FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Denis Iudin

Procedural Generator of Short Detective-like Stories

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Jakub Gemrot, Ph.D.

Study programme: Computer Science

Specialization: Computer Graphics and Game Development

Prague 2023

I thank my family and friends for their understanding and support. I also want to thank my supervisor Jakub Gemrot for his advice, kindness and patience.

Title: Procedural Generator of Short Detective-like Stories

Author: Denis Iudin

Department / Institute: Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Jakub Gemrot, Ph.D.

Abstract: Procedural generation of interactive stories is still an understudied area, most of the work on which is purely academic, while the application of these technologies in a practical, for a wide audience, area promises good prospects. The reason lies in a large set of factors that complicate the practical use of such systems. In this work, we solve some of them by proposing a new algorithm for this, with the help of which we create a system that generates interactive stories. We visualize the narratives of these stories, and on their basis we create a text quest on the Twine platform.

Keywords: text game, procedural content generation, Twine

Contents

Introduction

Video games are an important part of modern culture and lifestyle. The number of people who are more or less fond of them is over three billion people around the world [1]. Also, in 2011, video games were officially recognized by the US government and the US National Endowment for the Arts as a separate art form, along with theater, cinema and others [2]. Moreover, the revenues of the gaming industry for 2021 amounted to $180 billion [3].

An important part of this industry is the production of high-budget games designed for a mass audience. However, their development requires significant work of large teams, consisting of hundreds of specialists in various fields, most of whom work on the creation of game content: sounds, music tracks, geometric models, textures, levels, animation, texts, narration, quests [4, 5]. With the passage of time and the development of technology, the amount of content produced only increases, requiring more and more financial resources and time. It would expect these factors to automate and speed up the content creation process, however, with rare exceptions, most content is still produced manually. As a result, content creation is seen as a bottleneck in mind of development budget and production time [4, 6]. Procedural content generation methods are a way to solve this problem, able to speed up the production of content by automating this process and thereby take some of the burden off developers. The essence of this approach is that game content is not created manually, but by a computer that executes a well-defined algorithm, with limited or without human participation [4, 5].

This topic has been researched for several decades, and during this time, procedural content generation methods have been used to create many types of content. For example, in Elite Dangerous [7] and No Man's Sky [8], procedural generation methods are used to create a huge (count goes into the billions) number of detailed star systems, in Borderlands [9] they are used to generate items with various properties, taking into account those features that the player uses most often, in Minecraft [10] are used to generate biomes and ecosystems, and separate systems are also used in many games at once, for example, SpeedTree [11] for generating vegetation and CityEngine [12] for generating urban environments [4, 13]. A separate type of procedural content generation, also developing over time, is the generation and adaptation of narratives [14]. By narrative, we, according to the definition given in The Cambridge Introduction to Narrative, mean the representation of an event or a series of events. Moreover, "event" here can be replaced by "action". Without an event/action, the text will be only a description, but with them it will be a narrative [15]. We will return to this issue below to consider it in more detail. However, at the moment, most of the methods for procedural generation of narratives are not adapted to function in a highly dynamic interactive environment. Therefore, they are not suitable for use in commercial video games intended for a general audience, either as a method for generating a scenario or as a method for generating quests (that are part of the narrative development mechanism). This is confirmed by the extremely few attempts to use them in commercial projects, which, at the same time, turn out to be very limited. For example, Left4Dead [16] generates scenarios by dynamically creating encounters, which can hardly be called narrative generation in the full sense of the word, nor is it quest generation [4]. The Elder Scrolls V: Skyrim [17] dynamically generates quests using the Radiant Story system, randomly filling quest templates with the correct entities: locations, enemy types, and reward options, which is a pretty straightforward approach. This system is not designed to generate large, long and complex quests [4, 18]. According to player feedback, quests generated by this system are significantly less interesting than quests generated by traditional methods [19]. It is also eloquent that not exists a single significant commercial game belonging to genres the essence of which is interactive storytelling - visual novels and text quests, which would use the methods of procedural narrative generation.

This situation is due to the fact that the generation of interactive narratives and quests for video games has a number of problems, the main of which is the dynamic interaction between the player, the environment, characters and the narrative itself as a whole. An important task in this case is to ensure the player's control over the environment and, at the same time, maintain the logical coherence of the narrative and the believable behavior of the characters. The essence of this problem is that any interaction, even the most insignificant at first glance, can affect the entire narrative as a whole. For example, killing a certain character important to the plot will make it impossible for him to appear in the story. Accordingly, this is a threat to the consistency of the narrative and requires a replanning (restructuring) of the narrative, taking into account this fact, right at runtime [20]. Another significant problem is the need to provide a sufficient level of variability, since the repetition of the same actions causes frustration for the players [21]. Another reason why commercial games do not use any complex form of quest generation, much less narrative, such as planning, is that it reduces the developer's authorial control over the system. This increases the likelihood that the generated quests will violate the overall author's intent or will have a meaning conflict with each other [22]. That is, due to the imperfection of the generation mechanisms and the reduction of author's control, the consistency and believability of the narrative are reduced.

Thesis goals

In this thesis, we will focus on the challenges developers face when generating interactive narratives and quests that we mentioned above, such as handling dynamic interactive environments, maintaining narrative coherence, and believable agent behavior. We will also pay attention to ways of introducing the author's (designer's) intentions into the system and methods of maintaining the narrative in accordance with them, so that developers can influence the algorithm by adjusting its parameters. To do this, we will formulate our concept of how this can be achieved and implement it in the form of a prototype of a software tool that allows us to procedurally generate a consistent and believable narrative which adapts to the actions of the player in an interactive environment, taking into account the parameters set by the author of the generated story.

We believe that such a system could be used as a tool for quest designers to help them calculate all possible options for the development of the narrative given the parameters and interaction factor with the player having the freedom to choose from at least several options in their actions. This will allow to fix or cut some of these options that previously remained unaccounted for, due to the limited development time and human abilities to predict non-linear sequences of events with many factors influencing them. Thus, this system will help them save time and resources either directly on development or on fixing errors caused by collisions in the narrative and player behavior that will appear in the future. Also, it could be used as a direct tool for creating the basis of quests (which, however, will need further processing by linguistic algorithms), and transferred them in finished form directly into a development tool, for example, into a platform for writing interactive literature, text games and visual novels - Twine [23], which will also reduce development time and costs.

We will also describe the algorithm underlying the developed prototype, which uses the principles of a multi-agent system with centralized regulation and the CSP methodology.

However, since the problem under consideration is quite complex and extensive, we do not set ourselves the goal of creating a solution that is immediately ready for commercial use. Instead, we will formulate a fairly general algorithm that has certain shortcomings, which we will implement in a software prototype (and therefore we will restrict ourselves to the area of dramatic adventure and detective stories) in order to test the prospects of our method as a whole.

Thesis structure

This thesis is organized as follows. It is structurally divided into three parts: theoretical, development and implementation. The theoretical part includes Chapter 1, in which we describe key terms and concepts, the context of our work, and Chapter 2, in which we present related works. The development part includes Chapter 3, in which is present the proposed method for generating narrative, and Chapter 4 showing use cases for our system. The third part, that describes the implementation, is presented in Chapter 5, which describes the details of our specific implementation of the tool for procedurally generating consistent narratives. At the end, we present conclusions and indicate the area of future work.

1. Background

In this chapter, we will introduce the terms and key concepts that we will use and refer to in the following. We will try to provide the reader with the amount of information that will be sufficient to ensure their understanding, but we will not conduct a complete analysis, since this work does not pretend to be an introductory material for a reader who is completely unfamiliar with them. Thus, to begin with, we will acquaint the reader with such a concept as Artificial Intelligence, consider certain areas of its application - planning and scheduling, multi-agent systems, procedural content generation. We will also stay in more detail on explaining what interactive storytelling and narrative generation are, and tell about their most important features. In the end, we will touch on the topic of video games, covering such concepts as game design and quests.

1.1 Artificial intelligence

It is not easy to give a precise definition for Artificial Intelligence, due to the fact that there is no unambiguous definition of human intelligence. The consequence of this is that there are many definitions of Artificial Intelligence, each of which is more or less vague. We will give a rather fresh, but already generally accepted, definition given by Stuart Russell and Peter Norving. They define Artificial Intelligence as "intelligent agents that receive information from the environment and take actions that affect that environment". This definition connects several different areas of Artificial Intelligence, holding them together through a machine capable of achieving a given goal, perceiving and influencing the environment [24]. It is especially convenient for us, since we are developing a multi-agent system and this type of Artificial Intelligence is closely related to the concept of “intelligent agents”. In addition to the concept of multi-agent systems, the area of artificial intelligence known as planning and scheduling is also important to us. We will describe them below.

1.2 Multi-agent system

A multi-agent system is a computerized system that consists of several interacting computational elements called intelligent agents. Agents have two important capabilities. First, they are capable of autonomous action, independently deciding which actions to take to achieve the goals for which they are programmed. Secondly, they are able to interact with each other. This interaction is broader than a simple exchange of data, and rather resembles the social activities of people, which they were inspired: cooperation and coordination [25]. Moreover, agents can differ markedly in their structure and complexity. For example, the simplest type of intelligent agents are reactive agents that simply perceive the environment and, based on their perception, act in the condition-action paradigm, i.e. react to external stimuli. More complex agents are capable of self-learning or have a set of personal characteristics and desires that influence their actions. Also, an important part of MAS is that agents act in some environment (software or physical), which they are able to perceive and which they are able to influence (change it).

An example of a scheme of the simplest intelligent agent (without a self-learning module) is shown in Figure 1:

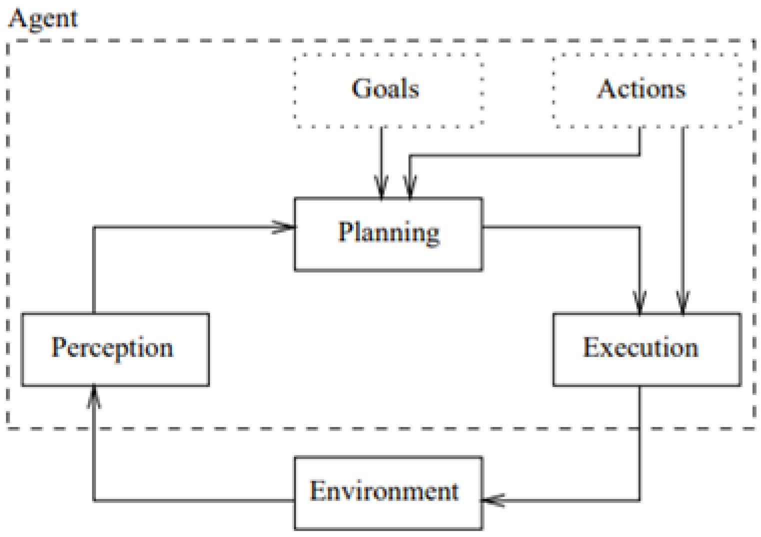


Fig. 1 Scheme of the simplest intelligent agent.

Our interest in MAS in the context of interactive storytelling and narrative generation is because they are successfully used to model social systems, demonstrating social behavior, due to the fact that they can be quite fine-tuned [26]. This quality makes it easy to relate agents to characters within a narrative, assigning them specific roles to demonstrate believable and rational behavior.

1.3 Automated planning and scheduling

Automatic planning and scheduling is an area of artificial intelligence tasks that determines the actions necessary to achieve the goal (planning) and group them into an ordered sequence (scheduling) for subsequent execution. Used in areas such as intelligent agents and autonomous robots.

Each specific problem, for the solution of which it is necessary to synthesize a plan, is called a planning problem. It consists in choosing a sequence of actions that, step by step, transforms the initial state of the system so that it corresponds to the goal state. The state is considered as a specific configuration of the system (a set of atomic facts, state variables). Actions performed by agents change state variables, which ensure the transition of the system from one state to another. The set of all states is called the state space. Structurally (and graphically), it forms a graph in which two states are connected by an edge if there is an action that can be performed to transform the first state into the second.

The term planning domain refers to the set of actions and state variables specific to a given planning problem. The planning problem itself is a broader concept than the domain that is its component, including also the initial state and the list of goal states.

Formally, we can formulate the planning problem as a triple:

$$(\Sigma, s0, g)$$

where Σ is the planning domain describing states and actions (transition between states), $s0$ is the initial state, and g characterizes the goal states.

Almost all planning algorithms are based on search and differ only in what space is explored (state space, plan space) and how exactly they do it (forward search, backward search, domain dependent) [27].

A serious complication with automatic planning is that the state space can grow extremely rapidly, even exponentially, along with the number of state variables. Because of this, it is subject to the curse of dimensionality and combinatorial explosion, which lead to the fact that the number of states can reach hundreds of millions and their exploration by brute force becomes impossible. To alleviate this problem, various heuristics are used to select the more relevant action and cut the search space to reduce the number of states to be explored to a more manageable number [28]. This problem is especially relevant when generating a narrative, since complex stories imply the presence of a large domain (in this case, the number of state variables is of primary importance) and, accordingly, the state space in them is astronomically large. But even in a simple domain, such as the one presented in S. Ware et al. [29], which contains only 4 locations, 9 agent beliefs, and 7 actions, the number of states was 300 million, and the number of edges was a billion.

The most popular languages for representing planning problems are STRIPS and PDDL, which we consider below.

1.3.1 Stanford Research Institute Problem Solver

STRIPS (STanford Research Institute Problem Solver) was the first system that was created directly for solving planning problems [30]. Subsequently, the name STRIPS was also used to refer to the formal language describing the input data of this system. This language gained popularity and became the basis for most modern languages for describing automatic planning problems.

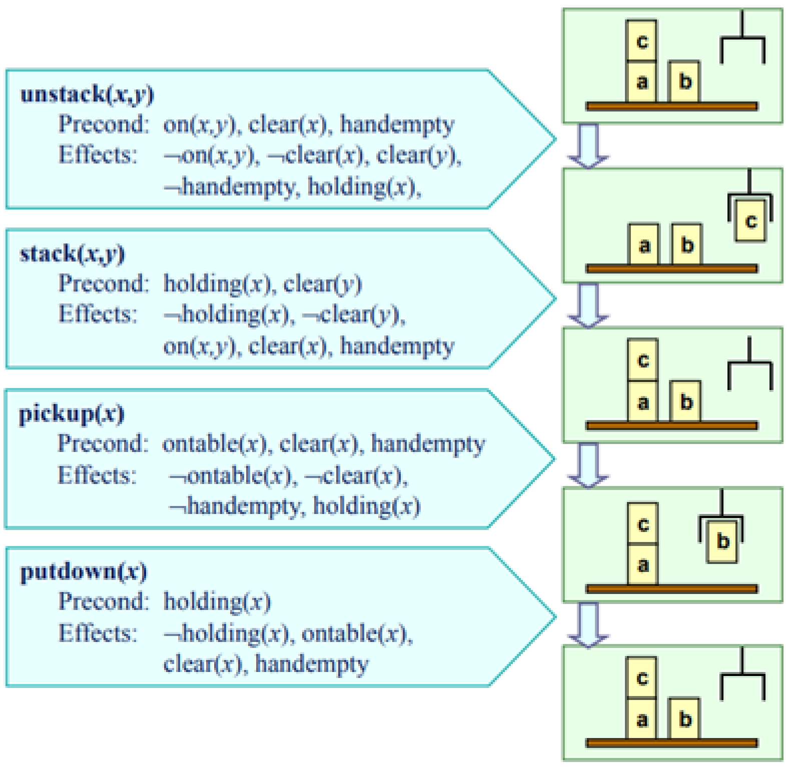
STRIPS offered a representation of the planning problem, which now is considered a classic. In it, states are a set of instantiated atoms, and the operator is a triple (name, precondition, effect), in which precondition and effect are sets of literals. Action itself is an instance of the operator:

$$precondition_{+} \subseteq s \land precondition_{-} \cap s = \emptyset \rightarrow (s - effects_{-}) \cup effects_{+}$$

That is, actions are grouped into parameterized templates called action (operator) schemas. For example, the "`stack`" action, which puts one cube on top of another in one of the classic problems. There may be many separate actions for each pair of cubes. But it is more convenient to group all such actions within the framework of one parameterized scheme "`stack(x,y)`" where the parameter `x` - denotes the cube that will be on top, and `y` - the cube that will be on the bottom. Specific actions are obtained by substituting specific constants for parameters. Only specific schemes of actions can be applied, i.e. the actions themselves.

The description of each action scheme consists of two main parts: a description of the preconditions which satisfy the action is applicable, and a description of the effect of the action. Preconditions are specified using a single formula. In order to find out whether the action is applicable in some considered state, it is necessary to check the truth of the precondition, i.e. prove that the precondition corresponds to the set of axioms of the considered state. The effect of an action is defined through two lists: a list of formulas that are added to the state, and a list of formulas that are removed from the state because they become false as a result of applying the action [27].

An example of action schemes in the STRIPS formalism is shown in Figure 2:



unstack(x,y)
Precond: on(x,y), clear(x), handempty
Effects: ¬on(x,y), ¬clear(x), clear(y), ¬handempty, holding(x),

stack(x,y)
Precond: holding(x), clear(y)
Effects: ¬holding(x), ¬clear(y), on(x,y), clear(x), handempty

pickup(x)
Precond: ontable(x), clear(x), handempty
Effects: ¬ontable(x), ¬clear(x), ¬handempty, holding(x)

putdown(x)
Precond: holding(x)
Effects: ¬holding(x), ontable(x), clear(x), handempty

Fig. 2 An example of a planning problem domain in the classical representation, where the constants are the blocks a,b,c, the predicates ontable(x), on(x,y), clear(x), holding(x), handempty, and the action schemes are visible on the image itself. Source: Roman Barták lectures slides, Planning and Scheduling, State space planning (forward, backward, lifting, STRIPS).

1.3.2 Planning Domain Definition Language

Planning Domain Definition Language (PDDL) is one of many languages based on the STRIPS notation. Its distinguishing feature is that it was created as an attempt to standardize languages for describing automatic planning problems. This was originally done to enable the International Planning Competition. This language turned out to be a good decision, as a result of which the community began to develop and improve it with each new competition, several separate modifications were created. Thanks to constant development, this language remains modern and relevant. The advantage of standardization provided by PDDL is that researches can be reused and easily compared, although this comes at the cost of losing some expressive power compared to domain-specific systems and languages [31].

Planning problems described using PDDL are divided into two files:

- Domain: for types, predicates, functions and actions.
- Problem: for objects, initial state and goal description.

Since this language is based on the STRIPS formalism, the descriptions created with it, although they have some specific features, in general look very similar to STRIPS.

Most modern planners use this language to represent planning problems, which makes it the most likely candidate for use in generating a narrative using multi-agent systems, where an important component will be the planning by agents of their actions.

1.3.3 Constraint satisfaction problem

Constraint satisfaction is a technology for describing and solving combinatorial optimization problems. Accordingly, the constraint satisfaction problem consists of:

- Finite set of variables.
- Domains - finite sets of values for each variable.
- Finite set of constraints, each of which is an arbitrary relation over a set of variables and can be defined extensionally (set of compatible tuples) or intentionally (formula).

Typical examples are: Sudoku (a type of puzzle with substitution of numbers according to a constraint), coloring a graph according to given constraints, or the N-Queens problem.

The CSP solution is a full variable assignment that satisfies all constraints.

Usage of the CSP formalism is also possible for solving planning problems (for example, CSP techniques are used in the Graphplan algorithm). The improvement of standard algorithms for use with CSP techniques leads to an increase in the efficiency of planning. The problem is that CSP uses static encoding, but the planning problem is dynamic because we do not know the length of the plan beforehand. The solution to this problem was found in looking for plans of a certain fixed length, and in case of failure, increasing this length. The planning problem can be recoded into a constraint satisfaction problem using state variables that will describe the properties of an object depending on the state as a function and which can be reduced to CSP variables with a domain corresponding to a range of values. The initial and goal states can then be encoded in the form of unary limiting conditions, and actions can be encoded in the form of conditions linking action variables to "adjacent" state variables. The constraint satisfaction technique is also used to extract the plan from the planning graph. The cost of increased efficiency is the exponential increase in problem size due to CSP domains, while maintaining the same number of variables [32].

In addition, CSP can be used to test the consistency of the finished graph. Which can also be used in the field of narrative generation, since maintaining the consistency of the narrative is one of the most important goals.

1.4 Procedural content generation

As we mentioned above, procedural content generation is an algorithmic process of creating content, both with and without human participation. The content created in this way can be of any type: images, sounds, texts, 3D models, game rules and mechanics, quests, etc [33]. This process can be either completely random or deterministic, for example, when creating an image, the color of each individual pixel will depend on its coordinates [34].

The benefits of using PCG are that this method allows to speed up the content production process, compared to completely manual production. At the same time, it allows to develop it in large quantities (billions of star systems in Elite Dangerous [7] or No Man's Sky [8] would be almost impossible to create manually) and at the same time provide a sufficient level of content diversity [4, 13].

In the case when content is generated with the participation of a person - the author or content designer, then this is called a "mixed initiative". In this case, the computer and the human designer work together to create content, but the ratio of their participation may be different. For example, a computer can only offer some options to a human, while he makes the final decision

[35]. On the other hand, a content designer may only be able to set a few own wishes, and the computer will do the rest of the work [36].

The main disadvantage of PCG is that the generated content may seem noticeably more artificial to players compared to human-generated content, which will frustrate them. We mentioned this effect above when talking about the Radiant Story system [19]. Usage of mixed initiative helps to cope with this disadvantage. In the context of procedural narrative generation, this problem becomes completely unacceptable, and tools for setting and regulating the author's intention are used to solve it.

1.5 Interactive storytelling

Here, before continuing, we would like to note that there are two terms that are often used synonymously and interchangeably - storytelling and narrative (the situation is similar with the terms story and narrative). We will define them to avoid further confusion. Storytelling theorist Olaf Bryan Wielk writes that story is the architecture of the elements of a story as a whole: events, characters, and their relationships. A narrative is a sequence of story elements presented in a specific order. Different narratives of the same story can be built by reordering the elements of the story [37]. Also, Professor J. Martin from Media Design School Dusseldorf writes that story - **what** is told. It is more creative, focused on the creative techniques and emotions of the user. It’s the content. Narrative - **how** it is told. Narrative is more about the technical side, the way the story is organized and represented. It’s a form [38]. Based on these statements, we will define storytelling as the creative aspect of creating a story. For example, in the context of storytelling, interactivity is understood as an artistic method. We will define narrative as the technical aspect of creating a story. For example, interactivity in its context would be understood as a technical feature of the way a story is told. As is evident, these are interrelated concepts, and one cannot exist without the other, the only difference is what is emphasized when using a particular term.

We give a diagram of the components of the stories in Figure 3:

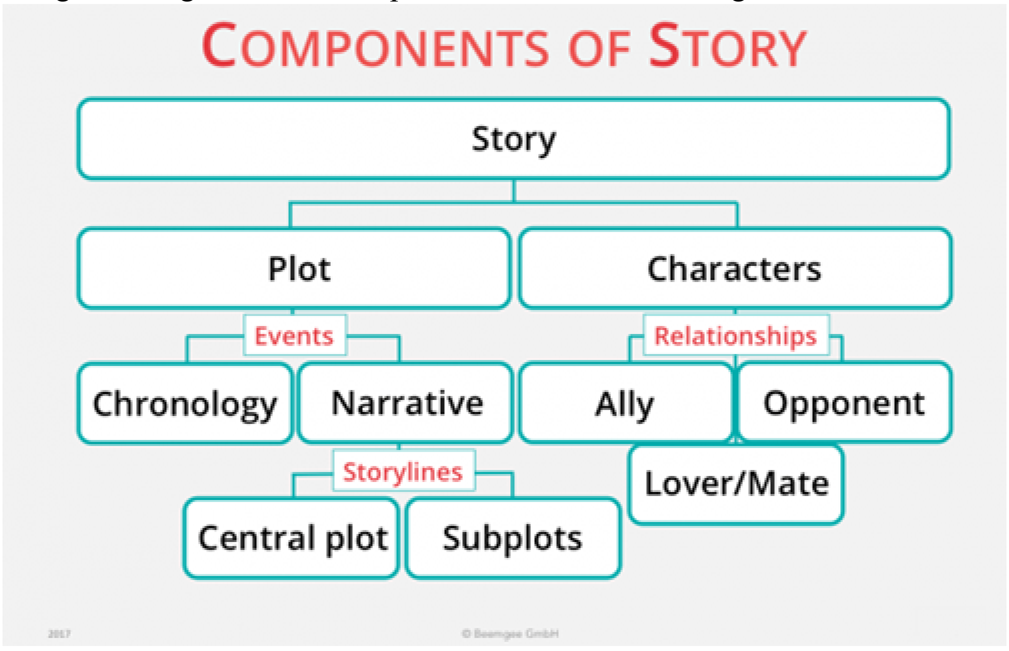


Fig. 3 Diagram of story components, showing that narrative is an integral part of a story, and from which it can be concluded that storytelling is the art of creating a story as a whole. Source: Story vs. Narrative [37].

Interactive storytelling is a story format in which the storyline is not firmly defined in advance. The author creates the basic structure of the story (setting), characters, story world (environment), the initial state of the story world, but each user experiences a unique experience based personally on their actions and interaction with the story. The architecture of interactive storytelling systems includes the elements necessary to manage the narrative (drama manager), the narrative experience (user model), and the behavior of the characters (agent model) [39]. Together, these subsystems generate believable characters (who act "like people"), change the story world in runtime and respond to user actions, and ensure the consistency of narrative discourse (i.e. events will unfold in a clear and consistent way).

The success of an interactive story depends on balancing the dramatic structure and providing a sufficient degree of user interaction. Also, the narrative experience must be markedly different depending on the choices made by the user when interacting with the story world, for which he must have some degree of freedom of action [40]. This can be achieved with varying degrees of success through branching, emergent, character-centric, and story-centric systems.

Several attempts have been made to formalize an evaluation system for interactive dramas (although all projects currently in existence are academic or at an experimental stage). Most of them end up using Likert scales to rate parameters such as player agency and fun, sometimes replaced by more specific "interestingness", "surprise", "degree of empathy for the characters", and others, depending on the purpose of the study [40]. A number of works use specific fitness functions, for example, to calculate the “novelty” of a story, or vice versa, to calculate a story’s adherence to classical structures [41]. However, the use of these methods provides only a rough quantification of user experience, leaving out much of the subjective interpretation that underlies human perception and interaction. More promising is the proposal by Mark O. Riedl and R. Michael Young to evaluate the degree to which users understand the goals of the characters in the story and their motivation [42].

1.5.1 Dramatic structure

An interactive story to be successful must provide the user with a narrative experience, which will be dramatically interesting to him. Dramatic structure plays a significant role in ensuring the dramatic interest of experience. Many theorists showed interest in the disclosure and synthesis of the dramatic structure from ancient times, when Aristotle himself was engaged in this, to the present day [40].

A three-act dramatic structure is considered classic, thanks to which conflicts arise logically, develop, culminate and resolve.

A classic three-act dramatic structure is shown in Figure 4:

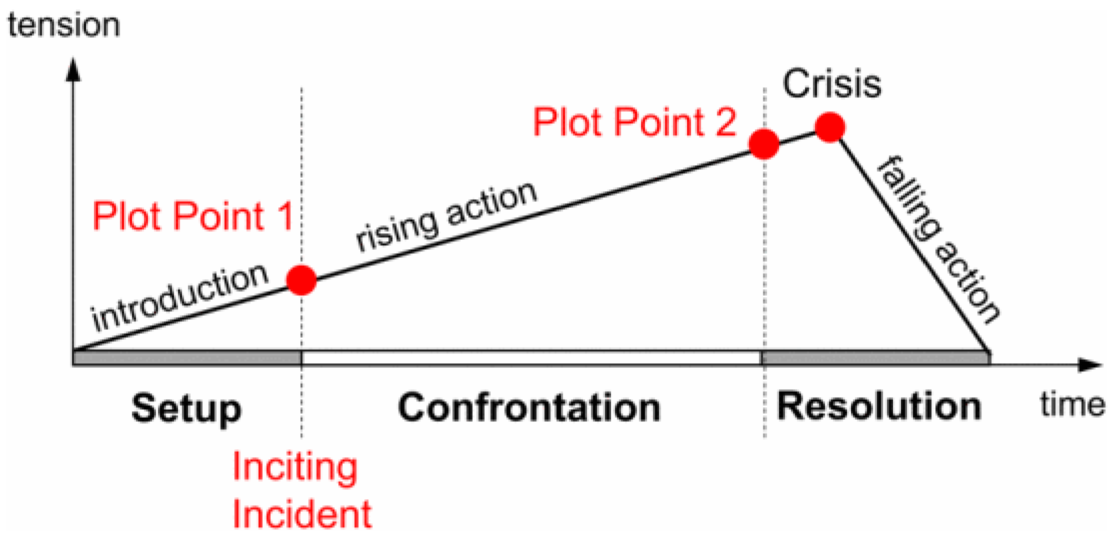


Fig. 4 Story dramatic structure as a three-act structure. Source: "Procedural generation of branching quests for games" [87].

The five-act structure proposed by Freytag, and known as the "Freytag pyramid", is also quite well known. It is believed that it outlines the main ups and downs that are usually found in an interesting drama.

The five-act dramatic structure is shown in figure 5:

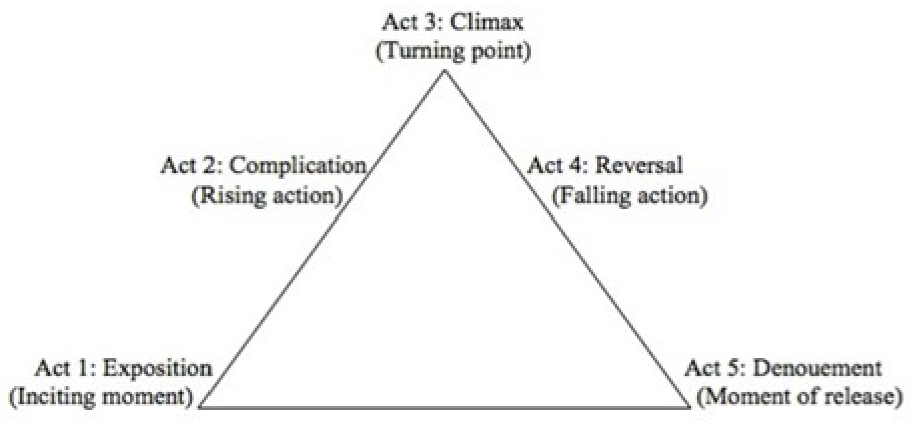


Fig. 5 Dramatic structure known as the "Pyramid of Freytag". Source: "Celebrity Bowl: More Marketers Than Ever Turn To Celebrity For Super Bowl Ads, But That's Not The Whole Story".

An interesting drama can arise without following one of the forms of the classical structure, for example, there may be no completion. However, the closer the drama follows the "dramatic arc", the more dramatic it seems to be. In one form or another, these dramatic structures have been used in many previous studies of interactive drama. For example, the Oz Project [43] and IDtension [44] require generated narratives to follow a "dramatic arc". The Facade uses a structure that the authors call neo-Aristotelian (it is a theory of interactive drama based on Aristotle's dramatic theory and its interpretation in the context of interactivity proposed by

Laurel, but modified to address player-added interactivity, in which the user's roles and constraints can be represented as a character in a drama) [45, 46].

Esslin argues that any drama must capture the attention and keep the audience engaged while being consistently entertaining [40].

1.5.2 Believable Agents

The agent model takes in information about the story world and generates possible actions for each character in the story, and it also handles the interaction of the characters with the user. Due to such property as autonomy, agents are able to demonstrate believable behavior. By this, we mean the perception of the agent's actions as if they were motivated by his inner beliefs, desires and emotions, that he acts intentionally [39, 47].

1.5.3 Consistency and coherence of narrative

The next two important properties required for a successful interactive narrative are coherence and consistency. Coherence expresses causal relationships between events and actions, forms a “continuous whole” from the narrative, and provides actions with a logical necessity. Consistency, on the other hand, expresses the absence of internal inconsistencies in the narrative and behavior of the characters, for example, the detection and elimination of inconsistencies caused by user actions that can annul the plot [48, 49].

1.5.4 Narrative experience

The interactive drama offers the user a story world in which he can exert real influence on the narrative he is experiencing. All of the above factors, such as dramatic structure, believable behavior of agents, coherence and consistency of the narrative, and, most importantly, the possibility of many different interactions with the story world and influence on it, are important components that make up an interesting narrative user experience. An interactive story that provides a continuous, realistic way of interacting leads to a more immersive experience. Since the very purpose of the existence of interactive stories is to allow the user to experience some kind of experience - dramatic, educational, communication or training, then ensuring its quality, fascination, interest and realism is the most important task in creating an interactive story, as well as the main criterion for evaluating its success. Computing systems that can reason about the narrative, manipulating it to provide a more immersive experience, are more efficient [42].

1.5.5 Story-Centric and Character-Centric Designs

There are two opposing approaches to the design of interactive dramas, driven by the desire to alleviate the efforts of authors that result from the fusion of interactivity and narrative. Authorship support is needed because, unlike traditional drama where the user is presented with only one storyline, interactive drama has the potential to have more storylines due to the user's ability to interact with the characters and the story world. Creating enough contingencies to create a rich interactive environment and provide a compelling experience is often beyond human capabilities or requires an unreasonable amount of time. One of these approaches focuses on the plot, the other focuses on the characters.

Both of these approaches are based on fairly old concepts. So, even Aristotle in his "Poetics" argued that the characters are secondary in relation to the action. Another, more modern, view was expressed by the dramaturgist Lajos Egri in 1949, suggesting that the plot unfolds based on the actions of the characters and that they can, in fact, "build their own story."

In line with the above concepts, story-driven processes for interactive drama focus on the overall structure of a story in terms of a story arc and seek to automate approaches to organizing events in such a way as to create a tightly structured story in which agents will act in the interests of following the story's development plan. They often use POP planners because they can automatically generate sequences of character actions (plans) to achieve story goals, and at the same time provide a plausible causal relationship between these actions. However, such plans give neither the author nor the user insight into the motivations of the characters and therefore cannot avoid creating action sequences that are perceived as contradictory in means of the characters' intended motivations. On the other hand, character-driven processes emphasize the development of individually believable and autonomous characters that the user can interact with and create a narrative through user-character interaction, without rigorously following a rigidly structured plan. For example, this approach is used by the Thespian system, which can ensure that characters are constantly motivated during interaction. It uses purposeful agents based on decision theory to control virtual characters. The character's motives are coded as the agent's goals. It provides an automated fitting system that provides the ability to adjust the motivations of the characters according to the intended development of the plot. The characters thus created will maintain their roles as long as the user's actions align with the expected development of the plot. But when the user deviates from it, the characters will react to his actions in accordance with their internal motives, trying to adapt to the changes. However, deviating from the author's intended plot paths risks turning the interaction into a poorly structured or inconsistent story. To prevent this, the author has to work out several potential ways of developing the story to adjust the beliefs and motivation of the characters in accordance with them.

Attempts are also being made to combine both approaches, for example in the paper "Integrating Story-Centric and Character-Centric Processes for Authoring Interactive Drama" by Mei Si, Stacy Marsella and Mark Riedl [50].

The diagram, represent correlation of the coherence of the plot and the believability of the characters is shown in figure 6:

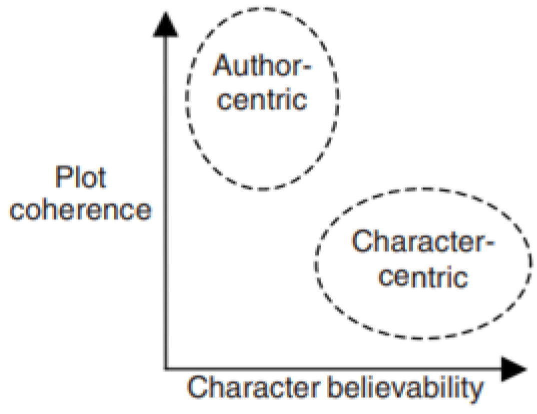


Fig. 6 Influence of story-centric and character-centric design on story properties. Source: "Character-Focused Narrative Generation for Execution in Virtual Worlds" [74].

1.5.6 Emergent narrative

With complete autonomy of agents, the user experience is completely determined by the uncoordinated (or weakly coordinated) decisions of the characters with the drama manager and user's own actions. This kind of interactive narrative is called emergent (i.e. spontaneous) narrative. The idea of this is that it is not always necessary to intervene to guide the user's narrative experience towards a particular conclusion. It may be sufficient to create believable agents with detailed motivations and personalities, and then the random causal sequence of their interactions with each other and the user will be perceived as a narrative. In such narratives, events occur as a result of the natural development of the simulation, and are not created by the author or explicitly planned in a way generated. Emergent systems are unique in that the author usually has little influence on the plot and instead focuses on creating the initial state of the story world. Examples of the use of emergent narrative are various simulation games (RimWorld [51], Minecraft [10], the Crusader Kings series [52]) and learning environments.

Emergent systems, however, can not only fully automate narrative and the story world, but can also include some form of narrative control to set some direction for the story's development [14].

1.5.7 Authorial Intent and Authorial Liberty

Another important topic that arises in the context of interactive narratives is author's intent and author's freedom.

Authorial intent is the ability of an autonomous interactive system to express the intentions of a human designer (author) when generating a narrative. However, a generative approach to interactive narrative, in which an automated system takes on part of the author's responsibility, moves the human designer away from directly shaping the user experience, because, as a rule, the human author is not present at runtime and cannot make decisions about how the user experience should be adapted at a given point in time. Thus, the creation of interactive narratives is often a process of predicting user actions under certain conditions and using computing systems and structures to form a response to them. That is, the creation of interactive narrative content in essence is endowing an automated computing system with the ability to make the same decisions that a human designer would make in response to user actions. And the goal of human designers is to embed their artistic vision and their author's intent into a computing system. This situation leads to the need to develop methods that would allow authors and designers to express their intention, through indirect influence, but more subtle and complex adjustment of the drama manager. For example, Mark Riedl suggests using intermediate states when creating a plan, the so-called "islands", which must be present somewhere in the plan, and not just determine the initial and final states of the story world. This will allow the formation of more complex narratives and will allow expressing the author's intentions by coding it in the form of "author's goals" - some states of the story world through which it must pass and which are determined by the author at the design stage. As an example of the complexification of narrative plans, he gives the following example: according to the author's intention, the narrative should develop in such a way that a character who starts out rich becomes poor, and then rich again. Such a phenomenon is quite a challenge for planners. That is, if they are not given any special instructions, but simply given an initial state in which the character is rich and an end state in which the character is also rich, the planner will simply indicate that there is no problem to solve. But it is possible to use "author goals" and have the planner embed the state in which the character is poor into the resulting narrative structure [53].

With the question of incorporating the author's intent into the generation of an interactive narrative, the question arises of how much freedom the author can be given to do so. T. Pedersen et al. [54] propose to define the degree of authorial freedom as a continuum that ranges from highly deterministic possibilities (Drama Manager) to complete creative freedom (Game Master). They interpret the two poles of this continuum as two extremes (lack of freedom versus complete freedom) in terms of what and how much the author can change. Depending on the author's or designer's intentions, different ends of the spectrum may be more or less suitable for his purposes. For example, if the author intends to convey a specific message, then a position closer to the Drama Manager (so as not to distort this message) would be more appropriate on the author's freedom continuum. If the goal is to convey some kind of experience of exploration or interaction, for example, during education or training, then the user will have to rely more on their own perception of the narrative. Thus, the distance between the author and the audience will be greater, which means that there will be a gap in interpretation between what the author wanted to say and the narrative that the audience perceived. This can give more freedom to adapt the narrative without breaking its consistency and coherence, which makes a position on the continuum closer to the Game Master more appropriate. T. Pedersen et al. consider this in the context of the fact that the human author has the ability to influence the narrative at runtime, but we extrapolate this also to the most common cases where this is not that. With the clarification that the author's freedom manifests itself during the setting up of the system (Drama Manager), and determining the boundaries of its ability to influence the adaptation of the narrative to the user's actions.

The author's freedom continuum is shown in figure 7:

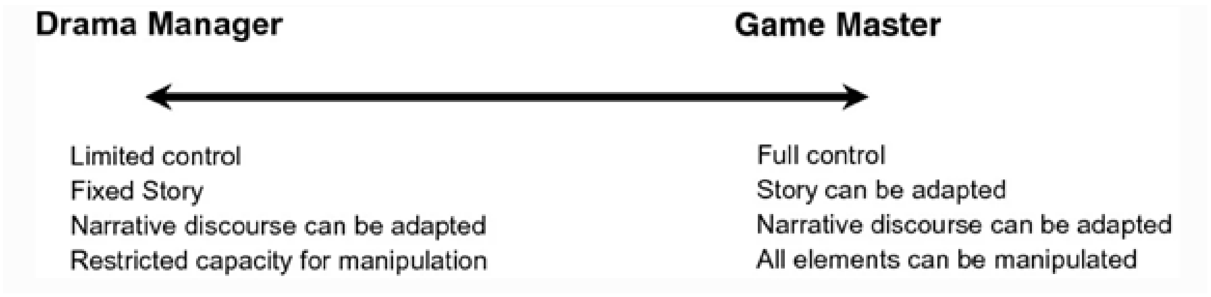

Drama Manager
Game Master
Limited control
Fixed Story
Narrative discourse can be adapted
Restricted capacity for manipulation
Full control
Story can be adapted
Narrative discourse can be adapted
All elements can be manipulated

Fig. 7 Author's freedom continuum. Source: "Considering Authorial Liberty in Adaptive Interactive Narratives" [54].

1.5.7.1 Drama manager

The Drama Manager is one of the components of the interactive narrative and is responsible for managing the development of the narrative by monitoring the flow of events in the story world, their coherence and consistency, according to the goals of the author, and, if necessary, collaborating with the agent model to control or correct the actions of the characters. It is important to note that in such a paradigm, communication between the drama manager and agents occurs relatively infrequently, allowing them to act autonomously for the most part.

Drama managers cannot change any of the rules in the story world, and their ability to adapt the narrative is limited to pre-prepared scenarios. Accordingly, T. Pedersen et al. consider the manager of drama as one of the extremes of the continuum of authorial freedom, in the direction of limiting freedom. By placing the author's freedom within these limits, he should only be allowed to change the story's narrative discourse, but not its rules or setting [54].

1.5.7.2 Game master

At the other end of the author's freedom continuum, they place the Game Master, defining the set of tools and functions available to him as similar to those available to the game master of tabletop RPGs like Dungeons and Dragons. Tychsen et al. [55] divide the functions of the game master toolkit into five groups: narrative flow, rules, engagement, environment, and virtual world. Collectively, these groups represent complete control over the narrative in terms of narrative rhythms, as well as the rules of the story world, even how exactly the player is involved in the interaction. By placing the freedom of the author within these limits, he must be given complete control over the narrative and the opportunity to influence all its aspects (both discourse and story as a whole) [54].

1.6 Games

One of the applications of interactive narratives is video games. They serve for entertainment and often use drama as their basis. Moreover, this can apply to games of any genre - strategies, shooters, RPGs, etc. However, often the interactivity of the narrative in them comes down to choosing one of several alternatives designed by the authors in strictly defined places, and, as a rule, this choice does not significantly change the story world. And the changes that take place in the story world are also planned in detail by the authors. The only exceptions are games that implement emergent gameplay, and which, in general, cope with the generation of emergent narrative (for example, the aforementioned RimWorld [51], Minecraft [10], the Crusader Kings series [52], also Dwarf Fortress [56]). However, their big problem is that they do a poor job of maintaining coherence and consistency. Also, they are often inferior in expressive power and ability to evoke empathy for characters, stories created with a more active participation of the author and more fully expressing his intentions [38].

1.6.1 Quests (Adventure games)

Quests in literature are the essence of many stories - they are adventures, actions. According to the analysis of Propp's fairytales [57], any adventure has a clear beginning - committing a crime or understanding a problem, after which the hero leaves on a dangerous journey in order to solve the problem. As a rule, there is no single adventure, in itself, it exists in the context of the story world, and around it other situations and adventures are formed - both before it begins, in the preliminary phase, which can only lead to it, and those that will happen after. For example, we can recall Homer's Odyssey, when, it would seem, the main big adventure is over - the Trojan War is over, and the hero simply returns home. However, in the process of returning, he experiences a large number of new adventures before he reaches home. Moreover, even here a new adventure arises when he has to deal with enemies. As Greimas notes [58], all these adventures are equally structured trials for the hero and other characters, and the continuation of the story as a whole depends on the outcome of each of them. Each completed adventure can recursively generate a new one, as happens in many of the tales of the Thousand and One Nights. According to Lima et al. [59], if the outcome of the adventure plan is non-deterministic, then they acquire an even more complex hierarchical structure, which leads to variability.

Quests are present in many computer games, being both a subsystem in them (for example, in role-playing games - RPG), and an independent genre. They represent missions or goals that must be completed by the player. It is possible to say that they originated in 1975, when the text game Colossal Cave Adventure [60] appeared in the quest genre, becoming the ancestor of several genres of computer games at once, based on quests - computer role-playing games, roguelike, and MUD (Multi-User Dungeons) . They have come a long way, and modern RPGs are starting to incorporate some interactive storytelling techniques, but as we mentioned above, their narrative structures are still very simple.

Lima et al. [59] also claims that during the last fourteen years there have been many attempts to create a theory of quests in computer games, however, no work has yet completed this task, because the concept of storytelling in video games is not properly understood. They mention a lot of conflicting opinions about video game narrative. Thus, Aarseth [61] describes it as a "post-narrative discourse", Tosca [62] claims that games are not run-time narratives at all, and only after the completion of the quest can it be told as a story, Jenkins [63] defends a hybrid concept combining games and narratology.

Since this genre is focused on a story, with an emphasis on plot and characters, and the player's interaction with them, we consider quests to be the most suitable video game genre for integrating procedural narrative generation systems. Also, this genre relies more than others on the narrative means of literature and cinema, which also brings it closer to interactive storytelling, many of the meanings, concepts and methods of which come from literature [64]. Equally important, the creation of the quests itself does not require advanced graphics or physical environment simulation technologies, they can even be completely text-based, which technically simplifies development.

2. Related works

The first attempt to explore interactive storytelling was James Meehan's Tale-spin system in 1977 [65]. This system produced original tales with morals in a fantasy setting. The stories created by this system were purely textual and had a lot of inconsistencies. One of the features of this system was that it contained a large amount of background knowledge about the story world, which was created in the process of how the story was told. Agents were implemented in such a way that they were semi-autonomous, had goals, emotions, and relationships [40].

The next large milestone was the "Universe" system [66] developed by Michael Lebowitz in the 1980s, which created endless stories in a soap opera setting. In it, an author had to set goals for the storytelling system, and then she used those goals and pre-created plot fragments to create a soap opera plot summary. The system created characters that were dynamically assigned roles in these fragments. The relationships of the characters were central to the story. "Universe" used a planner to select the sequence of actions that the characters in the story world should perform. The planner in the "Universe" included in the narrative sequence only those actions that contributed to the achievement of systemic goals, although systemic goals could be described at a high level of abstraction, for example, "keep lovers apart" [40].

The first main interactive drama research group was “The Oz Project” in the 1990s, led by Joseph Bates [43, 67]. Research has focused mainly on creating believable agents. This system placed the user in a virtual environment inhabited by autonomous animated agents called woggles. Each of these agents had a set of goals and beliefs, and worked autonomously to achieve them. The user was able to give instructions to one of these characters by interacting with it. Also, the characters interacted with each other. The group also explored interactive narrative generation based on the plot graph structure. So, for example, to provide an interesting experience for a user whose interest was determined by a specially defined rating function, a special module (drama manager) - discretely manipulated the desires of agents to change the narrative and direct the user experience towards those storylines that were rated as more interesting. Interactive narrative was the focus of this project, but was seen as a sub-problem in narrative generation in general, so they used an algorithm that works similarly with and without the presence of an interactive user [40].

Also in the late 1990s, was created the “Defacto” system, which used a rule-based approach to storytelling [68]. It had a database that stored rules about the relationship of characters, their goals, social norms, intentions. These rules were encoded in a format that allowed the system to reason about the intentions and actions of the characters. The result of the system was a list of causally ordered actions, the result of which was assigned the status of "success" or "failure", to achieve an outcome that is considered satisfactory and disturbing. Stylistically, this was designed so that the user took on the role of a character in a story world in an Ancient Greek setting. The story was dynamically created in text form, and the user could specify his actions by interacting with it. After the interaction phase, the user was shown a graphically generated story. But until the graphical output was made, the result of his actions was not known to him. “Defacto's” specific focus on a particular story world and setting limits its applicability to other themes. Also, its design was such that it does not provide a unique experience with each following use, becoming predictable [40].

Façade was the first project in the field of interactive storytelling that really got a lot of attention, including among a wide audience [69, 70]. Michael Mateas and Andrew Stern created this system and released it in 2005. A year later, Façade won the Grand Jury Prize at the Slamdance Independent Games Festival. The plot of Façade was that the player character was invited to visit by his friends, a couple living together. As the story progresses, he dives into the couple's marital problems. Through text input, the player could communicate with these characters, and what he said to them influenced the story, including its outcome. The entire environment was designed graphically and took place in the first person view. Architecturally, the Façade system included such structural elements as: drama manager, beats, characters, plot variables, actions, and natural language processing. The authors called "beats" are short sequences of actions that occur throughout the story. They have been explicitly pre-created, with all actions in each beat being predefined, and actions for roles assigned to ensure coordination between agents. The order in which these beats occurred could be different, having their own prerequisites for occurrence and introducing new ones as an effect, for other beats. Essentially, the plot graph was divided into short chunks of actions that were performed based on the state of the story world, rather than in a strictly defined order. Higher-level logic of goals and behavior of characters, close to autonomy, was used inside the beats, to achieve local goals. Each session of the game lasted about half an hour, and its variability was enough for several (up to 5) replays that retained its uniqueness [40].

Also in the early 2000s, the Liquid Narrative Group showed significant activity in the research of interactive narratives. They created The Actor Conference, a storytelling system that used both author-focused and character-focused designs to balance the conflicting concepts of plot coherence and high character believability [71]. This system used a blackboard architecture to coordinate agents. This transformed the process of narrative generation into a search in space for hypotheses or partial plans. Then they upgrade this system by adding a 3D plan execution environment to it and calling it Mimesis [20, 72, 73]. Because Mimesis was designed as a common architecture, it is theoretically capable of running on any game engine. A distinctive feature of this system was also that it sought to provide the user with the illusion of complete freedom of action. In the case that the user performed an action that violated the current plan, the system had two ways to respond to this - "adapt" or "intervene". Adaptation was possible in those cases when the plan was not strongly violated and it was possible to replan, including this action in the plan for achieving the goal of the story. For example, if according to the plan, a certain character was supposed to hack the vault, but the user tried to do it instead. In such a case, as a result of the replanning, the story world could be changed in such a way that the user would find this vault already open. If the user's action violated the plan too much and adaptation was impossible, then the system intervened in the user's actions. For example, if he tried to kill a character, later important for the continuation of the story, without which it is impossible to achieve the goals of the story. This would critically violate the consistency of the narrative, then the system would consider that he could not do this, for example, he missed. Technically, this is implemented so that when Mimesis receives a request to create a plan, it creates a directed acyclic graph. This graph is a plot structure, within which it can be changed if the user violates its original structure. However, such changes are not enough to create a wide variety of unique narratives [40].

The members of this group, in particular Mark Riedl, have written many theoretical papers as part of their research into various aspects and problems of narrative generation. These articles make a great contribution to the extension of the theory about the generation of interactive narratives, being relevant to this day. They described the generation of character-based interactive narratives [74], intelligent methods for controlling narratives in interactive systems [20], managing the interaction between agents and the user of a multi-agent interactive system [72], intention-oriented planner in a multi-agent environment [48], autonomous agents and their believability in an interactive environment [47, 75], the integration of plot-centric and character-centric designs in one interactive environment [42, 50], the inclusion of author's intentions in a system that generates an interactive narrative [53].

We have not mentioned all the interactive storytelling systems that implemented some degree of narrative control created in those years. Because we tried to describe and refer to the experience of the most important and relevant works for us. But also noteworthy are such systems as: IDA [76], U-DIRECTOR [77], PaSSAGE [78], IN-TALE [47], NOLIST [79], GADIN [80], OPIATE [81], DED [82], IDtension [44, 83], BARDS [84], FAtiMA [85]. A complete and thorough analysis of these systems was made by M. Arinbjarnar, H. Barber, D. Kudenko in their work "A Critical Review of Interactive Drama Systems" [40].

Among modern research, we would like to highlight the works of Edirlei Soares de Lima, Bruno Feijó and Antonio Furtado. In "Hierarchical Generation of Dynamic and Nondeterministic Quests in Games" [59], they use hierarchical task decomposition and automatic planning to create non-deterministic quests with multiple alternative outcomes in order to achieve interactive dynamic narratives. In this paper, they write that events in interactive narratives and games have a conceptual similarity to the functions identified by Propp [57]. In this case, in their opinion, a promising approach to reconcile interactive narratives and games is to provide semantics for such events using planning models for this. Also, by analyzing Propp's functions, they came to the conclusion that these functions fit well with the STRIPS technique, where the preconditions for the execution of a certain function will first be fulfilled by the effects of other functions. In such a case, narrative plots could be seen as the result of backward planning. They write that an effective approach in developing interactive storytelling elements in quests is to develop dynamic non-deterministic planning models. Then, in the works "Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning" [86] and "Procedural generation of branching quests for games" [87], they describe and use procedural content generation techniques (in particular, combining genetic algorithms and automatic planning algorithms) to create interactive quests with branching storylines based on the generated narrative structure.

3. Narrative generation

In this chapter, we describe in detail the algorithm developed by us and the principles of operating the system based on it. We will start the description at a high level, describing the general architecture of the system, and its input and output. Then we will go into details, describing in more detail the individual elements of the algorithm, describing the mechanism for creating our specific multi-agent environment, agent and player models, the planning mechanism, and the mechanism for ensuring the consistency and coherence of the generated narrative.

3.1 System's input and output

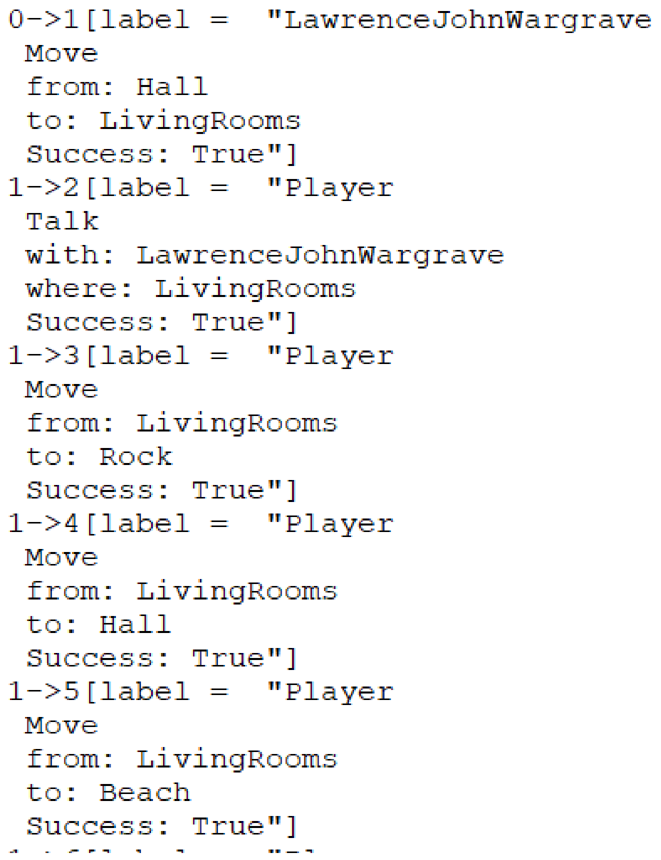
The input of the system is the intentions of the author, expressed as system settings (example of input interface in Figure 24). Based on these settings are determined the rules by which the system will work: the set of agents, their goals, the set of locations, the degree of randomization of the location parameters when they are created, the actions available and not available to agents, the degree of randomization of the result of actions, constraints, technical parameters depend on this (number of nodes, seed for randomization). It also affects the generation of PDDL files that are passed to the planner, i.e. on the behavior of agents.

The system output is a story graph. Based on which, the program generates two files. The first is a file in the DOT format that describes the graph in the language of the same name. This description contains information about the nodes, including their type (obtained in the usual way, or obtained as a result of the counter-action of the system, whether it is the goal node), information about the connections of the nodes to each other (oriented edges), as well as actions (including their internal parameters: which agent acted, where, which agent was the target, whether the action was successful or not). It also contains technical information about what kind of graph to build (directed graph). Based on this information, it is possible to visualize the graph using any program that works with the DOT format.

Excerpts of one of the files this format are shown as an example in Figures 8 and 9:

```
digraph G {
0 [shape ="circle" label =" 0"]
1 [shape ="circle" label =" 1"]
2 [shape ="circle" label =" 2"]
3 [shape ="circle" label =" 3"]
4 [shape ="circle" label =" 4"]
5 [shape ="circle" label =" 5"]
6 [shape ="circle" label =" 6"]
8 [shape ="circle" label =" 8"]
12 [shape ="circle" label =" 12" style = "filled" fillcolor =
"yellow"]
13 [shape ="circle" label =" 13"]
14 [shape ="circle" label =" 14" style = "filled" fillcolor =
"aquamarine"]
15 [shape ="circle" label =" 15" style = "filled" fillcolor =
"aquamarine"]
16 [shape ="circle" label =" 16" style = "filled" fillcolor =
"yellow"]
```

Fig. 8 An excerpt from a dot file.



```
0->1[label =  "LawrenceJohnWargrave
 Move
 from: Hall
 to: LivingRooms
 Success: True"]
1->2[label =  "Player
 Talk
 with: LawrenceJohnWargrave
 where: LivingRooms
 Success: True"]
1->3[label =  "Player
 Move
 from: LivingRooms
 to: Rock
 Success: True"]
1->4[label =  "Player
 Move
 from: LivingRooms
 to: Hall
 Success: True"]
1->5[label =  "Player
 Move
 from: LivingRooms
 to: Beach
 Success: True"]
```

Fig. 9 An excerpt from a dot file.

An example of visualization is shown below in Figures 19, 20, 21, 22.

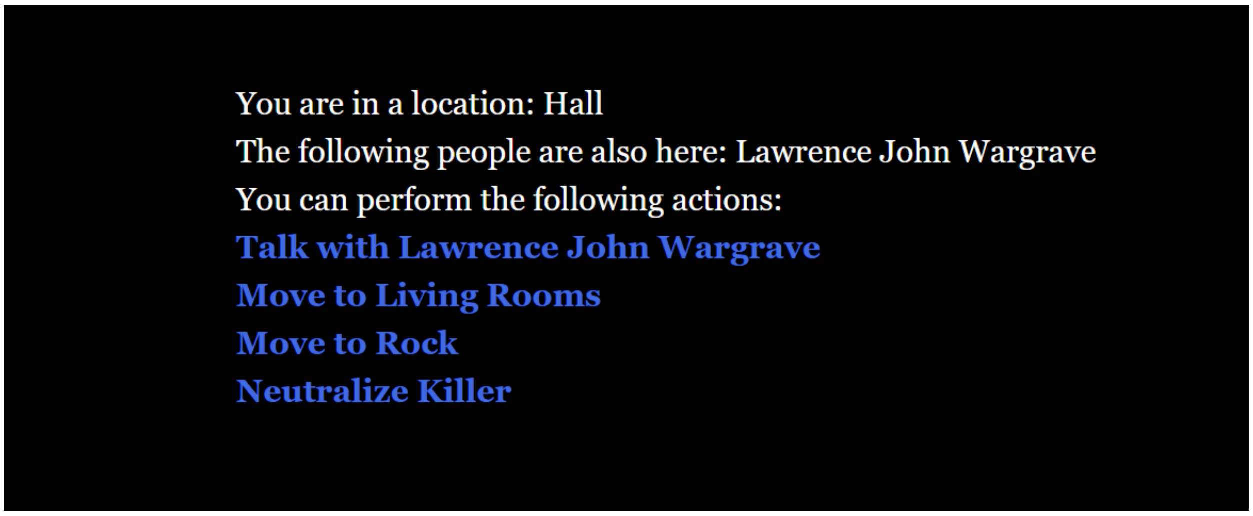
The second file generated by the system is an HTML file intended for the Twine platform. It also stores information about the story graph, but not about everything, only about those states in which the player has to act, and about goal states. The situation is similar with edges and the actions associated with them. For each node is stored information about it and assigned textual description (description the state of the world stored in it, and links with other nodes). When entering the key from the OpenAI API, these texts are generated using the DaVinci-003 model, which provides more colorful, human-like descriptions. Also, the text description is associated with some actions and states resulting from their performance, not contained in the graph initially, but generated in the process of converting to a format understood by Twine to provide the player more information (for example, to report the result of an action to find evidence). In addition to information about the graph, this file contains instructions for the Twine platform on exactly how to run and playback this story, up to pointing out the required or undesired interface elements and their associated features. This file can be played correctly only with Twine.

An excerpt from one of these files is shown in Figure 10:

```
<tw-storydata
name="DragonAge"
startnode="0"
>
<style role="stylesheet" id="twine-user-stylesheet"
type="text/twine-css">tw-icon[title="Undo"], tw-
icon[title="Redo"] { display: none; }</style>
<tw-passagedata pid="0" name="Node 0" tags="" position="-200,400"
size="200,200">
You are in a location: Lothering
The following people are also here:
You can perform the following actions:
[[Move to   Mages   Tower -&gt;Node 7]]
[[Move to   Orzammar -&gt;Node 9]]
[[Move to   Brecilian   Forest -&gt;Node 11]] </tw-passagedata>
```

Fig. 10 An excerpt from an HTML (Twine) file.

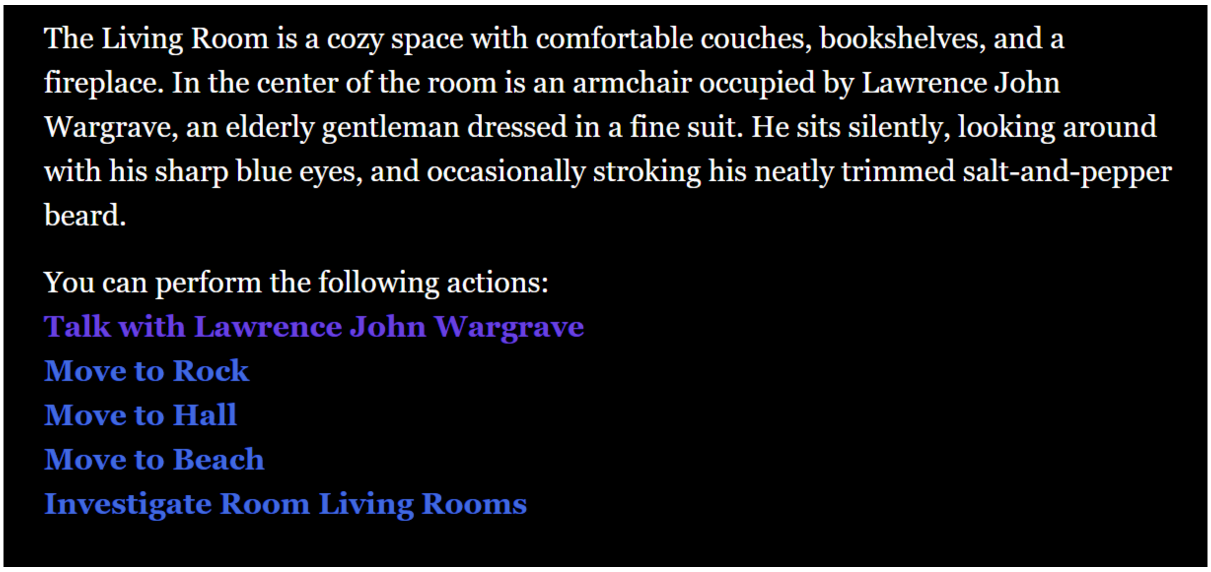
Figure 11 shows the playback of an interactive story obtained from the generated narrative using Twine:



You are in a location: Hall
The following people are also here: Lawrence John Wargrave
You can perform the following actions:
Talk with Lawrence John Wargrave
Move to Living Rooms
Move to Rock
Neutralize Killer

Fig. 11 An interactive story playback.

Figure 12 also shows an interactive story playback in Twine, but using the DaVinci-003 text generation model from OpenAI:

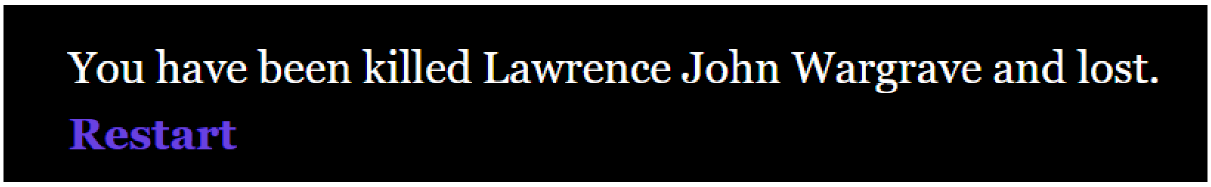


The Living Room is a cozy space with comfortable couches, bookshelves, and a fireplace. In the center of the room is an armchair occupied by Lawrence John Wargrave, an elderly gentleman dressed in a fine suit. He sits silently, looking around with his sharp blue eyes, and occasionally stroking his neatly trimmed salt-and-pepper beard.

You can perform the following actions:
Talk with Lawrence John Wargrave
Move to Rock
Move to Hall
Move to Beach
Investigate Room Living Rooms

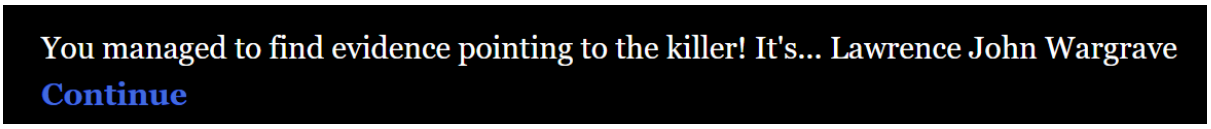
Fig. 12 An interactive story playback (with using OpenAI).

Figures 13, 14, and 15 show examples of the interactive story's reaction to the player's actions: service messages that are not part of the story, but inform the player about its progress and the results of his actions.



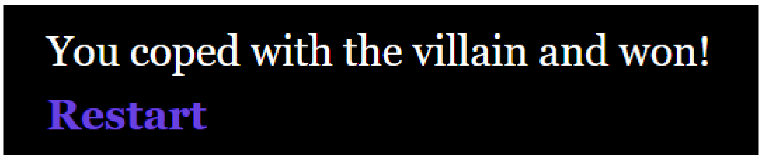
You have been killed Lawrence John Wargrave and lost.
Restart

Fig. 13 An example of how an interactive story reacts to a player's loss.



You managed to find evidence pointing to the killer! It's... Lawrence John Wargrave
Continue

Fig. 14 An example of the reaction of the interactive story to the successful discovery of evidence by the player.



You coped with the villain and won!
Restart

Fig. 15 An example of the reaction of the interactive story to a player's victory.

3.2 Architecture

As shown in Figure 16, the architecture of our proposed story generation system consists of the following subsystems: story world, drama manager, planner, constraints manager, output module.

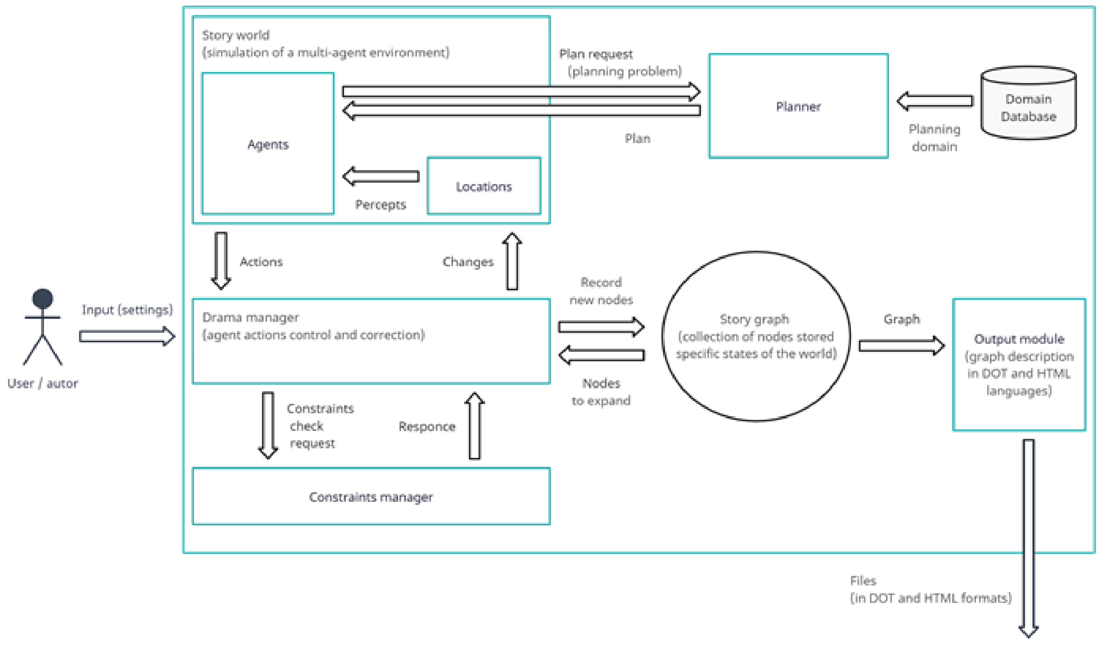


Fig. 16 Story generator architecture.

The architecture of our system is based on several logical interconnected modules that exchange information between each other, performing transformations on it.

The user (author) enters his preferences and intentions using the program interface, which allows him to configure it.

The representation of the environment is an imitation of a certain physical environment, divided into separate areas (locations) with certain characteristics in which software agents operate and interact (as well with each other). The state of this environment, including the agents in it, is the state of story.

A story graph is a data structure consisting of nodes, each of which stores a specific state of the story world, reached at some point and recorded in it. It is oriented, the edges are a representation of the transition from one state to another. Transitions are carried out through the actions of agents.

The drama manager controls the admissibility of changing the state of the story world by agents (by using actions effects), making such changes to them, if necessary, so that they do not violate the constraints, as well as the consistency and coherence of the narrative. From the environment representation, it receives information about the changes that the acting agent is going to make to the current state.

The constraint manager is used to create, store, and validate the constraints imposed on the story world.

The planner is used by agents to create action plans. It makes plans based on the planning domain and the planning problem. Interacts with the representation of the environment in such a way that it receives a request from the acting agent to search for a plan, the found plan is returned to the requesting agent.

The output module is responsible for converting the generated narrative (story graph) into other formats. For example, translating it into the dot language, for rendering, or into Twine's own language, for playback on this platform.

3.3 Story world creation

The story world can be defined as a set of settings, objects, and agents. Agents, through actions, change the parameters and properties of objects and other agents, including themselves. Settings determine what exactly can be changed, to what extent and in what way. The aggregate configuration of all objects and agents, i.e. their own states, is the state of the story world. Accordingly, by their actions, agents are able to change the state of the story world in which they operate.

As the main type of objects, we distinguish locations - simulating the environment in which agents are located and act. Each location has a number of parameters, such as: name, connections (neighborhood, which makes it possible to move from one location to another) with other locations, the presence or absence of evidence against an antagonist in this location, a list of links on agents located in it.

The state of each individual agent is determined by the following parameters: name, role, status (alive/dead), initiative value, emotional state variables (for example, the agent can be scared or angry), desire variables (for example, the killer wants to complete his long-term plan and lure another agent into a trap without being distracted by other goals), beliefs (reflect the agent's view of the story world, including himself).

The settings define things such as whether a connection between locations is necessary to move between them, or an agent can go to any one, as well as how many actions an agent can perform in one “plot cycle”, for example, visit several locations in during one “turn”, or only one, whether agents can interact with each other while in different locations (for example, talk on the phone) or not, and also determines the sequence of actions of agents, the probability of success of their actions (for example, some percentage probability or always success), as well as the very principle of selecting available actions.

3.4 Agent model

We define agents (characters) using a tuple:

```
C = {R, B, E, D, G, I, A}
```

where:

• `R` is the role of the agent. Within different settings, agents are given different roles that determine their behavior. This can be the role of a hero, or an ordinary average character, the role of an antagonist, or the role of a villain's minion, etc.

• `B` is the beliefs of the agent, which he forms, perceiving the story world. As we already mentioned, agents can only perceive a limited part of the story world at a time - only within the location in which they are located. But they remember a lot of what they managed to perceive before. For example, the roles of other agents, the locations where they saw other agents, the fact of talking with them, and their own attitude towards them.

• `E` is the agent's emotions, which can change depending on the situation in which he finds himself. For example, he may be frightened when he finds the body of another agent, or angry when he recognizes the antagonist, which will affect his behavior.

• `D` is the agent's desires that arise during the execution of actions that last several "plot cycles" and that are needed to make his actions more logical and predictable. For example, if an agent intended to move to a certain location, but not a neighboring one, then in the next "plot cycle" he will retain the desire to continue the path, instead of doing something else, although this is not due to the current state of the story world, but only the internal state of the agent.

• `G` is the agent's goals, i.e. the specific state of the story world that the agent seeks to achieve in the long term.

• `I` is the agent's intention. Each time the agent receives a request from the narrative generator, the agent plans, seeking to find a way to fulfill its intention, which it re-formulates with each new request. It is possible to call this a short-term goal. The agent formulates an intention based on his role, his goals, emotions, desires and beliefs.

• `A` is a set of actions available to the agent for selection. The availability of actions depends on the role of the agent, the current environment in the story world (the context of the environment), his emotions and beliefs (for example, the agent managed to reveal the initially unknown antagonist and now the action to neutralize the antagonist will appear in the set of actions available to him, otherwise it will not be available to him).

We believe that the agent model formulated in this way makes it possible to flexibly customize its behavior and ensure high believability of its actions, which is an important quality for emergent narratives and when using Character-Centric design.

3.5 User model

In our case, the user can be understood in two ways. On the one hand, it is the author who uses our system to generate the story. On the other hand, it is a player who will be able to play quests created using our system. Their interests and needs are noticeably different.

It is important for the author to generate an interactive story in which he can integrate his intentions. That is, to have an influence on the process of its generation. If the author is a designer who wants to calculate the possible paths of the created quest, then he needs to specify his domain database as accurately as possible. To do this, we provide the user with authorship tools that allow him to choose the setting, the type of goals for agents, determine the presence or absence of evidence in the location, set constraints imposed on the story world, and configure the behavior of different types of agents. We also provide the ability to configure a more "service" nature, that is, to determine the randomization of certain elements of the history (for example, set the seed of generation, randomly connect locations, randomly determine the outcome of fights, etc.) and specify preferences when visualizing the story graph (for example, hide "empty" actions). However, all these settings can only be made before the start of generation, in the preliminary phase. During the generation itself, the user no longer has the opportunity to influence it.

It is important for the player to have an interesting narrative experience. Its important components are the believability of agents and the variability of narratives. Therefore, we strive to give the player as much freedom of action as possible, limiting it only to the scope of common sense and the contextual availability of an opportunity to perform an action in a particular situation: he can interact with agents (for example, talk), interact with the environment (for example, look for evidence), move between adjacent locations, fight, etc. We do not seek to cut off its capabilities for shortening the story graph. In addition, it is also important for quest designers to foresee all possible actions of the player. Believability of agents is their high autonomy. The player character is also an agent by its architecture, having all of its properties that can be read by autonomous agents. They perceive the player as well as an agent, so they can try to interact with him as with other agents: talk, give information, attack, try to trap, etc.

3.6 Planning Domain

Domain Database (DD) in our system is expressed by the following tuple:

DD = {P, L, O}

where:

• P is a set of atomic expressions representing a set of relations expressed as pairs p_x = (objectType$_x$, objectName$_x$) that describe all objects in the story world. For example: (ROOM, hall), (AGENT, Journalist). An atomic expression is an expression of the form predicateName(t_1,…,t_n), where predicateName is the name of the predicate, and t_1,…, t_n are ground terms.

• L is a set of ground literals that describe the properties and relationships of all objects in the story world, in the form l_x = (property$_x$, objectName$_x$) or l_y = (relation$_{y0}$, objectName$_{y1}$, objectName$_{y2}$). For example: (ALIVE, Journalist) or (connected, (ROOM, hall), (ROOM, kitchen)). A literal is an atomic expression or its negation, allowing the removal of a statement from the current state of the story world.

• O are planning operators expressing various events and actions in the story world. Each o ∈ O, in accordance with the STRIPS formalism which we have already described, is defined as:

o = (name(o), precondition(o), effect(o)),

• name(o) is the name of the operator.

• precondition(o) is a set of literals that describe the preconditions of o (precondition must be true for this operator to be possible, but it can be either positive or negative).

• effect(o) is a set of literals that describe the effects of o (these literals can also be either positive or negative, after applying the operator effect they will become true for the current state of the story world).

An example of a story domain and action schemes for a detective story according to the STRIPS formalism implemented by us using the PDDL language is shown in Figure 17:

```
(define (domain detective-domain)
(:predicates (ROOM ?x) (AGENT ?x) (ANTAGONIST ?x) (ENEMY ?x)
(PLAYER ?x) (USUAL ?x) (alive ?x) (died ?x) (wait ?x) (in-room ?
x ?y) (connected ?x ?y) (complete-quest ?x) (want-go-to ?x ?y)
(thinks-is-a-killer ?x ?y) (found-evidence-against ?x ?y)
(scared ?x) (angry-at ?x ?y) (explored-room ?x ?y) (contains-
evidence ?x) (talking ?x ?y))

(:action Kill
:parameters (?k ?victim ?r)
:precondition (and (ROOM ?r) (ANTAGONIST ?k) (AGENT ?victim)
(alive ?k) (alive ?victim)
(in-room ?k ?r) (in-room ?victim ?r))
:effect (and (died ?victim) (not (alive ?victim))))

(:action TellAboutASuspicious
 :parameters (?k ?a ?place ?suspicious-place)
 :precondition (and (ROOM ?place) (ROOM ?suspicious-place)
(ANTAGONIST ?k)(AGENT ?a) (alive ?k) (alive ?a)
(in-room ?k ?place) (in-room ?a ?place) (not (= ?place ?
suspicious-place)))
 :effect (and (in-room ?a ?suspicious-place)))

(:action move
 :parameters (?a ?room-from ?room-to)
 :precondition (and (ROOM ?room-from) (ROOM ?room-to)
(ANTAGONIST ?a) (alive ?a)
 (in-room ?a ?room-from) (not (died ?a)) (not (in-room ?a ?room-
to))(connected ?room-from ?room-to))
 :effect (and (in-room ?a ?room-to) (not (in-room ?a ?room-
from))))

(:action nothing-to-do
 :parameters (?a)
 :precondition (and (ANTAGONIST ?a) (alive ?a))
 :effect (wait ?a))

)
```

Fig. 17 An example of a detective domain encoded in the PDDL formalism.

In this figure, we see a specific planning domain instance. It defines the predicates that we mentioned above, denoted by `P`. Then there is a list of actions (we denoted them as `O`) that operate on these predicates. Each action consists of a set of parameters, a set of preconditions, and a set of effects.

The list of parameters specifies the variables with which this action will work. In the process of building a plan, the planner will try to substitute in their place the objects of the story world (they are constants from existing grounded literals, which we denoted as `L`) in such a way that the formula in the preconditions has the evaluation "true".

The preconditions describe a number of predicates, as well as propositional connectives that form a logical formula. The arguments to these predicates are expressed as variables specified in the parameter list. As mentioned above, in the process of building a plan, the planner will try to substitute objects of the story world (agents, locations) in place of these variables as arguments to these predicates, so that this logical formula is true (this happens only if each predicate with substituting variables corresponds to one of the grounded literals). Only when the formula in the preconditions has such an evaluation, the action is considered possible to perform.

The effects describe another logical formula, similar in structure to the one described in the preconditions. However, in this case, it does not serve to check the conditions for performing an action, but describes the change that occurs in grounded literals when a given action is performed.

Consider this using the `Kill` action as an example:

We have three parameters: "`k`", "`victim`" and "`r`". We also have a number of preconditions: the variable "`r`" must be a parameter of the `ROOM(x)` predicate, the variable "`k`" must be a parameter of the `ANTAGONIST(x)` predicate, the variable "`victim`" must be a parameter of the `AGENT(x)` predicate, the variables "`k`" and " `victim`", each separately, must be parameters of the predicate `alive(x)`, the variables "`k`" and "`r`" must be parameters of the predicate `in-room(x, y)`, similarly, the variables "`victim`" and "`r`" must be parameters predicate `in-room(x, y)`. It should be understood that the variable "`k`" must represent the antagonist-agent, the variable "`victim`" must represent the normal-agent, the variable "`r`" must represent the location, the killer agent and the victim agent must be alive at the time when the action starts and be in the same location. The task of the planner is to find grounded literals that will match the given predicates with the specified variables as parameters. This means finding agents of the appropriate types and a location whose properties will meet the specified conditions in the world state. At the end is described the effect of the action: a new grounded literal is created corresponding to the predicate `died(x)` with the variable "`victim`", and a negation is added to the grounded literal corresponding to the predicate `alive(x)` with the variable "`victim`". This means that the result of the action will be to change the status of the victim agent from alive to dead.

3.7 Planning Problem

In our system, planner solves planning problems expressed using a STRIPS-like formalism. These problems are dynamically formulated by agents, according to their current intentions (desired state of the story world). The intentions of agents are formed from a set of their character (realized in the form of assigned roles), goals, and beliefs about the story world obtained in the course of observing it (at the same time, the agent does not survey the entire story world, but only the location in which he is, but he has a memory of what he observed earlier.

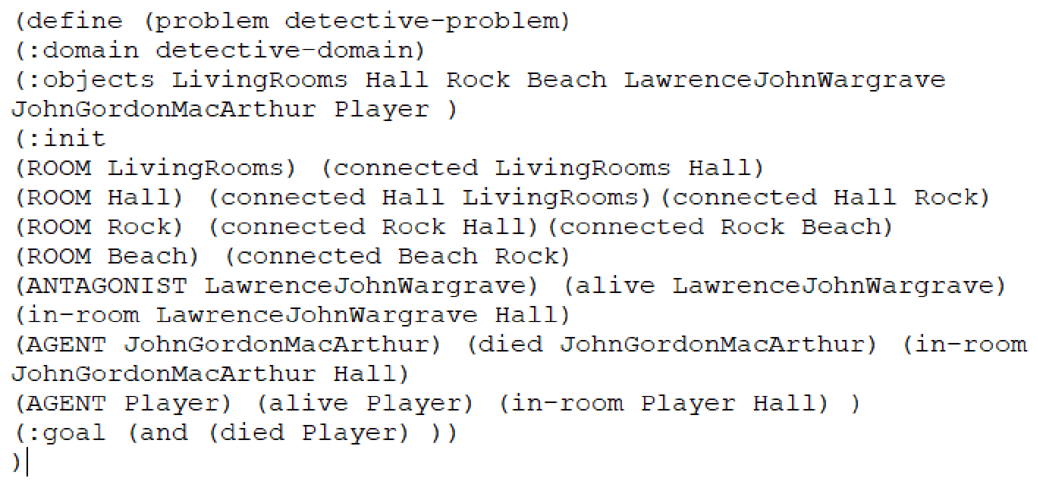
We express the planning problem (`PP`) with a tuple:

$PP = (DD, S_0, I)$

Where `DD` is the planning domain (domain database), S_0 is the initial state of the story world (it is also valid to consider it the current state of the story world, since the agents are formulating a planning problem an ongoing story), and `I` is the intention of the planning agent expressed as the desired state of the story world.

It is possible that the question will arise why, when formulating the planning problem, we did not mention the constraints that are imposed on the story world, forbidding it to move into certain states. The reason is that agents do not know about them, so they do not consider them when planning. These constraints are used by the drama manager, which has the ability to interfere with the actions of agents, violating their plan. They are part of the regulation of the narrative, but not of the problem of planning.

An example of a story problem for a detective story according to the STRIPS formalism implemented by us using the PDDL language is shown in Figure 18:



```
(define (problem detective-problem)
(:domain detective-domain)
(:objects LivingRooms Hall Rock Beach LawrenceJohnWargrave
JohnGordonMacArthur Player )
(:init
(ROOM LivingRooms) (connected LivingRooms Hall)
(ROOM Hall) (connected Hall LivingRooms)(connected Hall Rock)
(ROOM Rock) (connected Rock Hall)(connected Rock Beach)
(ROOM Beach) (connected Beach Rock)
(ANTAGONIST LawrenceJohnWargrave) (alive LawrenceJohnWargrave)
(in-room LawrenceJohnWargrave Hall)
(AGENT JohnGordonMacArthur) (died JohnGordonMacArthur) (in-room
JohnGordonMacArthur Hall)
(AGENT Player) (alive Player) (in-room Player Hall) )
(:goal (and (died Player) ))
)
```

Fig. 18 An example of a detective problem encoded in the PDDL formalism.

In this figure, we see a specific instance of the planning problem. It contains a reference to the planning domain that will be used to resolve it (we denoted it as `DD`), a list of objects, the initial state of the story world (denoted us as `s0`), and a goal (denoted as `I`).

The list of objects corresponds to entities in the story world, such as agents and locations.

The initial state is described as a set of grounded literals, whose constants are the objects mentioned above.

The goal is the agent's intention, expressed as a change in the initial state in such a way that it contains the literal specified in the goal and does not contain those that contradict it. The task of the planner is to find such an action in the chosen planning domain that would change the initial state so that it corresponds to the goal.

Consider this using the problem presented above as an example:

It lists all locations and agents that exist in the story world and a description of the state of the story world in which was located the agent who created this problem, as well as the goal of this agent. It is important to consider that, in creating this planning problem, the agent uses his beliefs, which may be wrong, but not accurate system information. In this case, this agent is “`Lawrence John Wargrawe`”, located in the “`Hall`” location. His goal is to kill the agent "`Player`", who is in the same location. He also has information about the topology of the story world and the status of agent “`John Gordon MacArtur`”, who was killed by him in the same “`Hall`” location. In this case, the plan that the planner will return will be simple and will consist of just one step: apply the “`Kill`” action, using the objects "`Lawerence John Wargrawe`", "`John Gordon McArtur`", "`Hall`" as its parameters, i.e. will look like: "`Kill (Lawerence John Wargrawe, John Gordon McArtur, Hall)`".

3.8 Story Planning

Quite often, the story worlds consist of a large number of different objects, the number of which goes to tens, hundreds and even thousands of entities: characters, locations, objects, and so on. In addition, as part of the interaction with the story world and the characters in it, users can perform dozens of different actions. Characters can also perform a number of actions while interacting with each other and with the story world. This seriously complicates the planning problem, since the state space, given so many variables in the planning domain, reaches a huge size of hundreds of millions of nodes [29]. Finding a narrative plan in a space of this size is already an extremely difficult optimization problem, but we also impose additional requirements on it, related to the logical coherence of the sequence of events, and the overall structure of the narrative, in order to ensure its consistency.

In accordance with this, we believe that the classical search methods are not efficient enough, further losing their effectiveness as the duration and complexity of the story increase (i.e. the planning domain increases). By efficiency, we mean, among other things, the speed of search. This factor is important, given the fact that we want to create a tool that allows to accelerate the development of interactive stories and video games. If its use requires a supercomputer and days of computing, its usefulness will be questionable. Therefore, our task was to simplify the search problem and provide a sufficient level of performance.

A feature of our system is that it only generates stories, while the platform for their playback is an external module. For example, it could be Twine. This imposes specific constraints on story generation, since we cannot control an external module at runtime, but can only load a predefined sequence of instructions into it. Because of this paradigm, the result of the program is a plan that takes into account all the possible actions of the player in advance and is encoded as a story graph, which can then either be rendered or converted into a set of instructions for Twine.

To achieve these goals, we combined the automatic planning method with the principle of emergent narrative. We starting with a story graph consisting of a single node, the initial state. We then give agents the ability to act completely autonomously to achieve their goals. At each step of the algorithm, considering the selected state of the story world, the narrative generator determines whose turn from agents to perform an action, and then sends a corresponding request to this agent. Then we add the new state of the story world, obtained as a result of the implementation of the agent's action, to the graph. The action itself is also added, encoding in the form of edge connecting two nodes - the one in which it was applied and the one it caused to appear. To reduce the number of possible nodes and edges, we resort to cutting them off, ensuring that in any given state of the story world, each agent will choose exactly one particular action. To do this, we use his models of emotions, desires and beliefs, taking into account also his role and goals, so that he chooses his current intention, then we call the planning procedure, which will build a plan for the agent to achieve his intention, and then choosing the first action from the plan, assign him suitable parameters using the CSP technique. To keep the plans up to date, we use the "turns" system, in which agents plan and perform actions in turn. These moves cyclically replace each other as soon as all agents capable of acting in the current state of story world perform an action. We do not undertake to direct the user's actions, because we strive to give him as much freedom as possible, therefore they are the source of the story graph branching factor. The creation of the story graph is completed when each branch of the graph reaches the goal state. After that, we render the resulting story graph and convert it into a plan for Twine. And although we adhere to the principle of emergent narrative, we need to ensure the coherence and consistency of the narrative, which this kind of narrative does poorly. Therefore, we will use the drama manager, for soft centralized regulation in cases where it is necessary. In essence, we combine Story-Centric (in the form of centralized regulation with the help of a drama manager) and Character-Centric (in the form of highly autonomous agents and emergence) designs.

3.9 Convergence model and constraints

To maintain the coherence and consistency of the narrative, as well as maintain the author's intention, we use a method that we call "convergence". His idea is that all the narratives of a generated story converge to some common configuration, while being otherwise completely independent of each other and unique. Those, some events are bound to happen (or vice versa, cannot happen), regardless of how the storyline developed before. For example, if an agent must die solely at the hands of an antagonist, then no other agent can kill him. Alternatively, if the antagonist must survive the first few "plot cycles", then in this case no one can harm him as long as the "convergence" "protects" him.

To do this, we use constraints, a set of rules that the drama manager follows. They describe parts of the configurations of both the state of the story world and the story graph itself, which are considered to violate one of the properties of the narrative that we maintain. If such a violation is detected, then the drama manager intervenes, which prevents the agent (or player) from taking an action whose effect would lead to a violation. Instead, he performs a counter-action that replaces the violating action. For example, if an agent performs a “move” action to go to a location where he is currently not allowed to be. Then the drama manager will perform a counter-action that will send the agent to another nearby location, which can be interpreted as the fact that the agent got lost and did not go where he intended.

Some of the constraints can be set explicitly: for example, using the author's interface. It is possible to call these constraints dependent on the story domain and the author's intent. This is of the kind such as prohibition of movement between locations, control of the agent's status during the required number of “plot cycles”, requirements to perform one action before performing another, etc. The other part of the constraints is an integral part of the algorithm, they support basic coherence and consistency, and do not depend on the domain. For example, they check the status of an agent before requesting an action - an agent with the status "dead" cannot perform actions. Also, the narrative should not enter the cycle, or enter the dead end, and the actions of the agents should be evenly distributed.

4. Use cases

In this section, we will give examples of specific instances of the system, considering their input, author's intentions, and generation results.

Example 1 - Dragon Age:

Here the inputs are the following settings: setting - fantasy, subsetting - dragon age, target types - based on agent statuses, maximum number of nodes - 350, no random encounters, "do nothing" actions are hidden, the protagonist must survive, the antagonist and enemy agents must be cunning, the protagonist and regular agents must be aggressive towards the antagonist.

With this input, the characters are the protagonist Gray Warden and the antagonist Archdemon. The world consists of the locations Lothering, Mages Tower, Brecilian Forest, Orzammar, Denerim, Deep Roads. The following actions are available to the protagonist: move, speak, fight, help mages, help templars, help elves, help werewolves, help Prince Belen, help Lord Harrowmont. The following actions are available to the antagonist: move, fight. The following constraints are imposed on the world:

1) The antagonist cannot leave the Deep Roads location until the protagonist is in the Denerim location;

2) The protagonist cannot leave the Mage Tower, Brecilian Forest and Orzammar locations until he performs another action than Move;

3) The protagonist cannot die;

4) The protagonist cannot return to the location where he has already been;

5) The antagonist cannot move to any location other than the Denerim location.

The goal of each agent is to kill the other.

The locations mentioned above represent the main locations in which the game took place, and the actions represent the main choices and events in it. Constraints force the agents to follow the structure of the game's story, playing through them in a unique order.

The author's intent is to simulate the player's progression through the Dragon Age Origins game (main story events only). At the same time, the author is not interested in the results in which the player fails. This will allow him to evaluate all possible combinations of ways of passing. If this were done during the development process, the purpose of this could be to collect information to assess the possibility of strengthening the connection of some events with each other (i.e. subsequent events would be more directly related to the previous ones, contain references to them).

As a result of the generation was obtained a file in dot format, which was then rendered into an image in PNG format, and also an HTML file, which, when loaded into the Twine system, will allow to bypass this graph in an interactive form, i.e. play as an interactive story.

On the graph, we clearly see all the possible ways to complete the main plot of Dragon Age (48 unique paths), divided into eight groups. These groups are determined by options for the order in which the story choices are made (three main locations, with the ability to move between them in free order, in each of which two choices are available). Returning to the author's intentions, the author can analyze this graph and, based on it, come to the conclusion that it is too costly to additionally work each possible path of the planned plot individually. However, if he will work out not every possible way, but only general patterns (of branching order of passing), then this is quite possible, since in this case the number of such patterns is small.

Figure 19 shows a fragment of the rendered graph from Example 1:

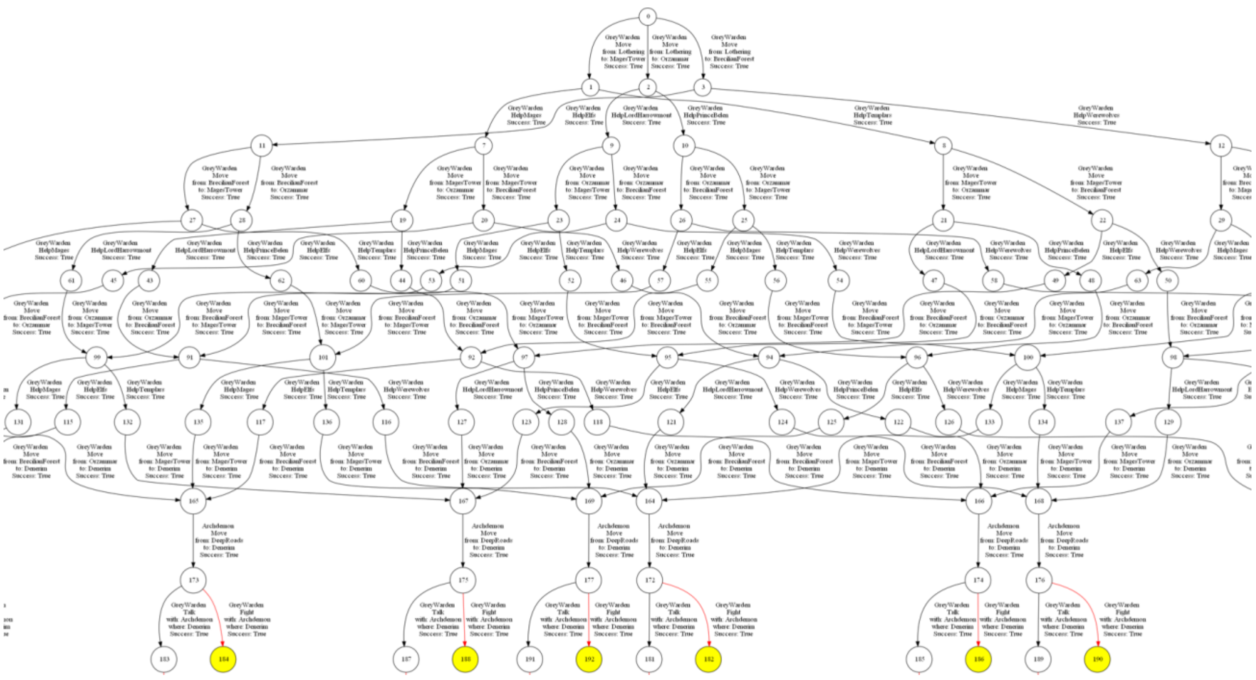


Fig. 19 A fragment of the rendered graph from Example 1.

Example 2 - Random Detective:

In this case, the inputs are the following settings: setting - detective, types of goals - based on agent statuses, seed - 0, maximum number of nodes - 350, location links are determined randomly, the result of fights is determined randomly, the initiative of agents is determined randomly, empty actions are skipped in the visualization, agents can find an evidence when searching the location with a probability of 75%, the protagonist must survive, the antagonist and enemy agents must be cunning and talkative, the protagonist and ordinary agents must be aggressive towards the antagonist.

With this input, the characters are the protagonist (Player), John Gordon MacArthur (usual agent) and Lawrence John Wargrave (antagonist). The world consists of the Living Rooms, Hall, Rock and Beach locations. The following actions are available to the player and the usual agent: move, talk, investigate the location (in search of evidence), neutralize the killer. The following actions are available to the antagonist: move, talk, entrap, kill.

In this case, the author's intention is to obtain a narrative that corresponds in form to the average detective story-taking place in a limited space (as a guideline serves Agatha Christie's story "Ten Little Indians"). The author has no preferences for a specific development of events, but the protagonist must survive in any scenario. The end goal may be to create the basis for an interactive story.

As a result of the generation was obtained a file in dot format, which was then rendered into an image in PNG format, and also an HTML file, which, when loaded into the Twine system, will allow to bypass this graph in an interactive form, i.e. play as an interactive story.

On the graph, we see a small generated story that matches the intentions of the author. In it, the protagonist (player) has the opportunity to interact with the environment in different ways: talk with other agents, explore locations, finding evidence and neutralize the killer, or even be attacked by him. At the same time, this story world "lives", not concentrating only on the player. If the player does nothing, or goes the wrong way, another agent will find the evidence himself and expose the killer.

Figure 20 shows the rendered graph from Example 2, with the requirement that the protagonist must survive:

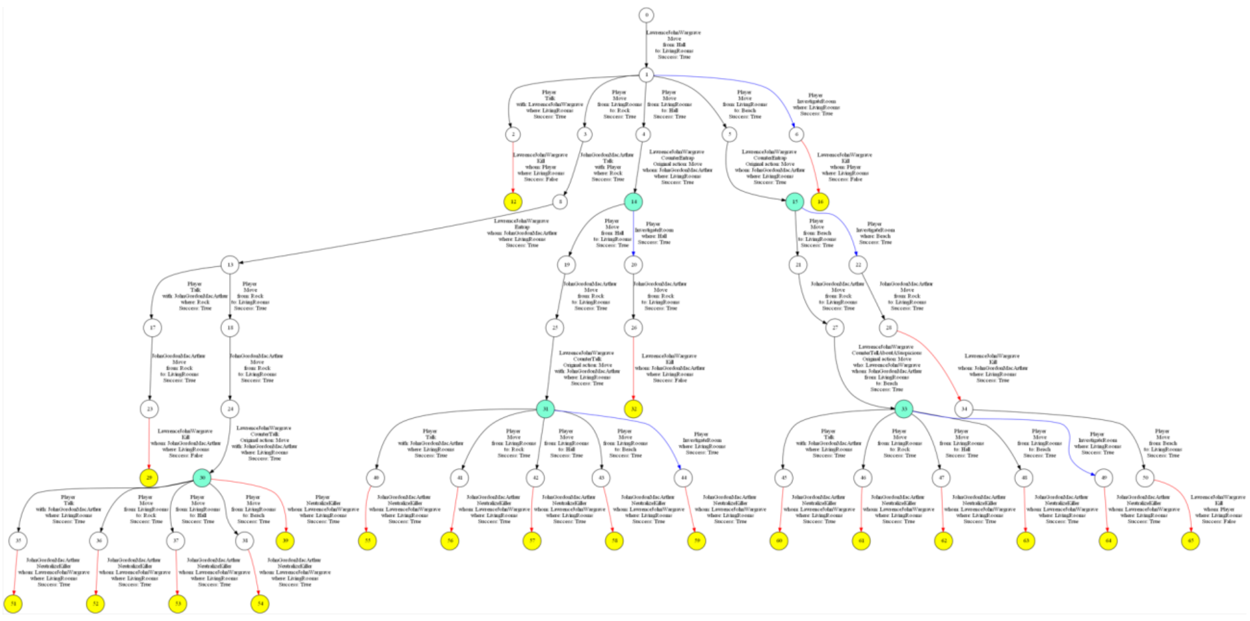


Fig. 20 The rendered graph from Example 2, with the requirement that the protagonist must survive.

However, we can change the conditions a bit, removing the requirement that the protagonist must always survive and adding the random results of the fights. This will lead to some changes in the graph. As it is possible to see, the graph has become larger as the antagonist manages to "survive" more time.

Figure 21 shows the rendered graph from Example 2, without the requirement that the protagonist must survive:

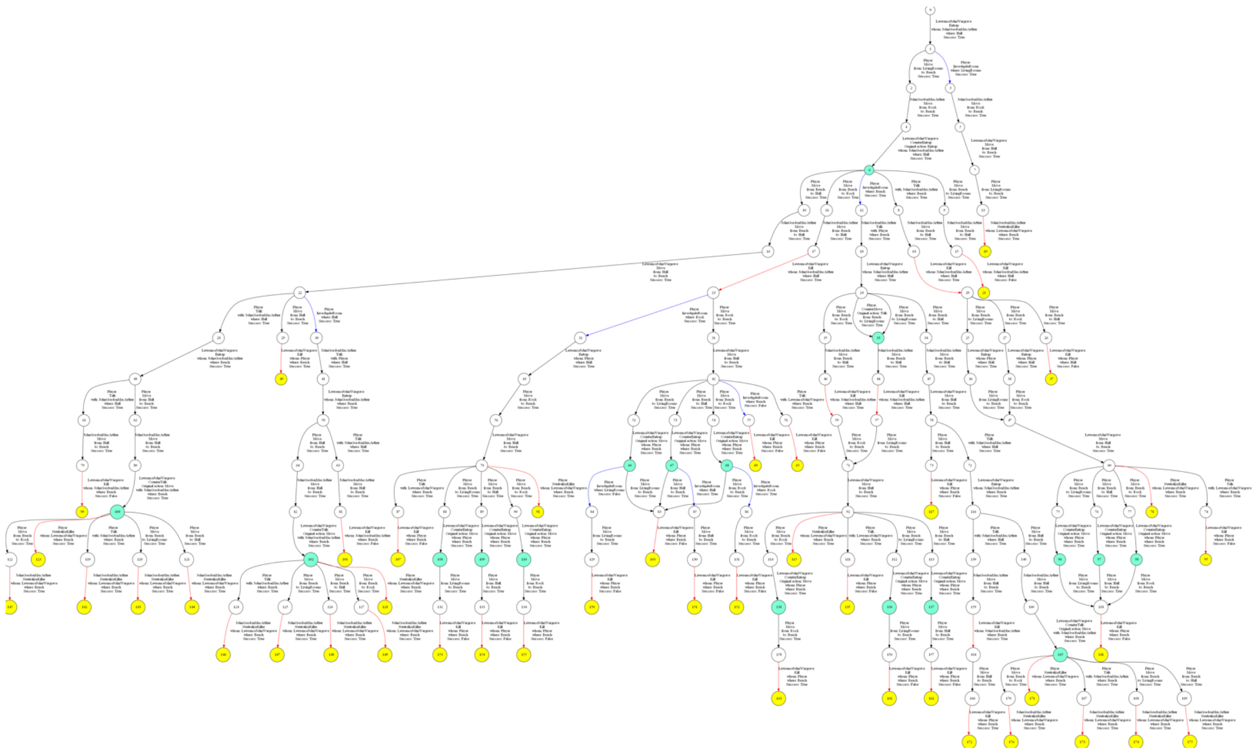


Fig. 21 The rendered graph from Example 2, without requirement that the protagonist must survive.

5. Implementation

In this chapter, we will describe the details of the technical implementation of our system, presenting a general description of its architecture. In particular a detailed description of the algorithm as a whole and a description of the implementations of the interface, environment, agents, graphs, states, actions, and constraints. As well as a description of the integrated third-party modules.

5.1 Overview of software choice

5.1.1 General concept

From the very beginning, we planned to use some kind of IDE to create our system, so that it would help facilitate the development process and implement all the subsystems of interest to us. Initially, we considered using a game engine like Unity [88] because of their power. Then we considered this idea unsuitable, since these software tools provide excessive capabilities for our needs, for which we would probably have to pay in performance, which would negatively affect the implementation process, since we did not have enough powerful computer equipment. In addition, our goal was to simplify the overall design, not complicate it. With these considerations in mind, we decided to go for a simpler IDE. It was also important to consider the choice of language, as different IDEs support different languages. We concluded that a reasonable choice would be to use one of the JIT languages, which provide both good computational speed and convenience. They also have a wealth of expressive possibilities. The most well-known languages of this type are C# and Java. This narrowed down our final choice to the already familiar for our Microsoft Visual Studio [89] and IntelliJ IDEA [90].

When choosing between MVS and IntelliJ IDEA [90], we took into account several factors. First, each of them supports different languages, and it was necessary to choose between C# and Java. Since we had much more experience with C# than with Java, was chosen exactly this language. Performance on our hardware also mattered. MVS ran stably and without freezes, while IntelliJ IDEA [90] took a long time to start up and had occasional performance issues. We also took into account the convenience of creating a graphical interface, in which MVS provides more options than IntelliJ IDEA [90], thanks to the Windows Forms API. Therefore, in the end, we decided to use the MVS IDE [89] (Microsoft Visual Studio Community 2017 version) and the C# language, in which we wrote all the code.

To simplify the design, we planned to divide it into several modules that could be connected or disconnected as needed. So, a third-party system, Fast Downward [91], is responsible for automatic planning, and Graphviz [92] is responsible for visualizing interactive story graphs. We eventually abandoned some of the originally planned external modules, since they only complicated the design and use of the system, having implemented their functionality on our own.

5.1.2 Fast Downward

Fast Downward [91] is a classic planning system based on heuristic search. It is capable of dealing with common deterministic planning problems encoded using the PDDL language. Like some other famous planners, Fast Downward [91] is a progression planner that searches the state space of the world for forward planning problems [94]. We used the seq-opt-lmcut planner configuration (sequential optimizing planner type with Landmark-Cut heuristic, one of the most popular heuristics in optimal planning due to its efficiency in quickly finding optimal paths in problems with a small number of key points and its universality for different tasks).

5.1.3 Graphviz

Graphviz [92] is a set of utilities for automatically visualizing graphs, described using the DOT language.

The main tool from this set of utilities is "dot", which accepts a text file in the same language as input. That file describes the nodes and their connections in the graph, as well as various additional properties, such as their name, color, etc. "Dot" itself recognizes all connections of the graph and arranges it in such a way as to minimize the number of intersections of edges in space. However, manual adjustment or the introduction of preferences is not available. The resulting graph can be displayed as a graphic (PNG), vector (SVG) or text file (PDF), and a number of other formats are also supported. Also included in this set are tools such as "neato" - a tool for creating a graph based on a "spring" model ("spring model", "energy minimized"), "twopi" - a tool for creating a graph based on a "radial" model, "circo" is a tool for creating a graph based on the "circular" model, "fdp" is a tool for creating an undirected graph based on the "fdp" model, and some others.

We use this main tool to automatically generate a PNG image immediately after generating a DOT file describing the generated story graph. However, this DOT file can be manually used for visualization with another tool or another version of Graphviz.

An example of a short detective story created by our generator, visualized using graphviz is shown in Figure 22:

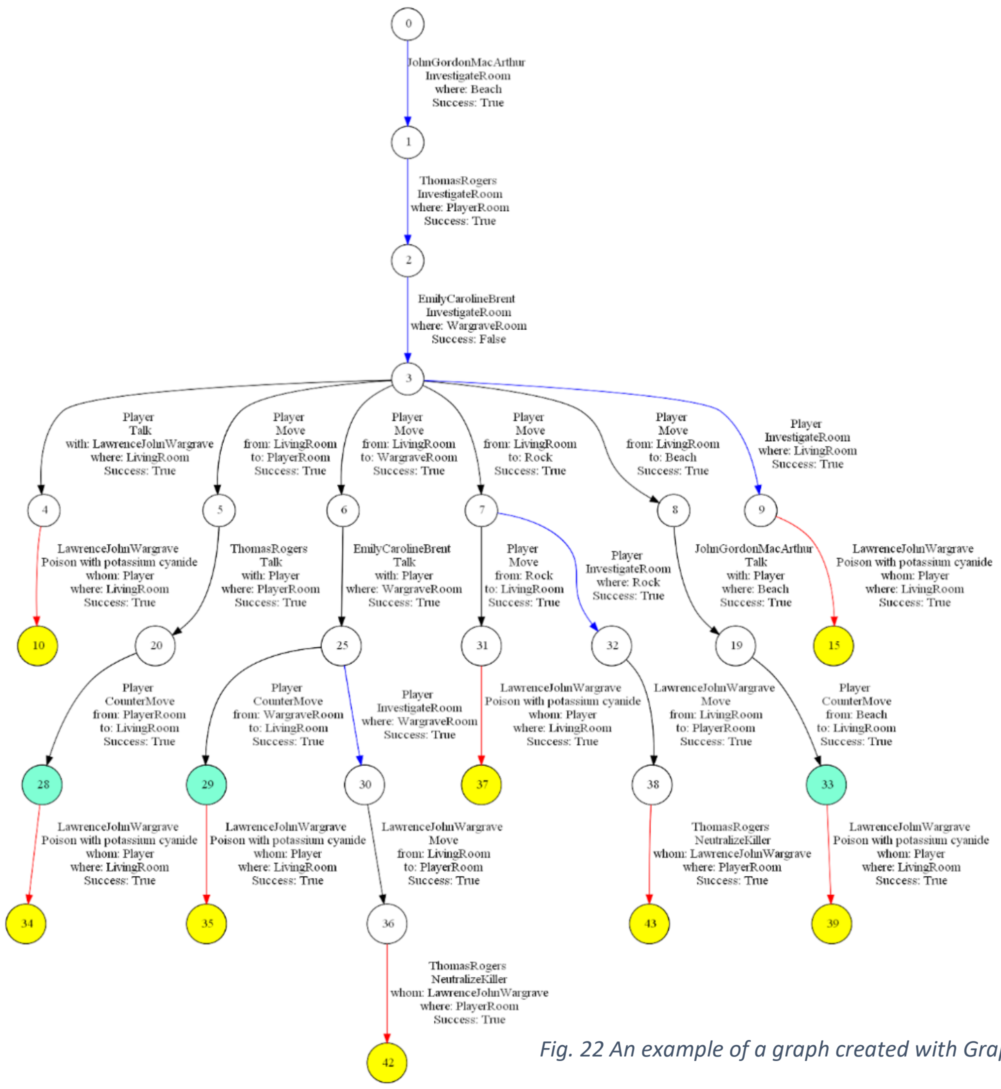


Fig. 22 An example of a graph created with Graphviz and dot.

5.1.4 Twine

Twine is a platform for writing interactive fiction, text-based games, and visual novels. Its great advantage is ease of use - it allows users to export stories to HTML files that can be used locally or uploaded to a website, viewed with any browser, and does not require programming skills, since it uses its own markup language. In addition, its advantages are an intuitive interface and undemanding both to the resources of the platform itself (which, moreover, is available online), and to the games and stories created with its help. The workspace is presented as a visual scheme with blocks-nodes (in which text is placed) and edges-transitions between them, implemented as links. At the same time, the final view of the story can be flexibly modified using JavaScript and CSS, entering it directly in the editor.

This module is also optional. Our system generates files in HTML format, which can be loaded into Twine, which in this case is not a development tool, but a platform for running an interactive story. Twine itself is available both as an online service and as a downloadable application.

An example of the Twine graphical workspace (from where stories and quests also can be launched) is shown in Figure 23:

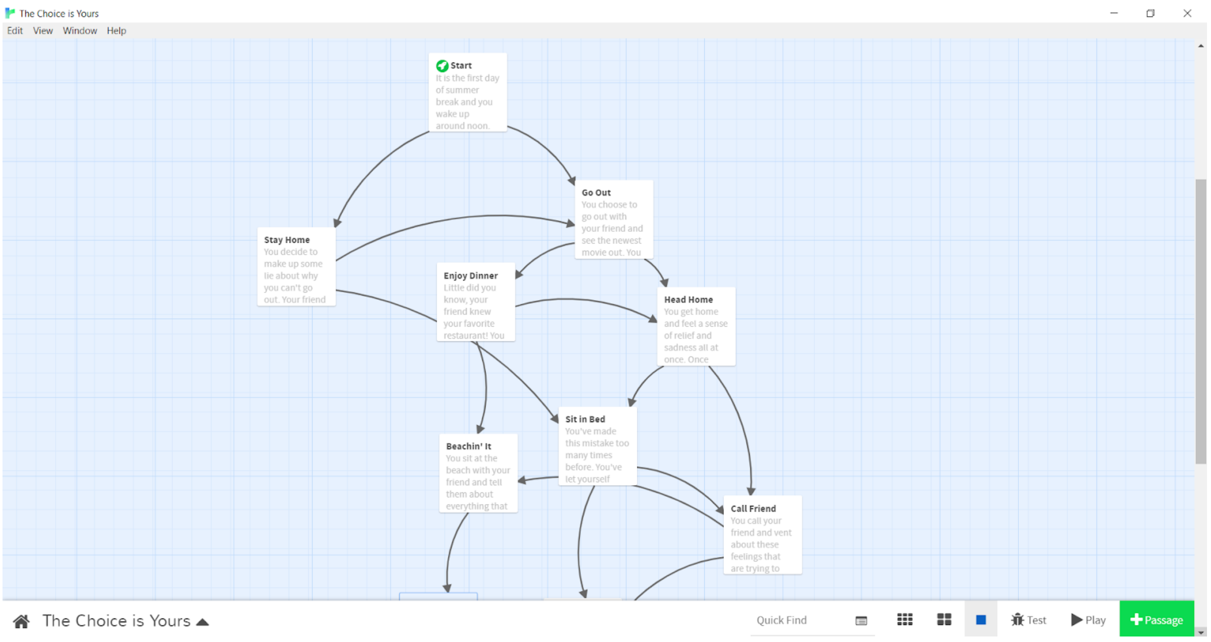

The Choice is Yours
Edit View Window Help
Start
It is the first day of summer break and you wake up around noon.
Go Out
You choose to go out with your friend and see the newest movie out. You
Stay Home
You decide to make up some lie about why you can't go out. Your friend
Enjoy Dinner
Little did you know, your friend knew your favorite restaurant! You
Head Home
You get home and feel a sense of relief and sadness all at once. Once
Sit in Bed
You've made this mistake too many times before. You've let yourself
Beachin' It
You sit at the beach with your friend and tell them about everything that
Call Friend
You call your friend and vent about these feelings that are trying to
The Choice is Yours
Quick Find
Test
Play
Passage

Fig. 23 Twine workspace example.

5.2 Generator

5.2.1 General algorithm

The generator operation process is controlled by the logic implemented in the `StoryAlgorithm` class. In this class are created and stored instances of most of the other generator modules. Their work in it is synchronized, and the data of their work is stored and transferred to those modules that need them in the future. This happens in the following order: user-entered data (including settings) is read, the initial state of the story world is created and initialized, instances of the necessary constraints are created, planning domains are generated, an instance of the story (narrative) graph and its root are created, then graph nodes are expanded cyclically, until while it is possible. After that, the story graph is visualized, and a control file for Twine is created on its basis.

Reading the data entered by the user and the settings selected by the user occurs in the `ReadUserSettingsInput` method, where they are written to the `currentStoryState` entity. True to its name, it stores the current state of the story, including some of the settings that are specific to a particular story. Since at the time of reading the entered data and settings, the story has not yet begun to be generated, this variable will not contain anything other than information about the settings, as a result of which it will be used to initialize the root node of the story graph. Also in this method, data is passed to the `pddlModule` entity, which implements interaction with the PDDL language, in particular, encoding information about the planning problem and the planning domain for the current state of the story and writing them to a file. To work correctly, it needs information about some settings. Finally, some of the settings directly affect the definition and number of goal states, because this method also passes data about them to the `goalManager` entity, which is an instance of the class responsible for storing and processing information about goal states, as well as methods for manipulating them.

The initial state of the story world is created and initialized using the `CreateInitialState` method. The first step here is the generation of locations, using the `CreateLocationSet` method, which creates a set (dictionary) of pairs (static and dynamic parts of each created location), each of which together makes up a single entity. Then, in a similar way, a set of agents is generated using the `CreateAgents` method. The notable difference is that the "agent" is a more complex entity than the "location", storing more data and methods. As a result, additional methods are used to create and initialize agents in the future. For example, to prioritize their actions (refer to them when expanding nodes), using the `DistributionOfInitiative` and `OrderAgentsByInitiative` methods, passing information about the goal states of the story (using the `goalManager's AssignGoalsToAgents` method) and their beliefs. During this process, the `CreateInitialState` method accesses the user's settings, in particular, the choice of setting, since this has a significant impact on the number of locations, agents, their roles and names, as well as the definition of target settings, and some others. Then, using the `CreateEnviroment` method, the location data is added to the `currentStoryState`. Agents are also added there, one by one, but immediately after they are created (using the `AddAgent` method called in the `CreateAgent` method). This is because locations are given data about the agents located in them, so they are “forced to wait” until the generation of agents is completed. This is necessary in order to be able to know exactly where each of them is in the starting state.

Constraints are generated in the `CreateConstraints` method. In this method, depending on the settings, instances of constraints of a different type are created (each type is represented by the corresponding class), to which additional information is passed (a more detailed specification of the type of constraint, specific parameters, such as the validity period of the constraint). The `constraintManager` entity is responsible (including being used in this method) for constructing, storing and providing constraints.

The planning domain is generated using the `GeneratePDDLDomains` method from the `PDDL-Module` class, in which a PDDL file is generated to be passed to the planner. Depending on the settings, its content may be different, but in general, it describes the planning domain: locations, their connection with each other, the presence of evidence in them, agents, their status, beliefs, being in locations, a description of possible actions to perform. Also, when creating this file, a task is defined (based on the current state of story, the role of the agent and, accordingly, its goal, and some other parameters), the execution of which will bring the agent closer to fulfilling its goal (achieving the goal state of the story world). After creation, this file is saved in the directory where the program files are located.

Then it created a root node for the graph, to which is assigned a start state, and also assigned information about which agent will act first. This node is placed in the graph and marked as the root node.

When the root node is ready and added to the empty graph, ran the `CreateStoryGraph` method. In it, in a loop built a story graph. The loop is structured as follows: the root node is passed to the `BFSTraversing` method, after which it is expanded, and then its descendants are expanded. We describe this process in more detail below. Next, is launched the `BFSGoalAchieveControl` method, which checks for the presence of goal states in the graph (the number of required goal states is determined by a constant, i.e. configurable). Then checked the condition: a positive result of the `BFSGoalAchieveControl` method and that the number of nodes in the graph does not exceed a certain set number (depending on the settings). If all conditions are met, then this method returns the completed story graph. Otherwise, the graph is cleared and its construction starts again (since the generation process has a number of adjustable elements with randomness, then, depending on the settings, the result of new attempts to be either almost identical or noticeably different).

Returning to the `BFSTraversing` method, we can say that, step by step, node by node, the graph is directly constructed. To do this, we used a breadth-first graph traversal algorithm. That is, a queue and a list of visited nodes are created. Then, in a loop, limited on the one hand by the presence of nodes in the queue (the root node enters it first), and on the other hand by the maximum allowable number of nodes (it was explained above), the nodes are removed and expanded from the queue one by one (except for those whose state is already marked as goal, they are not expanded). To do this, with helps the `GetActualAgentNumber` method, is determined which of the agents will be called by the `Step` method. In the `Step` method, the selected agent is called with help the `ActionRequest` method provided by the `storyworldConvergence` class entity. In this method, the previously selected agent's perceptions and beliefs are updated (using the `RefreshBeliefsAboutTheWorld` method), then, based on its current perceptions, beliefs, and goals, the agent calls pddlModule and uses it to create a PDDL file describing the planning problem (using the `GeneratePDDLProblem` method). After preparing the file, the agent calls the planner, providing it with PDDL files describing the planning domain and the planning problem, and receiving in response an action plan (using the `CalculatePlan` method). This plan is provided by the planner in the format of a text file, so the agent has to parse it into “understandable” (for self) action entities that the agent adds to the data structure - the plan. After that, the agent checks which of all actions are currently available for him to perform by going through each type of action and comparing its preconditions with his own role, state, beliefs, setting, settings and state of the world, after which he remembers in a separate list those actions that can perform (using the `ReceiveAvailableActions` method). Then, by calling the `ChooseAction` method, the agent decides what action to perform, comparing the plan received from the planner with the list of actions available to it. If the first action specified in the plan is in the list of available actions, the agent remembers it as selected and continues. Otherwise, it decides not to perform any action at all (an empty action). After selecting an action to perform, the agent accesses the `cspModule` entity, using the `AssignVariables` method it provides, assigning variables to actions. For example, in the case of a movement, the variables - the initial and final locations are read directly from the plan and are substituted into the entity representing the action. In other cases, the most appropriate variables are selected for substitution into actions, with using the CSP methodology, since not all constraints can be reasonably encoded using PDDL, which means that the planner may substitute them incorrectly. When the agent has selected the action that is going to be performed and substituted all the necessary variables into it, then using the `ActionControl` method, it accesses the `storyworldConvergence` entity. It must handle the action itself, change the state of the story according to its effects, and add a new graph node, and in case of collisions, handle them properly.

In the `ActionControl` method, the first step is to determine how the agent performed the action - successfully or not (using the `ProbabilityCalculating` method). Then performed a some number of checks: checking that all constraints are satisfied (using the `ConstraintsControl` method), checking for the presence of dead end nodes in the graph (using the `DeadEndsControl` method), checking for the presence of a node that is identical to the already existed node in the graph (using the `DuplicateControl` method), checking for the occurrence of cycles in the graph (with using the `CyclesControl` method). The scheme of all checks is generally the same - the current state of the story world is cloned, the checked action is applied to it, as a result of which the effects of this action change this test state. Then created a special test node, into which inserted the newly created test state, after which checks specific to it are performed in each case. After that, all test objects are destroyed. Based on the results of the checks, different reactions to them are possible. If all checks are passed successfully, then called the ApplyAction method to apply the effects of the action already directly on the current state of the story world, after that a new node is created and it added to the story graph. Also, if all the checks are passed successfully, except for checking a duplicate of the newly created node (i.e. the exact same node that was obtained by applying the effects of the action on the current state of the story is already present in the story graph), then calls the DuplicateNodeConnecting method. It will connect the current expandable node and that node in the graph, similar to the received test one, with a new edge. If detected a constraint violation, but all other checks are successful, then calls the ActionCounteract method, in which the system tries to find a more appropriate action to use, taking into account the current state of the story world and graph configuration. As in the case of an action performed by an agent, cspModule is used to assign variables to this action, and the potentially selected action, before being applied, passes the appropriate checks using the CounterreactionControl method. Further, similarly, if the new received state (and, accordingly, the node) is unique, the ApplyAction method is called, but if such a node already exists in the graph, the DuplicateNodeConnecting method is called. It should be clarified here that although the chosen action is technically considered to be the chosen agent, it is actually chosen by the storyworldConvergence entity, and this mimics the circumstance of why the agent's action was unsuccessful and what happened instead. Also, if the detected "duplicate" node is the parent of the considered current node (or itself), then the ActionCounteract method is also called, as well as in all other combinations of failed checks not mentioned here.

After the Step method completes, in the loop checks all the edges of the newly expanded node to check the nodes at the other ends of the edges. If the discovered nodes are not in the list of already visited nodes (often these are newly created nodes), they are added to the queue, as well as to the list of visited nodes.

After building the story graph, it is checked using the BFSGoalAchieveControl method. This method, using the graph width traversal algorithm, identifies all nodes in which the target state has been reached and counts their number.

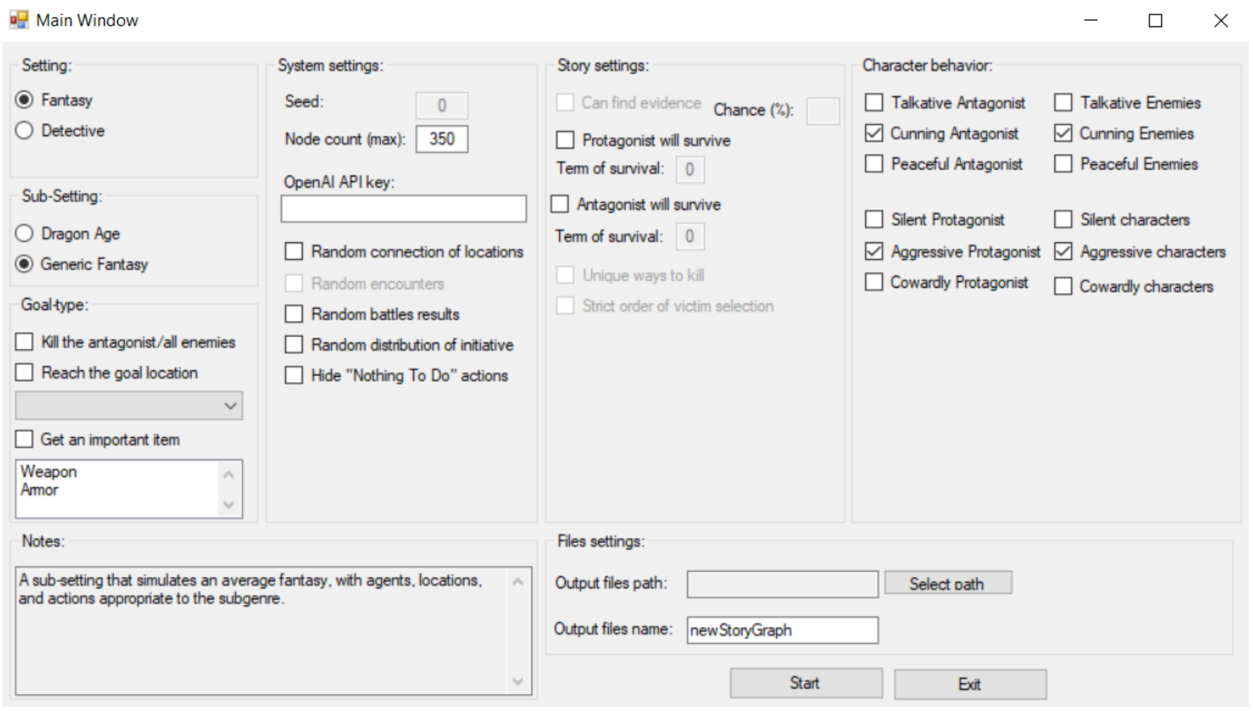
When the story graph is built and validated, this graph is passed to the graphConstructor entity. This entity, using the CreateGraph method, creates a file in the .dt format, in which it describes the previously generated graph in the dot language so that it can be visualized by any program that recognizes this language. For example, using Graphviz [92]. Then we visualize it by running and configuring Graphiz using the Graphviz and GraphvizWrapper classes methods.

After that, the story graph is passed to the twineGraphConstructor entity, which, using the CreateTwineGraph method, creates a file in HTML format. This file describes the generated graph using the syntax of the Twine system.

After that, the algorithm ends.

5.2.2 User interface

Figure 24 shows the graphical user interface.



Main Window

Setting:
[✓ Fantasy]
Detective

Sub-Setting:
Dragon Age
[✓ Generic Fantasy]

Goal-type:
Kill the antagonist/all enemies
Reach the goal location
Get an important item
Weapon
Armor

System settings:
Seed: 0
Node count (max): 350
OpenAI API key:
Random connection of locations
Random encounters
Random battles results
Random distribution of initiative
Hide "Nothing To Do" actions

Story settings:
Can find evidence Chance (%):
Protagonist will survive
Term of survival: 0
Antagonist will survive
Term of survival: 0
Unique ways to kill
Strict order of victim selection

Character behavior:
Talkative Antagonist
[✓ Cunning Antagonist]
Peaceful Antagonist
Silent Protagonist
[✓ Aggressive Protagonist]
Cowardly Protagonist
Talkative Enemies
[✓ Cunning Enemies]
Peaceful Enemies
Silent characters
[✓ Aggressive characters]
Cowardly characters

Notes:
A sub-setting that simulates an average fantasy, with agents, locations, and actions appropriate to the subgenre.

Files settings:
Output files path:
Select path
Output files name: newStoryGraph

Start
Exit

Fig. 24 Graphical user interface.

The user interface is a form divided into several zones, each of which contains a certain set of controls grouped according to their functional similarity.

In the Setting and Sub-Setting sections the user can select the setting in which the narrative will be generated. In the Goal-type section, it is possible to select the types of goals that agents will be guided. In the System settings section are grouped randomization settings for various generation elements: whether to randomly generate links between locations or use preset ones, whether random encounters should be present, whether the outcome of fights should be determined randomly, whether agents should act in a random order or in a predetermined one. It is also possible to define here: the seed for the random number generator, the maximum number of nodes in the story graph, whether nodes and edges created by "empty" actions should be hidden, and enter the key from the OpenAI API. In the Story settings section, settings are specified that apply specifically to the generated story: is it possible and with what chance for agents to find evidence against the antagonist, should the protagonist and antagonist be guaranteed to "survive" a certain time and if so, how long, unique descriptions for the actions of the antagonist and whether he should follow a certain order of action in order to achieve his goals. In the Character behavior section is configured the behavior of agents: whether they should behave more aggressively or more peacefully, be more sociable and cunning or vice versa, etc. In the Files settings section is possible to choice and display the path where will be written the output files created by the program and also possible choose their name (it will be the same for a .dot file with a description of the story graph and an image generated on its basis). In the Notes field is displayed auxiliary information: initially, there is a brief instruction for configuring and launching the program, then there is displayed information about selected element, and after launch there is displayed the status of the program (what exactly action it performs, i.e. on what stage is it). Also there is a Start button, clicking on which will start the generation process, if all the minimum required settings have been specified.

This interface was created using the Windows Forms system. It provides the ability to select and place all the required elements in the graphic editor (`GroupBox` - for creating sections, i.e. combining other elements into groups, `RadioButton` - a logical element that allows select (set the value to "true") only one element this type from of the entire group, `CheckBox` - also a logical element, but not limited in number, `TextBox` - a standard text field, with the possibility of user input, `Label` - displays a predefined text, without support for user input, `Button` - an element to which are attached a some number of events (triggers), allowing to flexibly define its behavior, for example, perform certain actions when pressed). Once placed, the behavior of these elements can be configured in the appropriate class. In our case, this is the `MainWindow` class. In this class also declared an object of the `StoryAlgorithm` class that controls the run of the program. Here it directly receives information about the settings selected by the user (set by interacting with interface elements), as well as a command to start the generation process.

5.2.3 Implementation of the environment

The environment in which the agents operate is implemented using the `LocationStatic` and `LocationDynamic` classes. These classes represent an abstraction of a certain location (room/area) in which agents and various objects can be located. These locations are "logistically" connected, in the sense that a link can be established between each pair of locations. Such a connection, when the corresponding rule is activated, will serve as a condition for the possibility of moving agents between these locations.

The representation of a single entity by dividing it into two classes is due to the fact that part of the data is set once during the existence of this entity, and part of the data can change an indefinite number of times during the program's operation, overwritten. Accordingly, static data and dynamic data use their own classes. They are combined into a single entity using data structures such as `KeyValuePair` and `Dictionary`.

The `LocationStatic` class contains information about the name of the location and what other locations are associated with it, as well as methods for setting/getting this information, and also changing it.

The `LocationDynamic` class stores information about which agents are in it, information about the presence (or absence) of evidence and information about items that are stored in this location. Similarly, methods for setting/getting this information and changing it.

5.2.4 Agent implementation

Agents are implemented in a similar way, where a single entity representing an agent is divided into two classes. One of these classes stores immutable or rarely changed information (this is the `AgentStateStatic` class), and the second information that changes during operation (this is the `AgentStateDynamic` class). Also, to combine two components into a single entity, used the `KeyValuePair` and `Dictionary` data structures.

The `AgentStateStatic` class stores information about the agent's name and role.

The role is implemented using an enumerator and serves as a behavior template for agents, without the need to manually define it individually for each. For example, the availability of actions, goals, as well as some features of obtaining perception are determined by being tied to the role of the agent.

The `AgentStateDynamic` class stores information: about the agent's plan, the list of actions available to him, his status (alive/dead), information about the agent's goals, his "emotional" state (scared, angry), his intentions (go to a certain place, or lure another agent to a specific location) and list of items belong to agent. It also stores information about the agent's beliefs, and his assumption of the world around him (environment, other agents). And, of course, it contains methods for assigning, receiving and changing this information, including by accessing third-party modules. For example, to the planner.

The emotional state of an agent implemented using separate classes, each of which serves to express a certain emotion. Objects of these classes are stored in the dynamic part of the agent. These classes are generally similar to each other, containing information about whether the agent is experiencing a given emotion at the current moment or not, as well as some specific information about it. For example, if one agent is angry with another agent, then its `AgentAngryAt` class object will store information about the agent with which he is angry (a reference to him).

The agent's beliefs are expressed primarily by holding its own instance of the `WorldContext` class. It stores what the agent thinks about the world and other agents. We note separately that he only believes, but does not know, since this information may be incomplete and incorrect. So, it stores: the agent's assumption about where he is, about other agents in the location where he is, in general - about the agents and locations present in the story world (of which the agent is aware, and not about all in general) . Beliefs about agents are also stored using the special `BeliefsAboutAgent` class, rather than directly using classes that represent agents specifically. This class stores the agent's assumptions about the name, role, status, location, and anger level of some other agent. Also, beliefs are additionally expressed remembering by agents certain facts. For example, about finding evidence or a list of explored locations.

5.2.5 Player implementation specifics

Player representation has its own challenges. He must have several options for choosing how to act, and at the same time it is difficult for us to assume what exactly he will do. At the same time, since we are creating a plan that provides for all the options for the development of the story, we cannot leave it to itself somehow decide during the runtime. The player's actions must also be part of the plan, and must not conflict with it.

We consider the player as an agent, but with a unique role. When `storyworldConvergence` sends an action request to this agent, it is handled differently. The `ActionRequest` method has a separate fork that overrides the algorithm if the current agent has the role of "player". Unlike the situation with an ordinary agent, the stage of drawing up a plan is skipped here. After determining the actions available to the agent-player, no one is chosen to be performed, but all they begin to be performed sequentially (using a cycle). However, some actions may have options within themselves. For example, when one certain location is connected to several others and it is possible to move to each of them. In this case, each such action option is processed as a separate action.

It is worth bearing in mind that we are considering a human player as someone who only controls the character assigned to control, but is not him. Therefore, the beliefs of the player agent, his goals, perceptions, and everything else are calculated in the same way as for other agents. And the set of actions available for performing (from which, in fact, the player will choose) depends on the same set of factors as with ordinary agents.

5.2.6 Implementation of the story graph, graph nodes and states of the story world

The state of the story world is also implemented with a division into two classes. The one that stores immutable or rarely changed information (`WorldStatic` class). And the one that stores frequently changed information (`WorldDynamic` class). Also, similarly, they are combined into a single entity using the `KeyValuePair` and `Dictionary` data structures.

`WorldStatic` stores information about a list of all locations in the story world (meaning their static information, name and links to other locations), as well as settings relevant to this story. The settings are displayed using enumerators and booleans values.

`WorldDynamic` stores information about the current state of all locations and information about the current state of all agents.

The story graph is represented using the `StoryGraph` class, which stores a set of nodes and a link to the root node.

Each node is implemented using the `StoryNode` class. It stores: the state of the story world, information about the active agent, serial number and information about connections with other nodes. Information about these connections is expressed using a set of edges that connect two nodes in the graph. The `Edge` class is used to implement edges. Each object of this class stores references to two nodes that are connected by this edge (marked as top and bottom). And also information about the action, the application of which by the agent led to the formation of this edge (the transition of the story world from one state, which is stored in the upper node, to a new state, which is stored in the lower node).

5.2.7 Implementing goal states

By the goals that agents are trying to achieve, we understand a certain state of the story world that satisfies a number of agent-specific conditions. For example, in a detective setting, the goal of ordinary agents and the player may be to neutralize the killer. Therefore, all states in which the killer is dead, in this example, will be considered as a goal. When constructing a graph, nodes with such states in which this condition is met are considered finite and are not expanded.

For representation, we use the `Goal` class. This class stores an indication of the types of goal and the goal state itself, in which the condition has already been met, allowing it to be considered the goal standard. Goal types come in three types and are represented using the `GoalTypes` enumerator. These include the status (alive/dead) of certain agents, the presence of an agent in a certain location, an agent possession by a certain object. Dividing goals into several types allows them to be more flexible in their specification, and check only the necessary aspects of the state when checking its compliance with the goal, rather than making a full comparison.

The control of whether a certain state corresponds to the goal state occurs in the `ControlToAchieveGoalState` method of the `GoalManager` class. In it, the check occurs in such a way that first the key specifications of a particular state are read (whether the player is alive, whether the antagonist is alive, the number of surviving agents, etc.), and then, in accordance with the specified type of goal, made specialized checks corresponding to it.

Also, the agent's goal type affects the choice of intent when building a PDDL file describing a planning problem. For example, an agent who has the goal of finding an antagonist will systematically move around and research locations in search of evidence. While the antagonist himself will try to stay in the locations along with a small number of other agents, or lure them into empty locations in order to eliminate them later.

5.2.8 Implementation of actions

Actions serve to transform one state of the world (current) into another (next). They have a precondition that must be met for the action to be applied, as well as effects, which are things that will be changed in the state of the world.

They are implemented using the `PlanAction` virtual class. This class contains all the variables and methods that are used by the derived classes. In particular: arguments, success status, description, methods for adding arguments, displaying a description, checking preconditions, applying effects (if the action succeeds), and behavior when the action fails.

Based on this class created separate descendant classes for each required action. For example: `Move` to move, `Talk` to talk, `Kill` to change the status of an agent, etc. Each individual class redefines: its arguments, descriptions and algorithms for checking preconditions and applying effects, in order to match the specifics of a particular class.

To check preconditions used the `CheckPreconditions` method. It is passed a verifiable state of the story world, which is checked for compliance with the preconditions. For example, in the `Move` class checks the status of the agent performing the action (passed as an argument), as well as the presence of this agent at the point of departure and its absence at the point of arrival (the names of both are also passed as arguments). If all preconditions are met, then the method returns a boolean value - true, if at least one is not met, then returns a value - false. Accordingly, agents can at least simply add them to their list of actions available for execution only when this check is passed successfully.

There is also a `PreCheckPrecondition` method, used in cases where arguments have not yet been passed to this class, to write them to variables. This method allows to use them as his own parameters.

To apply effects used the `ApplyEffects` method. Unlike the `CheckPreconditions` method, it is passed not a copy of the state of the world, but a reference to the current state of the story world, which this method directly changes. For example, in the same `Move` class, the moving agent is removed from the location in which it is located and added to the one it was heading to. The agent's beliefs also change accordingly. In this case, the passed arguments are only used to identify (by name or id) their originals in a mutable state. All manipulations take place precisely with the state, directly with locations and agents getting from it, but with nothing passed as an argument.

If the action failed then called the `Fail` method. This method defines the appropriate behavior for the action. In the already mentioned `Move` class, the action will simply be marked as unsuccessful (with the help of the corresponding boolean variable), but in the `Kill` class (with the rule for random battle results enabled), this method describes the outcome of the action directly opposite to that which happens when it is successfully executed, those the killer dies.

5.2.9 Constraint implementation

Constraints are used to maintain the consistency and coherence of the story, as well as to support the author's intent. They represent various conditions, the fulfillment (or non-fulfillment) of which by the agents and the player will force the system to react to their behavior, responding to their actions with counter-actions (or, in fact, replacing their action with another one).

Constraints are implemented using the abstract class `WorldConstraint`. Based on this class implemented derived classes for more specialized constraints. To restrict access to a location used the `RestrictingLocationAvailability` class, which defines 11 types of possible options for restricting movement or being in locations. For example: a ban on re-visiting, an indication that a location should be visited only once, or at least once, a ban on moving before another agent arrives at a certain place or leaves a certain place, etc. Each type has its own validation algorithm, even within the same class. But the validation method inherited from `WorldConstraint` requires immediately providing it with all the necessary data that may be required for any type of constraint (checked state, graph, check type, target locations, target agents, expiration, etc.). The output of the check result always occurs in a boolean format, implying an answer to the question - is the constraint satisfied or not. To limit the communication between agents used the `ForbiddenToSpeak` class. For restrictions on the status (alive/dead) of agents used the `ConstraintAlive` class. To restrict the direct actions of agents used the `ActionsRestricting` class.

During the generation process, for each specific constraint (with a specific type and parameters) created its own object that stores all the necessary information for a specific constraint. Examples of such specific constraints are: "it is not allowed to put an antagonist in the status of "dead" during the first five turns", "a specific agent must visit each location at least once", "a hero agent cannot leave the village location without completing actions to complete the quest" and so on. The presence of this or that constraint is determined by the settings. All generated objects representing constraints are added to the target manager's list, which stores them.

While validating the action (for its correctness), in the loop, all objects representing the constraints are accessed (their list is provided by the constraint manager), with a call to their method that checks the compliance of the resulting state of the story world with the specified conditions.

5.3 External module – planner

To implement planning by agents, we use a third party program called Fast Downward [91]. To it are transferred two files: the planning domain and the planning problem, in the PDDL format. In response, this planner returns a text file with a plan.

For direct work with Fast Downward [90] used the `FastDownwardWrapper` class, which defines two methods: `Run` and `RunFastDownward`. In the `Run` method created a process, its settings are specified, and then the process is started. In this case, it means launching the command line with the required command to activate Fast Downward [91] and specify the files to use for it. The `RunFastDownward` method defines parameters for running the `Run` method. Such as: launching the command line, specifying the executable file, choosing a method for planning, the name of the used PDDL files. Then from it is launched the `Run` method. And it returns a boolean value characterizing the success of the launch (or, conversely, failure).

The `FastDownward` class contains methods for: launching the wrapper, processing the plan (reading the file with the plan, parsing the data in it, converting them into program entities - objects of certain classes perceived by the generator program) and error detection.

5.4 External module - graph visualizer

To visualize graphs, we also use an external module, a program called Graphviz [91]. It is presented in several versions, available both for installation on a personal computer and for online use. Since we are creating a file in the .dot format, and any of the versions of this program can process it, it is permissible to use any version. Our implementation uses the desktop version, which is launched using the command line. This choice was made due to the fact that web versions allocate limited resources to users for generation and often do not allow scaling the resulting image in good quality. While in some cases we generate very large graphs. Generating such large graphs requires some time and computing power, and quality-preserving scaling is necessary in order to be able to consider their arbitrary details.

As in the case of Fast Downward [91], for the direct work of Graphviz [92] we use the `GraphvizWrapper` class, which defines two methods: `Run` and `RunGraphviz`. Functionally, they are generally similar to methods from the `FastDownwardWrapper` class. Similarly, in the `RunGraphviz` method, the wrapper is initialized and launched. In the `Run` method, is created, initialized and configured (indicating the using executable file, working directory, commands, etc.) the process, which starts processing the system-generated .dot file and generating the corresponding visualization image based on it.

The creation of a file in the .dot format, which is passed to Graphviz [92], is described in the `GraphConstructor` class. It occurs in such a way that all nodes and edges in the graph are traversed in a cycle, their characteristic specificities are determined (for example, is the state stored by the node, goal or not). Then, in accordance with the received data, generated a text corresponding to the dot graph description language, which is added to a file of the appropriate format. Some user-defined settings (for example, do not display empty actions on the graph) affect the appearance of the generated graph. In this case, special algorithms are used to eliminate unnecessary nodes (and edges, respectively), and then correctly connect the nodes and edges remaining in the graph, preventing the occurrence of "breaks".

5.5 External module - a platform for writing interactive literature, text games and visual novels

As a platform that could playback the story based on the graph we generated, at the same time allowing the player to interact with it interactively, we chose Twine. This platform is available both as a desktop program and as a web application. It also provides both the ability to create user stories (acting as an editor) and upload already created ones. To do this, it uses HTML files, as well as its own language (in fact, there is even a choice of several ways to present information, but we worked with the default one). To start working with this platform, it is enough to upload the generated HTML file into it, no additional actions or settings are required. A running story (or a text-based game) appears as text in a window, which, depending on the purpose, is formatted accordingly (for example, text, clicking on which will lead to a transition to another state, is highlighted in blue, underlined and has larger font).

Since the interface for interacting with Twine is only graphical, we did not create any wrappers for working with this platform. Interaction is carried out manually. Methods for converting the generated story graph into a format understood by Twine are described in the

TwineGraphConstructor class. It does this by adding the necessary code that tells Twine how to interact with the given story, how to display, what title, where the entry point, etc., into the appropriate file. Then there is a loop through the generated graph, in which each node and each edge is considered separately. Based on their type, properties, and stored information, an appropriate block of code is created to write to the HTML file for Twine. For example, read information in which location the player-agent is located, what other agents are there, converted into a connected sentence in natural language (English), which is written in the same capacity to the created file, so that the player has information about the environment. In the OpenAI class is described the CompletionRequest method, with the help of which (by creating an HTTP Client and using it to send a POST request to the specified URL, specifying authorization parameters and request type, and catching the response using HttpResponseMessage) the graph constructor for Twine can use the model to generate text, passing it its own generated text as a prompt, and receiving from it a more extended version, more colorful and similar to that written by a human. The edges are also converted to text, but already allowed to interact with them, redirecting to a new state when clicked. This text itself is created on the basis of the action recorded in this data structure. Since the player may be able to view and act only in the story world on their turn when the player agent had the opportunity to do so, states in which the player agent could not act are not included in the description. It is transformed in such a way that the edges (actions) lead either to those nodes in which the player agent acts, or to those that are created by the constructor to provide additional information. Thus, the actions of other agents themselves remain "behind the scenes" for the player, he is faced only with their results.

Conclusion

Within the framework of this thesis, we explored various approaches to creating interactive stories, figured out what makes them interesting, what are their most important features, and how they can be applied in practice. Based on this, we have formulate our own goal - on base procedural narrative generator to create a tool that allows to visualize all possible ways of developing a narrative in a given domain (which, from our point of view, should be useful to game designers, instead of calculations "manually"), and to generate interactive stories (quests). We found out that this method of using procedural narrative generators is still a little-studied area.

To fulfill our goal, we have developed an algorithm that allows speeding up the creation of an interactive narrative, which usually takes quite a long time. At the same time, we applied an approach that we see as quite innovative – a combination of high emergence and centralized regulation with the help of a Drama Manager. We believe that it allows achieving several goals at once: to speed up the generation of narrative, and combine the believability of agents from emergent narratives with the consistency and coherence of narratives based on centralized regulation and planning. We were encouraged by the results, because initially we were going to limit ourselves to very short narratives in the detective genre and use a very small domain for this. However, in the end we were able to significantly complicate the domain and add a second genre. At the same time, generation still takes place in a reasonable time and narratives of more complex stories are created than we originally planned. In addition, we managed to integrate into our system the ability to use language models for text generation based on neural networks from OpenAI, which allow us to express the narrative generated by our system using texts similar to those created by a human, which allows us to create of full value interactive stories with internal coherence, consistency and expressed in the appropriate way. We also showed that it is possible to visualize the narrative generated by our system in an understandable way, and to create an interactive story (quest game) from it, which will be possible to play on a third-party platform.

Finally, we described our technical implementation of the story generation system. In the process of working on it, we somewhat went beyond the originally planned area of work, because we saw several interesting opportunities.

Future work

In the process of working on this thesis, we saw many opportunities to expand it. For start, we would like to improve the author's capabilities in the sense of modifying and editing the story. We see a rather successful and elegant solution for it in using downloadable "scripts", in the form of separate files that the author could edit, completely customizing the domain without having to correct the system code for such changes. We also see further opportunities for improvement by expanding the model of relationships between agents to make them even more plausible. Another interesting opportunity for further work is to introduce and control dramatic structures, which would improve the experience that such systems should provide to users. A logical step for improvement is also seen to complicate quests, adding sub-goals to them, opportunities to achieve the main goal without finishing the story, but starting a new one in the already changed story world. We also consider an increase in the number of platforms to which the generated narrative can be transferred to create a quest to be a good opportunity for improvement. For example, limit ourselves not only to Twine, but also expand to Squiffy and Unity [87].

We understand that all this together is a huge and complex job, but even each of the mentioned features individually can significantly improve our system. The implementation of several at once will be a noticeable advance in the field under study, since there are not so comprehensive works in it yet.

Bibliography

1. CLEMENT, Jessica. Number of video gamers worldwide in 2021, by region. In: *Statista* [online]. 25.10.2022 [accessed 2023-05-26]. Available from: `https://www.statista.com/statistics/293304/number-video-gamers/`

2. STRAUSS, Ben. Video Games Are Art, Says U.S. Government. In: *Business Insider* [online]. 18.05.2011 [accessed 2023-05-26]. Available from: `https://www.businessinsider.com/video-games-are-art-says-us-government-2011-5`

3. WIJMAN, Tom. The Games Market and Beyond in 2021: The Year in Numbers. In: *Newzoo* [online]. 22.12.2021 [accessed 2023-05-26]. Available from: `https://newzoo.com/insights/articles/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming`

4. HENDRIKX, Mark, Sebastiaan MEIJER, Joeri VAN DER VELDEN and Alexandru IOSUP. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications* [online]. 2013, 9(1), p. 1-22 [accessed 2023-05-26]. ISSN 1551-6857. Available from DOI: `10.1145/2422956.2422957`

5. TOGELIUS, Julian, Alex J. CHAMPANDARD, Pier L. LANZI, Michael MATEAS, Ana PAIVA, Mike PREUSS and Kenneth O. STANLEY. Procedural Content Generation: Goals Challenges and Actionable Steps. *Artificial and Computational Intelligence in Games* [online]. 2013, 6, p. 61-75 [accessed 2023-05-26]. ISSN: 1868-8977. Available from DOI: `10.4230/DFU.Vol6.12191.61`

6. AMATO, Alba. Procedural Content Generation in the Game Industry. In: KORN, Oliver and Newton LEE, ed. *Game Dynamics* [online]. Cham: Springer International Publishing, 2017, 31.03.17, p. 15-25 [accessed 2023-05-26]. ISBN 978-3-319-53087-1. Available from DOI: `10.1007/978-3-319-53088-8_2`

7. FRONTIER DEVELOPMENTS. *Elite Dangerous* [software]. 2014 [accessed 2023-05-26]. Available from: `https://www.elitedangerous.com/`. System requirements: OS Windows 8/10 64-bit, CPU Quad Core CPU (4 x 2Ghz), memory 6 GB RAM, GPU Nvidia GTX 470/AMD R7 240.

8. HELLO GAMES. *No Man's Sky* [software]. 2016 [accessed 2023-05-26]. Available from: `https://www.nomanssky.com/`. System requirements: OS Windows 7, CPU Intel Core i3, memory 8 GB RAM, GPU Nvidia GTX 480.

9. GEARBOX SOFTWARE. *Borderlands* [software]. 2009 [accessed 2023-05-26]. Available from: `https://borderlands.com/`. System requirements: OS Windows XP/Vista, CPU 2.4 Ghz or equivalent processor with SSE2 support, memory 1GB RAM (2GB recommended with Vista), GPU 256mb video RAM or better.

10. MOJANG STUDIOS. *Minecraft* [software]. 2009 [accessed 2023-05-26]. Available from: `https://www.minecraft.net/`. System requirements: OS Windows 7, CPU Intel Core i3 3210 | AMD A8 7600 APU or equivalent, memory 4 GB RAM, GPU Nvidia GeForce 400 Series or AMD Radeon HD 7000 series with OpenGL 4.4.

11. INTERACTIVE DATA VISUALIZATION, INC. *SpeedTree* [software]. 2002 [accessed 2023-05-26]. Available from: `https://store.speedtree.com/`. System requirements: OS Windows 7/8/10/11, CPU Multi-core Intel series or higher, Xeon or AMD equivalent, memory 4GB RAM.

12. ESRI R&D CENTER ZURICH. *CityEngine* [software]. 2008 [accessed 2023-05-26]. Available from: `https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview`. System requirements: OS Windows 8/10, CPU 2 GHz dual-core (minimum Core2 Duo compatible Intel/AMD), memory 16 GB RAM, GPU Nvidia—GeForce 600 and later / Quadro 600 and later or AMD-Radeon HD 7000 and later.

13. FREIKNECHT, Jonas and Wolfgang EFFELSBERG. A Survey on the Procedural Generation of Virtual Worlds. *Multimodal Technologies and Interaction* [online]. 2017, 1(4) [accessed 2023-05-26]. ISSN 2414-4088. Available from DOI: `10.3390/mti1040027`

14. KYBARTAS, Ben and Rafael BIDARRA. A Survey on Story Generation Techniques for Authoring Computational Narratives. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2017, 9(3), p. 239-253 [accessed 2023-05-26]. ISSN 1943-068X. Available from DOI: `10.1109/TCIAIG.2016.2546063`

15. ABBOTT, H. Porter. *The Cambridge Introduction to Narrative* [online]. Cambridge University Press, 2014 [accessed 2023-05-26]. ISBN 9780521887199. Available from DOI: `10.1017/CBO9780511816932`

16. TURTLE ROCK STUDIOS. *Left4Dead* [software]. 2008 [accessed 2023-05-26]. Available from: `https://www.l4d.com/`. System requirements: OS Windows 7/Vista/XP, CPU Pentium 4 3.0GHz, memory 1GB RAM, GPU 128 MB, Shader model 2.0, ATI 9600, NVidia 6600 or better.

17. BETHESDA GAME STUDIOS. *The Elder Scrolls V: Skyrim* [software]. 2011 [accessed 2023-05-26]. Available from: `https://elderscrolls.bethesda.net/skyrim`. System requirements: OS Windows 7/8.1/10 (64-bit Version), CPU Intel i5-750/AMD Phenom II X4-945, memory 8 GB RAM, GPU Direct X 9.0c compliant video card with 512 MB of RAM.

18. Skyrim:Radiant. In: *Unofficial Elder Scrolls Pages* [online]. 2011- [accessed 2023-05-26]. Available from: `https://en.uesp.net/wiki/Skyrim:Radiant`

19. Are Radiant Quests Having An Adverse Effect Open World Games and Unique Quests Within?. In: *Reddit* [online]. 15.10.2016 [accessed 2023-05-26]. Available from: https://www.reddit.com/r/Games/comments/57no58/are_radiant_quests_having_an_adverse_effect_open/

20. YOUNG, Robert and Mark RIEDL. Towards an architecture for intelligent control of narrative in interactive virtual worlds. In: *Proceedings of the 8th international conference on Intelligent user interfaces* [online]. New York, NY, USA: ACM, 2003, 12.01.2023, p. 310-312 [accessed 2023-05-26]. ISBN 1581135866. Available from DOI: 10.1145/604045.604108

21. JOHNSON, Daniel. Animated Frustration or the Ambivalence of Player Agency. *Games and Culture* [online]. 2015, 10(6), p. 593-612 [accessed 2023-05-26]. ISSN 1555-4120. Available from DOI: 10.1177/1555412014567229

22. HROMADA, Tomáš, Martin ČERNÝ, Michal BÍDA and Cyril BROM. Generating Side Quests from Building Blocks. In: SCHOENAU-FOG, Henrik, Luis Emilio BRUNI, Sandy LOUCHART and Sarune BACEVICIUTE, ed. *Interactive Storytelling* [online]. Cham: Springer International Publishing, 2015, 11.12.2015, p. 235-242 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-319-27035-7. Available from DOI: 10.1007/978-3-319-27036-4_22

23. KLIMAS, Chris. *Twine* [software]. 2009 [accessed 2023-05-26]. Available from https://twinery.org/

24. FAGGELLA, Daniel. What is Artificial Intelligence? An Informed Definition. In: *Emerj Artificial Intelligence Research* [online]. 21.12.2018 [accessed 2023-05-26]. Available from: https://emerj.com/ai-glossary-terms/what-is-artificial-intelligence-an-informed-definition/

25. WOOLDRIDGE, Michael J. *An introduction to multiagent systems*. New York: J. Wiley, 2002, 488 p. ISBN 0-471-49691-x.

26. LAWTON, Graham. AI can predict your future behavior with powerful new simulations. In: *New Scientist* [online]. 2.10.2019 [accessed 2023-05-26]. Available from: https://www.newscientist.com/article/mg24332500-800-ai-can-predict-your-future-behaviour-with-powerful-new-simulations/

27. GHALLAB, Malik, Dana S. NAU and Paolo TRAVERSO. *Automated planning and acting* [online]. New York, NY: Cambridge University Press, 2016 [accessed 2023-05-26]. ISBN 9781107037274. Available from DOI: 10.1017/CBO9781139583923

28. BÜCHNER, Clemens, Patrick FERBER, Jendrik SEIPP and Malte HELMERT. A Comparison of Abstraction Heuristics for Rubik's Cube. In: *Proceedings of the ICAPS 2022 Workshop on Heuristics and Search for Domain-independent Planning* [online]. 16.06.2022 [accessed 2023-05-26]. Available from:

https://ai.dmi.unibas.ch/papers/buechner-et-al-icaps2022wshsdip.pdf

29. WARE, Stephen G., Edward T. GARCIA, Alireza SHIRVANI and Rachelyn FARRELL. Multi-Agent Narrative Experience Management as Story Graph Pruning. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* [online]. 2019, 15(1), p. 87-93 [accessed 2023-05-26]. ISSN 2334-0924. Available from DOI: 10.1609/aiide.v15i1.5229

30. FIKES, Richard E. and Nils J. NILSSON. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* [online]. 1971, 2(3-4), p. 189-208 [accessed 2023-05-26]. ISSN 00043702. Available from DOI: 10.1016/0004-3702(71)90010-5

31. FOX, Maria and Derek LONG. PDDL+: Modeling continuous time dependent effects. In: *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space* [online]. 27.10.2002 [accessed 2023-05-26]. Available from: https://planning.wiki/_citedpapers/pddlplus2003.pdf

32. BARTÁK, Roman. Constraint Satisfaction for Planning and Scheduling. In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling* [online]. 3.06.2004 [accessed 2023-05-26]. Available from: https://ktiml.mff.cuni.cz/~bartak/ICAPS2004/download/ICAPS_2004_Tutorial.pdf

33. SHAKER, Noor, Julian TOGELIUS and Mark J. NELSON. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research* [online]. Cham: Springer International Publishing, 2016 [accessed 2023-05-26]. Computational Synthesis and Creative Systems. ISBN 978-3-319-42714-0. Available from DOI: 10.1007/978-3-319-42716-4

34. MOURATO, Fausto. *Enhancing automatic level generation for platform videogames* [online]. Lisbon, 2015 [accessed 2023-05-26]. Dissertation. NOVA University Lisbon, FCT NOVA - NOVA School of Science and Technology. Dissertation supervisors Manuel Santos, Fernando Birra. Available from: https://run.unl.pt/bitstream/10362/18497/1/Mourato_2015.pdf

35. PEREZ, Mijael R. Bueno, Elmar EISEMANN and Rafael BIDARRA. A Synset-Based Recommender Method for Mixed-Initiative Narrative World Creation. In: MITCHELL, Alex and Mirjam VOSMEER, ed. *Interactive Storytelling* [online]. Cham: Springer International Publishing, 2021, 04.12.2021, p. 13-28 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-030-92299-3. Available from DOI: 10.1007/978-3-030-92300-6_2

36. SMITH, Gillian, Jim WHITEHEAD and Michael MATEAS. Tanagra: A mixed-initiative level design tool. In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games* [online]. New York, NY, USA: ACM, 2010, 19.06.2010,

p. 209-216 [accessed 2023-05-26]. ISBN 9781605589374. Available from DOI: 10.1145/1822348.1822376

37. WIELK, Olaf. Story vs. Narrative. In: *Beemgee* [online]. 18.12.2014 [accessed 2023-05-26]. Available from: https://www.beemgee.com/blog/story-vs-narrative/

38. MARTIN, J. Interactive Narrative vs. Interactive Storytelling. In: *Between drafts* [online]. 8.04.2022 [accessed 2023-05-26]. Available from: https://betweendrafts.com/2022/04/08/interactive-narrative-vs-interactive-storytelling/

39. BOSTAN, Barboros and Tim MARSH. Fundamentals Of Interactive Storytelling. *AJIT-e: Online Academic Journal of Information Technology* [online]. 2012, 3(8), p. 19-42 [accessed 2023-05-26]. ISSN 1309-1581. Available from DOI: 10.5824/1309-1581.2012.3.002.x

40. ARINBJARNAR, Maria, Hether BARBER and Daniel KUDENKO. A Critical Review of Interactive Drama Systems. In: *AISB Symposium* [online]. 08.04.2009 [accessed 2023-05-26]. Available from: https://www.cs.uky.edu/~sgware/reading/papers/arinbjarnar2009critical.pdf

41. HERVAS, Raquel, Antonio SANCHEZ-RUIZ-GRANADOS, Pablo GERVAS and Carlos LEON. Calibrating a Metric for Similarity of Stories against Human Judgment. In: *International Conference on Case-Based Reasoning* [online]. 28.09.2015 [accessed 2023-05-26]. Available from: https://ceur-ws.org/Vol-1520/paper14.pdf

42. RIEDL, Mark and Robert YOUNG. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research* [online]. 2010, 39, p. 217-268 [accessed 2023-05-26]. ISSN 1076-9757. Available from DOI: 10.1613/jair.2989

43. BATES, Joseph. *The Oz Project* [software]. 2002 [accessed 2023-05-26]. Available from: https://www.cs.cmu.edu/afs/cs/project/oz/web/oz.html

44. SZILAS, Nicolas. *IDtension* [software]. 2003 [accessed 2023-05-26]. Available from: http://tecfa.unige.ch/perso/szilas/IDtension/install/. System requirements: having Java installed (JRE).

45. LAUREL, Brenda. *Computers as Theatre*. Boston: Addison-Wesley Longman Publishing Co, 1991, 211 p. ISBN 978-0-201-51048-5.

46. MATEAS, Michael. A Neo-Aristotelian Theory of Interactive Drama. In: *Papers from the 2000 AAAI Spring Symposium* [online]. 20.03.2000 [accessed 2023-05-26]. Available from: https://cdn.aaai.org/Symposia/Spring/2000/SS-00-02/SS00-02-011.pdf

47. RIEDL, Mark and Andrew STERN. Believable Agents and Intelligent Story Adaptation for Interactive Storytelling. In: GÖBEL, Stefan, Rainer MALKEWITZ and Ido IURGEL, ed. *Technologies for Interactive Digital Storytelling and Entertainment* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 1-12 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-540-49934-3. Available from DOI: `10.1007/11944577_1`

48. RIEDL, Mark and Robert YOUNG. An Intent-Driven Planner for Multi-Agent Story Generation. In: *Autonomous Agents and Multiagent Systems, International Joint Conference* [online]. New York City, New York, USA, 2004, vol. 2, p.186-193 [accessed 2023-05-26]. Available from DOI: `10.1109/AAMAS.2004.10054`

49. KAPADIA, Mubbasir, Jessica FALK, Fabio ZÜND, Marcel MARTI, Robert W. SUMNER and Markus GROSS. Computer-assisted authoring of interactive narratives. In: *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* [online]. New York, NY, USA: ACM, 2015, 27.02.2015, p. 85-92 [accessed 2023-05-26]. ISBN 9781450333924. Available from DOI: `10.1145/2699276.2699279`

50. SI, Mei, Stacy MARSELLA and Mark RIEDL. Integrating Story-Centric and Character-Centric Processes for Authoring Interactive Drama. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* [online]. 2008, 4(1), p. 203-208 [accessed 2023-05-26]. ISSN 2334-0924. Available from DOI: `10.1609/aiide.v4i1.18698`

51. LUDEON STUDIOS. *RimWorld* [software]. 2013 [accessed 2023-05-26]. Available from: `https://rimworldgame.com/`. System requirements: OS Windows XP, CPU Core 2 Duo, memory 2 GB RAM, GPU Intel HD Graphics 3000 with 384 MB of RAM.

52. PARADOX DEVELOPMENT STUDIO. *Crusader Kings* [software]. 2004 [accessed 2023-05-26]. Available from: `https://www.paradoxinteractive.com/`. System requirements: OS Windows XP, CPU Pentium II 450 MHz, memory 64MB RAM, GPU 4 Mb Video Card.

53. RIEDL, Mark. Incorporating Authorial Intent into Generative Narrative Systems. In: *Proceedings of the AAAI Spring Symposium on Intelligent Narrative Technologies II* [online]. 2009 [accessed 2023-05-26]. Available from: `https://faculty.cc.gatech.edu/~riedl/pubs/riedl-aaai-ss09.pdf`

54. PEDERSEN, Thomas Anthony, Tilde Hoejgaard JENSEN, Vladislav ZENKEVICH, Henrik SCHOENAU-FOG and Luis Emilio BRUNI. Considering Authorial Liberty in Adaptive Interactive Narratives. In: WÖLFEL, Matthias, Johannes BERNHARDT and Sonja THIEL, ed. *ArtsIT, Interactivity and Game Creation* [online]. Cham: Springer International Publishing, 2022, 10.02.2022, p. 181-188 [accessed 2023-05-26]. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. ISBN 978-3-030-95530-4. Available from DOI: `10.1007/978-3-030-95531-1_13`

55. TYCHSEN, Anders, Michael HITCHENS, Thea BROLUND and Manolya KAVAKLI. The game master. In: *Proceedings of the Second Australasian Conference on Interactive Entertainment* [online]. Sydney: Creativity & Cognition Studios Press, 2005, p. 215–222 [accessed 2023-05-26]. ISBN 978-0-9751533-2-1. Available from: `https://dl.acm.org/doi/pdf/10.5555/1109180.1109214`

56. BAY 12 GAMES. *Dwarf Fortress* [software]. 2006 [accessed 2023-05-26]. Available from: `https://www.bay12games.com/dwarves/`. System requirements: OS XP SP3 or later, CPU Dual Core - 2.4GHz+, memory 4GB RAM, GPU 1GB of VRAM - Intel HD 3000 GPU/AMD HD 5450/Nvidia 9400 GT.

57. PROPP, Vladimir. *Morphology of the Folktale* [online]. Second edition. Austin: University of Texas Press, 1968 [accessed 2023-05-26]. ISBN 9780292748095. Available from DOI: `10.7560/783911`

58. GREIMAS, Algirdas Julien. *Sémantique structurale: recherche de méthode*. Paris: Librairie Larousse, 1966, 262 p. ISBN 2030703141.

59. SOARES DE LIMA, Edirlei, Bruno FEIJÓ and Antonio L. FURTADO. Hierarchical generation of dynamic and nondeterministic quests in games. In: *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology* [online]. New York, NY, USA: ACM, 2014, 11.11.2014, p. 1-10 [accessed 2023-05-26]. ISBN 9781450329453. Available from DOI: `10.1145/2663806.2663833`

60. CROWTHER, William. *Colossal Cave Adventure* [software]. 1976 [accessed 2023-05-26]. Available from: `https://jerz.setonhill.edu/intfic/colossal-cave-adventure-source-code/`

61. AARSETH, Espen. Quest Games as Post-Narrative Discourse. In: RYAN Marie-Laure, ed. *Narrative across Media: The Languages of Storytelling*. Lincoln, NE, USA: University of Nebraska Press. 2004, p. 361-376. ISBN 978-0-8032-8993-2.

62. TOSCA, Susana. The quest problem in computer games. In: GOBEL, Stefan et al., ed. *Proceedings of 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment 2003* [online]. Darmstadt: Fraunhofer IRB Verlag. 2006 [accessed 2023-05-26]. Available from: `https://www.researchgate.net/publication/273946826_The_Quest_Problem_in_Computer_Games`

63. JENKINS, Henry. Game Design as Narrative Architecture. *Computer* [online]. 2004, 44(3), p. 118-130 [accessed 2023-05-26]. Available from: `http://www.madwomb.com/tutorials/gamedesign/Theory_HenryJenkins_GameDesignNarrativeArchitecture.pdf`

64. HITCHENS, Joe. Special Issues in Multiplayer Game Design. In: LARAMEE, François, ed. *Game Design Perspectives*. Hingham, MA, USA: Charles River Media, Inc., 2002. ISBN 978-1-58450-090-2.

65. MEEHAN, James. TALE-SPIN, An Interactive Program that Writes Stories. In: *Proceedings of the 5th international joint conference on Artificial intelligence* [online]. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1977, vol. 1, p. 91-98 [accessed 2023-05-26]. Available from: `https://www.ijcai.org/Proceedings/77-1/Papers/013.pdf`

66. LEBOWITZ, Michael. Planning stories. In: *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. Seattle, WA, USA: Psychology Press, 1987, p. 234–242. ISBN 9780805801668.

67. BATES, Joseph. Virtual Reality, Art, and Entertainment. *Presence: Teleoperators and Virtual Environments* [online]. 1992, 1(1), p. 133-138 [accessed 2023-05-26]. ISSN 1054-7460. Available from DOI: `10.1162/pres.1992.1.1.133`

68. SGOUROS, Nikitas. Dynamic, user-centered resolution in interactive stories. In: POLLACK, Martha, ed. *Proceedings of the Fifteenth international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, vol. 2, p. 990-995. ISBN 9781558604803.

69. MATEAS, Michael and Andrew STERN. Façade: An experiment in building a fully-realized interactive drama. In: *Game Developers Conference*. San Jose, CA, USA, 4-8 march, 2003.

70. MATEAS, Michael and Andrew STERN. *Façade* [software]. 2005 [accessed 2023-05-26]. Available from: `https://www.playablstudios.com/facade`. System requirements: OS Windows.

71. RIEDL, Mark. Actor Conference: Character-focused Narrative Planning. In: *The Georgia Institute of Technology website* [online]. 2002 [accessed 2023-05-26]. Available from: `https://faculty.cc.gatech.edu/~riedl/pubs/tr03-000.pdf`

72. RIEDL, Mark, C. J. SARETTO and Robert YOUNG. Managing interaction between users and agents in a multi-agent storytelling environment. In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems - AAMAS '03* [online]. New York, NY, USA: Association for Computing Machinery Press, 2003, p. 741-748 [accessed 2023-05-26]. ISBN 1581136838. Available from DOI: `10.1145/860690.860694`

73. YOUNG, Robert, Mark RIEDL, Mark BRANLY, Arnav JHALA, Richard MARTIN and C. SARETTO. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development* [online]. 2004, 1(1), p. 1-

29 [accessed 2023-05-26]. Available from: https://faculty.cc.gatech.edu/~riedl/pubs/jogd.pdf

74. RIEDL, Mark and Robert YOUNG. Character-Focused Narrative Generation for Execution in Virtual Worlds. In: BALET, Olivier, Gérard SUBSOL and Patrice TORGUET, ed. *Virtual Storytelling. Using Virtual RealityTechnologies for Storytelling* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, p. 47-56 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-540-20535-7. Available from DOI: 10.1007/978-3-540-40014-1_6

75. RIEDL, Mark and Andrew STERN. Failing Believably: Toward Drama Management with Autonomous Actors in Interactive Narratives. In: GÖBEL, Stefan, Rainer MALKEWITZ and Ido IURGEL, ed. *Technologies for Interactive Digital Storytelling and Entertainment* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 195-206 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-540-49934-3. Available from DOI: 10.1007/11944577_21

76. MAGERKO, Brian. Story Representation and Interactive Drama. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* [online]. 2005, 1(1), p. 87-92 [accessed 2023-05-26]. ISSN 2334-0924. Available from DOI: 10.1609/aiide.v1i1.18721

77. MOTT, Bradford W. and James C. LESTER. U-director: a decision-theoretic narrative planning architecture for storytelling environments. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems* [online]. New York, NY, USA: ACM, 2006, 08.05.2006, p. 977-984 [accessed 2023-05-26]. ISBN 1595933034. Available from DOI: 10.1145/1160633.1160808

78. THUE, David, Vadim BULITKO, Marcia SPETCH and Eric WASYLISHEN. Interactive Storytelling: A Player Modelling Approach. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* [online]. 2007, 3(1), p. 43-48 [accessed 2023-05-26]. ISSN 2334-0924. Available from DOI: 10.1609/aiide.v3i1.18780

79. BANGSO, Olav, Ole JENSEN, Finn JENSEN, Peter ANDERSEN and Tomas KOCKA. Non-Linear Interactive Storytelling Using Object-Oriented Bayesian Networks. *Proceedings of the International Conference on Computer Games: Artificial Intelligence* [online]. 2004 [accessed 2023-05-26]. Available from: https://vbn.aau.dk/ws/files/132449/nolist.pdf

80. BARBER, Heather and Daniel KUDENKO. Generation of Adaptive Dilemma-Based Interactive Narratives. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2009, 1(4), p. 309-326 [accessed 2023-05-26]. ISSN 1943-068X. Available from DOI: 10.1109/TCIAIG.2009.2037925

81. FAIRCLOUGH, Chris. *Story Games and the OPIATE System* [online]. Dublin, 2004 [accessed 2023-05-26]. Ph.D. dissertation. University of Dublin - Trinity College.

Dissertation supervisor Padraig Cunningham. Available from: https://www.scss.tcd.ie/publications/theses/phd/TCD-CS-PHD-2004-02.pdf

82. ARINBJARNAR, Maria and Daniel KUDENKO. Schemas in Directed Emergent Drama. In: SPIERLING, Ulrike and Nicolas SZILAS, ed. *Interactive Storytelling* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, 27.11.2008, p. 180-185 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-540-89424-7. Available from DOI: 10.1007/978-3-540-89454-4_24

83. SZILAS, Nicolas. IDtension: A narrative engine for interactive drama. *1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment* [online]. 2003 [accessed 2023-05-26]. Available from: http://nicolas.szilas.free.fr/research/Papers/Szilas_tidse03.pdf

84. PIZZI, David, Fred CHARLES, Jean-Luc LUGRIN and Marc CAVAZZA. Interactive Storytelling with Literary Feelings. In: PAIVA, Ana C. R., Rui PRADA and Rosalind W. PICARD, ed. *Affective Computing and Intelligent Interaction* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 630-641 [accessed 2023-05-26]. Lecture Notes in Computer Science. ISBN 978-3-540-74888-5. Available from DOI: 10.1007/978-3-540-74889-2_55

85. AYLETT, Ruth, Joao DIAS and Ana PAIVA. An affectively driven planner for synthetic characters. In: LONG, Derek, Stephen Smith, Daniel Borrajo and Lee McCleskey, ed. *Proceedings of ICAPS06 International Conference on Automated Planning and Scheduling* [online]. Cumbria, UK: AAAI, 2006, p. 2-10 [accessed 2023-05-26]. ISBN 978-1-57735-270-9. Available from: http://www.macs.hw.ac.uk/~ruth/Papers/planning/ICAPS06.pdf

86. DE LIMA, Edirlei Soares, Bruno FEIJÓ and Antonio L. FURTADO. Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning. In: *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* [online]. Rio de Janeiro, Brazil: IEEE, 2019, p. 144-153 [accessed 2023-05-26]. ISBN 978-1-7281-4637-9. Available from DOI: 10.1109/SBGames.2019.00028

87. DE LIMA, Edirlei Soares, Bruno FEIJÓ and Antonio L. FURTADO. Procedural generation of branching quests for games. *Entertainment Computing* [online]. 2022, vol. 43 [accessed 2023-05-26]. ISSN 18759521. Available from DOI: 10.1016/j.entcom.2022.100491

88. UNITY TECHNOLOGIES. *Unity* [software]. 2005 [accessed 2023-05-26]. Available from https://unity.com/. System reqirements: OS Windows 7/10 or macOS HighSierra 10.13+, CPU X64 architecture with SSE2 instruction set support, GPU DX10/11/12-capable or Metal-capable Intel and AMD.

89. MICROSOFT. *Microsoft Visual Studio* [software]. 1997 [accessed 2023-05-26]. Available from: `https://visualstudio.microsoft.com/`. System requirements: OS Windows, CPU 1.8 GHz or faster, memory 1GM RAM.

90. JETBRAINS. *IntelliJ IDEA* [software]. 2001 [accessed 2023-05-26]. Available from: `https://www.jetbrains.com/idea/`. System requirements: OS Windows 10 or macOS 10.15+ or Linux supports Gnome and KDE, memory 2GB RAM.

91. HELMERT, Malte and Silvia RICHTER. *Fast Downward* [software]. 2004 [accessed 2023-05-26]. Available from `https://www.fast-downward.org/`. OS Windows 10 or macOS 11 or Ubuntu 3.8, Python 3.8.

92. AT&T LABS. *Graphviz* [software]. 1991 [accessed 2023-05-26]. Available from: `https://graphviz.org/`

93. HELMERT, Malte. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* [online]. 2006, vol. 26, p. 191-246 [accessed 2023-05-26]. ISSN 1076-975. Available from DOI: `10.1613/jair.1705`

List of Abbreviations

PCG - Procedural Content Generation

AI - Artificial intelligence

MAS - Multi-agent system

APS - Automated planning and scheduling

STRIPS - Stanford Research Institute Problem Solver

PDDL - Planning Domain Definition Language

CSP - Constraint satisfaction problem

IS - Interactive Storytelling

POP - Partial-Order planning

EN - Emergent narrative

DAG - Directed acyclic graph

RPG - Role-playing game

NPC - Non-playable character

IDE - Integrated Development Environment

MVS - Microsoft Visual Studio

JIT - Just-in-time (compilation)

API - Application Programming Interface

CSS - Cascading Style Sheets

List of Figures

A Attachments

A.1 Source code

The source code is hosted on the Github repository (https://github.com/vonHimik/NarrativeGeneratorForTwine). As well there is an archive with a compiled program, ready for use immediately after unpacking. This archive also contains ready-to-use generated files for Twine corresponding to the examples given in the Use Cases section. It is possible to download them to Twine to see the final result, compare the interactive story and the presented visualization of its development paths.

A.2 User documentation

The repository (and also archive with the program) contains instructions (readme) for installing and using the program. Besides, it contains auto-generated documentation in HTML format that describes all used classes, methods, and variables.