "The total number of paths is. . . "

Task 2: In PATH #7, find the ending value for the story variable $anger.
Ideal steps:

1. Under the section "Analysis Conditions" select the option "Variable Ending Value".
2. Press the "Verify" button.

3. On the log, at the bottom of the “Variable Ending Value” section, under PATH #7 there will be a line that says “Variable anger ending value is: . . . ”

Task 3: In how many paths does the story variable $anger reaches an ending value of 0?
Ideal steps:

1. Under the section “Analysis Conditions” select the option “Variable Ending Value”.
2. Press the “Verify” button.
3. On the log, at the bottom of the “Variable Ending Value” section, there will be the ending value for each of the story paths. Identify the ones where the value is 0.

Task 4: In what paths and *Passages* does the story variable $anger has values higher than -4 and lower than -2?
Ideal steps:

1. Under the section “Analysis Conditions” select the option “Variable Value by given Threshold”.
2. Underneath it, insert -4 on the MIN box and -2 on the MAX box
3. Press the “Verify” button.
4. On the First Dialog Tree, the nodes of the *Passages* where the story variable $anger has values higher than -4 and lower than -2 will be coloured in light blue.

Task 5: Which ending is reached more often?
Ideal steps:

1. Under the section “Analysis Conditions” select the option “Ending Hit Percentage”.
2. Press the “Verify” button.
3. On the log, at the bottom of the “Ending Hit Percentage” section, there will see that the ending that is reached most often is Ending 3 (61% hit percentage) with 11 paths.

Task 6: How many paths lead to Ending 2?
Ideal steps:

1. Under the section “Analysis Conditions” select the option “Ending Hit Percentage”.
2. Press the “Verify” button.
3. On the log, at the bottom of the “Ending Hit Percentage” section, there will see that there are 6 paths that lead to Ending 2.

Task 7: Which *Passages* are visited in all paths?

Ideal steps:

1. Under the section “Analysis Conditions” select the option “Stroke Points”.
2. Press the “Verify” button.
3. On the First Dialog Tree, the nodes of the *Passages* that are visited in all narrative paths have a thicker outline

Task 8: Do all paths reach an ending?

Ideal steps:

1. Under the section “Analysis Conditions” select the option “Lost Plot”.
2. Press the “Verify” button.
3. On the log, on the “Lost Plot” section, there will be message that says “All paths reach an ending. Everything OK”.

Task 9: Is there a *Passage* that is never visited?

Ideal steps:

1. Under the section “Analysis Conditions” select the option ““Lost Plot”.
2. Press the “Verify” button.
3. On the log, on the “Lost Plot” section, there will be message that says “All Passages are reached. Everything OK”.

Task 10: What is the text in *Passage* Choice 4?

Ideal steps:

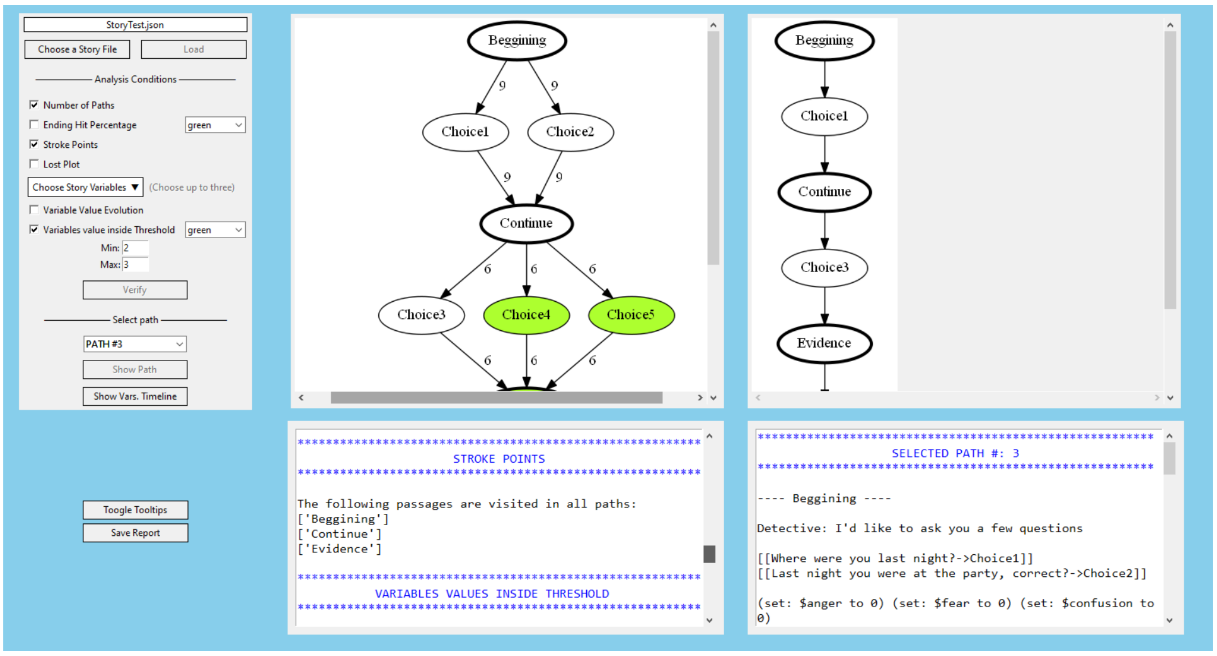
1. Under the section “Analysis Conditions” select the option “Number of Paths”.
2. Press the “Verify” button.
3. On the log, on the “Number of Paths” section, identify a path that visits Choice 4. An example is PATH #4.
4. Under the section “Select Path”, select PATH #4.
5. Press the “See Path” button.
6. On the second log, scroll down to “Choice 4”. The text on “Choice 4” will be “Suspect: And what if I did? Are you accusing me of something, detective? I do not like that tone of yours...!”

5.5 Phase 2 methodology

In the following section, we describe in detail how Phase 2 of user testing was performed. During this phase, in order to test participants were asked to use the tool to identify problems in a branching story with various technical issues and suggest possible solutions.

5.5.1 Prototype

Figure 5.2 shows the second and final prototype of the **Story Validator** that was used during Phase 2 of user testing. This prototype reflects the architectural application that was presented in Chapter 4 and explained in detail in Section 4.4.



StoryTest.json

[✓] Number of Paths
[ ] Ending Hit Percentage: green
[✓] Stroke Points
[ ] Lost Plot
[ ] Variable Value Evolution
[✓] Variables value inside Threshold: green
Min: 2
Max: 3

PATH #3

```
************************************************************
                        STROKE POINTS
************************************************************

The following passages are visited in all paths:
['Beggining']
['Continue']
['Evidence']

************************************************************
              VARIABLES VALUES INSIDE THRESHOLD
************************************************************
```

```
**********************************************************
                 SELECTED PATH #: 3
**********************************************************

---- Beggining ----

Detective: I'd like to ask you a few questions

[[Where were you last night?->Choice1]]
[[Last night you were at the party, correct?->Choice2]]

(set: $anger to 0) (set: $fear to 0) (set: $confusion to
0)
```

Figure 5.2: Story Validator second prototype

Based on the feedback gathered during Phase 1 of user testing, we improved the first prototype and added some new features, including:

- The option to test more than one story variable simultaneously.
- A "Toggle Tooltips" button, which allows the user to hover different elements on the tool's menu to display a short message detailing how each Analysis Conditions works.
- A "Print Report" button, which creates a full pdf report of the story analysis that the user can access outside the tool's workspace.
- A dotted chart graph, which displays a timeline with the changes in the values of each story variable select for analysis throughout the story path selected.

- Each section separator is now colored blue. The user can now also select the color in the ”Ending Hit Percentage” and ”Variable Value inside Threshold” options

5.5.2 Procedure

During this phase, we performed a follow-up user test of the second prototype, where the users were given a branching story (created using Twine) with various problems and were asked to use the **Story Validator** second prototype to identify those problems. They were then asked to propose solutions/changes to these problems.

During the study, we observed their performance, took notes, and measured the time each participant took to identify the problems, or until they gave up. These sessions were also recorded.

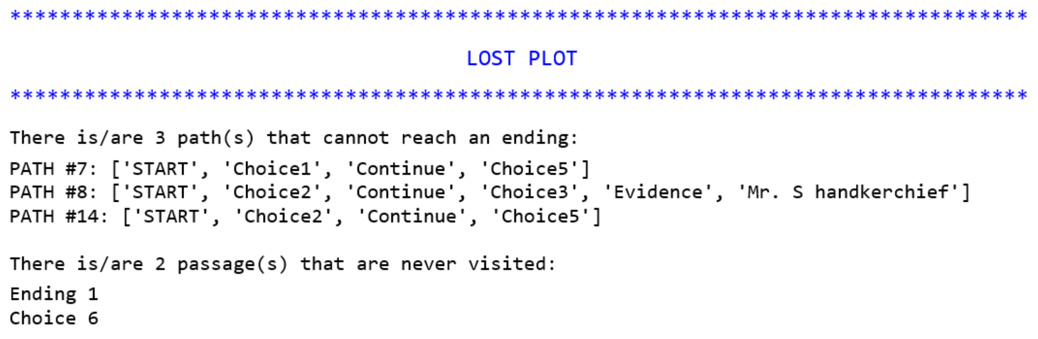
Afterwards, the subjects were asked to score the usability of the tool using the SUS and respond to a questionnaire.

5.5.3 Scenarios and expected performance

The test story that was given for the participants to analyse had the following issues:

Problem 1: The Path #7 does not reach an Ending Passage.
Problem 2: The Path #8 does not reach an Ending Passage.
Problem 3: The Path #14 does not reach an Ending Passage.
Problem 4: The Passage “Ending 1” is never visited in any path.
Problem 5: The Passage “Choice 6” is never visited in any path.

All the previous problems can be identified in the **Story Validator** under the Analysis Condition “Lost Plot”, which reveals *Passages* that can not be visited and Paths that do not reach an ending, if existent.



```
******************************************************************************
                                   LOST PLOT
******************************************************************************

There is/are 3 path(s) that cannot reach an ending:
PATH #7: ['START', 'Choice1', 'Continue', 'Choice5']
PATH #8: ['START', 'Choice2', 'Continue', 'Choice3', 'Evidence', 'Mr. S handkerchief']
PATH #14: ['START', 'Choice2', 'Continue', 'Choice5']

There is/are 2 passage(s) that are never visited:
Ending 1
Choice 6
```

Figure 5.3: Analysis Condition “Lost Plot”

However, the reasoning behind each of these problems is unique, and the user needs to make use of the tool in order to grasp what exactly is causing each of these issues.

Analysing Problems 1 and 3

These two problems have a similar cause: both stop at the *Passage* “Choice 5”, as it is shown under the results of Lost Plot. Alternatively, the users can identify this issue by observing the Path Dialog Tree after picking either path #7 or #14, under the option “Select Path” or by looking at the Main Dialog Tree.

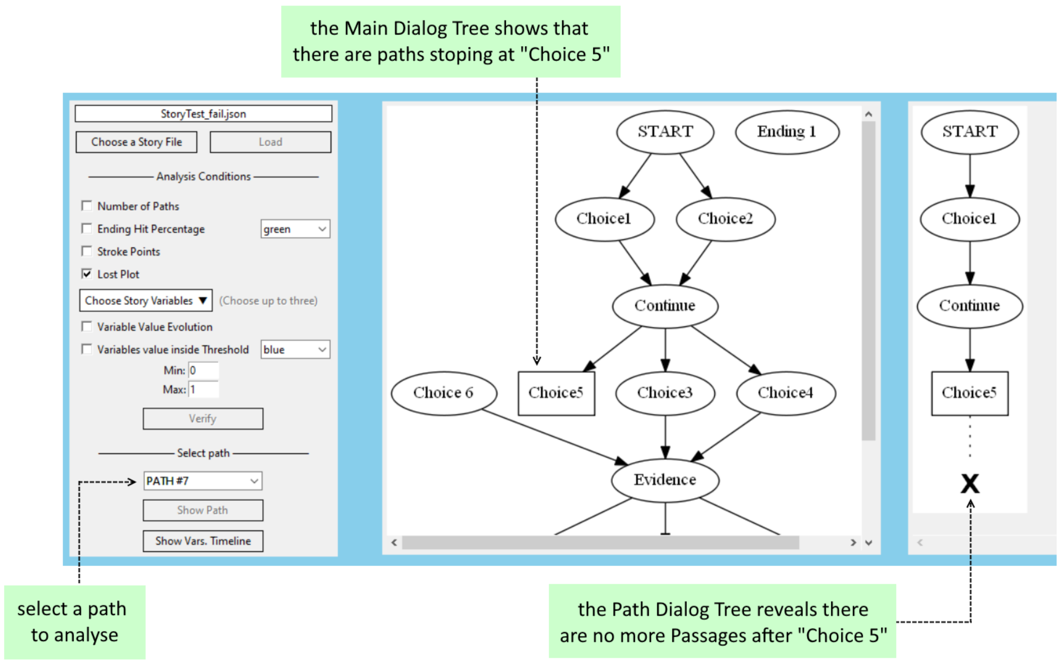


Figure 5.4: Analysing Problems 1 and 3

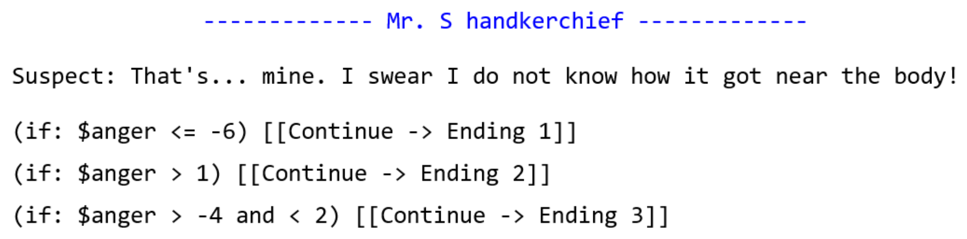
Upon reading the results in the Path Report Log, the user will then realise the issue in question was due to the fact that, during the creation of the story, the writer did not add any *Links* to the *Passage* “Choice 5” and consequently the playthrough stopped abruptly.

Analysing Problem 2

The user is able to identify this issue by proceeding in a similar manner to the previous two problems. In the case of Path #8, the story stops at the *Passage* “Mr S handkerchief”, as it is displayed under the results of Lost Plot. Alternatively, the users can identify this issue by looking at the Main Dialog Tree or by observing the Path Dialog Tree after picking path #8, under the option “Select Path”.

Still, the reasoning behind the occurrence of this problem is distinct from the other two previous problems. Upon reading the results in the Path Report Log, the user will then realise that the *Passage* in question has an <<if>> macro, that depending on the value of the variable $anger, it selects one of the [Ending Passages] to visit.

By selecting the Analysis Condition “Variables Value Evolution”, on the Path Dialog Tree, the user can consult that the value of the variable $anger at that point is -5. However, the <<if>> macro does not have a condition that in includes the value -5 and, consequentially, the playthrough stops abruptly. In order to fix this issue, the designer has to either rewrite the <<if>> macro to include the value -5 or make adjustments to the value of the variable $anger in previous *Passages* to meet the necessary conditions.



```
------------- Mr. S handkerchief -------------

Suspect: That's... mine. I swear I do not know how it got near the body!

(if: $anger <= -6) [[Continue -> Ending 1]]
(if: $anger > 1) [[Continue -> Ending 2]]
(if: $anger > -4 and < 2) [[Continue -> Ending 3]]
```

Figure 5.5: The Passage Mr. S handkerchief

Analysing Problem 4

As it is displayed under the results of Lost Plot, the *Passage* “Ending 1” is never visited in any path. At the same time, on the Main Dialog Tree (5), the user can observe that the *Passage* “Ending 1” appears as a single node with no edges (i.e., *Links*) connected to it.

As it was understood on the previous analysed problem, in this story there is a <<if>> macro that, depending on the value of the variable $anger, selects one of the [Ending Passages] to visit. Therefore, the user can conclude that the Passage “Ending 1” is never visited because the variable $anger not once holds values that satisfy the condition that selects the “Ending 1”. To fix this issue, the writer has to rewrite the <<if>> macro or make adjustments the variable $anger in previous *Passages* to meet the necessary conditions.

Analysing Problem 5

As it is displayed under the result of Lost Plot, the *Passage* “Choice 6” is never visited in any path. Alternatively, on the Main Dialog Tree (5), the user can observe that the *Passage* “Choice 6” appears as a single node connected to the *Passage* “Evidence”, but without any parent nodes. Consequently, since a Twine’s story always starts at the Starting Point (i.e., the root node), there are no paths that traverse through the *Passage* “Choice 6”. To fix this issue, the writer has to create a connection to “Choice 6” somewhere in the story.

6 Results and Analysis

Contents

In the following chapter we are going to be stating our results gathered from the usability tests and critically analyse them in order to establish some conclusions regarding our solution.

6.1 Statistical data from the questionnaires

After each experiment, the participants were asked to fill in a demographic questionnaire (see Appendix B) to help us identify the profile of our sample population. This questionnaire provided us with meaningful information about the users, regarding their background with Twine, games, and game design.

Phase 1 consisted of 3 female participants and 2 males, between the ages 18 and 24 (60%), all of Portuguese nationality. Most participants admitted they played games either regularly (40%) or every day (40%), with some (60%) enjoying games with interactive stories and branching narratives. Out of the 5 participants, 3 stated they had knowledge in game development, manly working as a Game Programmer and/or Game Writer. These users reported the following problems when using their preferred authoring tool for writing the narrative:

- Difficulty keeping track of game variables and/or objects;
- Difficulty identifying ”dead-ends”, meaning moments in the game, that once reached, prevent the player from continuing playing;
- Difficulty keeping track plot moments in the game that the player skips unintentionally, losing the plot coherence;
- The lack of ability to keep track of user experience (before playtesting)

Moreover, only one participant was familiar with Twine’s authoring tool and considered themselves to have an “Intermediate” expertise on the use of the tool. The most preferred authoring tool to write stories for games between all the users, was a basic word processing tool, such as Microsoft Word, Google Docs or LibreOffice Writer.

Phase two consisted of eight female participants and twelve males, all of Portuguese nationality. Except for one participant, all others admitted they played video games either every day (35%) or regularly (60%). Nine of the twenty participants said to have some knowledge in game development (as Game Programmers, Game Designers or Game Writers). Five of those participants were familiar with Twine’s authoring tool prior to the test and considered themselves to have “Intermediate” expertise.

6.2 Results for Phase 1 of User Testing

6.2.1 Total completion rates

For determining whether a certain percentage of users can complete a task, we measured how many participants were able to succeed or fail each task. We assigned a binary value of "1" if the participant managed to complete a task and "0" if they failed or gave up.

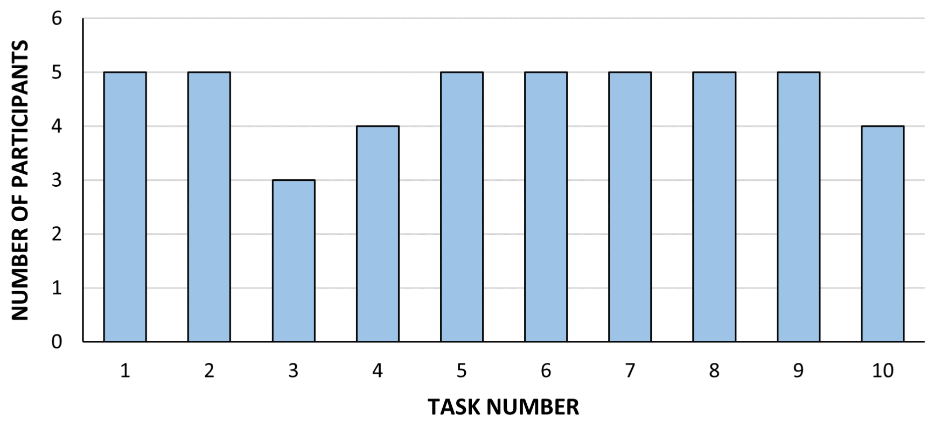


Figure 6.1: Task Completion Rate Results [Phase 1]

Figure 6.1 shows that 2 of the participants in the study group failed to complete Task 3, which involved counting the number of paths that reached an ending with a variable anger value of 0. These incidences seemed to occur due to the rise in difficulty from tasks 1 and 2 to task 3.

Additionally, there was 1 participant that was not able to complete task 4, which implicated finding the Passages in which the variable $anger had values between -4 and -2. This participant in question did not understand they needed to insert -4 on the MIN box and -2 on the MAX box, under the "Variable Value by given Threshold" section. Instead, they analysed one path at a time, searching for Passages where the variable $anger had values within that range. Eventually, they gave up, stating "This is taking me too long. I feel like I am doing something wrong.".

Finally, the same participant was unable to complete task 10, which requested they find the text inside the Passage Choice 4. This participant did not realise they could find what they were looking for under the Path Results Log. For all the previous tasks, the answers were under the Main Results Log, and because they were used to looking for answers to each task there, they did not realise they should look for the solution on the other log.

In conclusion, the completion rate for task 3 is 60%, and for task 4 and 10 is 80%. For all the other tasks (1, 2, 5, 6, 7, 8 and 9) the completion rate is 100%, meaning every participant was able to complete them.

6.2.2 Task Level Usability score

After completing each task, the participants were asked to respond to a Single Ease Question (SEQ), where they were asked how difficult or easy a task was to complete on a scale of 1 (very difficult) to 7 (very easy).

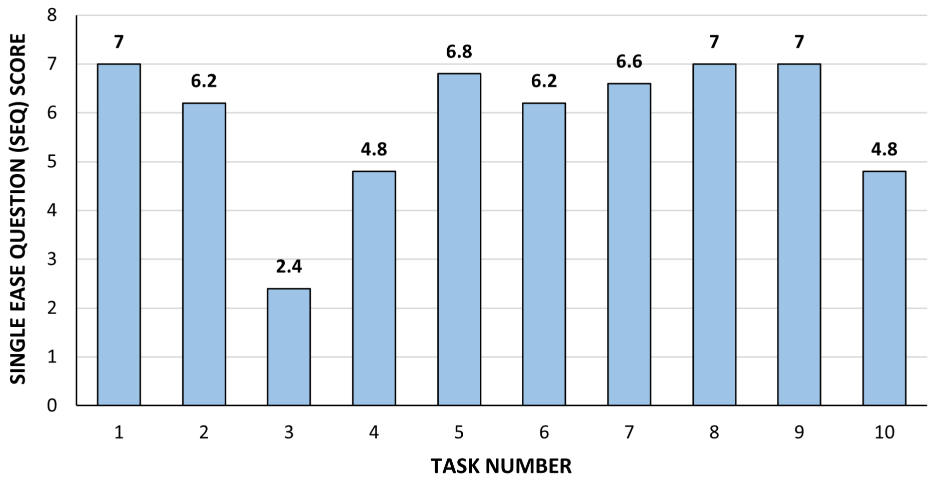


Figure 6.2: Average values of Single Ease Question (SEQ) [Phase 1]

The results show that the average user found Task 3 the most difficult, followed by Tasks 4 and 10 (see Figure 6.2). Tasks 1, 8 and 9 were the easiest to complete, with an average score of 7, according to all participants.

These results match the ones in section 6.2, as the tasks that some participants were unable to complete (3, 4 and 10) were the ones that were considered the most difficult. Regarding task 3, besides the increase in difficulty compared to the previous tasks, some participants also mentioned that the task was also difficult to understand. On the other hand, several participants considered that task 10, while easy to understand, was difficult to complete. The users knew right away they could find the Choice 4's textual contents in the Tree Report Log but had trouble finding a path that travelled through Choice 4. It was not clear for some of them that selecting the option "Number of Paths", under the section "Analysis Conditions" would facilitate the discovery of the correct path. Instead, some of the participants opted to select one path at a time, under the "Select Path" section, until they eventually found a path that reached the *Passage* Choice 4, which took longer. While this method is legitimate, in a situation where the story being analysed has a large quantity of paths, it can become a lengthy and tedious process and, therefore, not desirable.

6.2.3 Average Task Completion Time

During the experiments, we measure how long each user took to complete every single task. The time was quantified from when the task was handed out until the user finished the task.

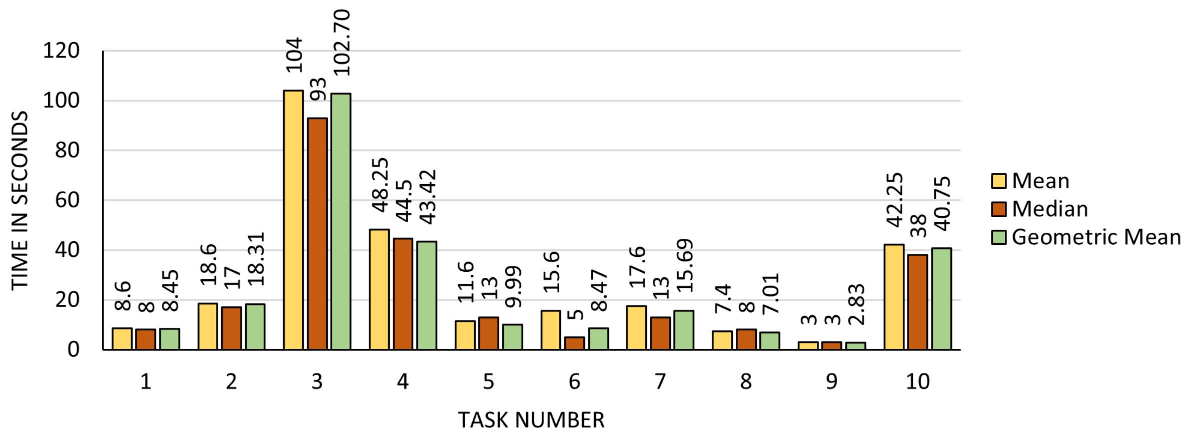


Figure 6.3: Task Completion Time for each task type [Phase 1]

The results presented in Figure 6.3 show that most users took the highest time completing Task 3, spending an average of 104 seconds (almost 2 minutes) on it, followed by task 4 and 10, where users spent an average of 48 and 42 seconds, respectively. These results are equivalent to the ones found in the Task Completion results (Section 6.2.1) and in the task Level Usability score results (Section 6.2.2). As expected, the tasks that were considered more challenging were the same ones that took longer to complete.

On the other hand, the task that took the least time on average to complete was Task 9, with an average of 3 seconds. In Task 9, we asked the participants if there were any *Passages* that were never visited on any path. After discussion, we concluded that this was due to the fact that the response to Task 9 was in the same place as the response to Task 8, and therefore users took a short time to complete the task.

Note that while their times were still counted, participant 1 (P1) did not complete Task 3, and participant 4 (P4) did not complete Task 3, Task 4, and Task 10. Therefore, their values are not added up to the Average Task Completion Time for those tasks.

6.2.4 System Usability Scale

At the end of the test session, we gave a formalised questionnaire to each test participant. In this questionnaire, we asked users to answer a System Usability Scale (SUS), regarding their experience and satisfaction with the tool. The SUS is a set of 10 statements, which the users rank on a Likert scale, ranging from 1 (strongly disagree) to 5 (strongly agree).

The ten standard questions of the System Usability Scale (SUS), initially defined by John Brooke [64], 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.

After gathering all the participants' answers, we calculated the SUS score for each. According to a study made by Bangor, Kortum, and Miller [65], the SUS scores fall into a range of categories: "worst imaginable", "poor", "OK", "good", "excellent" and "best imaginable"; which provide an adjective rating that correlates with a given score. Since the participants P2, P4 and P5 got SUS scores above 85.58, their scores are considered "excellent". Participant P1 score is higher than 72.75, so their score is considered "good". Finally, participant P3 score is considered "OK", since they had a SUS score above 52.01. All the results are presented in Table 6.1.

Participants	1	2	3	4	5	6	7	8	9	10	SUS score	Adjective
P1	5	2	4	1	3	1	5	1	4	3	82.5	Good
P2	4	2	4	1	4	1	5	1	5	1	90	Excellent
P3	3	2	4	4	4	1	5	1	2	3	67.5	OK
P4	5	1	4	2	4	1	5	1	5	2	90	Excellent
P5	5	2	4	1	4	1	5	1	3	1	87.5	Excellent

Table 6.1: System Usability Scale (SUS) Results [Phase 1]

Additionally, the average SUS score is 83.5 (SD = 9.45) which, according to the same study, means that our solution, regarding its overall usability, is considered "passable". However, the participant P3 SUS score (67.5) suggests that the tool had some usability issues. A product with a score below 50 would be considered "unacceptable". Fortunately, the study revealed no such cases.

6.2.5 Usability suggestions

The following section describes the different suggestions proposed by the participants regarding the tools usability during Phase 1 of the user testing.

Suggestion 1 – Help understanding how the tool works
Some users reported having initial issues when trying to understand how the tool functions. While everyone was quick to learn, most still suggested with would had been easier had they been given a tutorial demo to explain the tool and how it worked. As one participant stated "Once you start clicking some check boxes you learn pretty quickly how [the tool] works [...] but having a help button or something similar would have helped a lot." and added "[the tool] is very overwhelming at first."

Suggestion 2 – Easier to read log report
In cases where multiple analysis conditions were selected, users had issues while navigating through the log report box, and often had to keep scrolling up and down to locate what they were seeking. Participants noted that if the box were bigger or if the analysis conditions were separate from each other, it would be easier to navigate.

Suggestion 3 – Desire to analyse more than one variable
One of the users (20%) reported the desire to view an analyse more than one variable at a time. While this did not hinder their ability to complete the tasks, their suggestion was noted as a hurdle that could arise in the future and therefore needed fixing.

6.3 Results for Phase 2 of User Testing

6.3.1 Total Completion Rate

To calculate the completion rate for Phase 2, we measure how many participants were able to identify each of the problems indicated in section 5.5.3 and how many were then able to work a solution for each. Once again, we assigned a binary value of "1" if the participant managed to complete a task and "0" if they failed or gave up.

First, we gathered that most participants were able to identify all the problems, except for three of them that were not able to identify Problem 2: The Path #8 does not reach an [Ending Passage]. This was the situation were story stopped at the *Passage* "Mr S handkerchief" and never reaches an ending. We believe that the reason for the misidentifying of this problem was due to the fact that these participants in question, instead of using the Analysis Condition "Lost Plot" to identify the issues with the story, they found them by observing the Main Dialog tree. While this method is legitimate, none realized that

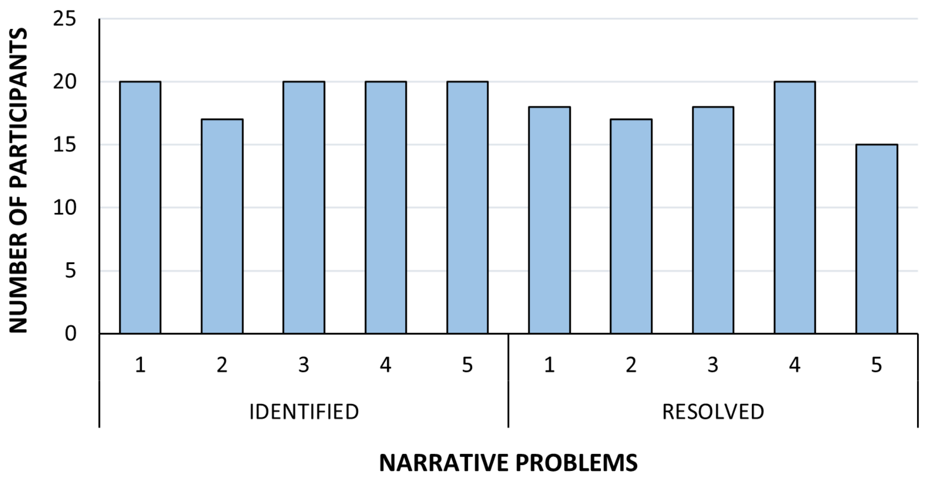


Figure 6.4: Task Completion Rate Results [Phase 2]

while 4 paths reached the *Passage* “Mr S handkerchief” only 3 of them reached an ending. Consequently, none of these users solved the problem in question. The completion rate for the identification and resolution of Problem 2 is 85%. Additionally, two users (90%) were not able to solve the Problems 1 and 3: The Path #7 and #4 do not reach an [Ending Passage]. These problems refer to the situation were narrative stopped at the *Passage* “Choice 5”. Alternatively, Problem 5: The *Passage* “Choice 6” is never visited in any path; seemed to be the one that was harder to solve, at least for five of the participants (75% rate of completion). For all the other situations the completion rate is 100%, meaning every participant was able to complete them.

6.3.2 Task Level Usability score

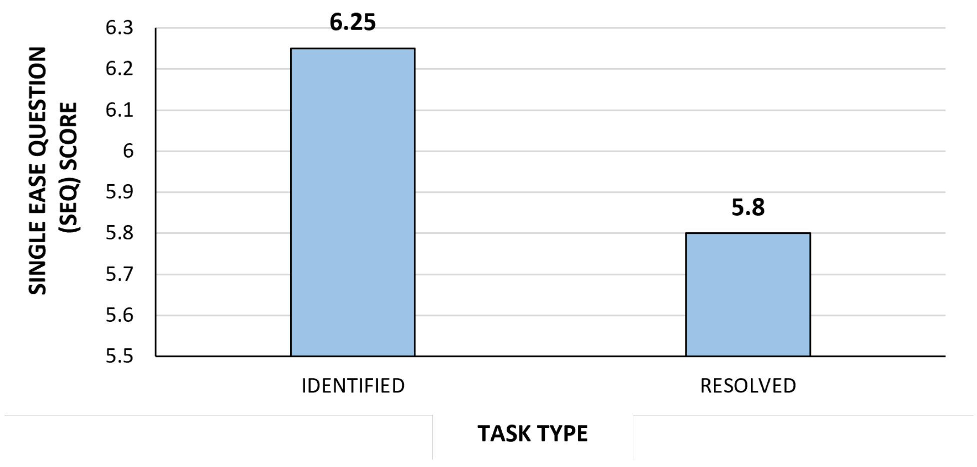


Figure 6.5: Average values of Single Ease Question (SEQ) [Phase 2]

After finishing identifying and solving the problems, the participants were asked to respond to a Single Ease Question (SEQ), where they were asked how difficult or easy a task was to complete on a scale of 1 (very difficult) to 7 (very easy).

According to the results, the average SEQ score for identifying the problems is 6.25 and for resolving the problems is 5.8. While using the tool to pinpoint the problems was easy and only required a few “clicks”, solving the problems required more energy to analyse the results and deduct a solution.

6.3.3 Average Task Completion Time

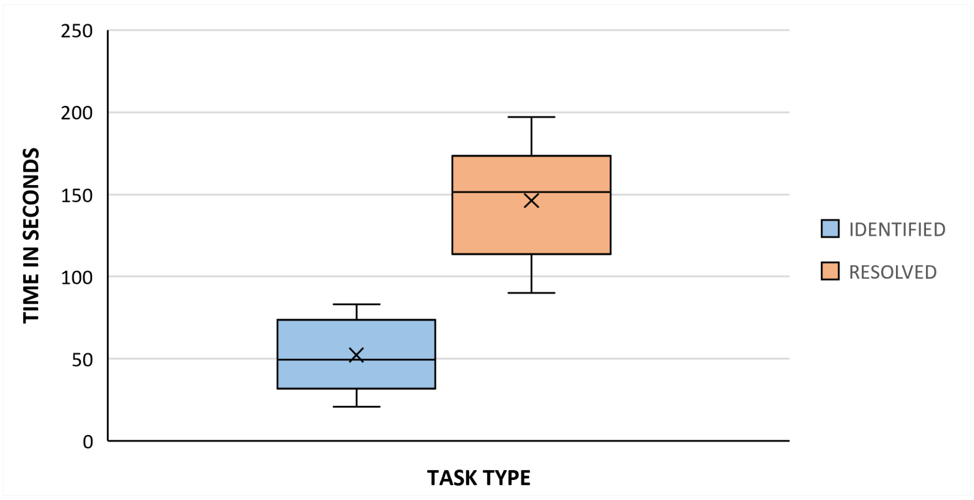


Figure 6.6: Task Completion Time for each task type [Phase 2]

During the experiments, we measure how long each user took to identify all the problems and solve them. The time was quantified from when the task was handed out until the user finished the task.

Figure 6.6 shows that when comparing the time it took users to identify all the problems to the time it took solve them, that the former took much longer. On average the time it took users to identify all the problems was 52.3 seconds (SD = 22.66), and to solve them it took them on average 146.2 seconds (SD = 31.84).

Overall, the time values for both finding and solving problems prove that the tool is efficient and that, with little habit, users can quickly use the tool to pinpoint and repair errors in their interactive narratives.

6.3.4 Usability Scale (SUS) System

Once again, at the end of the test session, we gave a formalised questionnaire to each test participant. In this questionnaire, we asked users to answer a System Usability Scale (SUS), regarding their experience and satisfaction with the tool.

Participants	1	2	3	4	5	6	7	8	9	10	SUS score	Adjective
P1	4	1	4	1	5	1	5	1	4	1	92.5	Excellent
P2	4	2	4	2	5	1	5	1	5	1	90	Excellent
P3	5	1	5	1	5	1	5	1	5	1	100	Best Imaginable
P4	3	1	4	2	5	1	5	1	4	2	85	Good
P5	3	2	4	1	5	1	5	1	4	2	85	Good
P6	4	1	4	1	5	1	5	1	5	1	95	Excellent
P7	4	1	4	2	5	1	5	1	4	3	85	Good
P8	5	1	5	1	5	1	5	1	5	1	100	Best Imaginable
P9	4	1	4	2	5	1	5	1	5	1	92.5	Excellent
P10	5	1	5	1	5	1	5	1	4	2	95	Excellent
P11	4	1	3	1	5	1	5	1	4	2	87.5	Excellent
P12	4	1	4	1	5	1	5	1	4	1	92.5	Excellent
P13	5	1	5	2	5	1	5	1	4	2	92.5	Excellent
P14	4	1	4	1	5	1	5	1	4	3	87.5	Excellent
P15	5	2	5	1	5	1	5	1	4	1	95	Excellent
P16	3	1	5	2	5	1	5	1	5	2	90	Excellent
P17	4	1	4	1	5	1	5	1	5	1	95	Excellent
P18	5	1	5	1	5	1	5	1	5	1	100	Best Imaginable
P19	5	1	4	1	5	1	5	1	5	2	95	Excellent
P20	4	1	5	1	5	1	5	1	4	2	92.5	Excellent

Table 6.2: System Usability Scale (SUS) Results [Phase 2]

Since the participants P3, P8 and P18 got SUS scores of 100.0, their scores are considered “best imaginable”. Participants P4, P5 and P7 score is higher than 72.75, so their score is considered “good”. Finally, all the other participants scores are considered “excellent”, since they had a SUS score above 85.58. All the results are presented in Table 6.2.

On average the SUS score of our system is 92.4 (SD = 4.76). Our average SUS score proves that our system is considered a “truly superior product”, according to [65], however, we nonetheless received some suggestions for improvements on the tool.

6.3.5 Usability suggestions

The following section describes the usability suggestion that users gave us during Phase 2 of the user testing.

Suggestion 1 – Exploring using clickable Passages

Some users suggested that, besides exploring the narrative through each path, it would be useful to have the option to explore through the use of *Passages*. What they proposed was the possibility to interact directly with the tree , by clicking on the nodes.

7 Conclusion

Contents

In this dissertation, we have underlined the current challenges concerning the authoring process of Interactive Narratives.

After analysing several past works pertaining to the authoring of Interactive Narratives, we noticed how there was a lack of tools that provided ways for the author to test their narrative while considering the player's agency, during the game's developmental stage. More often than not, these works opt for online Artificial Intelligent (AI) approaches (such as drama managers) that, during gameplay, dynamically adapt the narrative and resolve conflicts created by unintended player's actions. This might lead to situations where the system takes control of the story, replacing human authorship.

With this work, we set ourselves to develop a tool for testing interactive dialogues for video games, that allows human authors to express their artistic intentions without feeling constrained. This approach has been designed to facilitate the development of interactive narratives in stages before human playtesting by letting the author explicitly test different hypotheses and narrative properties to identify possible design mistakes. The tool's GUI allows for a clearer picture of the interactive narrative authoring process, by providing a visual representation of the narrative structure through the use of tree structures, that run through different test conditions.

After a thorough analysis of our test results, we concluded that we met our requirements. Several users said that they found the tool to be an essential asset for the creation of interactive stories, even though many expressed the desire to have more testing features and the option to interact directly with the tree.

Overall, we believe that, as a first approach to this type of systems, our prototype managed to achieve the proposed objectives; however, as it stands, there is room for improvement.

7.1 Future Work

In order to improve our prototype, we have defined a set enhancements and new functionalities that we plan to add in the future.

Firstly, we plan to add the option to explore other narrative properties. For example, through re-utilising the "tag" method, we can study the distribution of different Passages and how often the player visits them in different playthroughs. Additionally, we would also like to add the option to explore the narrative through each Passage and not just through Paths.

As mentioned in the previous Section 2.2.1 and explored in detail in Section 3.2, player play-styles are reflected not only in how they interact with the narrative but also on how they expect the story to unfold. Therefore, we also have plans to make improvements to the tool in the future, through the implementation of an artificial agent that simulates multiple automated play sessions, emulating the behaviour of different player-types based on player models, in order to predict user experience and better evaluate interactive games. We plan on achieving this through the usage of derived heuristics that reflect play traces and human behaviour. The weight of each of these heuristics could be adjusted to play-styles that mirror those of different types of players.

Bibliography

[1] J. Hidders, "'s answer to "What are BFS and DFS in a binary tree?" Quora." 2019, September 19, online; Retrived 27-January-2020. [Online]. Available: https://www.quora.com/What-are-BFS-and-DFS-in-a-binary-tree/answer/Jan-Hidders

[2] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[3] E. Aarseth, "A narrative theory of games," in *Proceedings of the international conference on the foundations of digital Games*, 2012, pp. 129–133.

[4] H. Qin, P.-L. Patrick Rau, and G. Salvendy, "Measuring player immersion in the computer game narrative," *Intl. Journal of Human–Computer Interaction*, vol. 25, no. 2, pp. 107–133, 2009.

[5] M. O. Riedl and V. Bulitko, "Interactive narrative: An intelligent systems approach," *Ai Magazine*, vol. 34, no. 1, pp. 67–67, 2013.

[6] S. Dinehart, "Dramatic play," 2009, online; Retrived 5-November-2020. [Online]. Available: http://www.gamasutra.com/view/feature/4061/dramatic_play.php

[7] M. Mateas, "The authoring challenge in interactive storytelling," in *Joint International Conference on Interactive Digital Storytelling*. Springer, 2010, pp. 1–1.

[8] P. Mirza-Babaei, N. Moosajee, and B. Drenikow, "Playtesting for indie studios," in *Proceedings of the 20th International Academic Mindtrek Conference*, 2016, pp. 366–374.

[9] P. Mirza-Babaei, V. Zammitto, J. Niesenhaus, M. Sangin, and L. Nacke, "Games user research: practice, methods, and applications," in *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, 2013, pp. 3219–3222.

[10] N. Moosajee and P. Mirza-Babaei, "Games user research (gur) for indie studios," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2016, pp. 3159–3165.

[11] J. Bates, “Virtual reality, art, and entertainment,” *Presence: Tele-operators and Virtual Environments*, vol. 1, no. 1, pp. 133–138, 1992.

[12] C. Klimas, “Twine - an open-source tool for telling interactive, nonlinear stories,” 2020, online; Retrived 5-January-2020. [Online]. Available: https://twinery.org/

[13] T. Rothamel, “The ren’py visual novel engine,” 2020, online; Retrieved 16-November-2020. [Online]. Available: https://www.renpy.org/

[14] M. O. Riedl, “Incorporating authorial intent into generative narrative systems.” in *AAAI Spring Symposium: Intelligent Narrative Technologies II*, 2009, pp. 91–94.

[15] B. Books, *Choose Your Own Adventure Book Series*. Bantam Books: NYC, 1979 - 1998.

[16] C. Moser and X. Fang, “Narrative structure and player experience in role-playing games,” *International Journal of Human-Computer Interaction*, vol. 31, no. 2, pp. 146–156, 2015.

[17] J. Majewski *et al.*, “Theorising video game narrative,” *Bond University*, 2003.

[18] R. Aylett, “Emergent narrative, social immersion and “storification”,” in *Proceedings of the 1st International Workshop on Narrative and Interactive Learning Environments*, 2000, pp. 35–44.

[19] E. Adams, *Fundamentals of game design*. Pearson Education, 2014, pp. 172–173.

[20] Bethesda Game Studios, “Fallout 3,” Rockville, Maryland: Bethesda Softworks, 2008.

[21] S. Dow, B. MacIntyre, and M. Mateas, “Styles of play in immersive and interactive story: case studies from a gallery installation of ar façade,” in *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, 2008, pp. 373–380.

[22] S. P. Hernandez, V. Bulitko, and M. Spetch, “Keeping the player on an emotional trajectory in interactive storytelling,” in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[23] M. O. Riedl and R. M. Young, “From linear story generation to branching story graphs,” *IEEE Computer Graphics and Applications*, vol. 26, no. 3, pp. 23–31, 2006.

[24] A. Cayley, “Xxviii. on the theory of the analytical forms called trees,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, pp. 172–176, 1857.

[25] E. F. Moore, “The shortest path through a maze,” in *Proc. Int. Symp. Switching Theory, 1959*, 1959, pp. 285–292.

[26] C. E. Leiserson and T. B. Schardl, "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)," in *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, 2010, pp. 303–314.

[27] C. P. Trémaux, "École polytechnique of paris (x: 1876)," in *French engineer of the telegraph in Public conference, December*, vol. 2, 2010, pp. 0980–603.

[28] S. Even, *Graph algorithms*. Cambridge University Press, 2011.

[29] R. Sedgewick, *Algorithms in c++, parts 1-4: fundamentals, data structure, sorting, searching*. Pearson Education, 1998, vol. 1.

[30] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[31] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[32] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," *arXiv preprint arXiv:1908.01423*, 2019.

[33] P. W. Weyhrauch, *Guiding interactive drama*. Carnegie Mellon University, 1997.

[34] M. J. Nelson, D. L. Roberts, C. L. Isbell Jr, and M. Mateas, "Reinforcement learning for declarative optimization-based drama management," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006, pp. 775–782.

[35] M. Mateas and A. Stern, "Architecture, authorial idioms and early observations of the interactive drama façade," 2002.

[36] ——, "Façade: An experiment in building a fully-realized interactive drama," in *Game developers conference*, 2003, pp. 4–8.

[37] ——, "Structuring content in the façade interactive drama architecture." in *AIIDE*, 2005, pp. 93–98.

[38] ——, "A behavior language: Joint action and behavioral idioms," in *Life-Like Characters*. Springer, 2004, pp. 135–161.

[39] M. O. Riedl and A. Stern, "Believable agents and intelligent story adaptation for interactive storytelling," in *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*. Springer, 2006, pp. 1–12.

[40] M. O. Riedl, A. Stern, D. Dini, and J. Alderman, “Dynamic experience management in virtual worlds for entertainment, education, and training,” *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning*, vol. 4, no. 2, pp. 23–42, 2008.

[41] J. Porteous and M. Cavazza, “Controlling narrative generation with planning trajectories: the role of constraints,” in *Joint International Conference on Interactive Digital Storytelling*. Springer, 2009, pp. 234–245.

[42] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen, “Interactive storytelling: A player modelling approach.” in *AIIDE*, 2007, pp. 43–48.

[43] R. Laws, *Robin’s laws of good game mastering*. Steve Jackson Games, 2002.

[44] T. Fullerton, C. Swain, and S. Hoffman, *Game design workshop: a playcentric approach to creating innovative games*. AK Peters/CRC Press, 2018.

[45] A. Zook, E. Fruchter, and M. O. Riedl, “Automatic playtesting for game parameter tuning via active learning,” *arXiv preprint arXiv:1908.01417*, 2019.

[46] C. Holmgard, M. C. Green, A. Liapis, and J. Togelius, “Automated playtesting with procedural personas with evolved heuristics,” *IEEE Transactions on Games*, 2018.

[47] L. Mugrai, F. Silva, C. Holmgård, and J. Togelius, “Automated playtesting of matching tile games,” in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–7.

[48] S. Devlin, A. Anspoka, N. Sephton, P. I. Cowling, and J. Rollason, “Combining gameplay data with monte carlo tree search to emulate human play,” in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[49] S. Stahlke, A. Nova, and P. Mirza-Babaei, “Artificial playfulness: A tool for automated agent-based playtesting,” in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, p. LBW0176.

[50] L. E. Nacke, C. Bateman, and R. L. Mandryk, “Brainhex: preliminary results from a neurobiological gamer typology survey,” in *International conference on entertainment computing*. Springer, 2011, pp. 288–293.

[51] C. Bateman, R. Lowenhaupt, and L. E. Nacke, “Player typology in theory and practice.” in *DiGRA Conference*, 2011.

[52] C. Bateman and R. Boon, *21st Century Game Design (Game Development Series)*. Charles River Media, Inc., 2005.

[53] I. Biederman and E. A. Vessel, “Perceptual pleasure and the brain: A novel theory explains why the brain craves information and seeks it through the senses,” *American scientist*, vol. 94, no. 3, pp. 247–253, 2006.

[54] M. Bernstein, “Collage, composites, construction,” in *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, 2003, pp. 122–123.

[55] T. Rothamel, “Quickstart — ren’py documentation,” 2020, online; Retrieved 20-December-2020. [Online]. Available: https://www.renpy.org/doc/html/quickstart.html

[56] S. Mascarenhas, M. Guimarães, R. Prada, J. Dias, P. A. Santos, K. Star, B. Hirsh, E. Spice, and R. Kommeren, “A virtual agent toolkit for serious games developers,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–7.

[57] M. Guimarães, S. Mascarenhas, R. Prada, P. A. Santos, and J. Dias, “An accessible toolkit for the creation of socio-emotionalagents,” in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2357–2359.

[58] R. S. Aylett, S. Louchart, J. Dias, A. Paiva, and M. Vala, “Fearnot!–an experiment in emergent narrative,” in *International Workshop on Intelligent Virtual Agents*. Springer, 2005, pp. 305–316.

[59] J. Dias and A. Paiva, “Feeling and reasoning: A computational model for emotional characters,” in *Portuguese Conference on Artificial Intelligence*. Springer, 2005, pp. 127–140.

[60] F. Correia, T. Ribeiro, P. Alves-Oliveira, N. Maia, F. S. Melo, and A. Paiva, “Building a social robot as a game companion in a card game,” in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2016, pp. 563–563.

[61] W. Westera, B. Fernandez-Manjon, R. Prada, K. Star, A. Molinari, D. Heutelbeck, P. Hollins, R. Riestra, K. Stefanov, and E. Kluijfhout, “The rage software portal: Toward a serious game technologies marketplace,” in *International Conference on Games and Learning Alliance*. Springer, 2018, pp. 277–286.

[62] M. Cavazza, F. Charles, and S. J. Mead, “Character-based interactive storytelling,” *IEEE Intelligent systems*, vol. 17, no. 4, pp. 17–24, 2002.

[63] A. Ortony, G. L. Clore, and A. Collins, *The cognitive structure of emotions*. Cambridge university press, 1990.

[64] J. Brooke, “Sus: a “quick and dirty’usability,” *Usability evaluation in industry*, p. 189, 1996.

[65] A. Bangor, P. T. Kortum, and J. T. Miller, “An empirical evaluation of the system usability scale,” *Intl. Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

A

Story Transcript

START

On June 9, 1954, a man invites a group of guests to his secluded New England mansion, for a dinner party. However, during the party, Miss M. (one of the guests) is found dead in the library.

You are the detective tasked with solving the murder. You arrive at the scene and collect evidence. Your main suspect is Mr. S who was seen arguing with the victim during the party. The evidence is not sufficient, and you will need a confession.

You bring the crime suspect back to the station for questioning. Mr. S is a very wealthy man and if you upset him during questioning, he will ask for his lawyer and you will lose the case. Choose your next options carefully, present the right evidence and make him confess for his murder!

Detective: I would like to ask you a few questions...

[[Where were you last night? -> Choice 1]]
[[Last night you were at the party, correct? -> Choice 2]]

(set: $anger to 0) (set: $fear to 0) (set: $confusion to 0)

Choice 1

Suspect: If you don't know, why am I here detective?

[[Continue]]

(set: $anger += 1) (set: $confusion += 2)

Choice 2

Suspect: Yes, indeed. I was invited.

[[Continue]]

(set: $anger -= 1)

Continue

Detective: I would also like to ask...

[[How well did you know Miss M? -> Choice 3]]
[[I heard you had an argument with Miss M prior to her murder. You know that makes you our main suspect? -> Choice 4]]
[[How was the food at the party? -> Choice 5]]

Choice 3

Suspect: Oh, how well did I know her? Too well! That woman owed me money for months and always came with the excuse of how "life was difficult". Yeah, right ... that dress she was wearing was certainly not cheap. That liar! I .... *sweating* I mean, I am sorry for my language, detective.

[[Continue -> Evidence]]

(set: $anger -= 2) (set: $fear += 3)

Choice 4

Suspect: And what if I did!? Are you accusing me of something, detective? I don't like that tone of yours...!

[[Continue -> Evidence]]

(set: $anger += 2)

Choice 5

Suspect: Yes, it was delightful... but I don't understand how is that relevant?

[[Continue -> Evidence]]

(set: $anger += 1) (set: $confusion += 2)

Evidence

Detective: Near the body we found this item. Can you tell me more about it?

Show an item:

[[Mr. S handkerchief]]
[[Miss M lipstick]]
[[Party Invitation]]

Mr. S handkerchief

Suspect: That's... mine. I swear I don't know how it got near the body!

(if: $anger <= -4) [[Continue -> Ending 1]]
(if: $anger > 1) [[Continue -> Ending 2]]
(if: $anger > -4 and < 2) [[Continue -> Ending 3]]

(set: $anger -= 2) (set: $fear += 3)

Mr. S handkerchief

Suspect: I'm sorry, is this interrogation a joke? This clearly belongs to a woman!

(if: $anger <= -4) [[Continue -> Ending 1]]
(if: $anger > 1) [[Continue -> Ending 2]]
(if: $anger > -4 and < 2) [[Continue -> Ending 3]]

(set: $anger += 2)

Party Invitation

Suspect: It the invitation for the party. All the guests received one exactly like that.

(if: $anger <= -4) [[Continue -> Ending 1]]
(if: $anger > 1) [[Continue -> Ending 2]]
(if: $anger > -4 and < 2) [[Continue -> Ending 3]]

(set: $confusion += 2)

Ending 1

Suspect: Ok, ok I confess! I was the one who killed her!

|ENDING 1: Good Detective|

Ending 2

Suspect: Enough of this! I want my lawyer.

|ENDING 2: Lawyer|

Ending 3

Detective: This is getting nowhere!

|ENDING 3: Not enough evidence|

B

User Survey

Users Survey

1. What is your gender?

- Female
- Male
- I do not want to disclose
- Other: ______________________________

2. How old are you?

- Under 18
- 18 to 24
- 25 to 34
- 35 to 44
- 45 to 54
- 55 to 64
- Over 65

3. What is your occupation?

____

4. What is your nationality?

- Portuguese
- Other: ______________________________

5. Do you enjoy writing stories?

- Yes
- No

6. Do you enjoy games with interactive stories or branching narratives?

- Yes
- No

7. Do you have experience in game development?

- Yes (go to question 8)
- No (go to question 12)

8. What role(s) did you partake in game development?

- ☐ Game Animator
- ☐ Game Audio Engineer
- ☐ Game Designer
- ☐ Game Programmer
- ☐ Game Artist
- ☐ Game Writer
- ☐ Game Marketer/PR
- ☐ Game Tester
- ☐ Other: ______________________________

9. If you have worked on writing a story for your game before, which software did you use?

- ○ Twine
- ○ Tinderbox
- ○ Basic word processing tool (e.g., Microsoft Word, Google Docs, LibreOffice Writer)
- ○ Other: __________________________

10. How do you classify your expertise with this software?

- ○ Expert (I use this software to create all my stories)
- ○ Intermediate (I have used this software several times before and I understand all its basic concepts)
- ○ Beginner (I have only used this software once or twice)

11. During the creation of your stories, have you encountered any of the following conflicts?

- ☐ Keeping track of game variables and other objects
- ☐ Keeping track of "dead-ends" (sections that stop the game abruptly, preventing the player from continuing)
- ☐ Keeping track of "lost plot" (sections that the player skips unintentionally)
- ☐ Keeping track of "unescapable loops" (dialog loop that the player cannot escape from regardless of the choices they make)
- ☐ Keeping track of user experience
- ☐ Other: ______________________________

12. Have you heard of Twine's authoring tool?

- ○ Yes (go to question 13)
- ○ No (go to question 14)

13. How do you classify your expertise with this tool?

- Expert (I use Twine to create all my interactive stories)
- Intermediate (I have used Twine several times before and I understand all its basic concepts)
- Beginner (I have only used Twine once or twice)
- Novice (I have only heard of Twine, but I have never used it)

14. [SUS] Regarding the Story Validator tool, rate the following statements on a scale from 1 (strongly disagree) to 5 (strongly agree)

	1	2	3	4	5
I think that I would like to use this system frequently.					
I found the system unnecessarily complex.					
I thought the system was easy to use.					
I think that I would need the support of a technical person to be able to use this system.					
I found the various functions in this system were well integrated.					
I thought there was too much inconsistency in this system					
I would imagine that most people would learn to use this system very quickly.					
I found the system very cumbersome to use.					
I felt very confident using the system.					
I needed to learn a lot of things before I could get going with this system.					