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 2022

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. [8] 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. [9] And the revenues of the gaming industry for 2021 amounted to $180 billion. [10]

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. [1][3] 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 terms of development budget and production time. [1][2] 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 no human participation. [1][3]

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 and No Man's Sky, procedural generation methods are used to create a huge (count goes into the billions) number of detailed star systems, in Borderlands they are used to generate items with various properties, taking into account those features that the player uses most often, in Minecraft are used to generate biomes and ecosystems, and separate systems are also used in many games at once, for example, SpeedTree for generating vegetation and CityEngine for generating urban environments. [1][4] A separate type of procedural content generation, also developing over time, is the generation and adaptation of narratives. [5] 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 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. [1] The Elder Scrolls V: Skyrim 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. [1][11] According to player feedback, quests generated by this system are significantly less interesting than quests generated by traditional methods. [12] In such genres as text quests or visual novels, the very essence of which is interactive storytelling (visual novels) and a quest system (text quests), we cannot designate a single significant commercial game for a mass audience that would use methods of procedural content 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. [6] 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. [7] Another reason why commercial games don't 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 semantic conflict with each other. [62]

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 that 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 broadcasting them in finished form directly into a development tool, for example, into a platform for writing interactive literature, text games and visual novels - Twine, 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 work. The development part includes: chapter 3, in which we describe the construction of our framework, and also chapters 4 and 5, which present the proposed method for generating narrative and, as a specific subtype of it, the method for generating quests. The third part, which describes the implementation, is Chapter 6, 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. [13] 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 that are inspired: cooperation and coordination. [14] 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).

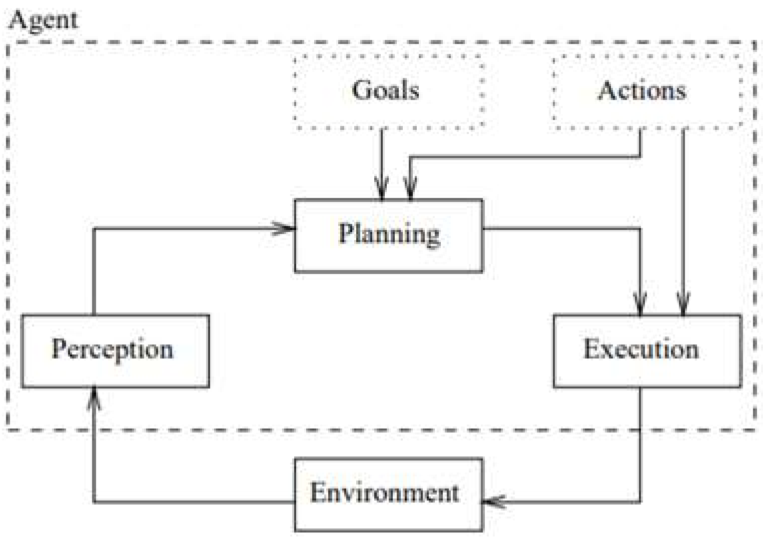


Fig. 1 Scheme of the simplest intelligent agent.

Our interest in MAC in the context of interactive storytelling and narrative generation is due to the fact that they are successfully used to model social systems, demonstrating social behavior, due to the fact that they can be quite fine-tuned. [15] 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). [20]

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. [21] 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. [22], 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

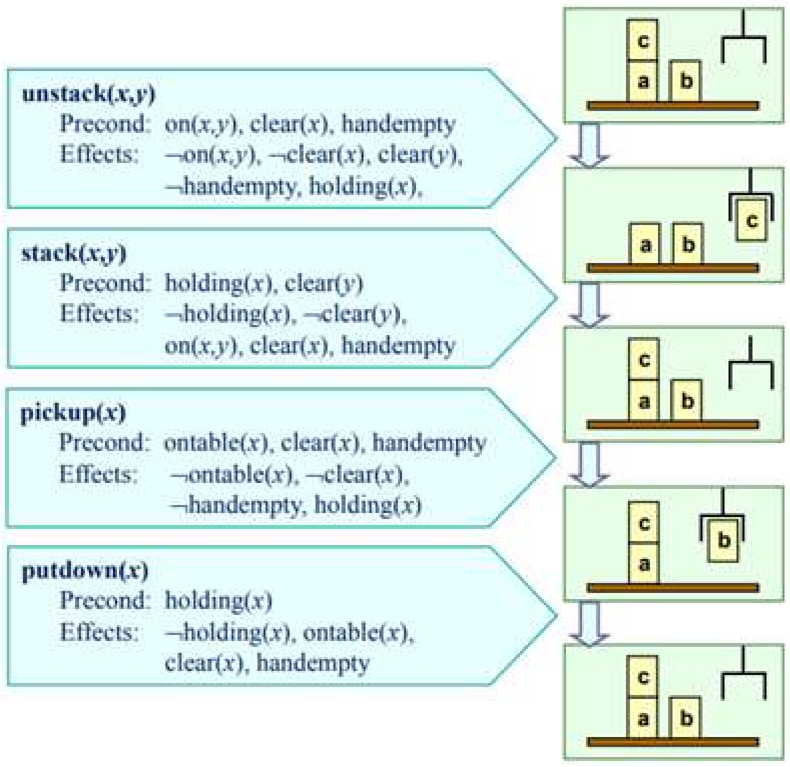
STRIPS (STanford Research Institute Problem Solver) was the first system that was created directly for solving planning problems. [23] Subsequently, the name STRIPS was also used to refer to the formal language describing the input data of this system. This language gained popularity and became the basis for most modern languages for describing automatic planning problems.

STRIPS offered a representation of the planning problem, which is now 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, and the action itself is an instance of the operator:

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

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 problem. 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 given 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. [20]



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
c
a
b

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. [24]

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.

The use 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 don't 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. [25]

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. [16] 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. [17]

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 or No Man’s Sky would be almost impossible to create manually) and at the same time provide a sufficient level of content diversity. [1][4]

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 (for example, [18]). 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 (for example, [19]).

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. [12] The use 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. [26] 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. [27] 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.

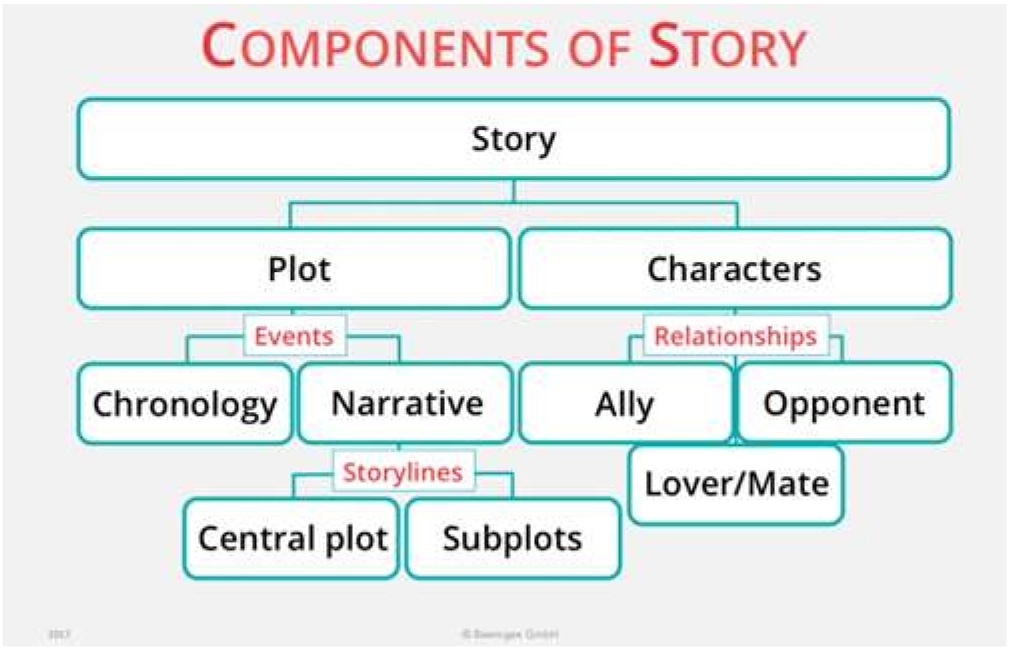


Fig. 3 Diagram of story components, showing that narrative is an integral part of a story, and from which it can be concluded that storytelling is the art of creating a story as a whole. Source: Story vs. Narrative [26].

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, storyworld (environment), the initial state of the storyworld, 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). [28] Together, these subsystems generate believable characters (who act "like people"), change the storyworld 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 storyworld, for which he must have some degree of freedom of action. [29] 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. [29] 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. [30] 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. [31]

1.5.1 Dramatic structure

For an interactive story to be successful, the narrative experience it provides to the user must be dramatically interesting to them. Dramatic structure plays a significant role in ensuring the dramatic interest of experience. Interest in the disclosure and synthesis of the dramatic structure was shown by many theorists, from ancient times, when Aristotle himself was engaged in this, to the present day. [29]

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

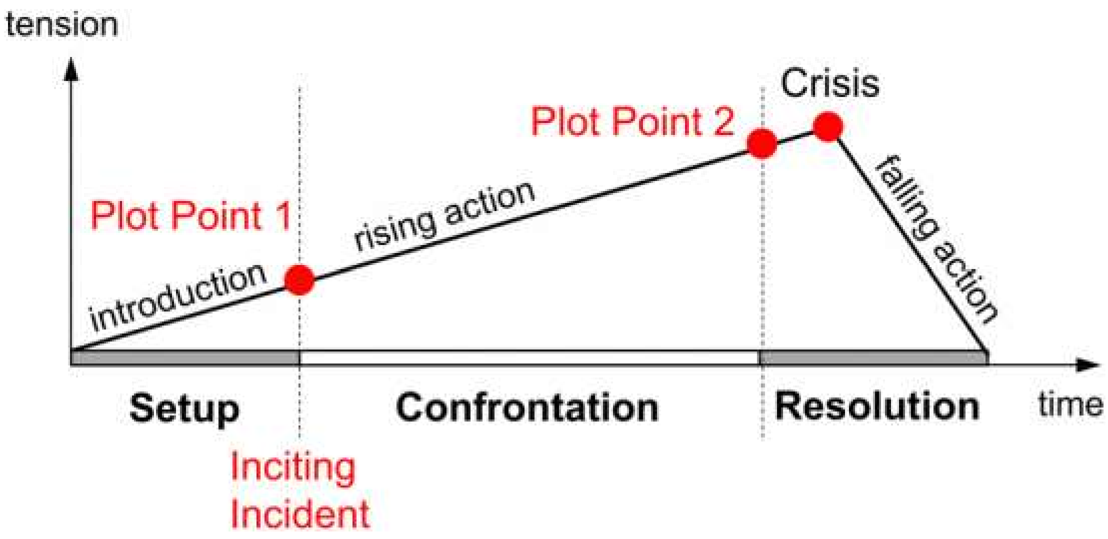


Fig.4 Story dramatic structure as a three-act structure. Source: "Procedural generation of branching quests for games" [61].

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.

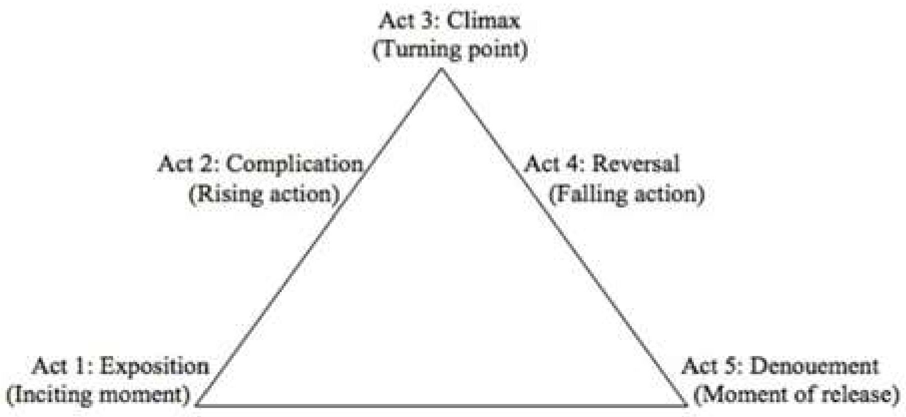


Fig.5 Dramatic structure known as the "Pyramid of Freytag". Source: "Celebrity Bowl: More Marketers Than Ever Turn To Celebrity For Super Bowl Ads, But That's Not The Whole Story".

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 and IDtension require generated narratives to follow a "dramatic arc". The Facade uses a structure that the authors call neo-Aristotelian.

Esslin argues that any drama must capture the attention and keep the audience engaged while being consistently entertaining. [29]

1.5.2 Believable Agents

The agent model takes in information about the storyworld 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. [28][32]

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. [33][34]

1.5.4 Narrative experience

The interactive drama offers the user a storyworld 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 storyworld 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. [31]

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 storyworld. 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 terms 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. [35]

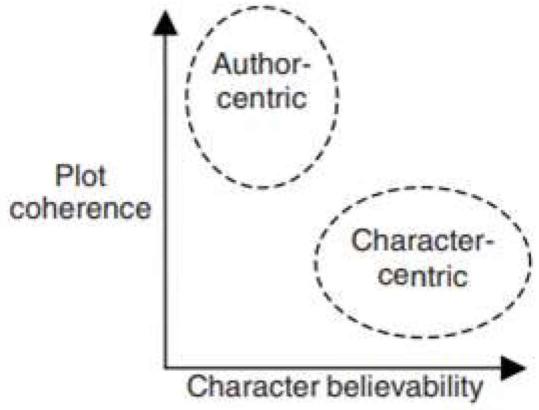


Fig.6 Influence of story-centric and character-centric design on story properties. Source: “Character-Focused Narrative Generation for Execution in Virtual Worlds” [48].

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 his own actions. This kind of interactive narrative is called emergent (that is, spontaneous) narrative. His idea 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 storyworld. Examples of the use of emergent narrative are various simulation games (RimWorld, Minecraft, the Crusader Kings series) and learning environments.

Emergent systems, however, can not only fully automate narrative and the storyworld, but can also include some form of narrative control to set some direction for the story's development. [5]

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 is, in essence, 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 storyworld. 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 storyworld 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. [36]

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. [37] 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). The two poles of this continuum are interpreted by them 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.

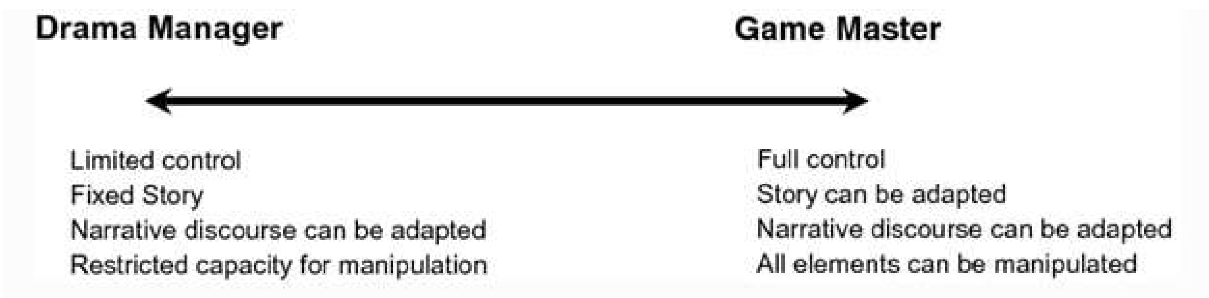

Drama Manager
Game Master
Limited control
Fixed Story
Narrative discourse can be adapted
Restricted capacity for manipulation
Full control
Story can be adapted
Narrative discourse can be adapted
All elements can be manipulated

Fig. 7 Author's freedom continuum. Source: “Considering Authorial Liberty in Adaptive Interactive Narratives” [37].

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 storyworld, 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 storyworld, 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, not its rules or setting. [37]

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. [38] 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 storyworld, 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, the opportunity to influence all its aspects (both discourse and story as a whole). [37]

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 storyworld. And the changes that take place in the storyworld 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, Minecraft, the Crusader Kings series, also Dwarf Fortress). 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. [27]

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 [63], 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 storyworld, 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. But in the process of returning, he experiences a large number of new adventures before he reaches home. And even here a new adventure arises when he has to deal with enemies. As Greimas notes [64], 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. [65], 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 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. [65] 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 [66] describes it as a "post-narrative discourse", Tosca [67] 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 [68] 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. [39] 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. [40] 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 storyworld, 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, relationships. [29]

The next large milestone was the "Universe" system [41] 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 storyworld 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." [29]

The first main interactive drama research group was “The Oz Project” in the 1990s, led by Joseph Bates. [42] 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. [29]

Also in the late 1990s, the “Defacto” system, which used a rule-based approach to storytelling, was created. [43] She 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 storyworld 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 storyworld 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. [29]

Façade was the first project in the field of interactive storytelling that really got a lot of attention, including among a wide audience. [44] 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 storyworld, 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. [29]

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. [45] 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. [6] [46] [47] 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 storyworld 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. [29]

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 [48], intelligent methods for controlling narratives in interactive systems [6], managing the interaction between agents and the user of a multi-agent interactive system [46], intention-oriented planner in a multi-agent environment [33] , autonomous agents and their believability in an interactive environment [32][49], the integration of plot-centric and character-centric designs in one interactive environment [31][35], the inclusion of author's intentions in a system that generates an interactive narrative [36].

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 [50], U-DIRECTOR [51], PaSSAGE [52], IN-TALE [32], NOLIST [53], GADIN [54], OPIATE

[55], DED [56], IDtension [57], BARDS [58], FAtiMA [59]. 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". [29]

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" [65], 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 [63]. 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" [60] and "Procedural generation of branching quests for games" [61], 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. Overview of the Framework

In this chapter, we will describe the framework that we have created for making a system for generating interactive stories, including such aspects of stories, like implementing intelligent agents in a multi-agent environment and empowering them with planning capabilities, visualizing the resulting story graphs, and executing the resulting interactive stories. We will first describe its general structure and concept, and then move on to describing the implementation details.

3.1 Concept

From the very beginning, we planned to use some IDE to create our system in order to facilitate the development process and implement all the subsystems that interest us. Initially, we considered using a game engine like Unity, but then we came to the conclusion that this was not a good idea, because these software tools provided excessive possibilities for our needs. For these possibilities, we would probably have to pay with the performance of both the resulting system and the framework itself, which would have a negative impact on the implementation process, since we did not have a sufficiently performance computer. In addition, our goal was to simplify the overall design, not complicate it. Based on these considerations, we decided to choose a simpler IDE. It was also important to take into account the choice of language, since different IDEs support different languages. We came to the conclusion that a reasonable choice would be to use one of the JIT languages, which provide both good computational speed and rich expressive capabilities, as well as convenience. The most well-known languages of this type are C# and Java. This narrowed down our final choice to Microsoft Visual Studio and IntelliJ IDEA.

When choosing between MVS and IntelliJ IDEA, we took into account several factors. First, each of them supports different languages, and it was necessary to choose between C# or Java. Since we had much more experience with C# than with Java, the choice was made in his favor. Performance on our hardware also played a role. MVS ran stably and without freezes, while IntelliJ IDEA took a long time to start 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, thanks to the Windows Forms API. Therefore, in the end, we decided to use the MVS IDE (Microsoft Visual Studio Community 2017 version) and the C# language, in which we wrote all the code.

To simplify the design of the framework, we planned to divide it into several modules that could be connected or disconnected as needed. So, an external system, Fast Downward, is responsible for automatic planning, and Graphviz is responsible for visualizing interactive narrative graphs. However, we eventually abandoned some of the external modules, since they only complicated the design and use of the system, having implemented their functionality on our own.

3.2 Fast Downward

Fast Downward 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 is a progression planner that searches the state space of the world for forward planning problems. [69] We used the seq-opt-lmcut algorithm.

This module is critical for the functioning of the system, so its presence is necessary. However, its installation is the most difficult among all the software tools that make this framework. To do this, it is necessary to download the program files and place them in the directory with our system, and also have Python version 3 installed on the target computer (make sure that python3 is on your PATH) and Visual Studio >= 2017, also CMake. Then, following the instructions from the official site [70], compile and create the build once.

3.3 Graphviz

Graphviz is a set of utilities for automatic visualization of graphs written using the DOT language.

The main tool from this set of utilities is dot, which accepts a text file in the DOT language as input. This language describes the nodes and their connections in a graph, as well as various additional properties, such as their name, color, and so on. dot itself recognizes all graph connections and arranges it in such a way as to minimize the number of intersections. But 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, twopi - a tool for creating a graph based on a "radial" model, circo - a tool for creating a graph based on a "circular" model, fdp - a tool for creating an undirected graph based on the fdp model, and some others.

This module is not critical for the functioning of the system, so its presence is not required. Our system independently generates .dot files that can be loaded into the required tool from this set. To do this, it is sufficient to download Graphviz from the official website and run the desired tool using the command console.

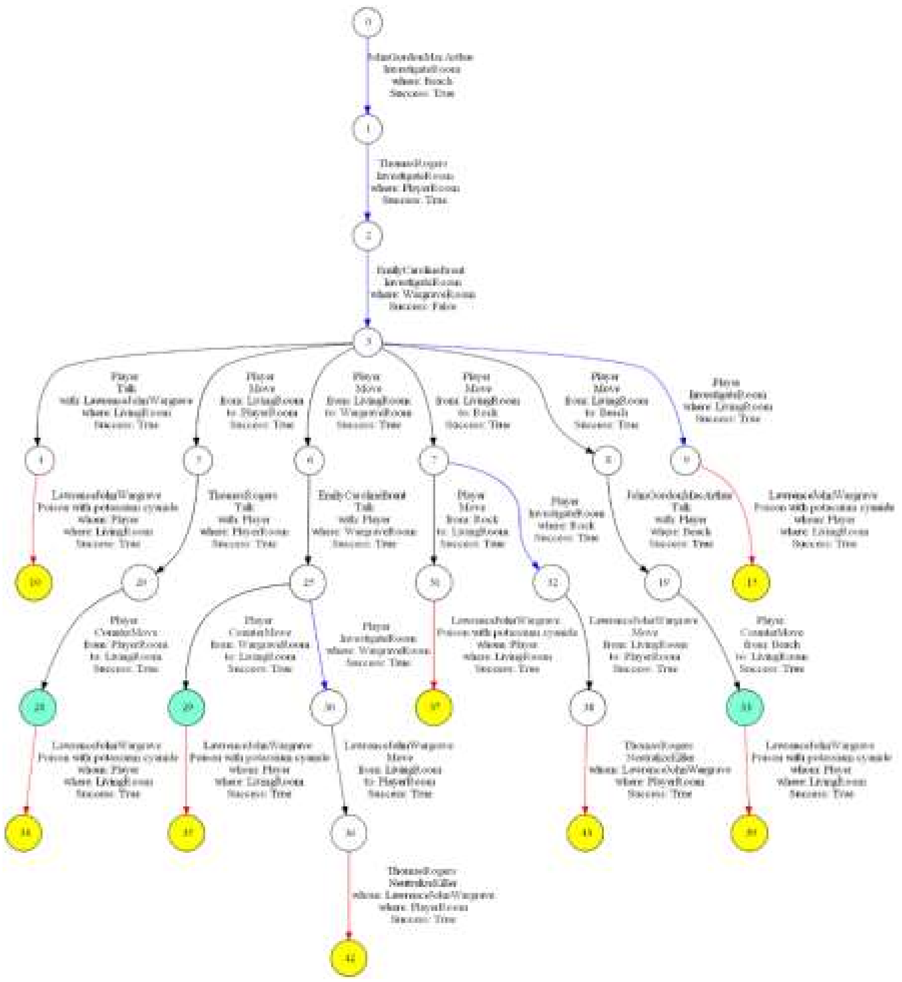


Fig. 8 An example of a graph created with Graphviz and dot.

3.4 Twine

Twine is a platform for writing interactive fiction, text-based games, and visual novels. A huge advantage of this platform is its ease of use, as it allows export stories to HTML files that can be used locally or uploaded to a website and viewed with any browser. Also, the use of this platform does not require programming skills, using its own markup language. Another advantage is 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 where the 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 software. It should be mentioned that the files we generate are suitable for version 2.3, but version 2.4 was released on 2022-07-10, which is not backward compatible with the previous version. Only the latest version is available on the website as an online service.

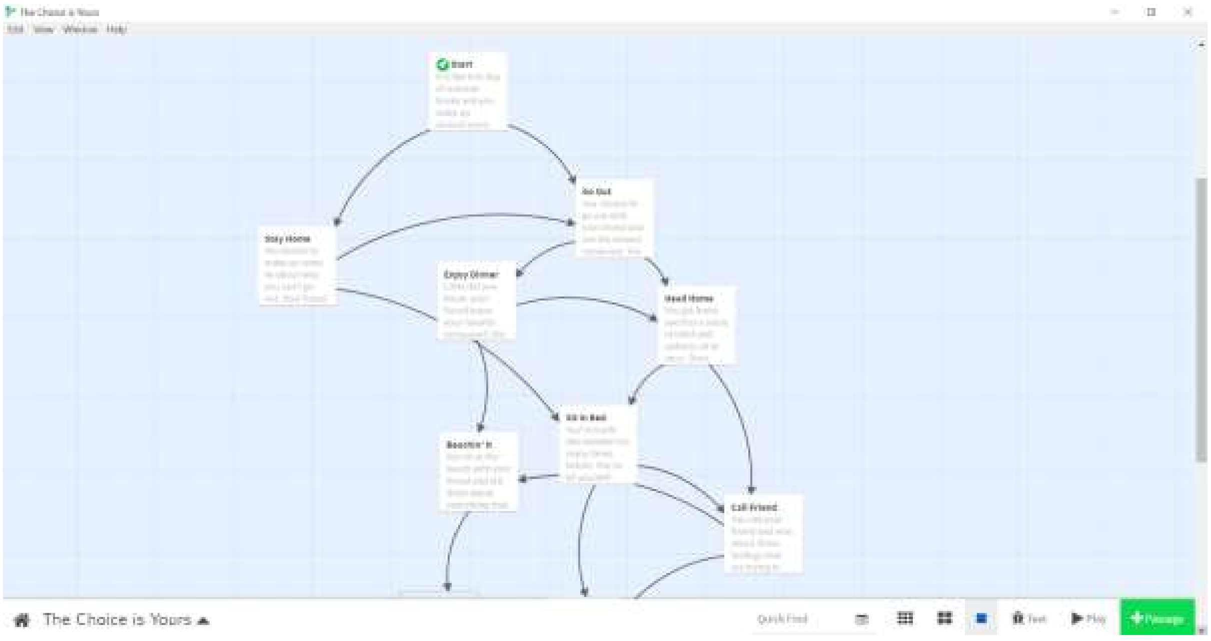


Fig. 9 Twine workspace example.

4. Narrative generation

In this chapter, we will describe the algorithm we have developed, which is the basis of the system we have created. First we will look at it at a high level, describing its architecture. Then we will explain in more detail how its individual elements solve the subproblems that arise when generating an interactive story.

4.1 Architecture

As shown in Figure 10, the architecture of our proposed story generation system consists of the following subsystems: storyworld, narrative generator, drama manager, planner, author interface, output module.

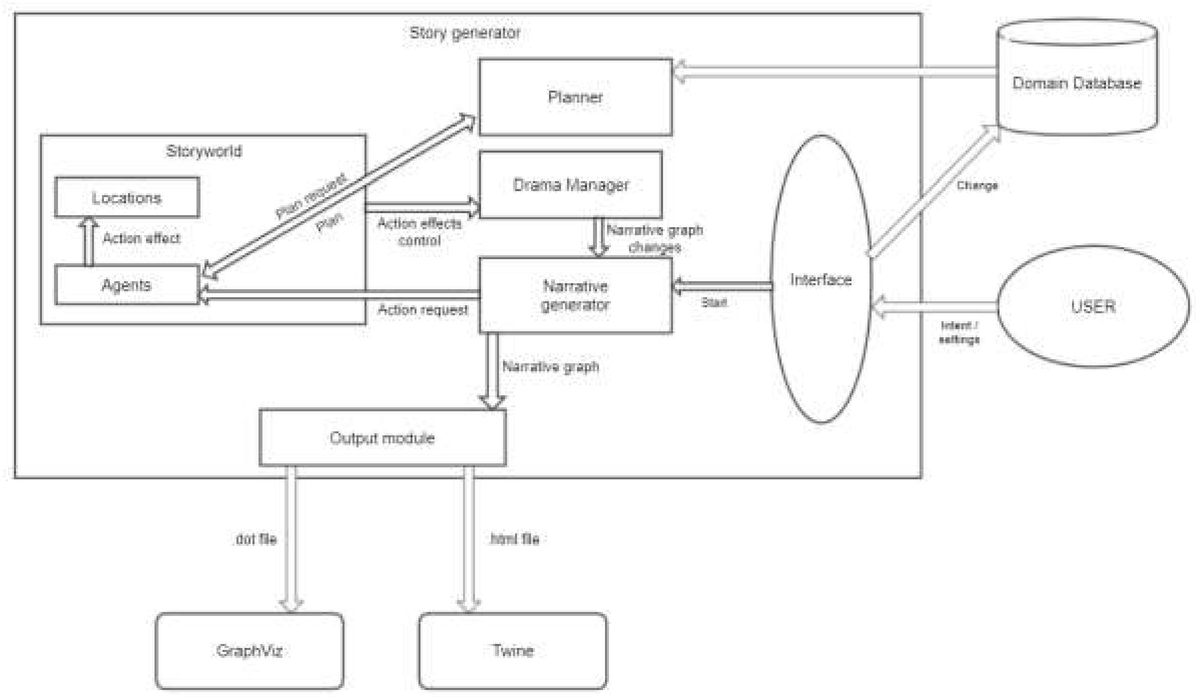


Fig. 10 Story generator architecture.

(1) The storyworld combines itself all the information about the environment in which agents operate, as well as about themselves. A specific configuration of the storyworld is called the state of the storyworld. Making changes to its configuration by agents, by performing actions and applying their effects, leads to a change in the states of the storyworld. If the goals of one of the agents are fulfilled in a certain state, it is considered to be the goal state. (2) The narrative generator manages the interaction of agents, calling them in turn to update beliefs, by performing an act of perception, followed by an act of planning, and then to perform the action chosen by the agent, after which it records the change in the state of the storyworld in the narrative graph. (3) The drama manager oversees the coherence and consistency of the narrative, as well as the observance of the author's goals. It controls this by checking that the current state of the storyworld fulfills the constraints that are imposed on it. If the agent's actions violate consistency or coherence, or contradict the author's goals, it intervenes by replacing the agent's action with another action that does not violate the specified restrictions. (4) The planner uses the observations extracted from the state of the storyworld by the agents and the information they provide about themselves to, in response to the agents' requests, find the right course of action for them, the execution of which will lead to the fulfillment of their intentions. (5) Using the authoring intent interface, the user modifies the domain database and the initial state of the storyworld, and defines the behavior of the drama manager. This has an impact on the generation

of the storyworld as it is based on the information that is stored in the domain database. (6) The output module takes the narrative graph as input and processes it, creating files, one in .dot format to render the narrative graph, the other in .html format to load the created interactive story into Twine, converting it into a quest.

4.2 Storyworld creation

The storyworld 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 cumulative configuration of all objects and agents, i.e. their own states, is the state of the storyworld. Accordingly, by their actions, agents are able to change the state of the storyworld 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, connection (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 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 storyworld, 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.

4.3 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 storyworld. As we already mentioned, agents can only perceive a limited part of the storyworld 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 location of 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 storyworld, but only the internal state of the agent.

• G is the agent's goals, i.e. the specific state of the storyworld 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 storyworld (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.

4.4 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 storyworld, 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.

4.5 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 storyworld. 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 storyworld, 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 storyworld.

• O are planning operators expressing various events and actions in the storyworld. Each $o \in O$, 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 storyworld).

```
(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. 11 An example of a detective domain encoded in the PDDL formalism.

4.6 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 storyworld). The intentions of agents are formed from a set of their character (realized in the form of assigned roles), goals, and beliefs about the storyworld obtained in the course of observing it (at the same time, the agent does not survey the entire storyworld, 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 storyworld (it is also valid to consider it the current state of the storyworld, 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 storyworld.

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 storyworld, forbidding it to move into certain states. The reason is that agents do not know about them, so they do not take them into account 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.

4.7 Story Planning

Quite often, the storyworlds 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 storyworld 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 storyworld. 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. [22] 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 history 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 narrative 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 start with a narrative 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 storyworld, the narrative generator determines whose turn from agents to perform an action, and then sends a corresponding request to this agent. We add the new state of the storyworld, 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 storyworld, 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 storyworld 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 narrative graph branching factor. The creation of the narrative graph is completed when each branch of the graph reaches the goal state. After that, we render the resulting narrative 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.

4.8 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. Or 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 storyworld and the narrative 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 offending 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.

4.9 Output

After the completion of the narrative graph generation process, the resulting graph is passed to the output module. There, it is first passed to the visuals graphs constructor, where, based on it and taking into account the author's instructions (for example, hide "empty" actions), a text file in the .dot format is created. This is done by sequentially forming descriptions for each node, depending on its properties, and then each edge. Also, from the information encoded in each edge, a text is created that describes the action and some of its details. Then the resulting graph is passed to the quest constructor, where an .html file is created based on it. In the case of Twine, things are made easier by the fact that it is also built on the architecture of nodes and links between them. Therefore, the nodes and edges of the graph are simply recoded into a format understandable for Twine, and supplemented with some service information that serves for the correct visual display, for example, the coordinates of each node in space or CSS settings.

5. Quests generation

In this chapter, we will briefly explain the relationship between the interactive story and the quest, and also explain how to get the quest from the interactive narrative.

5.1 Relationship between interactive story and quest

In this work, we focus on generating an interactive story whose narrative is able to adapt to user actions. We believe that it is a fundamental element of any system of which it is a part. This is due to the fact that, as we mentioned above, the two main components of a story are plot and characters. The plot is a sequence of events ordered in a certain way, including the actions of the characters, i.e. narrative. Characters are the actors of the story, between whom certain relationships are established. See figure 3. The sequence of events in which the characters act, and which is expressed in one way or another, is a story. By saying that a story is interactive, we mean that it does not have a clearly defined sequence of events in advance, and this sequence can change as a result of user interaction. The quest, in our understanding, is a specific case of interactive story, which also has its own storyworld. It also consists of characters, referred to in this context as NPCs, and a sequence of events determined by the actions of the player and NPCs. A specific feature of his storyline is that it is determined by a set of tasks facing the player. At the same time, in one particular storyworld, a quest can generate subquests, or chains of quests can appear in it, and a change in its state can affect them, which Greimas notes [64]. Given the above, we come to the conclusion that the existing general narrative can be converted into any required format in which it is present as an integral part, regardless of the specific features of this format, and there is no need to initially generate a separate, specifically formed narrative for each format.

5.2 Converting a universal narrative into a quest

Thus, we create a universal narrative, which can later be adjusted to the required format. For example, to create quests or a tool for calculating possible paths and outcomes of quests and stories invented by designers. And although there is no single generally accepted theory of quests, many researchers agree that their important distinguishing feature is the concept of "missions", "tasks". Those, they are focused on setting a number of tasks for the player to complete. In an interactive story, this is not necessary, there is a much greater emphasis on artistic features and experience. For example, in Façade [44] the player does not have the task of reconciling a married couple, even if the couple breaks up - this will not be considered a failure, the main thing is to let the player participate in this story. Accordingly, an absolutely necessary condition for converting a universal narrative into a quest is a clear statement of the task for the player. This is what we do by perceiving the player in the process of generating a narrative as an agent, albeit a very specific one, who has the same parameters as other agents, including goals. Accordingly, the state of the storyworld, in which the player’s goals are fulfilled, is the goal state, and when it is reached, the game ends (with the player fulfilling his goals, i.e., his victory). Presenting such a quest for an external module that will playback it remains a matter of specific technical implementation.

6. Implementation

In this chapter, we will describe the details of the technical implementation of our system, starting with a description of the implementation of the interface and the implementation of the author's intentions with it, then we will describe the implementation of the storyworld, after which – the implementation of the mechanism of the drama manager. We will conclude the chapter with a description of the output mechanism of the resulting files.

6.1 Authoring Tools

We implemented the support of the author's intentions in such a way that we allowed the user to choose the setting of the generated story, the type of goals of agents, some settings of the storyworld determine some constraints and influence the behavior of agents by setting preferences in their behavior.

The configuration process itself is carried out through interaction with the graphical interface. Not all the features that we put into it, we were able to implement, but we will tell you about it in more detail in the “future work” section.

This is directly implemented in such a way that an instance of the StoryAlgorithm class is placed in the class that controls the behavior of interface elements – MainWindow, which performs general control of the generation process. Having direct access to this instance, we bind calling its methods and changing its properties to interface elements. In the StoryAlgorithm class itself, we contain instances of classes of all the main modules of the system, using it, among other things, as an intermediary for transmitting the appropriate settings to them, using the ReadUserSettingsInput method.

6.2 Story graph

One of the most important components of our system is the narrative graph, which we implemented to the StoryGraph class. It contains a hashed set of all nodes – the states of the storyworld reached by the agents and the player, as well as a pointer to the root.

The nodes themselves store information about the corresponding state of the storyworld, the agent active in this state, links to other nodes with which the node is connected by faces, and, in fact, about the faces to which it is connected.

We also implemented the faces as a separate class that stores references to the nodes that it connects (upper and lower), as well as to the action that it models itself.

6.3 Storyworld representation

We realized the storyworld with the help of locations, the representation of which was divided into two classes. One stores information about a location that is guaranteed not to change during generation, such as its name and links to other locations. This is implemented in the Location Static class. The Location Dynamic class contains information about the agents in it, as well as the presence of evidence in the represented location. It also contains a link to its static part.

We also divided the class representing the state of the storyworld into two parts – static and dynamic, for the same reasons. The WorldStatic class contains a list of locations that make up the world of history, information about the setting, the number of the "plot stroke", as well as several settings. The World Dynamics class contains a link to its static part, complete and detailed information about the status of all locations and agents. Both of these classes differ in that they have complex, extensive, and detailed methods for comparing their instances.

6.4 Agents

Implementing the agent model, we divided it into two classes – static and dynamic.

Agent State Static stores static information that does not change during the generation of the narrative, for example, the name and role of the agent. It also stores methods for its assignment (used once, during creation), receipt (getters) and verification (hash function).

Agent State Dynamic contains the information about the agent that can change during generation. This is the current plan, a list of available actions, a link to the static part of the class (to simplify access to it), the status (alive / dead), a description of goals, various variables responsible for emotions and desires, beliefs, and some service, such as the value of the initiative or hash function. It also contains methods for assigning, modifying, obtaining and verifying these properties and parameters.

6.4.1 Beliefs

We implemented the agents' beliefs mainly through the WorldContext class. This class stores a reference to the location where the agent is currently located, a hashed set of copies of static parts of the locations of the storyworld (which allows the agent to know how the locations are interconnected), as well as two hashed sets that store his knowledge about other agents. The first of them, anotherAgentsInMyLocation, stores links to static parts of other agents that the player perceives as being in the same location with himself. The second set of agentsInWorld stores information about all agents in the storyworld, but in the format of a special class representing the agent's beliefs about other agents – BeliefsAboutAgent. This class is initially empty and is filled with information as the agent perceives other agents, meeting them and remembering their location, and also contains his assumptions about their role, initially neutral, which may change if he finds evidence against the antagonist.

6.4.2 Goals

We encoded the goals of the agents in the Goal class, presenting them as a pointer to the type of goal and a copy of the goal state of the world for the agent, in the form of an instance of the WorldDynamic class, which is used as a "reference" when the narrative generator checks whether the current state of the world is a target for someone.

To work with goals, we use the GoalManager class, which we use to define goals (taking into account the setting, the author's intentions, the agent's role), assign them to agents and then monitor their implementation.

6.4.3 Actions

We encoded each type of action as a separate class, the successor of the abstract PlanAction class. Each action contains a list of arguments – objects with which it interacts or information about which it should receive, information about its success or failure, a description (with which it is possible to give the same action, taking into account different circumstances, for example, different names, which will be displayed by different actions when output, functionally being one and the same). Also, each action contains a method that checks the prerequisites for its application (if they are not satisfied, the action will not be assigned to the agent in its list of available actions), and a method that changes the state of the story (an instance of the WorldDynamic class), which receives input.

A separate subspecies of actions are counter-actions, which are operated by the Drama Manager, implemented in the form of the StoryworldConvergence class. These counter-actions are also inheritors of the PlanAction class, functionally practically no different from ordinary actions. The counter-actions that the Drama Manager performs, we, for our own simplicity, in a working order, named according to the functionality that they have. Each of them at the same time "remembers" what action it replaces. So far it looks like instead of Neutralize Killer action, CounterMove action is performed, which may seem counterintuitive. However, when processed by language algorithms, it can look like this: "the character tried to shoot the killer, but the collapsed ceiling forced him to retreat to the nearest room, so he could not make a shot."

6.4.4 Perception

Agents have perception, receiving information from the system about the location they are in, as well as about other agents who are in it (although, of course, not complete). This is implemented in such a way that using the RefreshBeliefsAboutTheWorld method of the AgentStateDynamic class, the agent takes each "plot beat", i.e. when he gets the right to act again, he updates information about which location he is in (through this he can access an instance of the LocationDynamic class of a specific location to get more detailed information about the location), and also updates his idea of other agents he met in this location, for example, in his instance of the BeliefsAboutAgent class he removes them from copies of other locations by placing them in a copy of the location where they met.

6.4.5 Planning

Agent planning is implemented as follows. (1) Each "plot cycle", the PDDL-Module class that implements the management of the creation of PDDL files, creates a new instance of a file describing the planning domain for each agent role. (2) The agent active in this "plot cycle" receives an action request implemented by the ActionRequest method of the StoryworldConvergence class, to which the information about the required agent is transmitted by the StoryAlgorithm class. (3) In the ActionRequest method, the agent consistently updates his perception, and, accordingly, beliefs. Then, based on a number of its parameters, it sends a request to the PDDL-Module to create a pddl file with a planning problem based on its perception, beliefs, goals and emotions. (4) Requests from the scheduler a plan for itself, based on the already generated PDDL files for the domain and the planning problem. (5) The agent determines the actions available to him by going through the list of all actions and checking their compliance with the preconditions, checking that he can implement the plan. (6) Uses the CSP-module class to assign the most appropriate arguments to the selected action (We do not use the variables that the scheduler offers, since it is not convenient to pass all the parameters important for the application of the action to it. Therefore, we use the planner only so that it indicates which actions to use, determining the details of use independently.). (7) The resulting instance of the action is sent to the Drama Manager for control. If it does not threaten to violate any of the properties of the narrative, then it is applied. If it violates, then the Drama manager chooses a counter-action and commits it. After making changes to the narrative graph, the "plot tact" ends, and the StoryAlgorithm class selects a new state of the story world to consider (from those that have not yet been considered) and the agent who will act.

6.5 Graph rendering

When the system finishes generating the narrative graph, it returns the currentStiryGraph variable of the storyGraph class, which stores it. It is accepted by the Create Graph method of the

Graph Construction class. Also, this method receives a string variable graphName as input, which, as the name implies, is used to set the name of the file being created.

The creation of a text file describing a narrative graph takes place in three stages. At the first and third stages, new parts of the text are written to the global text variable for this class.

In the first, we go through all the nodes of the graph, determining their properties (for example, whether the state of the node is the target, or whether it was created as a result of a counteraction) and based on them creating a text description defining the shape of the described shape for visualization, its color and other parameters.

In the second, if the hide empty actions setting was set, we use several nested loops to go through the storygraph, cutting out "empty" actions from it and rearranging the connections of the edges so that their cut ends connect.

In the third, we cycle through all the faces of the normative graph, creating a description based on what action they represent and which nodes they connect.

6.6 Twine integration

Methods for converting a narrative graph into a file with a set of instructions for Twine are collected in a separate class TwineGraphConstructor. The process takes place by selecting from the nodes those in which the player should act, and then the faces that should lead to the next nodes, where the player makes the choice again (at the time of this work, the method of searching for subsequent nodes is broken, and we do not have time to fix it). Then the selected nodes and faces are designed in accordance with the code that Twine "understands" for their successful loading, correct display and launch on this platform.

Conclusion

As part of this thesis, we explored various approaches to creating interactive stories, figured out what makes them interesting and what are their most important features, and how they can be applied in practice. Based on this, we have developed our own goal of how to create a story generator that can also be used for practical purposes: to create a tool that allows to visualize all possible ways of developing a narrative in a given domain (which, in our opinion, should be useful to game designers, instead of calculations "manually"), as well as to generate quests. We found out that this method of using procedural story generators is still a little-studied area.

To fulfill our goal, we have developed an algorithm that allows to speed up the creation of an interactive story, which usually takes quite a long time. At the same time, we applied an approach that we see as quite innovative – a mixture of high emergence and centralized regulation with the help of a Drama Manager. We believe that it allows to achieve several goals at once: how to speed up the generation of story, and combine the believability of agents from emergent narratives with the consistency and consistency 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 stories in the detective genre and use a very small domain for this. But 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 "skeletons" of more complex stories are created than we originally planned. We also showed that it is possible to visualize the narrative generated by our system in an understandable way, and to create a 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, which we will describe below. Also, the overall complexity of coordinating all parts of the system did not allow us to avoid mistakes.

Future work

In the process of working on this thesis, we saw many opportunities to expand it. So, we would like to improve the author's capabilities in terms of modifying and editing the story. We believe that it would be a rather successful and elegant solution to allow this to be done 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 deep changes. We also see further opportunities for improvement by expanding the model of relationships between agents to make them even more plausible. We consider the introduction of language algorithms to be an essential task for further work, which would be able to create a real story from a "naked" narrative that could be compared with human creativity. We see an interesting opportunity for further work to introduce and control dramatic structures, at least the classical three-part one, which would improve the hoof 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 of not finishing the story, but starting a new one in the already changed storyworld. 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 yourself not only to Twine, but also expand to Squiffy and Unity.

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 no comprehensive works in it yet.

Bibliography

1. M. Hendrikx, S. Meijer, J. V. D. Velden and A. Iosup, "Procedural content generation for games: A survey", *ACM Transactions on Multimedia Computing Communications and Applications*, vol. 9, no. 1, 2013.

2. A. Amato, "Procedural Content Generation in the Game Industry" in Game Dynamics, Springer, 2017.

3. J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, et al., "Procedural Content Generation: Goals Challenges and Actionable Steps", *Artificial and Computational Intelligence in Games*, vol. 6, 2013.

4. J. Freiknecht and W. Effelsberg, "A Survey on the Procedural Generation of Virtual Worlds", *Multimodal Technologies and Interaction*, vol. 1, no. 4, pp. 27, 2017.

5. B. Kybartas, R. Bidarra, "A Survey on Story Generation Techniques for Authoring Computational Narratives", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 239-253, 2017.

6. R. Young, M. Riedl, “Towards an Architecture for Intelligent Control of Narrative in Interactive Virtual Worlds”, Proceedings of the 8th International Conference on Intelligent User Interfaces, 2003.

7. D. Johnson, "Animated Frustration or the Ambivalence of Player Agency", *Games and Culture*, vol. 10, no. 6, pp. 593-612, 2015.

8. J. Clement, “Number of video gamers worldwide in 2021, by region” [online, accessed 2022-06-30]. Available from: https://www.statista.com/statistics/293304/number-video-gamers/.

9. B. Strauss, “Video Games Are Art, Says U.S. Government” [online, accessed 2022-06-30]. Available from: https://www.businessinsider.com/video-games-are-art-says-us-government-2011-5.

10. T. Wijman, "The Games Market and Beyond in 2021: The Year in Numbers" [online, accessed 2022-06-30]. Available from: https://newzoo.com/insights/articles/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming.

11. UESP, "Skyrim:Radiant" [online, accessed 2022-06-30]. Available from: https://en.uesp.net/wiki/Skyrim:Radiant.

12. Reddit, "Are Radiant Quests Having An Adverse Effect Open World Games and Unique Quests Within?" [online, accessed 2022-06-30]. Available from: https://www.reddit.com/r/Games/comments/57no58/are_radiant_quests_having_an_adverse_effect_open/.

13. D. Faggella, "What is Artificial Intelligence? An Informed Definition" [online, accessed 2022-06-30]. Available from: https://emerj.com/ai-glossary-terms/what-is-artificial-intelligence-an-informed-definition/.

14. M. Wooldridge, "An Introduction to MultiAgent Systems", John Wiley & Sons Ltd, 2002, paperback, 366 pages, ISBN 0-471-49691-X.

15. G. Lawton, "AI can predict your future behavior with powerful new simulations" [online, accessed 2022-07-01]. Available from: https://www.newscientist.com/article/mg24332500-800-ai-can-predict-your-future-behaviour-with-powerful-new-simulations/.

16. N. Shaker, J. Togelius, M. Nelson, "Procedural Content Generation in Games: A Textbook and an Overview of Current Research", Springer, 2016.

17. F. Mourato, "Enhancing automatic level generation for platform videogames", 2015.

18. M. Perez, E. Eisemann, R. Bidarra, "A Synset-Based Recommender Method for Mixed-Initiative Narrative World Creation", International Conference on Interactive Digital Storytelling, Lecture Notes in Computer Science, vol 13138, Springer, 2021.

19. G. Smith, J. Whitehead, M. Mateas, "Tanagra: A mixed-initiative level design tool", Proceedings of the Fifth International Conference on the Foundations of Digital Games, pp. 209–216, ACM, 2010.

20. M. Ghallab, D. Nau, P. Traverso, “Automated Planning: Theory and Practice”, Cambridge University Press, 2016, hardback, 368 pages, ISBN 9781107037274.

21. C. Büchner, P. Ferber, J. Seipp, M. 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, 2022.

22. S. Ware, E. Garcia, A. Shirvani, R. Farrell, “Multi-Agent Narrative Experience Management as Story Graph Pruning”, Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-19), 2019.

23. R. Fikes, N. Nilsson, “Strips: A new approach to the application of theorem proving to problem solving”, Presented at the 2nd IJCAI, 1971.

24. M. Fox, D. Long, “PDDL+: Modeling continuous time dependent effects”, Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002.

25. R. Barták, “Constraint Satisfaction for Planning and Scheduling”, Proceedings of the 14th International Conference on Automated Planning and Scheduling, 2004.

26. O. Wielk, “Story vs. Narrative” [online, accessed 2022-07-05]. Available from: https://www.beemgee.com/blog/story-vs-narrative/.

27. J. Martin, “Interactive Narrative vs. Interactive Storytelling” [online, accessed 2022-07-05]. Available from: https://betweendrafts.com/2022/04/08/interactive-narrative-vs-interactive-storytelling/.

28. B. Bostan, T. Marsh, "Fundamentals of interactive storytelling", Online Academic Journal of Information Technology 3 (8), 2012.

29. M. Arinbjarnar, H. Barber, D. Kudenko, “A Critical Review of Interactive Drama Systems”, AISB Symposium, 2009.

30. R. Hervas, A. Sanchez-Ruiz, P. Gervas, C. Leon, “Calibrating a Metric for Similarity of Stories against Human Judgment”, Published in ICCBR, 2015.

31. M. Riedl, R. Young, “Narrative Planning: Balancing Plot and Character”, Journal of Artificial Intelligence Research, vol. 39, 2010.

32. M. Riedl, A. Stern, “Believable Agents and Intelligent Story Adaptation for Interactive Storytelling”, Proceedings of the 3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE), 2006.

33. M. Riedl, R. Young, “An Intent-Driven Planner for Multi-Agent Story Generation”, Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems, 2004.

34. M. Kapadia, J. Falk, F. Zünd, M. Marti, R. Sumner, M. Gross, “Computer-assisted authoring of interactive narratives”, In Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, 2015.

35. M. Si, S. Marsella, M. Riedl, “Integrating Story-Centric and Character-Centric Processes for Authoring Interactive Drama”, Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2008.

36. M. Riedl, “Incorporating Authorial Intent into Generative Narrative Systems”, Proceedings of the AAAI Spring Symposium on Intelligent Narrative Technologies II, 2009.

37. T. Pedersen, T. Jensen, V. Zenkevich, H. Schoenau-Fog, L. Bruni, “Considering Authorial Liberty in Adaptive Interactive Narratives”, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 422, 2022.

38. A. Tychsen, M. Hitchens, T. Brolund, M. Kavakli, “The game master”, Proceedings of the Second Australasian Conference on Interactive Entertainment, pp. 215–222, Creativity & Cognition Studios Press, 2005.

39. J. Hitchens, "Special Issues in Multiplayer Game Design", Game Design Perspectives, Charles River Media, ISBN 1584500905, 2002.

40. J. Meehan, “TALE-SPIN, An Interactive Program that Writes Stories”, Published in IJCAI, 1977.

41. M. Lebowitz, “Planning stories”, Ninth Annual Conference of the Cognitive Science Society, 1987.

42. J. Bates, “Virtual reality, art, and entertainment”, Presence: Teleoperators and Virtual Environments, 1 (1): 133–138, 1992.

43. N. Sgouros, “Dynamic, user-centered resolution in interactive stories”, Proceedings of the Fifteenth international joint conference on Artificial intelligence - Volume 2, 1997.

44. M. Mateas, A. Stern,”Façade: An experiment in building a fully-realized interactive drama”, Game Developers Conference, 2003.

45. M. Riedl, “Actor Conference: Character-focused Narrative Planning”, 2002.

46. M. Riedl, C. Saretto, R. Young, “Managing Interaction Between Users and Agents in a Multi-Agent Storytelling Environment”, Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi Agent Systems, 2003.

47. R. Young, M. Riedl, M. Branly, A. Jhala, R. Martin, C. Saretto, “An architecture for integrating plan-based behavior generation with interactive game environments”, Journal of Game Development, 1, 2004.

48. M. Riedl, R. Young, “Character-Focused Narrative Generation for Execution in Virtual Worlds”, Proceedings of the 2nd International Conference on Virtual Storytelling, 2003.

49. M. Riedl, A. Stern, “Failing Believably: Toward Drama Management with Autonomous Actors in Interactive Narratives”, Proceedings of the 3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE), 2006.

50. B. Magerko, “Story representation and interactive drama”, 1st Artificial Intelligence and Interactive Digital Entertainment Conference, 2005.

51. B. Mott, J. Lester, “U-DIRECTOR: A decision-theoretic narrative planning architecture for storytelling environments”, Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, 2006.

52. D. Thue, V. Bulitko, M. Spetch, E. Wasylishen, “Interactive storytelling: A player modeling approach”, AIIDE’07, 2007.

53. O. Bangs, O. Jensen, F. Jensen, P. Andersen, T. Kocka, “Non-Linear Interactive Storytelling Using Object-Oriented Bayesian Networks”, Proceedings of the International Conference on Computer Games: Artificial Intelligence, 2004.

54. H. Barber, “Generation of Adaptive Dilemma Based Interactive Narratives”, Ph.D. dissertation, University of York, 2009.

55. C. Fairclough, “Story Games and the OPIATE System”, Ph.D. dissertation, University of Dublin - Trinity College, 2004.

56. M. Arinbjarnar, D. Kudenko, “Schemas in directed emergent drama”, proceedings of the 1st Joint International Conference on Interactive Digital Storytelling ICIDS08, 2008.

57. N. Szilas, “IDtension: A narrative engine for interactive drama”, 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment, 2003.

58. D. Pizzi, F. Charles, J. Lugrin, M. Cavazza, “Interactive storytelling with literary feelings”, Proceedings of the Second International Conference on Affective Computing and Intelligent Interaction (ACII), 2007.

59. R. Aylett, J. Dias, A. Paiva, “An affectively driven planner for synthetic characters”, Proceedings of ICAPS06 International Conference on Automated Planning and Scheduling, 2006.

60. E. Lima, B. Feijó, A. Furtado, “Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning”, 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), 2019.

61. E. Lima, B. Feijó, A. Furtado, “Procedural generation of branching quests for games”, Entertainment Computing, 14 april, 100491, 2022.

62. T. Hromada, M. Černý, M. Bída, C. Brom, “Generating Side Quests from Building Blocks”, International Conference on Interactive Digital Storytelling, pp 235–242, 2015.

63. V. Propp, “Morphology of the Folktale”, University of Texas Press, 1968.

64. A. Greimas, “Sémantique Structurale”, 1966.

65. E. Lima, B. Feijó, A. Furtado, “Hierarchical Generation of Dynamic and Nondeterministic Quests in Games”, Proceedings of the 11th International Conference on Advances in Computer Entertainment Technology, 2014.

66. E. Aarseth, “Quest Games as Post-Narrative Discourse”, Narrative Across Media, University of Nebraska Press, 2004.

67. S. Tosca, “The quest problem in computer games”, Proceedings TIDSE, 2003.

68. H. Jenkins, “Game Design as Narrative Architecture”, Electronic Book Review, 2004.

69. M. Helmert, “The Fast Downward Planning System”, Journal of Artificial Intelligence Research 26, 191–246, 2006.

“Obtaining and running Fast Downward” [online, accessed 2022-07-15]. Available from: https://www.fast-downward.org/ObtainingAndRunningFastDownward.

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