universität wien

MASTERARBEIT

Titel der Masterarbeit

„Planning in Interactive Storytelling Systems“

verfasst von

Attila Torda, BSc.

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 066 935
Studienrichtung lt. Studienblatt:	Masterstudium Medieninformatik
Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs

Acknowledgements

I would like to express my gratitude for
Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs
for offering this thesis and supervising me.

I would like to thank my family for their support
throughout my student years.

Contents

List of figures

List of tables

Abbreviations

ABL	A Behaviour Language
AI	Artificial Intelligence
HTN	Hierarchical Task Network
ICAPS	International Conference on Automated Planning and Scheduling
IS	Interactive Storytelling
MMORPG	Massively Multiplayer Online Role-Playing Game
NLP	Natural Language Processing
NP	Non-deterministic Polynomial-time
NPC	Non-Player Character

1. Introduction

Suppose you go home, turn on your computer and want to watch a movie. You select a few properties, like genre and story arc, press a button, wait a while and then you watch a movie, which was never seen by anyone else before, because it was generated by your computer.

This is the basic idea of the research project “Automovie”, proposed by Dr. Helmut Hlavacs and Dr. Yohann Pitrey, at the University of Vienna. The goal of this project is to build automatically a movie from a limited set of instructions given by the user, which could be story arc, genre, etc.

The research project has two major challenges:

1. Generate a movie script from a high level description: define a structure for the script, define a hierarchical plot, find transitions between the states and allow for customization.

2. Generate a rendered movie from the script: go from the semantic description to a graphical representation, generate realistic objects, synthesize speech, and generate music.

This Thesis focuses on the first problem, namely Interactive Storytelling (IS).

Many games before included alternate story paths combined with multiple endings, or allowed the user to generate his own character, which influences the story, but in these games all the story lines and events are authored or hard-coded. In IS the story narrative is generated by an AI system.

In the near future Interactive Storytelling could be profitable in the domain of computer games. For movies the number of written movies scripts exceeds the demand by a huge margin, but for computer games it is a very time consuming task to write background stories for each of the characters, for example World of Warcraft contains nearly 60.000 NPC’s altogether and more than 15.000 quests.

Most Interactive Story projects didn’t include an extensive research in the field of script writing. This is the part where my project aims to improve. Therefore I chose several books on movie script writing to have a better understanding of the story creation process: -The Anatomy of the Story [2], Save the Cat [3], Storytelling in the New Hollywood: Understanding Classical Narrative Technique [4], How to Write Movie in 21 Days: The Inner Movie Method [5]. Of all these, The Anatomy of the Story served as a basis for my project implementation. It contains all the foundations to write a movie script.

The first influential Interactive Storytelling system was Façade, an “art/research experiment” created by Michael Mateas and Andrew Stern, released in 2005. This served as a basis for many research papers and publications.

Since then many IS systems emerged, some of them were experiments, while others were games. Nearly all of the IS systems were based on the concept of planning. Planning is a branch in Artificial Intelligence, which deals with the problem of generating an action sequence from a list of possible actions that leads from an initial state to a target state.

The goal of my project and this thesis is to produce a computer program capable of generating believable short stories with minimal user interaction. The program loads a framework of saved files, which contain a description of the world: objects and possible actions. Then it generates milestones from the data. This data is supervised by the user. Then the program searches a path between the milestones in a reasonable amount of time. This is achieved by using a planner. The result is a text file.

The research contribution to my project is multidisciplinary; it is composed of 2 main parts:

-formal and informal analysis of narrative stories: there is a lot of research done in finding recurring patterns in stories. The earliest work is Poetics by Aristotle. Later the three-act structure was developed which described stories as having 3 parts: beginning, middle and end. This was considered too generic and restrictive; therefore many formal and informal systems emerged.

-ai techniques in planning: how to get from a starting state to a finish state, with the given transitions. There are many formal languages and planners. The selection of a planner depends largely on the problem domain. Planning is NP-hard [6], even a classical planning problem, therefore a planning algorithm always involves heuristics.

The problem domain and the story world are based on the well-known TV series, DuckTales. DuckTales was selected because it has short episodes, only with a few repeatedly appearing characters with simple goals and with a simple story, easily understandable by kids. Turning this into code is a challenging task, and is essential to the success of the project.

Chapter 2 describes previous IS projects. Chapter 3 reviews the drama theory in movies. Chapter 4 discusses AI planning. Chapter 5 discusses the implementation of my project. Chapter 6 is about the testing and evaluation of the project.

2. Projects

There are many entirely different projects, with different types of interactivity. [7] categorises IS systems by 3 properties:

-Authorial intent: how much influence the author has over the story.

-Autonomous agents: independent characters, which act simultaneously in the interest of their own and the authorial intent.

-Player modelling: how much shall the world adapt to the players style

Name	Type	Planner	Interaction	Graphics	Setting
Façade	Interactive drama	ABL	Text, mouse	Custom 3D	3 person act
Passage	Game (RPG)	none	Keyboard, mouse	Unreal 3D	Fantasy tale
Scheherazade	Authoring tool	none (random walk)	none	none	Bank robbery
Rationale	Game-like experiment	STRIPS	Keyboard, mouse	Unreal 3D	Kitchen
INTALE	Game	ABL	Keyboard, mouse	Unreal 3D	Afghanistan
Madame Bovary	Experiment	HSP (STRIPS-like)	Holodeck	Unreal 3D	Romance novel
Friends	Experiment	HTN	Speech	Unreal 3D	TV Show (Friends)
ACME	Experiment	HTN	GUI	2D	TV Show (Road Runner)
Wide Ruled	Authoring tool	HTN	GUI	none	Custom

Table 1. A comparison of projects

Façade [8] [9] is an interactive drama experiment, probably the most cited IS system.

It starts by the player creating a character, by giving it a gender and a name. The player controls this character in a first person view. There are two AI-controlled autonomous agents (Trip and Grace, a married couple) in a flat, which are having a conversation and interacting with the environment (for example, drinking wine).

Figure 1. Screenshot from Façade

The planning is based on the concept of dramatic beats. A dramatic beat is a smallest unit within a story, a simple action/reaction pair [9].

The player is allowed to interact in with the objects and write text, which is then interpreted by a Natural Language Parser. The AI reacts to these interactions. The narrated events together with these interactions make the story.

Façade uses ABL (A Behaviour Language), which is a parallel planning language that supports user interaction. User interaction is done by Natural Language Processing: the user is allowed to type in any text, which is then processed and classified as a one of the many reaction types. Then the AI reacts to these by choosing one of the pre-narrated actions.

The story is about 20 minutes long, and it takes about 6-7 attempts to fully exhaust all the possibilities. The story is made in a way, that the tension gets higher by the time progresses, until it gets into the final climax, where the couple could decide to divorce or stay together.

Passage [10] is a great example of implementing interactive stories in video games, which is one of the likely commercial applications. Passage is a role-playing game, which adapts to the player's style.

Passage is based on player modelling, which divides the player into 5 categories: fighter, method-actor, storyteller, tactician, powergamer. Each decision a player makes gives points to some of these categories. This works like a histogram. Then at encounters in the game, the system decides in which category the player belongs to, and adapts to it.

In the evaluation the players played through two fixed and one adaptive story. The adaptive story was based on the classic story of Little Red Riding Hood, but to avoid bias, it was changed in a way, that it's not recognisable. The adaptive ones gameplay was rated more fun, especially by female players.

Scheherazade [11] uses a very different approach: crowd-sourced plot graphs. In this system stories fragments are written by humans, then it is turned into a graph, and a story is generated from it.

The story is generated by the following 3 steps:

1. A story corpus acquired, by using Amazon Mechanical Turks. These are very simple stories, one verb per sentence, so it is easily processed.
2. The algorithm then creates plot events; the result is improved by a second round of crowd-sourcing.
3. Mutual and optional events are identified.

After the graph is constructed, the story is generated by a random walk.

The project focused on a bank robbery story. In the evaluation phase the stories generated by the computer were compared to authored stories. The stories were given to judges to read them, and make changes on them to make them more coherent. The judges could delete, add, or move events.

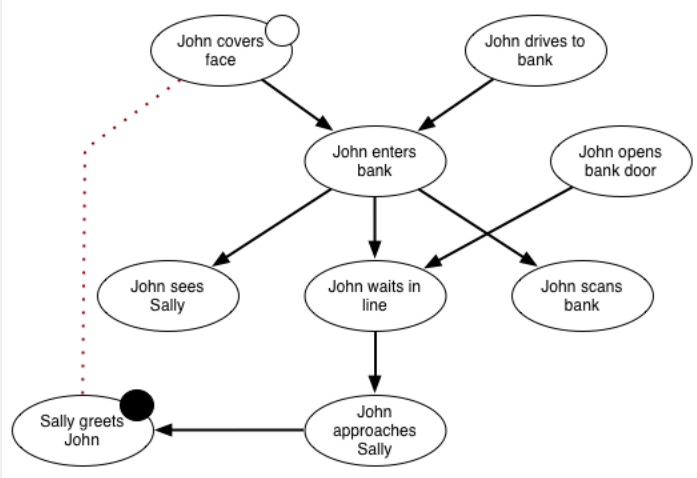

Figure 2. Excerpt from the Bank Robbery plot graph

The events with a blank circle are optional events, the ones with a filled circle are conditional events, the red dots mean mutual exclusion.

The results didn't find statistically significant difference in most measures. The number of deletions was higher for the computer authored stories, due to the events "open the door",

"wait in line". It is safe to conclude, that the stories generated are comparable in quality to the authored stories.

Rationale [12] is an AI system in a game-like setting.

The player is allowed to move around in a 3D environment. The player can interact with the objects around him, and then the AI tries to inflict damage on the player's Avatar.

The environment, "Death Kitchen" is based on the Movie "Final Destination".
The AI is based on planning as well, though the exact technical details of the planner implementation are not known, aside it is based on STRIPS.

Each Action is composed of a trigger: what initiates this action, e.g. the avatar uses something or the water hits something, a condition and an effect, which are known from previous planners. An example from the language:

```
Break-Fluid-Source (Agent#1, Watertap#2)

Triggers:    Event( HIT(Agent#1, WaterTap#2))

Conditions:  FluidSource(WaterTap#2)
Conductor(WaterTap#2)
Breakable(WaterTap#2)

Effects:     UpdateState(WaterTap#2, Damaged)
SetAnimation(WaterTap#2, Sprinkling)
UpdateState(Agent#1, Conductor)
SetAnimation(Agent#1, Soaked)
```

The exact gameplay is not defined yet.

INTALE [13] is a game where declarative story-planning and reactive agents are combined into a military leader training scenario. In this game the player witnesses an argument between two salesmen and a bomb is planted somewhere.

This system is based on Façade's ABL. This system has a different approach from the previous ones: each character is has its own agent, because it is a simulation, but there is a central Automated Story Director, which ensures that the story makes sense as a drama.

Madame Bovary [14] [15] [16] is based on the novel of the same name. The authors realised that psychology and character feeling are under–represented in IS search, therefore their system was based on a 19th century novel, which focuses on character feelings and emotions.

The system uses multi-threaded HSP planning.

Each feeling is associated with an intensity value: LOW, MEDIUM, HIGH.

Friends [17] is an example of a project that uses emergent situations. Unfortunately, the project doesn't have a name, so in this paper it is referred as "Friends".

The scenario is based on the TV Show "Friends", and in this episode Ross wants to invite the main female protagonist, Rachel on a date.

Each character is an autonomous agent, with its own plan. The planning is based on Hierarchical Task Network, which is a formalisation of an AND/OR graph. Several actors and player can interact with the characters, causing its plan to fail. This is not a problem, since failure is a part of drama and story.

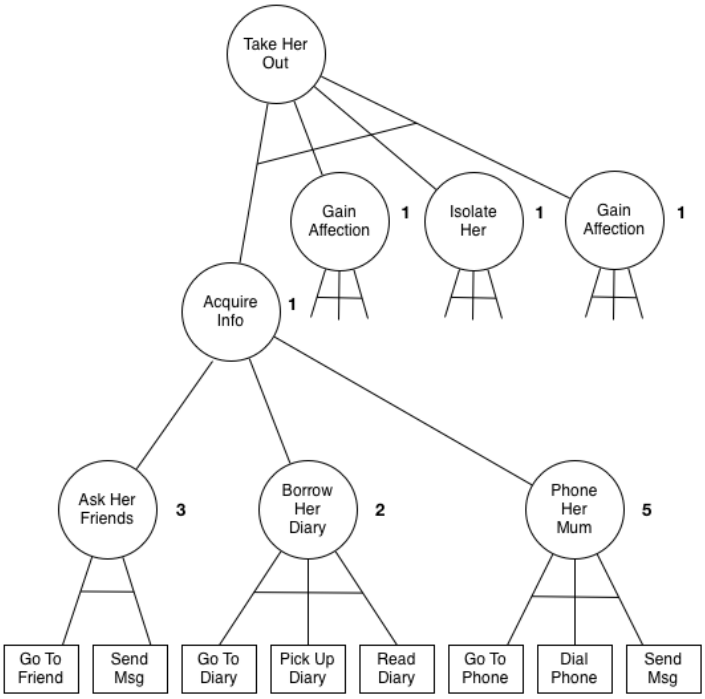

Figure 3. An HTN plan in Friends

The diagram illustrates the outlook of a hierarchical plan.

The plot is unpredictable, because of the following 5 factors: spatial locations (a character may not be reachable, or its path could be blocked), duration of actions, interaction between plans (there is a competition for resources, e.g. Rachel's diary), random outcome of some actions, user intervention

There is a separate mechanism that handles action repair, which is an alternative to re-planning. For example, Ross wants to steal Rachel's diary, but is unable to do so, because Phoebe is in the room. He either looks for another source of information (replan), or waits until Phoebe leaves the room (repair).

ACME [18] is another of a world based on physical modelling. It is based on the well-known cartoon the Road Runner and Coyote. In this system two autonomous agents are competing

against each other: the Coyote wants to capture the Road Runner, while the Road Runner wants to get away.

The player can add items to the scene, like a movie director. The Coyote then tries to catch the Road Runner using these items, but he always fails, similarly to the cartoons.

In this system the Road Runner and Coyote exist as finite state machine, for example the Road Runner can either run or stay still, which depends from its distance from the Coyote. The planning uses the HTN planning domain.

Wide Ruled [19] is an authoring tool, similar to mine, but with interaction features.

First the user selects from the window all the things related to the story: characters, environments, plot points, author goals. Then a window appears, where the story text is generated, here the user is allowed to interact with the story.

It is based on HTN planning as well. It contains a world composed of:

-characters: name, traits relationships
-environments: same as characters
-plot points
-goals and plot fragments

The algorithm works as following [20]:

1. Start with initial author goal, selected by author
2. Generator looks at every Plot Fragment for that author goal
3. Generator checks preconditions for Plot Fragments
4. Generator picks one Plot Fragment with satisfied preconditions, and then executes every story action in order (if a story action is a subgoal action, go to step 2).

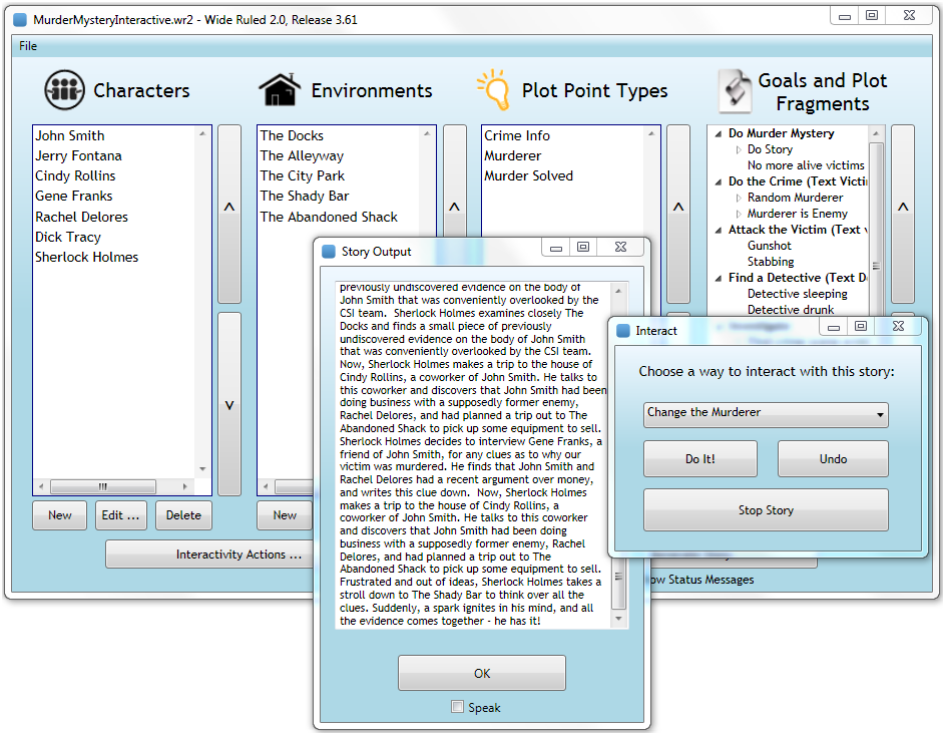

Figure 4. Wide Ruled in Action

The Sims, although not an IS system, but serves as an excellent example of autonomous agent simulation. It's a PC game, where the player guides through several characters' daily life. It's interesting to note, that Sims achieved a huge commercial success and became the best-selling PC Game at the time, despite the game doesn't have any real aim or objective, so it's not possible to win, therefore it's not a "real game". It's also an example for the problem of autonomous agents: despite it's very realistic, in itself cannot create entertaining stories (there is no such a thing as climax, or inciting event in the game).

3. Drama Theory

3.1. Introduction

Drama theory is an attempt to form theories from drama. The first influential work on the topic was Poetics (c. 335 BC) by Aristotle (384 BC – 322 BC). The second defining work was The Art of Poetry (Ars Poetica, c. 19 BC) by Horace. This remained the definitive work for centuries to come.

Aristotle divided the drama into several parts. Later the three-act structure emerged as the most-known dramatic structure. This divides the story into beginning, middle and end. This was the defining structure, until the 20th century, which saw an explosion in the number of books about story writing.

Vladimir Propp's "Morphology of the Folk Tale" (1928) was the foundation of modern narratology.

3.2. Definitions

It's essential to revisit some of the definitions connected to IS systems:

Agency is the player's ability to influence the plot. [21]

Interactivity is the player's ability to interact with any software, regardless of whether any story is involved, and it does not necessarily imply agency. [21]

Interactive story "is a story that the player interacts with by contributing actions to it. A story may be interactive even if the player's actions cannot change the direction of the plot." [21]

Interactive drama "is a first-person experience within a fantasy world, in which the user may create, enact, and observe a character whose choices and actions affect the course of events, just as they might in play." [21]

Immersion "is used informally to refer to a player or viewer's detachment from their true physical surroundings and their concentrated attention upon a game, story, task, or virtual space." [21]

Dramatic tension refers to the suspense an audience feels when experiencing a compelling story—the desire to know what will happen next. Authors create dramatic tension by engaging the audience's interest in characters or events and establishing a situation in

which something that the characters, and the audience, consider to be of value, is at risk. [21]

3.3. Modern Narratology

Many people attempted to develop a formal system for narrative stories. A short summary of the most important models [22]:

Vladimir Propp developed a symbolic notation to formalise the narratology Russian folktales. According to Propp, narrative functions are the basic primitives of folktales, which are independent from the characters executing them, as well as from the modalities. There are 31 such functions in Russian folktales. The ordering of theses cannot be changed, though it is possible to leave out arbitrary many parts.

Examples from the book:

I. ONE OF THE MEMBERS OF A FAMILY ABSENTS HIMSELF FROM HOME

1. The person absenting himself can be a member of the older generation (β1)
2. An intensified form of absentation is represented by the death of parents (β2)
3. Sometimes members of the younger generation absent themselves (β3).

II. AN INTERDICTION IS ADDRESSED TO THE HERO. (Definition: interdiction. Designation: γ)

1. Interdiction (γ1).
2. An inverted form of interdiction is represented by an order or a suggestion. (γ2)

III. THE INTERDICTION IS VIOLATED (Definition: violation. Designation: δ.)

Etc.

Greimas used the roles of actors based on Propps model. His system „The Actantian Model" each action breaks down into 6 components: the subject, which either wants or doesn't want to be cojoined with the object, the sender, who instigate the action, the receiver, who benefits from it. Additionally there is a helper and an opponent.

Barthes uses 5 interpretative codes to formalise story: ACT (Action), REF (Reference), SYM (Symbolic), SEM (Semantic), HER (Hermeneutic), where ACT and HER are determinants in the suspense of storytelling.

Bremond focuses his approach on characters: his model is centered on the Agent – Patient opposition, with characters having belief and motivation.

It's safe to say, Propp's system was the most influential, the story structure is the key to understand narrative stories. That's why I focused my research on story structure.

3.4. Story Structure

Many authors found the three-act structure for dramas general and too restrictive. Many models emerged, since then.

Freytag's pyramid divides drama into 5 parts [23]:

-*exposition*: where the characters, setting and background are revealed

-*rising action*: a related series of incidents build towards the point of greater interest

-*climax*: is the turning point, that changes protagonist's fate. In comedies it goes from negative to positive, in in tragedies it goes from positive to negative

-*falling action*: during which, the conflict between the protagonist and antagonist is resolved, which could result either in a positive or negative outcome. This may contain a moment of final suspense, in which the final conflict is in doubt

-*denouement*: conflicts are resolved, creating a normality or new equilibrium for the world.

It's safe to say, feature movies have the same structure, though a movie has a limited length of 2-3 hours. TV Series on the other hand are a bit different: a story usually doesn't conclude at the end of a TV episode, there are "cliffhanger" moments just before commercial breaks [4], and each episode presents more stories parallel.

The Anatomy of Story divides a structure of a movie it into 7 parts:

-Weakness and need: from the beginning of the story the hero usually has a psychological and in better stories a moral need as well. The hero must overcome these weaknesses to have a better life.

-*Desire*: is something the hero wants in the story. This is where the story becomes interesting. Desire is connected to need: need is something that the hero has inside, which he has to overcome to have a better life, desire is his goal outside the character.

-*Opponent*: the opponent is competing with the hero for the same goal.

-*Plan*: a plan is something that the hero will use to overcome the opponent and reach his goal.

-*Battle*: the battle is the final conflict between the hero and the opponent, which determines which character wins his goal

-*Self-revelation*: during the battle the hero has a revelation about who he really is. This is the most difficult act the hero performs during the entire story.

-*New equilibrium*: everything goes back to normal, and the desire is gone, but there is a major difference. Based on the self-revelation the outcome is either positive, or negative (the hero falls).

This model can be further analysed and extended into more parts. Save The Cat and The Anatomy of Story both further divide this into the following parts:

7 Parts	22 Steps	Blake Snyder Beat Sheet
Weakness and need	Self-revelation, need and desire	Opening Image
	Ghost and story world	Theme Stated
	Weakness and need	Set-up
	Inciting event	Catalyst
Desire	Desire	Break Into Two
	Ally or allies	B Story
Opponent	Opponent and/or mystery	Fun And Games
	Fake-ally opponent	
	First revelation and decision	
Plan	Plan	Midpoint
	Opponent's plan	Bad Guys Close In
	Drive	
	Attack by ally	
	Apparent Defeat	All is Lost
	Second revelation and decision	Dark Night Of the Soul
	Audience revelation	
	Third revelation	
	Gate, gauntlet, visit to death	
Battle	Battle	Break Into Three
Self-revelation	Self-revelation	Finale
	Moral decision	
New equilibrium	New equilibrium	Final Image

Table 2. Story structure in "The Anatomy of Story" and "Save The Cat"

Other formalisation of stories as in [2]:

Story Movement

Story movement defines how events are connected in the story. When it comes to IS systems the story paths are always linear, but it is worthy to take a look at other possibilities.

-Linear: the linear story takes a single character, from beginning to end. Most Hollywood films are linear.

-Meandering: the hero follows a meandering path, and covers a great deal of territory and characters.

-Spiral: the hero keeps returning to a single event or memory and progressively explores it.

-Branching: branching story is a system of paths that extend from a few central points.

-Explosive: this focuses on simultaneous actions.

3.5. Story Types

Movies can be grouped to genres: action, drama, comedy, horror, romance, but this grouping is too general, so many authors tried to have a better description of story types.

Save The Cat [3]

According to [3] movies can be grouped into 10 groups:

-*Monster in the House*: a simple set-up, a house and a monster. It is based on the primal instinct of survival, many examples include horror movies: Jaws, Alien.

-*Golden Fleece*: Road Movie, the hero goes on a journey, and the incidents he encounters make up the story, which give the hero a potential to grow. Star Wars, Back To The Future, Road Trip.

-*Out of the Bottle*: the name refers to the Djinn coming out of the bottle. This is when an unexpected big change happens, or a wish-fulfilment, which turns the world upside down, like Liar, Liar or Blank Check

-*Dude with a Problem*: “an ordinary guy finds himself in extraordinary circumstances”, a very popular genre with movies like Die Hard, The Terminator

-*Rites of Passage*: based on life-transitions, such as puberty, mid-life crisis, old age. Examples are: Ordinary People, American Pie.

-*Buddy Love*: is something typical to movies, buddy stories were invented so that the hero has someone to react to, a tool to the screenwriter to make display the characters thoughts, examples are 48 hours, Wayne’s World, Rain Man.

-*Whydunit*: here instead of the hero, the audience is the one that discovers something during the movie; a classical example is Citizen Kane.

-*The Fool Triumphant*: a fool takes up the role of the hero, who later turns out to be the wisest of use, and overcomes the much stronger bad guy; examples are Chaplin Movies, Forrest Gump.

-*Institutionalized*: is about a group of common cause, examples are American Beauty and One Flew Over the Cuckoo's Nest

-*Superhero*: is the exact opposite of Dude with a Problem, in this setting an extraordinary person finds himself in an extraordinary world, examples beside superhero movies are Gladiator and A Beautiful Mind

The Anatomy of Story [2]:

-*Journey plot*: the hero goes on a journey; he defeats the opponents and returns home. It usually covers a lot of time and space.

-*Three Unities Plot*: it covers one location, a short amount of time, and one story line

-*Reveals Plot*: the hero usually stays in one place and the opponents close in, and a great deal about them is hidden from the audience, resulting in many surprises.

-*Antiplot*: this plot focuses less on story and action, and puts the emphasis on the character.

-*Genre Plot*: are stories with predetermined characters, themes, worlds, symbols and plots, like the Superhero movies.

-*Multistrand Plot*: is the newest plot strategy, where the story is comprised of strands with different characters. Examples are Pulp Fiction or Cloud Atlas.

3.6. Characters

Making good characters is essential to the success of a story; even Aristotle had a great emphasis on characters (ethos) in Poetics. The following analysis is based on, yet again, The Anatomy of a Story.

Character traits

The place of each character in the story is defined by the following 6 properties:

-weakness
-need, both psychological and moral
-desire
-values
-power, status and ability
-how each face the central moral problem

Character types

The characters occurring in movies could be grouped into archetypes. One character may belong to more types, for example Luke Skywalker (Star Wars, 1977) is a prince-warrior-magician. Each character type has its strengths and weaknesses.

-*king or father*: leads his people or family with wisdom. He may force the people to act accordingly his own strict set of rules.

-*queen or mother*: provides care for the people, but may use guilt and shame to hold them close.

-*wise old man*: passes on knowledge and wisdom, but may force their students to think in a certain way

-*warrior*: the practical enforcer of what is right. Can live according to the motto "kill or be killed"

-*magician*: can control the hidden forces, but he could also misuse it to enslave others.

-*trickster*: uses confidence and trickery to achieve his goals, but may become a selfish liar.

-*artist or clown*: defines excellence or shows what doesn't work, but he could be the ultimate fascist.

-*lover*: provides care. Can lose himself in the other or force the other to stand in his shadow.

-*rebel*: stands out from the people, acts against the system, but often cannot provide a better alternative.

3.7. Other Problems

The problem of Amnesia

"Resolutions to Some Problems in Interactive Storytelling" defines the problem of Amnesia. That is, the characters of the story world are part of their world and know what's going on in them; on the other hand players have to be introduced to the rules. This problem could come up not only in games, but in movies and novels as well, especially in science fiction and fantasy. In TV Series viewers are gradually introduced to the world (in Naruto the authors exploit the dumbness of Naruto to explain the world to the viewers). In movies getting to the inciting event as soon as possible and gradually introducing the audience to the world is the preferred way as well.

The problem of awareness

"Reality is merely an illusion, although a very persistent one" – Albert Einstein

One of the problems in screenplay writing, and generally in stories to decide what to show and what to hide from the audience. For example, showing the audience that the hero sits on the toilet is unnecessary, except when there is a bomb in the toilet. Or showing the audience who was the murderer in the beginning of a thriller story kills the whole movie. A lot of effort is invested in this problem when creating a movie script.

In interactive story systems I didn't find any mention or research of this topic, though it would be worth investigating. In planning an action already implies something important happening. On the other hand systems based on something other than planning this might be a problem: consider the example in Scheherazade. The evaluation tests showed there were two completely unnecessary events in the graph: "wait in line" and "open door". Besides it's still a question whether the audience should be aware some events, like "Sally calls police" could be shown to the audience, or could be hidden from it.

It is not clear where should be this problem addressed: shall it be the script writing system or the rendering system? Showing the same scene from different views could add different interpretations to the viewer, so he might end up with a completely different perception of the story.

Character awareness is an easier problem. IDTension [24] provides a framework that provides predicates such as WISH, CAN, KNOW, WANT and a set of rules that produce the possible actions, thus effectively modelling character awareness.

Memento is a movie that plays around this problem.

4. Planning

4.1. Introduction to Classical Planning

Interactive Story generation is based on planners. There are many different kinds of planning languages. Unfortunately, there isn't a complete categorization of planning languages. Instead there is classical planning, and there are extensions to it.

-Planner language: planners are composed of a description of the initial state, a list of available actions, that change the state, and the description of the desired state.

A classical planning is composed of the following:

-*objects*: a list of objects.

-*states*: a list of predicate instances. Each instance is composed of a predicate and a list of objects. Note that classical planning doesn't allow negative statements.

-*actions*: deterministic, observable. Each Action has a precondition and a list of effects to add and remove.

-a single *agent*: there is only one plan by one agent, instead of many plans running concurrently. An example for more plans by more agents is the Madame Bovary (citation comes here)

Properties of planners according to [25]:

-A planner is *sound* if any action sequence it returns is a true solution

-A planner is *complete* if it outputs an action sequence or "no solution" for any input problem

-A planner is *optimal* if it always returns the shortest possible solution

Planners have a lot of practical uses asides from IS, including [26]:

-games such as sokoban, freecell, chess, checkers
-genome rearrangement
-analysing computer network security
-military training (SHOGUN)
-manufacturing (PARC)
-transportation, traffic control
-exploration mission (Mapgen)

-power supply restoration
-workflow composition
-scheduling applications
-Hubble Space Telescope

4.2. STRIPS, ADL and PDDL

STRIPS planner

STRIPS is a name of the planner developed for 1971 SRI International, and also the name for the formal language of this planner. The language is the basic for nearly all the planning languages, a notable exception is the HTN planning.

STRIPS is PSPACE-complete. Finding a plan is hard in the worst case.

In STRIPS states are represented as a conjunction of positive literals. Every action has a precondition and an add list and a remove list.

Formal definitions [27]:

-A state S is a finite set of logical atoms

-A STRIPS action o is a triple

$$o = (\mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o))$$

where pre(o) are the preconditions, add(o) is the add list, del(o) is the remove list of the action, each being a set of atoms. The result of applying a single STRIPS action to a state is defined as follows:

$$\mathrm{Result}(S, < o >) = \begin{cases} \big(S \cup \mathrm{add}(o)\big) \setminus \mathrm{del}(o) & \mathrm{pre}(o) \subseteq S \\ \text{undefined otherwise} \end{cases}$$

The result of applying a sequence of more than an action to a state is recursively defined as:

$$\mathrm{Result}(S, <o_1,..,o_n>) = \mathrm{Result}(\mathrm{Result}(S, <o_1,...,o_{n-1}>), <o_n>)$$

-A planning task P = (O, I, G) is a triple where O is the set of actions, and I (initial state) and G (goal state) are a set of atoms.

-Given a planning task P = (O, I, G), a plan sequence is P = <o_1, …, o_n> of actions in O that solves the task, i. e., for which G $\subseteq \mathrm{Result}(I, P)$ holds.

An example problem in STRIPS:

```
Objects: BoxA, BoxB, Ball
Initial State: In(Ball, BoxA), Container(BoxA), Container(BoxB), Object(Ball)

Goal: In(Ball, BoxB)

Actions:

Take(x,y)
Preconditions: In(x, y)
Postconditions:Free (x), not In(x, y)

Put(x, y)
Preconditions: Free(x), Object(x), Container(y)
Postconditions: In(x, y)

Solution:

Take(Ball, BoxA)
Put(Ball, BoxB)
```

Note that “not” refers to removing the predicate from the state, unlike in ADL predicates, STRIPS predicates don’t have a negative form, only the lack of a positive form.

A notable application is the video game F. E. A. R., which won several awards for its A. I., which used STRIPS planning.

ADL planning

ADL is advancement over STRIPS. ADL-A was proposed in 1987 as an improvement over STRIPS and had several extensions, namely ADL-B, ADL-C.

ADL planning is PSPACE-complete problem.

Improvements in ADL over PDDL [28]:

-Negative literals are allowed, for example the predicate

-STRIPS is based on the Closed World Assumption, meaning that unmentioned literals are false, while ADL is based on the Open World Assumption, where the unmentioned literals are unknown.

-Quantified variables are allowed in goals

-Goals may involve conjunctions and disjunctions

-Conditional effects are allowed

-There is an equality operator

-Objects have types

ADL plans can be translated into STRIPS plans. In fact, most of the efficient ADL planners work by translating the problem instance into STRIPS.

PPDL planner

PDDL is an effort to generalise planning problems. PDDL contains STRIPS, ADL and much more. It was developed in 1998, and had many versions since, the most recent being PDDL3.1.

There were several PDDL planning competitions in the past.

Notable features are: numeric fluents, plan-metrics, durative actions, derived predicates, timed initial literals, state-trajectory constraints, preferences, object-fluents.

A short description of the language, taken from [29]:

```
(define (domain sample-world)
      (:requirements :strips :equality :typing :conditional-effects)
      (:types location physob)
      (:constants (B - pyhsob)
      (:predicates (at ?x – physob ?l – location)
                     (in ?x ?y – physob))

(:action mov-b
      :parameters (?m ?l – location)
      :precondition (and (at B ?m) (not (= ?m ?l)))
      :effect (and (at b ?l) (not (at B ?m))
            (forall (?z)
                  (when (and (in ?z) (not (= ?z B)))
                        (and (at ?z ?l) (not (at ?z ?m)))))) )
```

This specifies a simple problem domain, where an object can be moved from one location to other location (it has to be a different one).

There exists many extension and variations of PDDL. Some examples:

-PDDL+: autonomous processes and events

-MAPL: modal operators, non-propositional state variables, events

-OPT: ontologies

-PPDDL: probabilistic effects

-RDDL: partial observity

4.3. Other planning languages

HTN planner

HTN stands for “Hierarchical Task Network”. This approach is completely different from the ones before and is used for domains where tasks are organized in a hierarchy.

The problem with HTN-planning, that it doesn’t have an accepted theoretical framework, there is much research to do in the formalization of HTN-planning.

HTN is useful when tasks can be organized in hierarchy. Actions are divided into abstract (non-primitive) and primitive actions. The plan starts with abstract operators, which are then decomposed into primitive actions by planning techniques. The final plan only contains primitive operations.

For an example HTN-tree, see the project Friends.

First-order planners

So far the planners considered were propositional planners, based on propositional logic. Currently they are the state-of-the-art solution to planning problems.

However, it is possible to use first-order logic to solve planning problems; these are called first-order planners [30]. *First-order logic* is an extension over propositional calculus, which uses quantified variables over objects: ∀ “for all” and ∃ “there exists”.

I’m going to describe two systems here, the Situation Calculus and the Event Calculus.

Situation Calculus

Situation Calculus is composed of situations, actions and fluent:

-Actions are first-order terms with parameters and can be quantified.

-Fluents are relations whose truth value varies over situations.

-A situation is either:
- *init*, the initial situation,
-*do(A, S*), the situation resulting from applying action A in situation S, if it is a legal action

There is a difference between a situation and a state. A state could be associated with multiple situations, if multiple sequences lead to the same result. A state is reachable if a sequence of actions exists, that can reach from the initial state to that state. Not all states are reachable.

Let us revisit our example from STRIPS:

do(Put(Ball, BoxB), do(Take(Ball, BoxA), init))

GOLOG is a programming language based on the Situation Calculus.

Event Calculus

Event Calculus [31] is another first-order system. This system models how the truth values change over time, which can be either discrete or continuous.

Events are modelled as occurring at a time. Event E occurring at time T is *event(E, T)*, this is analogous to the one found in situation calculus.

Events make some relations true and others no longer true:

- *initiates*(E,R,T) is true if event E makes primitive relation R true at time T.
- *terminates*(E,R,T) is true if event E makes primitive relation R no longer true at time T.

In event calculus relations are reified, meaning that they are represented as functions instead of predicates.

A simple example:

$holds(R,T) \leftarrow event(E,T_0) \wedge initiates(E,R,T_0) \wedge (T_0<T) \wedge \sim clipped(R,T_0,T)$.

This means that an event in the past ($T_0 < T$), which initiated f as an effect and it wasn't made false in the meantime (clipped). Clipped could be axiomatized or expressed as a formula.

Revisiting the STRIPS example:

```
initiates(take(Ag,Obj), carrying(Ag,Obj) ,T) ← poss(take(Ag,Obj) ,T) .

terminates(take(Ag,Obj) ,sitting_at(Obj,Pos) ,T) ← poss(take(Ag,Obj) ,T).

poss(take(Ag,Obj) ,T) ←autonomous(Ag) ∧ Ag ≠Obj ∧ holds(at(Ag,Pos) ,T) ∧
holds(sitting_at(Obj,Pos) ,T) .
```

Event Calculus can be run as a PROLOG program.

4.4. Preference-based Planning

Preference-based planning extends planning with a specification for measuring quality. This is useful because usually there are a lot of plans that solve a particular problem, but only a few of them are desirable or useful. This can be illustrated with the following real life example:

Suppose you need to travel from London to Amsterdam. There are many feasible ways to do this, either by rental car, train, bus, or plane.

If you want the cheapest solution, you should take the bus, which costs 50 euros and takes about 12 hours, on the other hand, if you want the fastest route, you should take a plane for 120 euros, and be there within an hour.

This motivates the extension of planning languages with preferences. A preference could be a state or a metric function, which ranks the plans. I'm going to discuss two systems here: PPLAN and HTNPlan-P.

PPLAN [32] is an extension to PDDL that supports preferences. This divides the preferences into 4 categories: state, action, trajectory, multi-dimensional preferences. Preferences are expressed by a declarative language. PPLAN is implemented in Prolog.

HTNPlan-P [33] is a preference based HTN planner. It uses an extension of PDDL 3 to decompose HTN, PDDL 3 introduced preferences. It uses many heuristic algorithms to improve the performance. This system differentiates between precondition preferences and simple preferences, and introduces a metric function, which expresses the quality of the plan.

4.5. Problems in Planning

Sussman Anomaly [34]

Sussman Anomaly is a problem in planners. It was described by Gerald Sussman in the 1970's. In this problem a planner has to put boxes on top of each other, such that A is atop B and B is atop C. This can be described as two goals:

1. A is atop B
2. B is atop C

The starting position looks like this:

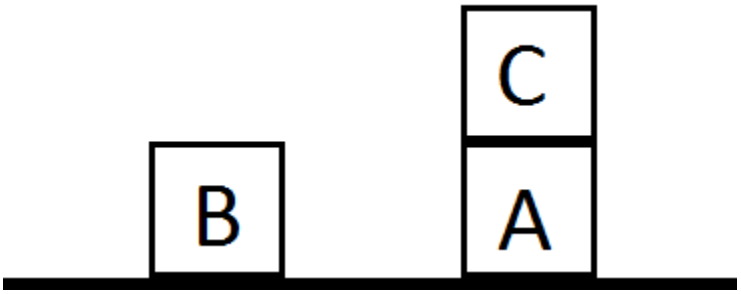

Figure 5. Sussman Anomaly

Suppose the planner starts by looking for a plan to solve 1. First it moves C from the top of A. Then it moves A on top of B. The first goal is solved now, but the agent cannot pursue 2 anymore, only by "unsolving" 1.

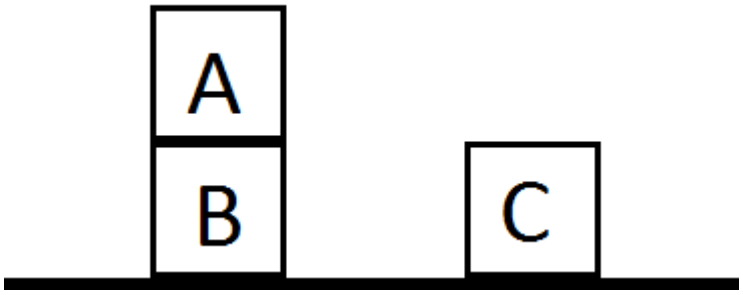

Figure 6. Goal 1 reached

Similarly, if the planner decides to pursue goal 2 first, it moves B atop on C, now goal 2 is solved, but goal 1 cannot be solved, it undoes the last step.

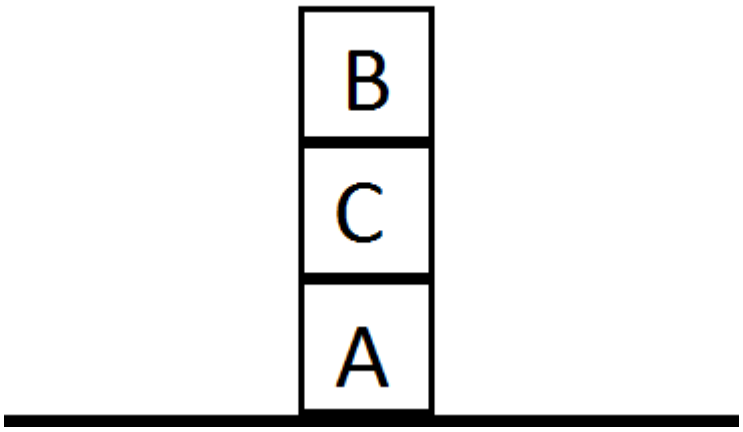

Figure 7. Goal 2 reached

Most modern planners can handle this problem but this illustrates the difficulty of making good planners.

Unfortunately there is a great fragmentation amongst planners, and the sources online are vague, I found many broken links and source files that not working. Therefore I decided to write my own planner, called SimplePlanner, which I can customize to meet my needs (see Implementation).

SimplePlanner is a STRIPS-like planner, with negative preconditions and a preference matrix. It uses the most straightforward planning strategy.

Planning strategies [35]

Most planning problems can be represented as a state-transition system, in which the world consists a finite number of states and transitions between states. Notable exceptions are HTN planning and constraint problems.

A state-transition can be represented as an arc-labelled directed multi-graph. Each node is a state and each arc is an action.

Planning can use both forward and backward search strategies (progression and regression planning).

Some planning strategies:

-*Heuristics*: the main approaches are different forms of relaxation: the original problem instance is simplified or the definition of a solution is simplified to be efficiently solvable.

-*Symmetry reduction*: tries to reduce the search space by recognising symmetries. These are caused by two objects being interchangeable. If A and B are interchangeable and a plan involves A and B, then there is a symmetric plan with the roles of A and B changed.

-*Partial-order reduction*: the idea is to recognise different forms of independence of actions. If two actions A and B are independent, in a sense that applying A and then B is the same as applying B first then A, then one only needs to consider one of these orderings.

-*Reduction to SAT*: plans can be found by solving a SAT problem. A SAT (Boolean satisfiability problem) is a problem of determining whether there exists an interpretation that satisfies a given Boolean formula. Simply put, is it possible to substitute the values true and false to logical expression in a way that it evaluates to true. For example, the formula “A AND NOT A” always evaluates to false, meaning it’s unsatisfiable. SAT is NP-complete.

4.6. Planner Implementations

PDDL has many implementations, but most of them don’t contain all the language features.

Unfortunately the literature for planning is badly documented: during my research I came across a lot of broken links, badly documented papers and program, and incorrectly working programs.

To measure their effectiveness there is a regular competition, called International Conference on Autonomous Planning and Scheduling (ICAPS). (www.icaps-conference.org)

The first one was organised in 1998, the most recent one in 2014. In 1998 there were only 5 planners competing in one category, but in 2014 there were 3 categories (the continuous probabilistic planning track was postponed to 2015), altogether (67 + 7 + 11 =) 85 submissions.

A paper released in 2007, “Planning Algorithms for Interactive Storytelling” by Barros and Musse also compared the state of the art planners. Unlike in the planning competitions the test is only composed of one problem, it would have been better to include different problems and problem domains as well. The task was a simple story problem, called Ugh’s story, where a caveman is required to obtain some information from the others.

The results are shown in the following table:

Algorithm	TH	EO	NP	CE	EP	Opt	POP	AC	NV	Time
FF	✓	✓	✓	✓	✓	×	×	×	×	0.015
GraphPlan	×	×	×	×	×	✓	✓	×	×	0.138
HSP	✓	✓	×	×	×	×	×	×	×	0.031
HSP*	✓	✓	✓	×	×	✓	✓	✓	✓	3.641 - 7.25 [1]
IPP	✓	✓	✓	✓	✓	[2]	✓	×	×	0.106/0.032 [3]
LPG-TD	✓	✓	✓	×	✓	×	✓	✓	✓	0.680 / 0.169 [4]
Marvin	✓	✓	✓	✓	✓	×	✓	×	×	0.017
Metric-FF	✓	✓	✓	✓	✓	×	×	[5]	✓	0.059 / 0.018 [4]
SatPlan_2004	✓	×	×	×	×	✓	✓	×	×	0.247 - 0.512
STAN 4	×	×	×	×	×	×	✓	×	×	0.093
TLplan	×	✓	✓	✓	✓	[6]	×	✓	✓	[7]

Table 3. Comparison of planners

Abbreviations: Type hierarchies (TH), equality operator (EO), negative preconditions (NP), conditional effects (CE), existential preconditions (EP), optimal (Opt), partial-order planner (POP), action costs (AC), numeric variables (NV), time to solve test problem (Time). Notes: [1] The hspb variant took considerably more time (4 min).
[2] Can be optimal or not, depending on how it is configured.
[3] First time is for optimal configuration; second time for nonoptimal configuration.
[4] First time with numeric variables; second time without numeric variables.
[5] Not explicitly supported, but can be easily emulated.
[6] Optimal as long as an appropriate control formula is used.
[7] We could not create a usable control formula for our test problem, so timing could not be assessed.

There are many extensions possible to classical planning problem, and combining these into a planner gives even more combinations [1]:

-include duration or cost with each action

-include negated preconditions

-type hierarchies: the ability to subclass objects

-equality operator: checks for equalities, for example I can't give a gift for myself

-conditional effects: when I give a gift to someone, he might like or not based on some condition

-optimality: guaranteed to find the best possible solution to a problem

-modal planning [36]: predicates inside predicates, for example intents in [36] "intends Jafar (married Jafar Jasmine)"

Some of these properties can be translated into a classical planning problem. Like types could be represented with more statements, for example Type(Scrooge, Hero), Subtype(Hero, Character), or modal planning problems can be compiled into classical problems as it was detailed in [36].

5. Implementation

5.1. Story World

In my project I went for the purely story based approach, because after reading books about movie script writing, I concluded the primary focus is placed on that and I wanted my application to be close to the real life one as much as possible.

The biggest challenge was to take an informal narrative system and try to turn it into computer code. Essentially the project tried to be the implementation of The Anatomy of Story, because in order to create good stories it's not good enough to have a good planner and a huge story world: a story has to be entertaining, and conform to many criteria as well. If the story is just a simulation and doesn't contain elements, such as revelation, apparent defeat, then it is completely useless.

Implementing the Story Generation part was a challenging task. First it had to be decided which story parts to include and how to implement them. Also, TV series have a slightly different story structure (the book about TV series) defines an additional part "Cliffhanger", is usually before advertisements and at the end of story, is usually short scene, which sets the dramatic tension very high.

I decided to split the system architecture into 2 levels. The top level takes care of creating the story milestones. This process wanted to mimic the way of creating a story draft in real life. There are some basic rules implemented and the program randomly selects between possible solutions.

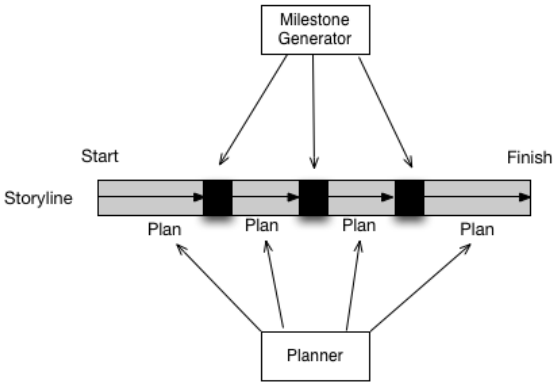

Figure 8. Milestones and planning

The second, lower level is the planning level. In this level the planner tries to find a way between the story milestones. This is where the Planner Executor and the Planner classes

could be found. This approach is unique; I wanted to use planning, but also to have some kind of a hierarchy, like in HTN.

Requirements

-simplicity: it's crucial that everyone without any understanding of story writing or programming is able to use the program. Also, programmers have to be able to reuse project components easily.

-completeness: have a full story from the beginning to the end

-believability: stories have to be fully interpretable by humans

-performance: make better stories than a person without any knowledge of story writing

-randomness: make different stories each time the application is run

-interactivity: the user has the ability the adjust preferences

-replanning or error correction: currently there is no mechanism for replanning. If, for some reason it's not possible to find a feasible solution between two milestones, the executor skips that planning state. The reason why users not able to select the available predicates in the first dialog roots in this problem: that would make the selection process way too complicated and may result in too many errors or aborted problems.

-live performance: generating a story shouldn't take long, ideally less than a minute.

Environments: there is no separate mechanism for handling environments. A place is implemented as an object in the planner world and represented by a predicate.
An alternative solution was considered: each story part takes place in a different environment, and the transition between the environments are handled by the story part generator.

Premise: it is a challenging part to implement a premise in software. The closest thing I could come up with is to include a story arc, which categorises the story into two parts: acquire scenario and lose scenario.

Character traits: character traits are left to the user to define. Each action could have a precondition certain character properties, e.g. Strong, leader, clumsy. The implementation is straightforward, each trait is a predicate.

Symbol web: Coming up with good symbols is hard, but fortunately there are common ways to do it. One example is superhero movies, where the superhero takes the properties of the animal it symbolises (it's not always the case, for example in DuckTales ducks can't fly or some can't swim) or horror movies, where the symbol web is based on the symbols of Christianity.

Morale: it's a very challenging task to implement story morale in software. A simple solution could be, that Scrooge realises the importance of the treasure he acquired, and gives it back (as it happened in a few episodes).

Story structure: the story has linear structure at the moment, implementing a more complex structure is beyond the scope of this Thesis.

Genre: thankfully to in dependencies in the files, it is possible to simply change the genre of a story. Each genre has its own file. Just simply change the dependencies in the main file be loaded, and the possible actions will then reflect that genre.

For example, it is possible to define a Fantasy genre, in a file which contains all fantasy related things, actions and predicates related to fantasy, like the “transform into frog” action.

Story parts: this is the heart of the application. This process sets the milestones in the program. Story parts have their own classes. It is possible to write new classes, there are only 7 included by default, because DuckTales is a simple story, it doesn't require 22 parts. Each story part generates a list of predicates to achieve and a list of variables. The process is randomised, so it doesn't give the same results when it's run multiple times.

Top-down structure:

One of the challenges of the project is to determine how much detail each action has. Generally, it makes sense, that each action is detailed at the same level. It is not quite clear that somewhere should some actions go, like a revelation scene, when a fake-opponent becomes an ally could be done as a story milestone or as an action, or there could be a variable introduced as revelations.

It was considered to make the story to have a top-down architecture. In this architecture every action could be detailed more. Like a fight scene or a car chasing scene could be progressively detailed into more detailed actions, and so on, similarly as in [17]. This was discarded because of its complexity.

It is unclear how to implement HTN plans for a complete story. Problems could arise when introducing cost variables because I haven’t seen any HTN planners involving this. Also when something changes at some point of the story it is not clear how this change could affect the other branches: like an important character dies, who could have a role at later branches.

Another consideration was that it should do some kind of a “cleaning” between milestones. Meaning that useless predicates are thrown away from the state, like in the introduction part the IDEA predicate is useless in the rest of the application so it could be just simply deleted, among many others.

Having a well-defined 3D environment would be helpful in terms of defining actions. If the project could be connected into such an environment, defining the possible actions would be less ambiguous.

5.2. Problem Domain

Our goal was to create believable DuckTales stories. Most of the DuckTales stories are based on 3 scenarios, but of course there are many episodes that are based on a different plot type (like the *Village Idiot* plot featuring Donald or Launchpad).

The first scenario is based on the Obtain Arc. This is probably the most common story arc in DuckTales stories: this is where our heroes decide to obtain a treasure and go on an adventure to achieve their goals. In the meantime a selected group of villains try to stop them from achieving their goals. There is a possibility of monster encounters and the use of science fiction and magic elements. The monsters or neutral characters sometimes reveal themselves as good characters.

This scenario is based on the *Journey Plot* or *Golden Fleece*.

The second one is the reverse of the first arc: the lose arc. This time introduction usually focuses on the villains. They have a plan to steal something from our Heroes, or take over the world in some way. This arc is when the villains attack our heroes and first they usually succeed, but then the heroes take back what they lost and goes everything back to normal.

This is the *Reveals Plot*.

There is a third possible scenario, which wasn't included in my wizard: the "something goes utterly wrong" scenario. This usually starts by Gyro inventing something entirely new, usually belonging to the world of science fiction. This invention first seems to be something revolutionary and very good, but it turns out to have a fatal flow, which causes all the problems. The world turns upside down, but in the end the heroes manage to repair everything.

This is the *Out of The Bottle* plot.

DuckTales is short and aimed at kids, so it is a very simple story, and is usually independent from other episodes. For example, episode 16 "The Money Vanishes" could be divided into 4 parts: set-up, attack, battle, resolution, where the battle part is a long feature, including many comical scenes, but the story itself doesn't progress a lot. Some other episodes on the other hand contain many turning points and revelations. After analysing many episodes, I decided to include the following Story Parts in my project, which are mostly based on [2]:

7 Story Parts

Based on the analysis of the story structure, I decided to divide the story into 7 parts. These milestones serve another reason as well: to make the planning easier by decomposing the story into more and easier reachable states. This is essential to the success of this project, especially considering the poor performance of SimplePlanner.

-Set-up: the DuckTales universe is composed of many characters. In this part we are introduced to our main heroes, the environment, but also the main villains come into play. We are also introduced to the object of this episode, which could be a treasure to be obtained by Scrooge, or something belonging to Scrooge, and targeted by villains, like Glomgold.

-Plan: our heroes and villains start working towards their goals.

-Attack: the villains close-in and the heroes' plan fails

-New Plan: the heroes have to come up with a new plan

-Battle: an action packed part, where the heroes and the villains are fighting over the treasure

-Apparent Defeat: is the all is lost moment, when it seems like everything is lost, and our heroes defeated.

-Final battle: using the new idea, our heroes defeat the villain

-Resolution: a new equilibrium is set; in TV Series it's usually everything goes back to normal

5.3. StoryWriter Program

The project was done in the Java programming language. It was selected because it's an efficient, platform independent high-level language, with a powerful core library, has its own garbage collector and it is widely used, a standard for many universities, thus it could contribute to later IS research.

To make the project reusable I decided to stick with the core Java libraries and not to use any outside libraries. Therefore I had to implement a few helper classes on my own, such as the Triple, and I copied some others from online resources, such as the CheckBoxList (these are referred in the documentation).

The compiler is version is 1.7, this is the minimum version required to compile the project, as it uses some of the Version 7 features.

The project was developed in the Eclipse IDE. This is pretty much a standard these days for most of the Java applications. The GUIs were hard coded, simply because I haven't found any good GUI creating add-ons in Eclipse, that's one of the biggest shortcomings of Java.

My project is composed of the following 5 packages:

interactivestory: the basic package containing the core classes needed to represent the story elements, such as Action, Predicate or StoryWorld. Also, it contains the StoryWriter class, where the entry point of the application is found.

-StoryWorld is the class where all the fields related to the story are stored. This is the central data structure of the whole application. This is where the parser loads the stored file.

-StoryWizard defines the workflow. It is composed of different classes extending the Dialogue class. This class goes through each class and passes the StoryWorld to them, which is being modified by the Dialogues. Adding and removing Dialogues is easy to the workflow, but currently there is only one such workflow is implemented.

-Action: a class for an action in planning.

Actions are created by the ActionBuilder class, which is based on the Builder pattern. The creation mechanism of Actions is complex, that's the reason behind using this pattern.

-ActionInst: holds an Action and an array of ObjectInst.

-Logical: an abstract class for logicals, which could be either a logical expression (LogicalExpr) or PredicateInst.

-LogicalExpr: holds a logical expression. It is composed of an operator, and one or two Logicals, which in turn might another LogicalExpr class.

-ObjectInst: represents an object in planning. Planning objects are uniquely identified by their names.

-Operator: represents a logical operator: AND, OR, NOT, NONE.

-Predicate: represents the concept of the same name in planning. A template for creating PredicateInst classes. It contains a name the number of arguments.

-PredicateInst: an instantiation of a predicate. It contains a Predicate class and an array of ObjectInst classes.

-State: represents a list of statements in planning.

-StoryVariables: a class that holds all the numeric variables related to planning. Currently it's composed of time, action, drama, comedy, romance, outcome, achieve. The distance between two StoryVariables class is calculated by using the Euclidean distance.

-StoryWizard: this defines the default workflow for the project. It holds a StoryWorld class and all the Dialogues for the project.

interactivestory.loader: this contains the parser that loads the files.

Parser: this class loads the files into a StoryWorld.

First a PDDL language was considered. Unfortunately, PDDL has a bad syntax in my opinion, and it didn't have a publicly available parser in Java.
I decided to come up with my own, very simple language.
After considering a few parser generators, I decided to write my very own Parser, because it allows me to customize to meet my requirements and it's possible to change it. It also makes the project less dependent on outside packages, which was one of the aims.

A Parser uses a class called TokenTree. A TokenTree has one field, a list of Type – Object pairs. Each Object in turn could be another TokenTree or an element like a PredicateInst. This is then converted into a LogicalExpr class.

The file formats allow a hierarchy of dependencies. In this case the parser goes through all the dependencies using a list, and then puts all the dependencies of the file at the end of the list. Then it goes backwards and puts each file, which is not already included on the top of the file. This way the User gets a concatenated file of all dependencies.

ParserTest: JUnit test cases for the Parser class.

interactivestory.planner: this class contains the planner implementation and the classes related to it.

-ExecutorCallback: an interface for getting the results of planning (Observer pattern)

-Planner: an interface for a planner. Every planner implementation should use this interface. This interface extends the Runnable class.

-PlannerCallback: the results of a planner thread are acquired by using this interface (Observer pattern yet again)

-PlannerExecutor: this class handles all the planner instances.

-SimplePlanner: the implementation of my planning algorithm.

interactivestory.utils: these are reusable Java components, a similar collection to Apache Commons. I didn't want to use any outside classes, so I decided to implement a few useful classes, missing from the Java Core.

-Pair: a generic immutable implementation of a tuple holding 2 objects.

-Triple: a generic immutable implementation of a tuple holding 3 objects.

-CheckBoxList: a list of checkboxes. The code was acquired from an online source.

-Utils: all the helper methods come here, such as concatenating 2 arrays.

-Permutation: a generic class, the input is a collection and it returns all the possible permutations of this collection by using the hasNext() and getNext() methods.

-Combination: a generic class, the input is a collection and a length, and returns all possible combinations of the elements in the collection in the given number of places. For example given the numbers 1-45 and the length 6, it returns all the possible combinations of the lottery.

-**interactivestory.wizard**: this contains the GUI elements, and classes that handle the User interaction. The Dialogues are extending the Dialogue class.

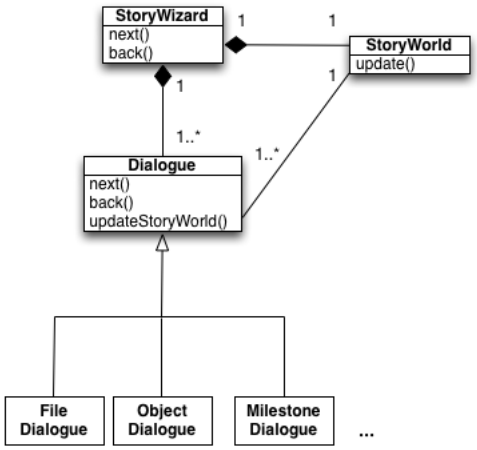

Figure 9. Dialogue class

The figure depicts the structure of a Dialogue class. It contains two methods that control the workflow: back() and next(), which tell the StoryWizard class, whether to make a step forwards or backwards in the workflow. When calling the next() method, the updateStoryWorld() method is called, which calls the corresponding updateXY() method of StoryWorld.

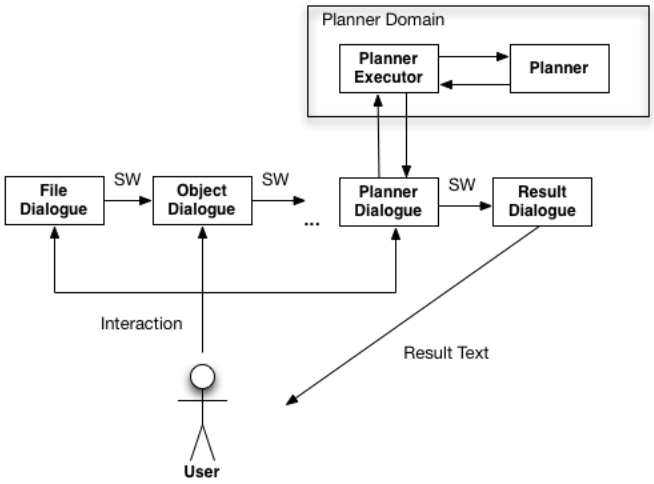

Figure 10. Default Workflow

A story is created the following way:

1. There is a default class that extends the Workflow class. This class doesn't have much use of yet, it is used simply for organization.

2. This class contains a StoryWorld class, which contains all the data related to the Story. This class is loaded by a Parser in the FileDialogue class.

3. Each time the User presses the next button on a dialogue, it updates the StoryWorld class by using the method updateStoryWorld(). Then the StoryWizard calls the next Dialogue in the workflow.

Dialogues in the order of occurrence in the default workflow:

-FileDialogue: a dialogue that allows the user to select which file to load and to which wizard to start: "planner only" jumps straight to the planner, while "story wizard" goes through the milestone generating steps.

-ComponentDialogue: allows the user to select/deselect which objects, actions and story parts to include in the project.

-ArcDialogue: select the properties of the story, such as main hero, treasure and story arc. The following figure depicts it:

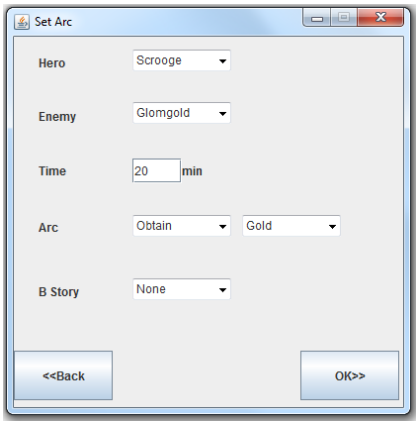

Figure 11. Arc Dialogue

Not that time and b story are not implemented yet.

-MilestoneDialogue: displays which story milestones are generated, shows all the predicate instances and story variables.

-PlannerDialogue: allows the user to set and start the planner. The user is able to set the number of planner threads and start the planning.

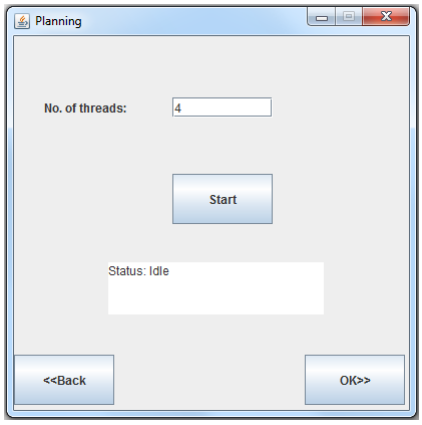

Figure 12. Plan Dialogue

-ResultDialogue: displays the results (if any) as a text.

5.4. Planner

My system architecture is unique in the way, from other planner architectures, that it's divided into 2 parts: an executor class, and several planner threads. The planner executor receives a list of states as the planning problem. After that, it starts several planner threads, which then try to solve the problem. It might be the case, that a thread can't find a feasible solution. The executor class then compares these solutions and picks the best ones for each state. Then it returns the complete result as a text.

SimplePlanner: I decided to write my own planner to suit my needs. Writing an optimal planner is out of the scope of this Thesis, in fact could be a Thesis of its own, so I decided to

make it as simple as possible. Not being an optimal is not a problem: some randomness is desirable; otherwise it would create the same story every time it's used. On the other it should be close to being optimal, otherwise the stories it creates are useless.

Summarizing the needs for my planner:

-Soundness, i.e. it has to produce correct solutions,
-solve simple problems quickly (<1s),
-solve a simple path finding problem,
-solve the Sussman Anomaly,
-produce feasible solutions, ones that don't have repetitive steps and cycles,
-handle preferences,
-solve complex solutions in a reasonable amount of time (<1min).

SimplePlanner version 1: it is based on a very simple idea: for each state, randomly go through each action and randomly substitute objects into it. If the objects satisfy the precondition, evaluate the action, and carry on to the next state, otherwise step back a state with some probability, or repeat.

This idea was too simple to give good results, so I decided to make it a bit cleverer.

SimplePlanner version 2: to improve the efficiency I came up with a new algorithm on my own. The idea is to try to work towards the target state.

1. Go through each action

2. For each action:
- get a list of missing statements
- check whether the added effects list contains any of these predicates
- if it does, then try to substitute the desired object into that place of the template
- return an array of substitutions, which may contain null fields.
- in the returned array randomly substitute objects in the null fields, until the precondition of that action is satisfied

example: there is the target FULL(DAGOBERT). When going through actions, the action EAT has the added effect FULL(x). So the planner substitutes Dagobert into x, so the template becomes EAT(Dagobert, y). Now the planner tries to randomly substitute objects into y, until the precondition "HAS(Dagobert, y) AND HERO(Dagobert) AND EDIBLE(y)" is fulfilled.

3. Collect these possible actions into a set. Then select the action that is closest to the target state, determined by the number of missing statements, and the distance between the Story Variables. Then the state created by this action is going to be the next state. If no such an action is found, make one step back with a probability, or repeat.

The planner was influenced by PDDL, with a few extensions. Properties of SimplePlanner:
-based on classical planning
-supports negated conditions
-doesn't support equality operators, but each substituted object has to be different in an Action, for example I can't give a gift for myself
-a cost is associated with each Action, but these are only for comparing results, it's not possible to have these costs in a precondition or a state.
-not optimal: not guaranteed to find the best solution. With a sufficiently large number of threads it usually finds the best solution or a solution close to it.

By definition, the planner is *sound*, because whenever it gives solutions, they are always correct solutions, but not *optimal* and not *complete* (?).

Evaluation: unfortunately, SimplePlanner version 2 didn't perform very well. It could solve simple problems very quickly. But then I performed some advanced tests, like path finding and the Sussman Anomaly problem and the planner couldn't solve it. This led me to change a few things in my planner.

SimplePlanner version 2.5

First of all, the possible next actions are stored at the State class in a deque. Each time a thread wants to acquire an action, it takes out from the head of the deque, and it puts it at the back of the list.

Secondly, the threads now can perform extensive search, which means they try all the possible actions with all the objects.

Because the program performs extensive search as well, I decided to make a limitation on the domain. Actions can have 3 parameters maximum and the world is limited to 20 objects altogether and 30 actions. This means each state has to search through 30 * 20 * 19 * 18 = 205200 possibilities.

A possible improvement over the search for actions could be is to search through all the possibilities of the planning and then at each state only update the actions that have predicates changed. This requires creating links between the actions and predicates.

5.5. Software Patterns and Best Practices

To have a reusable and stable implementation it is essential to stick to best practices guidelines and software patterns. I'm personally against the term *Software Pattern* because they are merely a subset of best practices, but because of its definition it omits useful practices, such as *immutability* or *pooling*, which are essential in Object Oriented programming. Therefore I decided to collect here not only the Software Patterns used in my project, but all the other important best practices as well.

-*immutability*: I made as many classes immutable as I could, because it makes pooling and thread-safety possible.

-*pooling*: all the ObjectInst, Predicate, PredicateInst classes, that satisfy the equality operation, point to the exact same class, similarly to the class String, providing efficient memory use. This might have been a bad decision, because pooling is not recommended for modern languages with a Garbage Collector.

-*Null object pattern*: whenever possible I initialized all the classes in the constructor, and the methods whenever possible return empty arrays or empty Strings instead of nulls.

-*MVC pattern*: all the user interaction and the GUI are found in the wizard package and the model is found in the core package. The user interacts with the dialogue (controller), which modifies the story world (model), which is then used by the planner domain, and then the results are displayed in a text box (view).

-*Object encapsulation*: I always preferred Lists to Arrays as recommended in [37]. Also, the ObjectInst class has only one field, a String.

-*Observer pattern*: the PlannerExecutor class relies heavily on this pattern

-*Dependency injection*: whenever possible, all the dependent classes are passed in the constructor

-*Strategy pattern*: it is possible to easily change the Planner included in my project to other planner implementations, they just have to implement the Planner interface, which is an extension of the Runnable interface, or it's possible to write a new PlannerExecutor.

-*Builder pattern*: Action classes are created using the ActionBuilder class

-*Classes instead of tuples*: the using of the generic classes Pair and Triple are often discouraged in Java. I used them several times, but refactored the code after realising it makes the code hard to read and less safe.

I listed only the most important best practices, many others are detailed in [37], such as using enumerations, using the @Override annotation, use typed classes instead of raw classes, etc.

5.6. Other Tasks

Testing:

I ran the different methods several times with different arguments to ensure they work. However, having a Parser that works 100% reliable is crucial to the project. Therefore I used automated testing with JUnit, to make sure the Parser does indeed work. I created test cases for each of the methods in the Parser class.

Documentation

JavaDoc is the standard way to document methods and classes. I also used annotations. I was aiming to document approximately every second or third line.

File format

I decided to come up with my own file format for storing the planning and the problem domain because I thought PDDL was hard to read. I didn't start with formal description of the file format, so I could play around with it.

```
Predicates: HAS 2, CHARACTER 1, HERO 1, VILLAIN 1, HERO 1, FREE 1, TREASURE 1;
Objects: Scrooge, Glomgold, Gold;
Start: HAS(Scrooge, Gold), CHARACTER(Scrooge), CHARACTER(Glomgold);
Finish: HAS(Glomgold, Gold);
Actions:

Steal(x, y, z)
@precondition: CHARACTER(x) AND CHARACTER(y) AND HAS(y, z)
@add: HAS(x, z)
@remove: HAS(y, z)
@change: TIME + 2
@output: $x steals from $y the $z
@end
```

A short summary of the file:

-the story world is composed of seven predicates, one of them taking two arguments

-there are 3 objects

-the starting state involves two predicates, the target state one predicate

-there is one action steal, which takes 3 arguments. The precondition is a logical expression composed of two AND statements and three positive predicates. It adds the predicate

HAS(x, z) and removes the predicate HAS(y, z). It increments the global variable TIME by 2. It outputs the text “$x steals from $y the $z” where each variable is substituted.

The BNF description of the file format:

```
grammar Plan;
prog: (objects)+ (predicates)+ (start)+ (finish)+ (actions)+ ;
objects: 'Objects: ' object (SEPARATOR object)* ';' ;
predicates: 'Predicates: ' predicate (SEPARATOR predicate)* ';' ;
start: 'Start: ' predinst (SEPARATOR predinst)* ';' ;
finish: 'Finish: ' predinst (SEPARATOR predinst)* ';' ;
actions: actionname (precondition)+ (add)+ (remove)+ ;

precondition: '-precondition: ' tree ;
add:  predinst (SEPARATOR predinst)* ;
remove: predinst (SEPARATOR predinst)* ;
tree: (expr)+ ;
expr: ('NOT' exprpi) | (exprpi binary exprpi) | predinst | ( '(' expr ')' ) ;
exprpi: expr | predinst ;

binary: 'AND' | 'OR' | 'XOR';
object: STRING ;
predicate: STRING INT ;
predinst: STRING '(' STRING (SEPARATOR STRING)* ')' ;

CHAR: [a-zA-Z] ;
STRING: [a-zA-Z]+ ;
INT: [0-9]+ ;
SEPARATOR: [\r\n]+ | ',' ;

//WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

For examples, see the section “Evaluation”.

6. Evaluation

6.1. Introduction and Problem Description

Evaluating and comparing interactive story creation programs is not easy, because there isn't standard or framework to compare them. On top of that interactive story systems have different goals: some create text, some others work in a game, and some others allow the user to interact them inside the story. The scenarios are very different: there are physics world like ACME or The Final Destination, there are multi agent planners like Madame Bovary, and single agent planner like Wider Ruled or that Aladdin thing

It's also difficult to judge compare the output of different interactive stories, because comparing stories always involves humans and cannot be done automatically.

Evaluating is therefore hard.

Test environment: Windows 7 operating system, iMac (21.5-inch, Mid 2010), 3.2 GHz Intel Core i3, 4 GB 1333 MHz DDR3, ATI Radeon HD 5670 512 MB.

6.2. Planner Tests

Before heading to the main scenario and generate real stories, I had to make sure the planner I'm using could solve large-scale problems. To do this, first it has to solve simple tasks, and if the planner is able to solve these, could solve the large-scale tasks.

I identified 3 different scenarios; each one measures a different attribute of the planner. The first task measure whether it is able to solve a simple task efficiently in a small amount of time. The second task measures whether it's able to forward track. The third task measures whether it's able to do error correction and back-stepping.

I think it's crucial but also satisfying for a planner to solve these 3 tasks within a short amount of time.

To measure time, I hard-coded a part in the Planner Executor, which measures the time between the start of the threads and the end of the last thread.

1. Testing for simple task: the goal of this scenario is to solve a very simple problem. It involves a few actors, and a few actions, it's about Scrooge stealing a cake from Glomgold and eating it.

The planner solved it fairly quickly.

2. Testing for path finding: this test involved a walk in a connected graph. In this case there were 9 environments defined (such as Prison, Jungle, Forest) and these were either connected (CONNECTED(Courtyard, Village)) or neighbours NEIGHBOUR(Home, Courtyard). Scrooge was located at Home and he had to get to the Dungeon.

Thread	Optimal	Time
1	0	0.18
4	3	0.28
16	2	0.36
50	5	0.53

Table 4. Results of the Path Finding problem

3. Testing for Sussman anomaly: this test was about to find out, whether the planner is able to solve the problem detailed in Sussman anomaly, and if yes, with what efficiency.

First the problem had to be described in the language my system was using. I took a look at several descriptions of the problem, like in [34], then I defined my own domain, I made it as short as possible.

Thread	Optimal	Time
1	1	0.032
4	1	0.047
16	4	0.114
50	5	0.25

Table 5. Results of the Sussman Anomaly problem

I include problem description here, it is also a good way to demonstrate the structure of the language:

```
Predicates: ONTABLE 1, FREE 1, ONTOP 2;
Objects: BoxA, BoxB, BoxC, Table;
Start: ONTOP(BoxC, BoxA), FREE(BoxB), FREE(BoxC), ONTABLE(BoxA), ONTABLE(BoxB);
Finish: ONTOP(BoxA, BoxB), ONTOP(BoxB, BoxC);

Actions:

MoveToTop(top, bottom)
@precondition: ONTABLE(top) AND FREE(bottom) AND FREE(top)
@add: ONTOP(top, bottom)
@remove: FREE(bottom), ONTABLE(top)
@change: TIME + 1
@output: put $top on top of $bottom
@end

MoveFromTop(top, bottom)
@precondition: FREE(top) AND ONTOP(top, bottom)
@add: FREE(bottom), ONTABLE(top), FREE(top)
@remove: ONTOP(top, bottom)
```

```
@change: TIME + 1
@output: remove $top from $bottom
@end
;
```

The optimal solution for the problem:

-remove BoxC from BoxA
-put BoxB on top of BoxC
-put BoxA on top of BoxB

Running the applictaion with a suffieciently high number of threads gives the optimal solution, for lower number of threads it gives highly unoptimal solutions.

The problem is probably there is a huge search space and achieving a goal involves a to look up a lot of steps ahead, something which is a weakness of my planner.

The other task related to testing is the fine tuning of planner variables. For example, there is a variable that decides the percentage of stepping back, or there maximum number of runs for a thread, that decides when it should give up on planning. I experimented a lot with finding the optimal variables, but I couldn't find a pattern in it at the moment.

6.3. Full Evaluation

Third scenario: this is the first full test of the application. It has a minimal test of characters and actions.

Unfortunately, the search space is huge and actions could take a lot of parameters, therefore the planning takes longer than it should be. It would have been a better idea, to use types, instead of representing types as predicates, because it would effectively cut down the planning time.

I came up with a minimal set of characters, environments, predicates and actions to test the system. The system should be able to generate a sound story.

The aim was to create two stories for each arc, and two stories with no arc.

Representing the environments and placing the characters in an environment turned out to be a challenging task. Each time a character does any kind of action it has to check whether the other character or object he is interacting with is present in the same environment. This lowers the performance of planning by a huge margin.

The first setting contained environments, and then I decided to remove it to speed up the planning. The second setting didn't contain any environments. Then I decided to reintroduce them, by the help of Story Parts: the story parts are now responsible for the

travel between the environments, for example the Introduction part makes the heroes move to the location of the dungeon.

To overcome the problem of checking environments at each action the planning domain has to be STRIPS and has to use the existential quantifier, as it was used in Ugh's story:

(exists (?p - place) (and (at ?speaker ?p) (at ?listener ?p))))

This means, the listener has to be at the same place as the speaker.

Having types in the planning domain would have been useful too. That is because it would cut down the search space for possible actions. Types could be interpreted as static predicates that cannot be changed.

The output results were slightly below my expectations.

When designing the problem domain and the StoryParts I had to be careful not to have any unreachable situations. It might be possible that at some earlier point of the story all the heroes got captured as a hostage, thus fulfilling the predicate HAS(Scrooge, Gold) now becomes impossible. Implementing a full error feedback would be very hard at this point, so the planner executor just skips at situations like this.

Obtain Arc

```
1. Setting:
-Hero: Scrooge
-Villain: Glomgold
-Arc: Obtain Gold
-Story Parts: 8 (All included)

Act 1

Launchpad goes from Home to Dungeon
Nephews goes from Home to Dungeon
Scrooge goes from Home to Dungeon

Act 2

Scrooge wanders around and suddenly finds an Map
BeagleBoys goes from Hideout to Dungeon
Glomgold goes from Hideout to Dungeon

Act 3

Glomgold lures Scrooge into a trap

Act 4

Launchpad has a cunning plan and frees Scrooge

Act 5
```

```
Scrooge learns the location of Gold from Map
Scrooge wanders around and suddenly finds an Gold

Act 6

Glomgold lures Scrooge into a trap
Glomgold steals from Scrooge the Gold

Act 7

Nephews has a cunning plan and frees Scrooge
Glomgold accidentaly loses Gold
Scrooge wanders around and suddenly finds an Gold

Act 8

Launchpad fights Glomgold and Launchpad emerges victorious
Launchpad fights BeagleBoys and Launchpad emerges victorious

The End
```

Lose Arc

-Hero: Launchpad
-Villain: BeagleBoys
-Arc: Lose NumberOneDime
-Story Parts: 6 (“Attack” and “New Plan” are excluded)

```
Act 1

Glomgold goes from Hideout to Home
BeagleBoys goes from Hideout to Home

Act 2

Nephews meets BeagleBoys
Glomgold meets Nephews
Glomgold sees tracks and identifies Launchpad
BeagleBoys meets Launchpad

Act 3

BeagleBoys fights Nephews and BeagleBoys emerges victorious

Act 4

Stranger goes from Dungeon to Home
Scrooge accidentaly loses NumberOneDime
Scrooge sees tracks and identifies BeagleBoys
BeagleBoys wanders around and suddenly finds an NumberOneDime

Act 5
```

```
Stranger confesses that he was promised a lot of money to act against our heroes
BeagleBoys accidentaly loses NumberOneDime
Launchpad wanders around and suddenly finds an NumberOneDime

Act 6

Launchpad fights Glomgold and Launchpad emerges victorious
Launchpad has a cunning plan and frees Nephews
Launchpad fights BeagleBoys and Launchpad emerges victorious

The End
```

6.4. Results

I monitored the CPU and memory usage during the evaluation. The CPU threads were overloaded. On the other hand the memory usage is quite low, around 25-150MB, most of the times it was around 75MB.

The output results were slightly below my expectations. I compared several results and found it a bit too repetitive. Actions with many parameters are rarely used and slow down the search process.

The problem description was changed several times. First time each story part included many predicates, some of them involving the location of characters, some involving holding objects or being in states. This produced poor results, it seemed like the planner couldn't handle too much traveling. Then I decided to exclude locations completely from the problem domain: this time it produced results, but the problem was that the story wasn't quite good. Then I decided to reintroduce locations, but each story part could have only a few predicates, and not involve a mixture of locations and character states. This was the final setting.

I also had to exclude actions with too many parameters, because it took the planner much more time to search and didn't produce good results. Ideally, an action should have 2 parameters, but not more than four. Because most of the actions involve a location, this limits the description of complex actions, thus the quality of the story.

I identified two problems: the planner is not optimal and can't solve complex problems and the domain description is weak.

7. Conclusions

My experiments showed that it is possible to integrate the art of script writing into computer generated stories . The emphasis is on story structure.

My ideas could be used for generating better stories in games, it could have a viable market in the MMORPG genre, especially considering the recent success of Guild Wars 2.

There is much space left to improve on the quality. One of them is to use a better planner. Unfortunately, because the great fragmentation amongst planners, it's not easy to find an optimal planner for this domain, especially considering many of the links I found were broken.

Even with a planner the options are limited, because the planning problem is NP-complete. Therefore a lot of research has to be made into the description of the domain model.

One of the wrong assumptions was made is that the planning language should be as simple as possible, as close to STRIPS as possible. It turned out; that it's the other way around, using more features of PDDL could make the system more efficient. It would be worth investing in research of preference-based HTN planners and preference-based predicate planners.

There is much research left on implementing the artistic side of the movie script generation, but because there are no strict rules on movie script writing as it was seen, it is a difficult task to come up with a general solution.

The major problem of this area is that there aren't any tasks or frameworks which could be used for comparing different solutions and it's hard to evaluate the results of a project.

8. References

[1] Leandro Motta Barros and Soraia Raupp Musse, *Planning Algorithms for Interactive Storytelling*. Universidade do Vale do Rio dos Sinos, Brazil, 2007.

[2] John Truby, *The Anatomy of Story: 22 Steps to Becoming a Master Storyteller*.: Faber & Faber, 2008.

[3] Blake Snyder, *Save the Cat!*: Michael Wiese Productions, 2005.

[4] Kristin Thompson, *Storytelling in the New Hollywood: Understanding Classical Narrative Technique*.: Harvard Univ Pr, 1999.

[5] Viki King, *How to Write Movie in 21 Days: The Inner Movie Method*.: William Morrow Paperbacks, 1993.

[6] Vinay Chaudhri. Knowledge Representation and Reasoning. [Online]. http://web.stanford.edu/class/cs227/Lectures/lec16.pdf

[7] Mark O. Riedl. and Vadim Bulitko, *Interactive Narrative: An Intelligent Systems Approach*., 2013.

[8] Manish Mehta, Steven Dow, Michael Mateas, and Blair MacIntyre, *Evaluating a Conversation-Centered Interactive Drama*., 2007.

[9] Michael Mateas and Andrew Stern, *A Behavior Language for Story-based Believable Agents*., 2002.

[10] David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen, *Interactive Storytelling: A Player Modelling Approach*., 2005.

[11] Boyang Li, Stephen Lee-Urban, George Johnston, and Mark O. Riedl, *Story Generation with Crowdsourced Plot Graphs*., 2005.

[12] Jean-Luc Lugrin and Marc Cavazza, *AI-based World Behaviour for Emergent Narratives*., 2006.

[13] Mark O. Riedl and Andrew Stern, *Believable Agents and Intelligent Story Adaptation for Interactive Storytelling*., 2006.

[14] David Pizzi and Marc Cavazza, *Affective Storytelling based on Characters' Feelings*., 2007.

[15] David Pizzi, Marc Cavazza, and Jean-Luc Lugrin, *Extending Character-based Storytelling with Awareness and Feelings*., 2007.

[16] David Pizzi, Fred Charles, Jean-Luc Lugrin, and Marc Cavazza, *Interactive Storytelling with*

Literary Feelings., 2007.

[17] Marc Cavazza, Fred Charles, and Steven J. Mead, *Emergent Situations in Interactive Storytelling*., 2002.

[18] David Olsen and Michael Mates, *Beep! Beep! Boom!: Towards a Planning Model of Coyote and Road Runner Cartoons*., 2009.

[19] James Skorupski and Michael Mateas, *Interactive Story Generation for Writers: Lessons Learned from the Wide Ruled Authoring Tool*., 2009.

[20] James Skorupski. (2013) Wide Ruled: An Author Goal-Based Interactive Story Generator. [Online]. http://skorupski.org/wiki/_media/wide_ruled/2013-02-wideruled_classpresentation.pdf

[21] Ernest W. Adams, *Resolutions to Some Problems in Interactive Storytelling*., 2013.

[22] Marc Cavazza and David Pizzi, *Narratology for Interactive Storytelling: A Critical Introduction*., 2006.

[23] Wikipedia. [Online]. https://en.wikipedia.org/wiki/Dramatic_structure

[24] Nicolas Szilas, *IDtension: a narrative engine for Interactive Drama*., 2003.

[25] Alan Fern. Classical Strips Planning. [Online]. https://web.engr.oregonstate.edu/~afern/classes/cs533/notes/strips-intro.pdf

[26] Minh Do. Practical Planning. [Online]. http://web.stanford.edu/class/cs227/Lectures/lec17.pdf

[27] Jörg Hoffman and Bernhard Nebel, *The FF Planning System: Fast Plan Generation Through Heuristic Search*., 2001.

[28] Wikipedia. [Online]. https://en.wikipedia.org/wiki/Action_description_language

[29] Malik Ghallab et al. PDDL - The Planning Domain Definition Language. [Online]. http://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf

[30] Scott Sanner and Kristian Kersting, "ICAPS-08 Tutorial on First-Order Planning Techniques," *18th International Conference on Automated Planning and Scheduling*, 2008.

[31] David Poole and Alan Mackworth. (2010) Event Calculus. [Online]. http://artint.info/html/ArtInt_336.html

[32] Tran Cao Son and Enrico Pontelli, "Planning with Preferences using Logic Programming," 2003.

[33] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith, "HTN Planning with Preferences".

[34] Sussman Anomaly. [Online]. https://en.wikipedia.org/wiki/Sussman_Anomaly

[35] Jussi Rintanen. A brief overview of AI planning. [Online]. http://users.ics.aalto.fi/rintanen/jussi/planning.html

[36] Patrik Haslum, *Narrative Planning: Compilations to Classical Planning*., 2009.

[37] Joshua Bloch, *Effective Java*, 2nd ed.: Addison Wesley, 2008.

[38] Vladímir Propp, *Morphology Of The Folk Tale*, 2nd ed., 1968.

[39] Mark O. Riedl and R. Michael Young, *Narrative Planning: Balancing Plot and Character*., 2010.

[40] Julie Porteous, Marc Cavazza, and Fred Charles, *Narrative Generation through Characters' Point of View*., 2010.

[41] David James Thue, *Player-informed Interactive Storytelling*., 2007.

[42] Michael Mateas and Andrew Stern, *Façade: An Experiment in Building a Fully-Realized Interactive Drama*., 2003.

9. Appendix

A. Abstract

English Version

Player influenced storylines existed for long in the world of computer games. Alternative endings and player choices improved the replayability of games, thus adding to the overall experience. However, all of these plot lines were author-created, as computer-generated stories didn't achieve the attention of the mainstream.

Interactive Storytelling tries to make stories less author-created and investigates the possibilities of creating stories with artificial intelligence. The earliest of this systems was Façade, which is defined by it's authors as an interactive drama, where the player's avatar interacts with two other non-player characters with text input and actions. This is then interpreted by the computer and a story is generated on-the-fly.

Since then many different interactive storytelling systems emerged. The default technical solution for this projects is to use a planner. Planner languages consist of the following: an initial state, a goal state and a list of possible transitions. A planner then finds a possible state transition from the initial to the goal state. Some planner languages contain a measure to rank the different solutions.

The goals of this thesis is to investigate different planning techniques and the theory behind writing stories as well as to produce a program that creates simple stories with using a planner.

First it analyses the existing projects and their technical solutions to the planning problem. The thesis also analyses the process of creating stories and how drama-theory could improve the results of computer-generated stories. Hence, a lot of research is done in modern narratology and movie script writing.

As next a new way is proposed to create an interactive storytelling program, which tries to mimic the process learned from drama theory. The planning domain and language is self-made and as simple as possible.

The project is evaluated in two steps: first the planner is evaluated and compared to other planners. Then several DuckTales stories will be created by the planner and these will be compared to the plot of DuckTales episodes.

Deutsche Version

Durch dem Spieler beeinflussten Geschichten existieren schon lange in der Welt der Computerspiele. Alternative Endungen und Spielerentscheidungen entwickelten die Wiederspielbarkeit der Spiele, wodurch die allgemeine Erfahrungswert ebenso gesteigert wurde. Jedoch waren all diese Handlungslinien vom Autor kreiert, da die vom Computer generierten Geschichten die Aufmerksamkeit der Mainstream nicht erreichen konnten.

Interaktive Storytelling versucht die Geschichten weniger vom Autor geschaffen zu sein und untersucht die Möglichkeiten der Schaffung von Geschichten durch künstlicher Intelligenz. Die älteste dieser Systeme war Façade, welche von den Autoren als interaktives Drama definiert wurde, in dem der Spieler´s Avatar mit zwei anderen Nicht-Spieler Charakters mit Texteingabe und Aktionen kommuniziert. Dieser wird dann durch den Computer interpretiert und die Geschichte wird zeitgleich („on-the-fly“) erzeugt.

Seitdem sind viele verschiedene iteraktive Storytelling-Systeme bekannt geworden. Der standard technische Lösung für diese Projekte ist einen sogennanten Planer zu verwenden. Planerpsprachen bestehen aus Folgendes: eine Anfangsphase, eine Zielzuphase und eine Liste der möglichen Übergänge. So findet dann ein Planer einen möglichen Phasenübergang von der Anfangsphase auf die Zielphase. Einige Planersprachen enthalten eine Maßnahme, um die verschiedenen Lösungen zu rankieren.

Die Ziele dieser Arbeit ist es verschiedene Planungstechniken und die Theorie hinter der schriftlichen Geschichten zu untersuchen sowie ein Programm zu erschaffen, welches einfache Geschichten mit einem Planer erstellt.

Zuerst analysiert sie die bestehenden Projekte und deren technische Lösungen iZm dem Problem der Planung. Diese Diplomarbeit analysiert ebenso den Prozess von Geschichtenschaffung und wie die Dramatheorie die Ergebnisse der computergenerierten Geschichten verbessern könnte. Folglich wurde viel Recherchearbeit in der modernen Narratologie und im Filmdrehbuchschreiben investiert.

Als Nächstes wird eine neue Methode vorgeschlagen, um ein interaktives Storytelling-Programm zu erstellen, welches den von der Dramatheorie gelernten Prozess imitiert. Der Planungsdomäne und Planungssprache ist selbst generiert und so einfach wie möglich gehalten.

Das Projekt wird in zwei Schritten evauliert: Zuerst wird der Planer ausgewertet und mit anderen Planern verglichen. Darauf folgend werden einige DuckTales Geschichten vom Planer erstellt und diese mit den Handlungen von DuckTales Episoden verglichen.

B. DuckTales Story Analysis

28. Sweet Duck Of Youth

Premise: Scrooge starts feeling old and decides to look for the Fountain of Youth

Heroes: Scrooge, Nephews, Launchpad
Opponent: Mysterious Conquistador
Key Object: Fountain of Youth
Place: Rainforest

Set-up/Motives
-Scrooge is getting old so he decides to find the Fountain of Youth

Middle
-Launchpad's helicopter gets shot and they lose Scrooge
-They start looking for Scrooge
-Scrooge gets trapped by a mysterious stranger with an axe
-Launchpad gets trapped too
-The nephews follow the mysterious stranger
-They escape Scrooge and Launchpad and have a talk with the stranger
-They find a hidden map inside the old stranger's armour

Turning Point
-They find the Fountain but it turns out to be a scam

Resolution
-Scrooge although didn't get any younger, he feels young again

Alternative description:

Introduction
-Scrooge starts to feel old and he finds about the Fountain of Youth

Plan
-Scrooge goes in a trip with Launchpad and the Nephews to find the fountain

Failure
-They lose each other, Launchpad and Scrooge gets trapped

New Plan (Second Plan)
-The Nephews decide follow the mysterious stranger

Battle
-They free Scrooge and Launchpad and trap the mysterious stranger

Revelation

-The mysterious monster (fake-opponent ally) is an adventurer looking (unsuccessfully) for the fountain as well

Act of God / Luck
-They find a hidden map inside the old stranger's armour

New Plan (Third Plan)
-They follow the instructions on the map

Turning Point / Revelation
-They find the fountain but it turns out to be a scam

Moral turning point
-Scrooge didn't get any younger but he feels younger which comforts him

Resolution
-They go home happily and decide to sell the water of the fountain

35. Scrooge's pet

Premise: Our heroes chase after a lemming who stole the code for the safe
Duration: 22:51 minutes

Heroes: Scrooge, Nephews, Launchpad
Opponent: Lulu (lemming)
Key Object: Safe code
Place: Scandinavia

Introduction
1:00- 3:30 - The nephews and Webby go fishing but without Scrooge because he's always busy

Turning Point / Intrigue / Desire
3:30 - 4:00 - They see a lot of pets and Webby comes up with the idea that leads to the plan

Plan
3:30 - 6:00 - They get a lemming called Lulu for Scrooge

Plan Fails
7:00 - Scrooge is not impressed with Lulu and starts chasing him
7:30 - Lulu escapes with the combination for the safe

New Plan
8:00 - They start chasing after Lulu

New Plan 2

9:30 - Lulu escapes on the ship and they go after him with a helicopter
11:00 - They arrive to their destination

Battle
11:00 - 14:00 - They are chasing after Lulu but he escapes

Turning Point / Attack
15:00 - Lemmings invade the village
17:00 - They have to find Lulu now otherwise he will disappear forever

Final Battle
18:00 - Launchpad tries to stop the Lemmings
19:00 - Another unsuccessful attempt

Apparent Defeat
20:00 - The Lemmings left the land

Act of God / Luck
21:10 - They find Lulu inside a cheese

Resolution
21:45 - Scrooge receives a new pet, a golden fish

C. About the Author

Name: Attila Torda
Website: www.attilatorda.eu
Born: 1989 - Székesfehérvár, Hungary
Languages: English, Hungarian, German, Japanese

A computer enthusiast started programming games at the age of 10. Main interests are Computer Graphics, Image Processing, and AI. Favourite programming languages are Java and C++.

Education

2012 – 2015 MSc. Medieninformatik, Universität Wien
2011 – 2012 MSc. Software Engineering and Internet Computing, Technische Universität Wien
2007 – 2011 BSc. Computer Science and Mathematics, The University of Manchester
2007 – Hungarian Matura (Érettségi)

Hobbies

Snowboarding, judo, kickboxing, playing the guitar, producing music