Diage: A Dialogue Generator

Nico Andrew Glas

19th March 2014

Abstract

Game development is resource heavy and time consuming. By using *procedural content generation*(PCG), game developers can save time perfecting and polishing the more critical parts of a game. A recent trend within game development is to use PCG to create worlds, levels, and quests. And as such we miss a more esoteric application for the techniques such as creating music or, indeed, generating narratives.

This document chronicles the development of a roguelike game and explores the potential of a procedurally generated narrative from a development perspective. I lay out the details of the narrative planner used to control the narrative and examine a modelling language that can be used in prototyping narrative scenarios.

Acknowledgements

I would first and foremost like to thank Joris Dormans for granting me this position within the research facility to let me work out any crazy idea I had in my head (and adding a few of his own, mind). Your work in the gaming field is an inspiration to us all. Thanks, to Stefan Leijnen, for your excellent guidance and providing me with the needed mental support. Without you and Joris, I shudder to think how this project would have gone.

My thanks, and most sincere apologies for all those late nights, to Anne-Marieke Zaal, for being the reason I stopped work and went home.

Additional thanks go to Gerjo Meier, for correcting all those stupid grammar errors and for all those silly discussion we have.

And of course, thank you, the reader, for taking your time and reading this document.

Contents

1 Introduction 9
1.1 Create-IT 9
1.2 Technology 9
1.3 Research Phases 10
1.4 Problem definition 10
1.5 Proof of concept 10
1.5.1 Rouge 11
1.6 Requirements and Constraints 11

2 The interactive story 13
2.1 Video Games and Interactive Stories 13
2.2 IS 14
2.3 Dynamic plot generation 14
2.4 Character-based storytelling 14
2.5 Generative narrative within games 16
2.6 Conclusions 16

3 Procedural Content Generation 17
3.1 Rouge 18
3.1.1 Mechanics 18
3.2 World generation 19
3.3 Cellular Automation 20
3.4 Evaluation 22

4 Narrative planning 25
4.1 Narrative planning in Rouge 25
4.2 Actor attributes 26
4.3 Initial Generation 26
4.4 Step Generation 29
4.5 Events 30
4.6 Planning flow 30
4.7 Evaluation 31

5 Diage Modelling Language 33
5.1 The structure of DML 33
5.1.1 Objects 33

5.1.2 Actors 34
5.2 Spaces 35
5.3 Actions and Accessors 35
5.4 The narration 35
5.5 DML in Rouge 38
5.6 Future work 38

Chapter 1

Introduction

During my coursework for my Bachelors degree in Information Technology, I stumbled upon the field of procedural content generation (PCG). I dedicated last two years of school to understanding and applying this field, and it silently turned into one of my 'specialities'. It all started with a research project that focussed on generating game worlds with Voronoi diagrams. After that I made a couple of games that used content generation for various things; from using audio to generate platforms for an *endless runner* type game, to cellular automata to create dungeons for a *dungeon crawler*. The one thing I always had an interest in, is video game narrative. My thoughts turned towards the idea of using procedural content generation techniques to create a narrative for video games. Together with Joris Dormans of the Create-IT research facility we set up a project to research the use of PCG in video game storytelling. This thesis is presented as part of my fulfilment for graduation and serves as a research report.

1.1 Create-IT

Create-IT applied research is one of the research institutes hosted by the University of Applied Sciences of Amsterdam. In this lab students, teachers and researchers all perform applied studies in the different sections of the IT world. Their goal is to educate future professionals in the uses of applied research, so these professionals can anticipate the ever changing field that is Information Technology. In the newly created Game Research Lab students and researchers alike contribute to the growing community of game developers; making the development of games easier or trying to understand current problems within the industry.

1.2 Technology

Rouge is built in C#.Net on the XNA framework, because of my familiarity within these techniques. I believe that my expertise within these areas made it easier and faster to develop the product in this setting than in any other. All the graphing and prototyping work work as explained in chapter 5 is done with the graphing software

Gliffy. The DML Creator is created with C#.Net and the Windows Presentation Foundation subsystem.

1.3 Research Phases

This project lasts for 20 weeks, and the following will indicate the phases of my research process.

- **Week 1-6** Literature study
- **Week 7-13** Creating a generative algorithm
- **Week 14-20** Proof of concept.

1.4 Problem definition

My main research question is *How does the development process of a roguelike game benefit from using a generative narrative?*. In this section I will state my research questions, and try to dissect them so all readers of this document have the same definitions and context.

Let me clarify the term roguelike. The genre started with the video game **Rogue** that was released in 1980, and was characterised by having "random" dungeons where the player has to navigate rooms and fight monsters. The ultimate goal of the game was to get to the highest level possible without dying once. The game never really "ended". The game was over when the player died, but after that the player got to start all over again on level 1 with a complete newly generated dungeon. As the game gets progressively harder when the player starts go get to other levels, the chances for the player to lose get higher. Now, back to the term "roguelike "; A game with no definitive end and permanent loss of game progress when the player dies. The previous years has seen a rise in popular roguelike games, **FTL: Faster Than Light** by *Subset Games* (2012) and **The Binding of Isaac** by *Edmund McMillen and Floris Himsl* (2011) being just some examples.

I specifically target *roguelikes* for their inherent use of content generation. The need to have a different set of content throughout a play-session is the key selling point of a roguelike game. This makes it the right genre to experiment in with any new type of content generation.

1.5 Proof of concept

As a proof of concept I propose to build a game that incorporates the findings of this research. The game I will make will be a roguelike for their inherent use of PCG techniques. Due to the limited duration and scope of this project, the game will be restrained to the most basic elements of a roguelike, the only addition being a dynamically generated narrative. This game should pose as the proof of my research and be demonstrable to verify my answers to my research questions.

1.5.1 Rouge

Rouge is a roguelike tile-based game that is created specifically for the demonstration of the Rouge narrative generation system. Created within my own framework *SilicaLib* created on top of the XNA game-framework created by Microsoft. Rouge is characterised by the fact that the world and the narrative is generated by the direct influence of the player.

1.6 Requirements and Constraints

The project has several requirements and constraints, which are:

- *Only* roguelike: All research done into generating content is targeted at roguelike games. This is done because said genre is small and already relies on procedural content.
- *Windows only*: I will develop only on *Microsoft Windows* for the duration of this project, thus limiting the resulting products to be available only on Windows.
- *Working demo*: The project must result in a working demo that shows the potential of a procedurally generated narrative.

Chapter 2

The interactive story

Most of the research done on narrative generation is contained within the field of Interactive Storytelling (IS). The average observer may not see the difference between a game with a story and a interactive story, and can be forgiven for this is still in open debate within their respective fields. However open the discussion may be there are, albeit subtle, differences. This chapter covers the work done in this field that relates to my own research, and discusses the contrast between the conception of interactive storytelling and video games.

2.1 Video Games and Interactive Stories

The notion that an interactive story is a distinctly different product than video games has been in open discussion ever since video games started having complex stories themselves. In the early years of video gaming (games like **Pong**, **Pac-Man** and **Tetris**) games usually didn't have a story[1], but then games like **The Legend of Zelda** by *Nintendo R&D4* became popular that did have a, albeit small, story. At this point in time the distinction between, and especially the semantics, became open for debate.

In this day and age contemporary games often don't get shipped without an attempt at a story, with the exception being *social* games like **Candy Crush** by *King* and **Bejeweled** by *PopCap Games*, the lines between an interactive narrative and video games become blurry. As matter of fact; the term *video games* has recently been under fire. The questions; what makes something a game? What defines a game? The web show *Extra Credits*[2] has a nice answer to this:

> [...] We're asked this question all the time, but, you know; I think it's the wrong question. It's a distraction. It does nothing but limits us. It's as if we started to ask: "Well is this really poetry?" when poets moved away from rigid meter or rhyming couplets. [...]

The fact is, we are in the middle of a semantic war. My opinion is that in another ten years the industry will have a new term to call these works of interactive

[1]That didn't stop people super imposing stories. See: `http://www.smbc-comics.com/?id=2736`

[2]See `http://extra-credits.net`

engagement. As we stand now games put more emphasis on doing and acting within a set of rules; the game mechanics. Whereas interactive stories revolve around the activity of acting out a story without the constriction of mechanics, usually because the only controls a user has are movement controls supplemented with a contextual button to interact with the environment.

2.2 IS

The early attempts to understand interactive storytelling came in the form of **Tale-Spin**[7]. Tale-Spin generated textual stories from data that a user created like; scenery, characters, and the problems that needed to be solved. Other work includes the *Oz Projects*[5] that used intelligent agent technology to tackle the challenges in interactive storytelling. In 2006 came the award winning **Faade** by *Michael Mateas and Andrew Stern*[6]. This interactive story focusses on the player who is a close friend of two AI characters. During a cocktail party at the AI characters' home the player can support the characters or try to drive a wedge between them. We can't mention interactive storytelling without mentioning former game designer Chris Crawford, who left the world of video games to work on IS. He wrote *Chris Crawford on Interactive Storytelling*[2] which is a deconstruction of the entire field and compares that with traditional video games.

2.3 Dynamic plot generation

One of the most cited works when dealing with generating plots for interactive systems is Sgouros' 1999 paper *Dynamic generation, management and resolution of interactive plots*[13]. The system proposed in this document dynamically moves the plot forward with the relations and interactions between actors serving as input. Figure 2.1 shows us the flow of Sgouros' *Plot Manager*; the central piece of his interactive story system. Sgouros shows us how we can abstract concepts like relations and events both causal and temporal and defines a syntax used to describe actions actors can make and goals they might have. His system generates so-called Aristotelian plots, where a conflict between antagonistic forces develops out of the initial situation. The plot will move through a sequence of conflicts and always terminates in a unambiguous solution.

2.4 Character-based storytelling

At the turn of the century we saw a shift from general story generation towards a more character driven approach. Researchers like Riedl and Mark[11][9] and Marc Cavazza[1] wanted to make characters within the story context be more outspoken and distinct and worked towards systems that made this possible. Most of my work has been based on and influenced by Mark Riedl and Michael Young who contributed to the computational planning of a story. With their papers[11][10][12] they tackle a multitude of challenges from story planners that use character intent

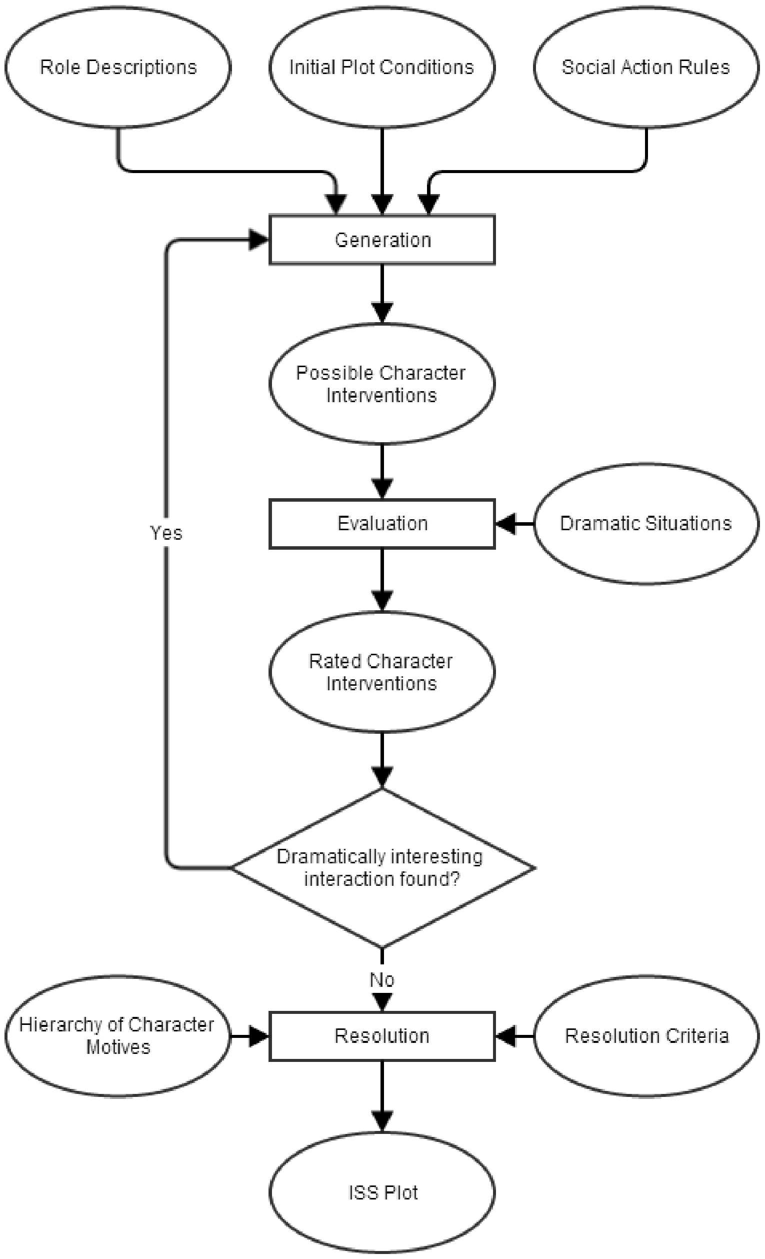


Figure 2.1: Flowchart of Sgouros' *Plot Manager*

to dealing with interactions between users and agents. They co-authored the IS system *Mimesis*[15], that allows storytellers to use a author-centric approach to generating different stories. My own planning system (further discussed in chapter 4) has taken a similar route as the systems proposed by Riedl and Young.

2.5 Generative narrative within games

I stated at the beginning of this chapter that most research on the subject of narrative generation takes place within the field of interactive storytelling, but there are some commercial games that have used a form of generative narration. In **The Elder Scrolls V: Skyrim** by *Bethesda Game Studios* the developers implemented a system they called *Radiant A.I.* that tries to dynamically react to the player's actions. For example, players with a high *Pickpocket* skill would get told to "keep their hands to themselves" by guards. The *Radiant Quest* system expands on this, giving players quests that suit their skills and send them to places previously unvisited. This context is usually used within games; representing a 'fixed' story, but with subtle variance that lets each player have their own experience within a game world.

2.6 Conclusions

The related work done within this field is vast. So vast perhaps, that to dissect it all greatly exceeds the scope wherein this thesis operates. But most of the research has been focussed on generating a narrative that is "fixed" from the beginning with some leeway towards actual context. The usage of a narrative planner is almost universal to ensure any kind of plot coherence, as is a form of author input to control the flow of a story. These basic elements create a interactive story system that can deliver powerful narratives with believable characters. My own concern is the involvement of the player. In most examples the player is merely an actor with a more volatile behaviour, instead of his own character. I strive for a system that really revolves around the player. The story made by the actions that the player did or did not do.

Chapter 3

Procedural Content Generation

In the world of game development we strive to create the best experiences for a wide and diverse audience. During the course of development there are a number of obstacles that can (and often will) hinder the progress of the developed game. Some of these hindrances come from processes that are essential to a game like; level- and world building, or story- and quest design. They take a proportionally large amount of development time and take a very specific mindset to create.

Content is a vague term, as it can be used do describe anything within a game. Ranging from levels and quests, to textures and music, everything that the player can experience within a game is referred to as content. Games such as the **Diablo** by *Blizzard Entertainment* use content generation to rearrange dungeons with every play-through, and **Elder Scrolls IV: Oblivion** by *Bethesda Game Studios* uses their *Radiant Quest* to generate quests suited to the player. The generation itself can take place at run-time starting a new generation during the actual game. In this way the game can take a different form at any given time, making the game at least a little different for ever player on every play-through. Another possibility is to generate content during the development cycle. This is mainly used to save development time, adding the benefit of creating an abundance of content that can be tweaked to create a more author-centric feel to the game. The term *Procedural* refers to the algorithms, and indeed procedures, used to generate the content. These procedures should result in predictable content and range from using cellular automata for generating dungeons, as is the case with Rouge, or using a Voronoi/Delaunay dual-graph to generate realistic worlds[1]. A good example is is the card game **Klondike**, alternatively called **Solitare/Patience**. In this simple card game, shuffling the deck is the procedure used to randomly reorder the cards with the resulting 'content' being the layout of the game.

Procedural content generation can be a viable option to spice up a game or help cut development time for various tasks. But it should be noted that the use of PCG is dependant upon the game that's being produced.

Rouge, as a roguelike, depends heavily on procedural content generation to serve new maps and levels to explore. *Roguelikes* are designed to be played over and over again, teaching the player to master the mechanics of the game, but not

[1]A reference to myself would be fucking pretentious, wouldn't it?

to memorize the worlds in which they play. In the spirit of research we ask the question: "Why does Rouge need procedurally generated content?"

3.1 Rouge

You are a nameless traveller that has been compelled to search for something. An object, a person, or just a place. You don't know yet, but it's up to you to find out. You will travel through various caves, forests and towns to pursue the whims of your heart. Every time you seem to solve a problem, another one turns up. Always moving you towards some inevitable doom. Are you born for heroism, or will you die unloved and unknown?

Choose your path, and see where the voices in your head take you.

Rouge is a tile-based roguelike game where the player travels through the world to discover a myriad of people and monsters that will react to the player in different ways depending on the players attributes. The setting is that the main character is a neurotic, or maybe even a little insane, adventurer. Voices in his head give him instructions (by way of the narrative planner) to do things. This can be finding an item, talking to a person, or beating up some monsters. In Rouge you play out the life of this character, and after he dies another person with the same affliction will become available for play.

3.1.1 Mechanics

Rouge is played in turns, with each action ending that character's turn. The main exception is movement; a character can move as many tiles as their speed attribute allows. An example would be that the player always starts with a speed of two, and thus can walk two tiles during his/her respective turn. Monsters in the starting area will have a speed of one, ensuring that the player will always move faster. Speaking to NPCs, fighting monsters and picking up items all cost a turn.

The items a player picks up can either be equipped or used, with a variety of bonuses and penalties. As seen in figure 3.1 the player is wearing some armour and is holding a combat-knife and shield. Attacking monsters would increase the player's attributes in the respective categories of the equipped items. In this instance, the armour and helm grant the player bonuses on defence, but penalties on speed, every time he/she equips heavy armour. The combat-knife will grant extra strength on small weapons, and so forth. Finding the right items contribute to making a character that suits the player's play style.

Getting a high score on a particular attribute will make NPCs behave differently around you. An example would be a hunter that is severely impressed with your archery skill, and give you his prized bow to use. But figure 3.2 shows us an NPC that is annoyed at the player for bringing an equipped weapon in to his house, and he would refuse giving the player information or items while in this state.

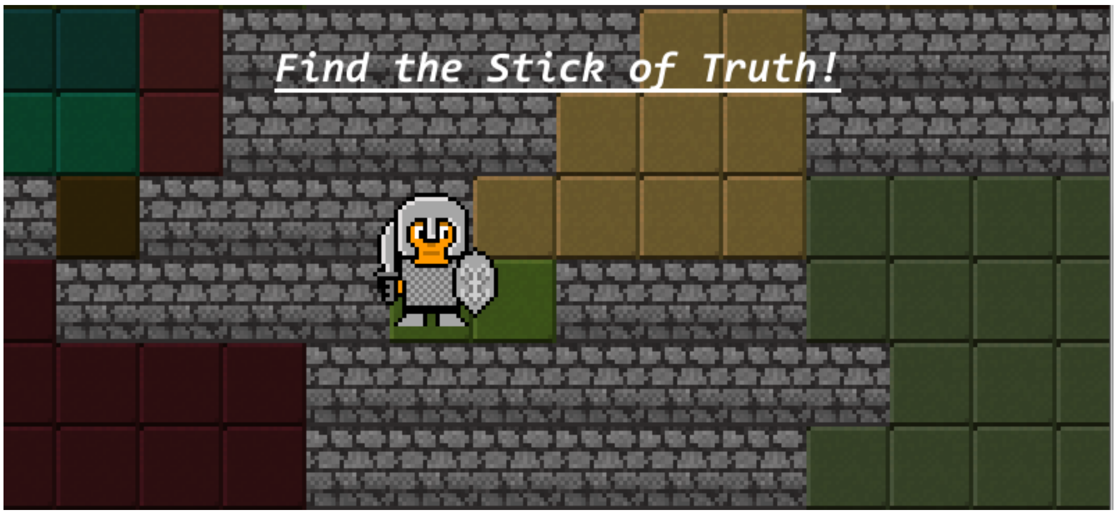


Figure 3.1: The narrator telling the player what to do.

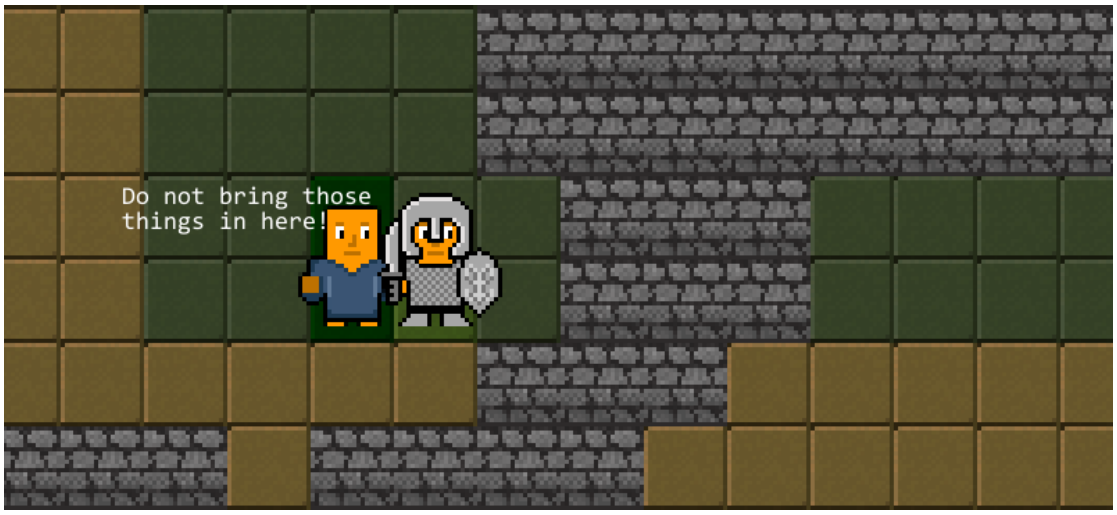


Figure 3.2: A NPC that is angry at the player for carrying a weapon in his home

3.2 World generation

Rouge can be played two ways; either with a persistent world or a generative one. In this chapter we will focus on the generative worlds and the persistent world will be covered in chapter 4.

Creating worlds in Rouge could be done a multitude of ways; either manually, dynamically generated, or generated for static purposes. Static generation means that the designer generates content that he/she can tweak and polish, fine-tuning it for purpose. Dynamic generation gives little control, every time a new game starts new content is generated. Dynamic generation limits the fine-tuning options that static generation has, but creates more novel content without the developers intervention.

The world within Rouge is dynamically generated to facilitate a large number of new worlds. If the developer had to manually create all the levels used within

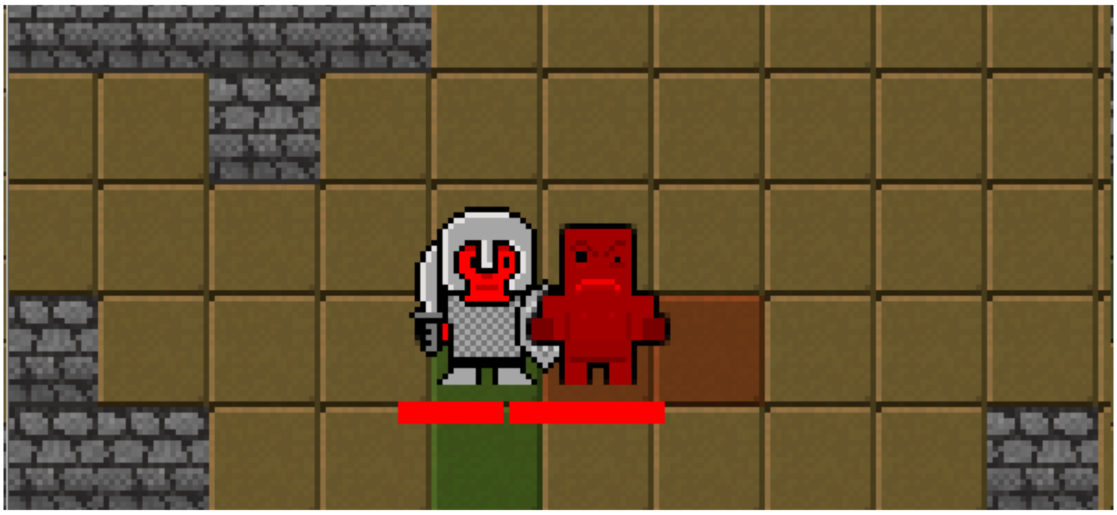


Figure 3.3: The player and a monster attacking each other

Rouge, either he/she would just have one level, or he/she would have to spend more time developing levels than developing the game itself. The second pressing reason for choosing a generative approach in contrast to a manual one is that level design might not be the designer's strong point, whereas software development is.

3.3 Cellular Automation

When level design isn't favourable, automating that process is. There are multiple well-known algorithms that solve this particular problem including but not limited to; *the drunken walk*, maze algorithms, and cellular automata. For the the world generation within Rouge I used a cellular automaton, for it's tried and tested use within other roguelike games and because I found it more technically challenging. Celullar automata work by defining a regular grid of cells, each in any number of states. Using rules pertaining to the neighbours of each cell, that cell can change its state. For example; we define the states *alive* and *dead*. After that we define several rules:

1. A cell goes into a *dead* state when it has less than 2 live neighbours.
2. A cell goes into a *dead* state when it has more than 3 live neighbours.
3. A cell goes into a *alive* state when it has exactly 3 neighbours.

With these definitions set the grid could be populated with a random amount of cells in live state, assuming that the dead state is the default. Each step of the automaton will cycle through the rules, checking to see if the conditions of a rule is set. After all the cells have been checked the rules execute and changes the grid. With these three simple rules, we can generate moving shapes and repeating patterns. The rules come from one of the most recognized cellular automata; **Conway's Game of Life**. Figure 3.6 shows an example of the emergent behaviour that these simple rules generate.

```
Input: tilemap
Output: new tilemap
let deadCells and liveCells be an empty collection of tiles;
foreach tile in tilemap do
    if tile.neighbours $\geq$ 5 then
        liveCells.push(tile);
    else if tile.neighbours $\leq$ 1 then
        deadCells.push(tile);
end
foreach tile in deadCells do
    tile.alive = false;
end
foreach tile in liveCells do
    tile.alive = true;
end
```

Algorithm 1: Cellular Automation algorithm as used in Rouge

During a step of this automaton, the algorithm walks through the game world and checks each tile for certain conditions pertaining to their neighbours. If these conditions are met, his state will be changed accordingly. The states a tile has in Rouge are a simple *Wall* or *Normal* state. When a tile has 5 or more wall neighbours, the tile becomes a wall themselves. However, when a tile has less than 2 wall neighbours, the tile becomes a normal. The first rule ensures that tiles group together, and the other rule destroys any singular 'island' walls.

As we will discuss in chapter 5 A game made with the Rouge system contains several entities. One of those entities is a *Space*. Spaces represent any or all spaces currently within a game-world, ranging from a single room to a entire city or even the world. Spaces in Rouge are designated areas where the player can find shops and resolve his/her objectives. These 'rooms' are generated using a simple tile-based flood-fill algorithm that circles around a selected tile. The algorithm has a maximum allowance that gets depleted by a tile's cost. This cost is higher for tiles that have fewer live neighbours than those that are encircled by them. This creates the effect that corridors are more expensive to fill (thus have a lower grouping rate) and large open spaces are easy to fill all the way through.

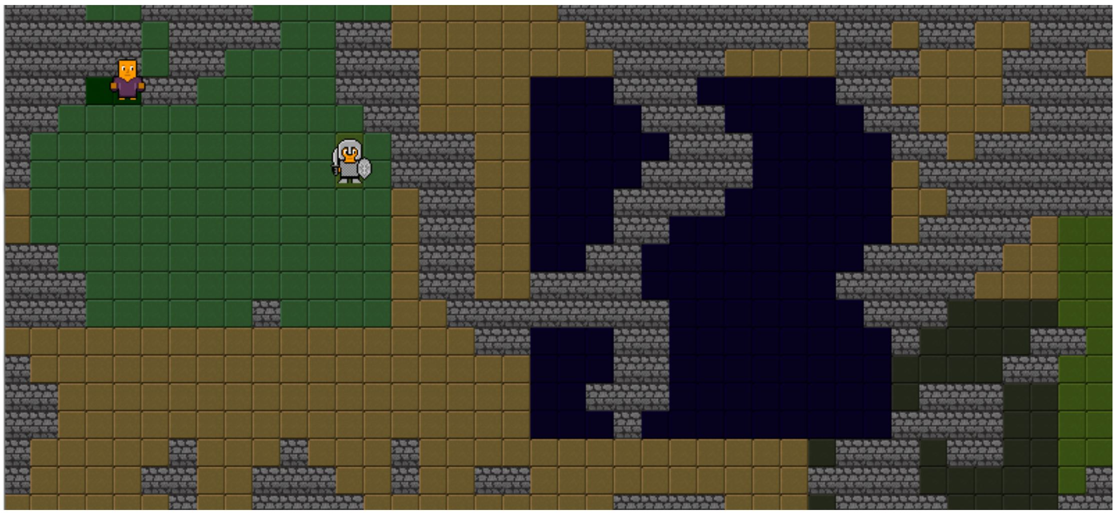


Figure 3.4: Space entities depicted by the areas with coloured tiles

3.4 Evaluation

We started discussing Rouge by asking the question: "*Why does Rouge need procedural content techniques*". Section 3.1 stated that world building could be done in three different ways; manual, dynamic generation, and static generation where I chose the dynamic route. This choice was made by the requirements that the game had. The game needed new worlds/levels for every new game that started. The manual option meant the designer had to develop all the worlds by hand, taking in a lot of time. Static generation would have given me a set of maps that the designer could polish into perfectly crafted worlds, which would be faster than manual creation and giving me pretty much the same results. Dynamically generating the worlds meant there would be more time implementing a more crucial element of the game: The narrative planner.

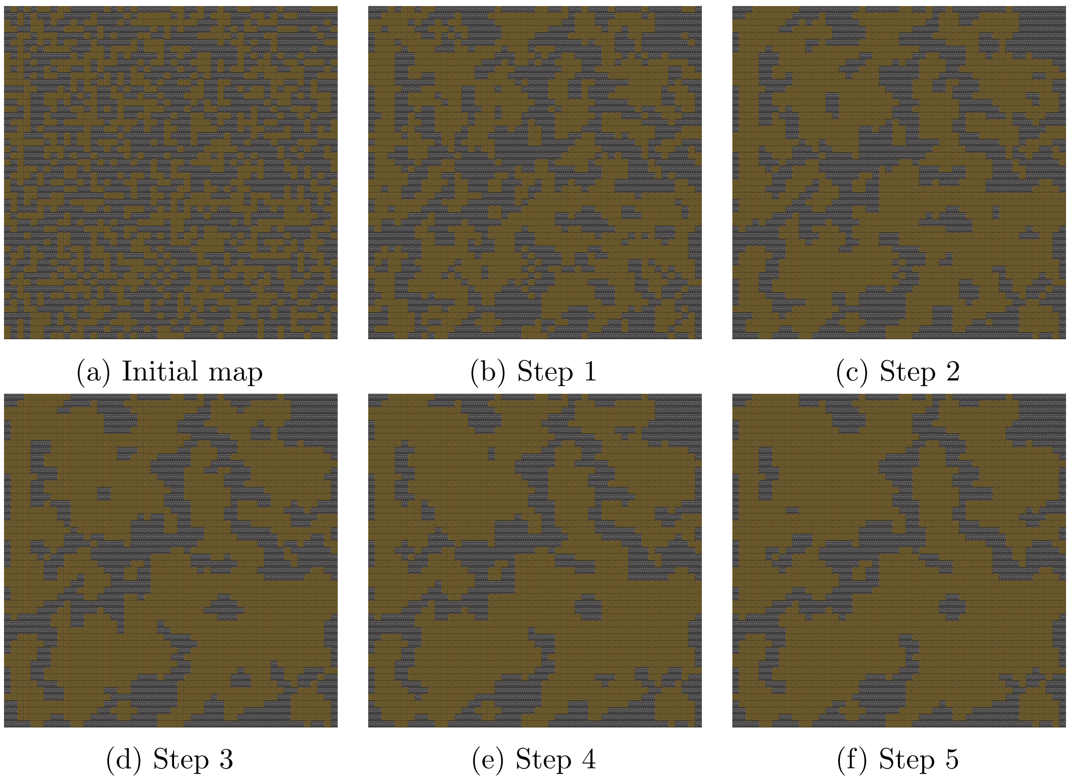


Figure 3.5: The generation of a world in Rouge

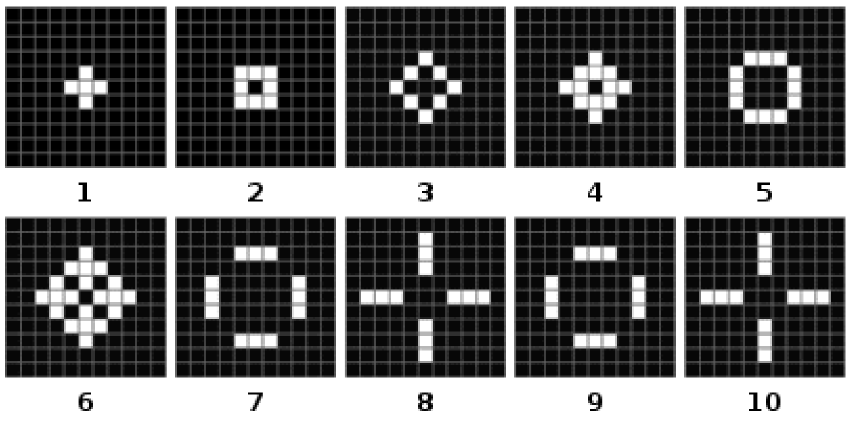


Figure 3.6: An example of the Game of Life generation steps

Chapter 4

Narrative planning

The game-play in Rouge benefits from an engaging narrative within the context of the game. When a character does something, the player should realise why this character does that specific action. Rouge needed a system that provides that context towards the interactions made within the game, but give a certain degree of author control over the narrative. Rouge itself tells the story of one person; the player's character. It chronicles what the protagonist lives through, the lives he touched, and the achievements and failures he made. This system should track the players actions, and propel the player into situations that have varying degrees of difficulty. Another functionality of this system was to remember the actions of the player, and to save the world state in such a way that the world could be revisited. When revisiting that world, a new player character could hear stories of the previous hero that could eventually evolve into myths and legends. The scale of renown is dependent on how many people the player has interacted with and influenced.

We state the question "Why does Rouge need a narrative planner" and observe what the planner does, and how it does it.

4.1 Narrative planning in Rouge

To achieve these results there are two distinct routes. One is using a *Believe - Desire - Intention*(BDI) system that grants NPCs their own agenda's and let the story unfold from that. The other would be to make a planner that acts as central director. BDI is a software model developed for the use of programming intelligent agents. The title alludes to every agent within the software model has their own beliefs, desires, and intentions and the agents should act upon those constraints. The model can create a highly sophisticated A.I. and by adding some additional constraints to make the agents biased to work with/against the player could create some interesting story settings. Alas, using a BDI system would mean that the system would become unpredictable to a certain degree. Directing the narrative would mean creating character archetypes and a whole lot of tweaking. And the most pressing issue: the NPCs are not the main focus of the game, nor the story. Creating a sophisticated AI for the NPCs would defeat the purpose of the game, telling the story of a protagonist. While there are parts of the BDI technique in Rouge, namely that nearly every actor within the game has a desire he wishes to fulfil, the latter option of a planner that acts as omnipresent director, is a more suitable option. It inspires the feeling of a pen and paper role-playing game, with the planner acting as Game Master. It iterates over the story during the end of every turn in the game, steering the character into dangerous encounters or interesting interactions with other characters. Besides that it keeps track of the attributes that characters have, and uses the world generation algorithm (as discussed in section 3.2) to create the initial world. This system, called *The Narrator*, actively drives the player towards that ultimate doom, giving more meaning to the phrase "*It's all about the journey, not the destination*". This heavily influenced the creation of the Narrator, as it doesn't need to concern itself with resolving the entire plot.

This chapter is dedicated to how the Narrator works and concludes with an observation on what other techniques would have been feasible.

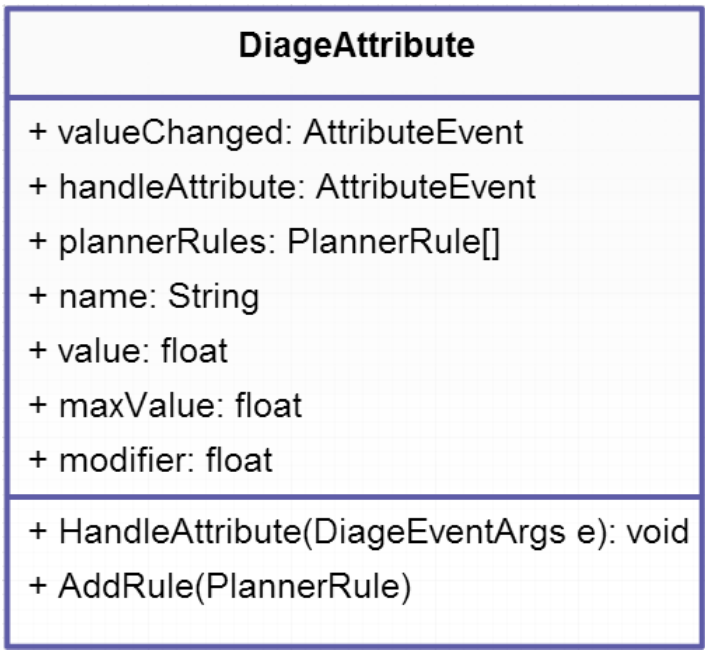
4.2 Actor attributes

To control and constrain the characters within Rouge the Narrator needed a subsystem that could unambiguously observe and measure their personalities and capabilities. From those requirements I made a key-value system in the form of the actor attributes. As discussed in section 5.1.2 Rouge has so-called *actor* entities. These actors represent all characters within Rouge, and they all have their own attribute set. These attributes are simple key-value pairs that can be used to describe a myriad of character statistics. Looking further the attributes can also describe behaviours that are executed whenever a actor interacts with another actor. In essence the actors don't really interact with each other, but rather their attributes do that for them. A simple example of an attribute would be a *strength* statistic that describes how much damage the actor can do to an other entity (see figure 4.1). Another example can be to add personality archetypes to an actor. Looking at the *Dungeons and Dragons* alignment table in figure 4.2 we could specify that the *Lawful Good* archetype will be friendly to all other actors on the *Good* and *Lawful* axes, but will become increasingly antagonistic towards an actor that veers towards the *Evil* and *Chaotic* side.

By exposing different attributes to the narrator the developer can direct the narrative towards their wishes. Besides specifying the actions and behaviours, the attributes can define actions that the narrator can use when planning a new story step. By looking at all exposed attributes within the given setting the narrator can assign actors to help or obstruct the player in his current mission.

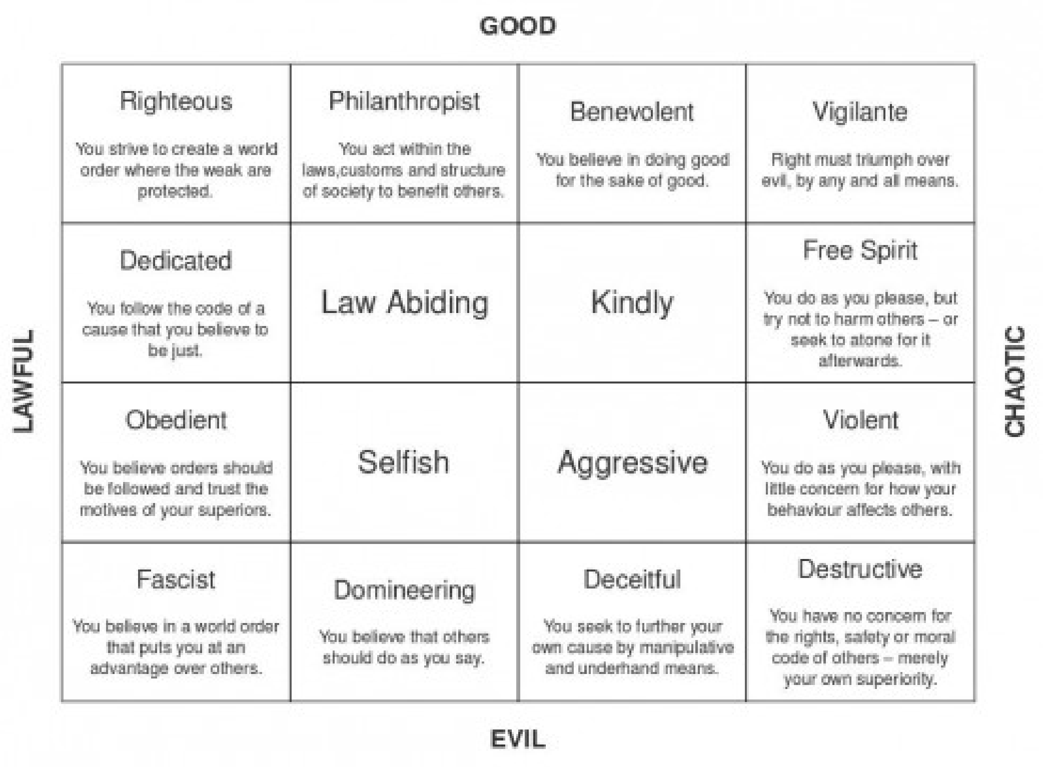
4.3 Initial Generation

Depending on the input given and the desires of the developer, The narrator does the initial generation of the game world and plots in varying degrees, as seen in algorithm 2. If no input is given at all, Rouge will just generate a random amount



```
DiageAttribute
+ valueChanged: AttributeEvent
+ handleAttribute: AttributeEvent
+ plannerRules: PlannerRule[]
+ name: String
+ value: float
+ maxValue: float
+ modifier: float
+ HandleAttribute(DiageEventArgs e): void
+ AddRule(PlannerRule)
```

Figure 4.1: Class description of the DiageAttribute



GOOD

LAWFUL — CHAOTIC

Righteous You strive to create a world order where the weak are protected.	**Philanthropist** You act within the laws,customs and structure of society to benefit others.	**Benevolent** You believe in doing good for the sake of good.	**Vigilante** Right must triumph over evil, by any and all means.
Dedicated You follow the code of a cause that you believe to be just.	**Law Abiding**	**Kindly**	**Free Spirit** You do as you please, but try not to harm others – or seek to atone for it afterwards.
Obedient You believe orders should be followed and trust the motives of your superiors.	**Selfish**	**Aggressive**	**Violent** You do as you please, with little concern for how your behaviour affects others.
Fascist You believe in a world order that puts you at an advantage over others.	**Domineering** You believe that others should do as you say.	**Deceitful** You seek to further your own cause by manipulative and underhand means.	**Destructive** You have no concern for the rights, safety or moral code of others – merely your own superiority.

EVIL

Figure 4.2: *Dungeons & Dragons* alignment sheet

of entities to populate the world. Input can be given either within the game code, or with a DML file (discussed in detail in chapter 5). Whether specified or not, the player character needs an initial constraint. This constraint is his/her first 'objective'. This initial constraint can be given within the input, otherwise Rouge will take an entity within the world and sets that as player constraint. Within the algorithm we see a section dedicated to custom rules. These rules can manipulate anything within the story setting. As an example I created a rule that randomly sets entities to a space as seen in procedure PopulateSpaces. The given example is purely non-deterministic, but is a simple matter to populate the spaces more evenly with the entities. This rule system is used throughout Rouge as a generalized form to give the developer more control on what a given entity can do.

```
Input: entities; customRules;
Output: Initial world state
let entities be all entities within current world state;
let customRules be the custom behaviour as specified by the developer;
foreach rule in customRules do
    rule.Invoke();
end
if $entities.count \leq 0$ then
    entities = GenerateRandomEntities(max);
end
if player.constraint == null then
    select random e from entities;
    player.constraint = e;
end
```

Algorithm 2: Initial planning

```
/* Populates the spaces with the current entities within world
   state                                                      */
Input: entities; spaces;
Output: All entities are randomly moved to a space
let entities be all objects within current world state, excluding spaces;
let spaces be all spaces within current world state;
while $entities.count > 0$ do
    select random s from spaces;
    entities.pop().MoveToSpace(s);
end
```

Procedure PopulateSpaces

4.4 Step Generation

When the initial generation is complete, the player should carry out his/her objective. When the objective has been completed, the planner will automatically start a step generation. This generation is like the initial generation but uses the actor attribute system and the player as extra input. The attribute system will be covered in a later section. This step generation looks at all the given variables and generates a new constraint for the player (see procedure GenerateConstraint). The longer the character's life, the more exact this generation will be; for any action taken by the player can be used in the generation.

```
Input: entities; actors; player; customRules;
Output: new world state
let entities be all entities, excluding actors and the player;
let actors be all actors, including the player;
foreach actor in actors do
    actor.rules.Invoke();
end
/* It's possible that one of the rules gave the player a
   constraint, so we'll check                              */
if player.constraint == null then
    GenerateConstraint(player);
end
foreach rule in customRules do
    rule.Invoke();
end
```

Algorithm 3: Step planning

```
Input: player
Output: new constraint for the player
if $rules.count \geq 0$ then
    foreach item in rules do
        rules.invoke(player);
    end
else
    select random e from entities;
    player.constraint = e;
end
```

Procedure GenerateConstraint(Player)

operator	meaning
sometime-before a b	*b* must be made true for the first time before *a*
sometime a	predicate *a* must be true at some stage of the narrative
at-end a	predicate *a* must be true at the end of the narrative

Figure 4.3: Event constraints as presented by Julie Porteous

4.5 Events

Riedl and Young[12] and Julie Porteous and Marc Cavazza[8] have demonstrated that stories are perfectly suited to be represented as a sequence of temporal and causal events. Porteous and Cavazza suggest to use events as a constraint for partial temporal order, using operators as seen in figure 4.3. These operators ensure that we have some control and gives us a partial temporal order, partial because not all of the events are ordered with respect to each other. The table is further expanded upon in their paper *Controlling Narrative Generation with Planning Trajectories: the Role of Constraints* [8]. The Narrator uses event constraints like this to layer plots and direct the player on what to do next. In the context of a roguelike we don't know when the story is going to stop, as it stops with the death of the player character. That could be within 2 minutes, but it could be several weeks. Rouge takes this into account by keep adding segments onto a story, sometimes using information gained from previous plots and at other times creating entire new spaces and NPCs therein. So the operators that Porteous uses we use within these small story steps instead of the entire narrative.

4.6 Planning flow

The previous sections described the processes that Rouge goes through when generating a new world. The flowchart in figure 4.4 displays the steps taken by the planner. The initial generation influences the world by setting up spaces, actors, and objects and manipulating these entities by their specific rules. After this, the player can, and will, influence the world by acting in it. The planner keeps track of his/her actions and saves this for future story generation [1]. The step generation gets activated automatically when the player's constraint is removed, i.e. a plot is resolved. This can also be manually activated by the author on whatever occasion or event he/she wishes. After the step generation the player gets his/her new constraint and, depending on the generative rules used, the world can be manipulated too. Be that actors that move, or new spaces that are generated. The step generation is highly dependant upon the player's attributes, as everything can react to the gain or loss between story steps. The actor attributes are a powerful tool in the Rouge arsenal, and the (game)world literary revolves around them. It has to be noted, that the design of Rouge works with semi-persistent worlds too. The world data can be saved to be used in a later play-through, giving the world a

[1]For example; a foe thought defeated returns for a rematch.

whole epic of one player, and starting the next.

4.7 Evaluation

I wanted a certain control on the narrative of Rouge, and I opted to use a narrative planner to give me those results. The only sensible evaluation is that a combination of BDI techniques and a narrative planner would be the most favourable, but due to time constraints that would not have worked. My choice to use a narrative planner was made from the need to have a central system to give commands to. These commands are now given in the form of the actor attributes, but there is a point to be made that the attributes are easily extended to encompass a BDI framework. Future work with this system could certainly encompass this change, but I stand by my choice with the time frame I had. The narrative planner was needed to convey the author-centric approach that I sought, whereas a bare-bones BDI framework would not.

The narrative planner shines in it's ability to be controlled by very simple constraints. Simply limiting the exposed attributes can heavily influence the Narrator's decision making. Another method of control is increasing the Narrator's bias towards certain attributes, so they will be used before any other. The downside is that, as the Narrator acts as every NPC, the other actors in the story can feel monotone and boring. In the same vain, the NPCs never act with, or react to, each other. Everything is focused upon the story of the protagonist, but the world feels empty without him. The latter part could be resolved by using a more sophisticated AI system, such as a BDI model for NPCs, and letting the actors react to each other, coordinated by the Narrator, to give a more natural feel to the story and the world.

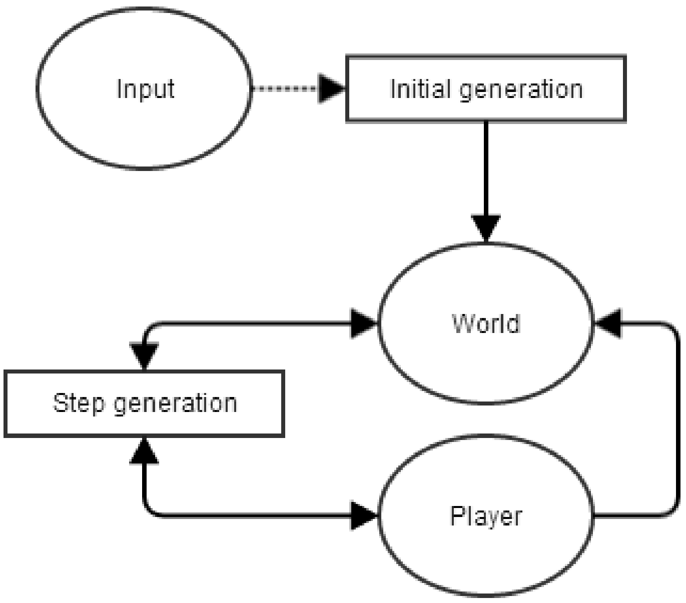


Figure 4.4: Simplified Narrator flowchart

Chapter 5

Diage Modelling Language

To help me visualise the stories I wanted to tell within Rouge, I needed a tool in which I could quickly draw up a narrative scenario to test different operators that the Narrator would perform. For this purpose I developed the Diage Modeling Language, or DML. DML is a very simple modeling language akin to those found in the UML specification. With DML I could create a scenario and declare certain operators to create a finite amount of possible story outcomes. It served it's usage as a prototyping tool and set the benchmark for what would become the Narrator.

Some functionalities that Rouge has were not conceived yet during the development of DML. The most noticeable are the actor attributes. That system was brought into the Narrator after the half way point of research and development time, in which DML was not actively used any more.

DML served its purpose for the early development phase by allowing me a quick and easy way to set up story prototypes, define the working operators that a narrative planner could use, and discover what kind of variations a - then hypothetical - planner could come up with.

This chapter serves as a run down of the DML specification, listing the applications and possible future work relating to it.

5.1 The structure of DML

DML is structured with the use of entities. These entities in three forms; `Actors`, `Objects`, and `Spaces`. The latter is also an information container which I will discuss in section 5.2. The following section will only cover the entities that affect the story directly; the objects and actors. In conjunction with `actions` and `accessors` these entities convey story information and plot progression.

5.1.1 Objects

Rouge objects form the basis of all entities. They represent the props and items that we find in the world that have narrative importance. For example; if the Player walks into a shop, Rouge does not specify all items that one could buy in the shop, but only those that have plot importance. In terms, these objects

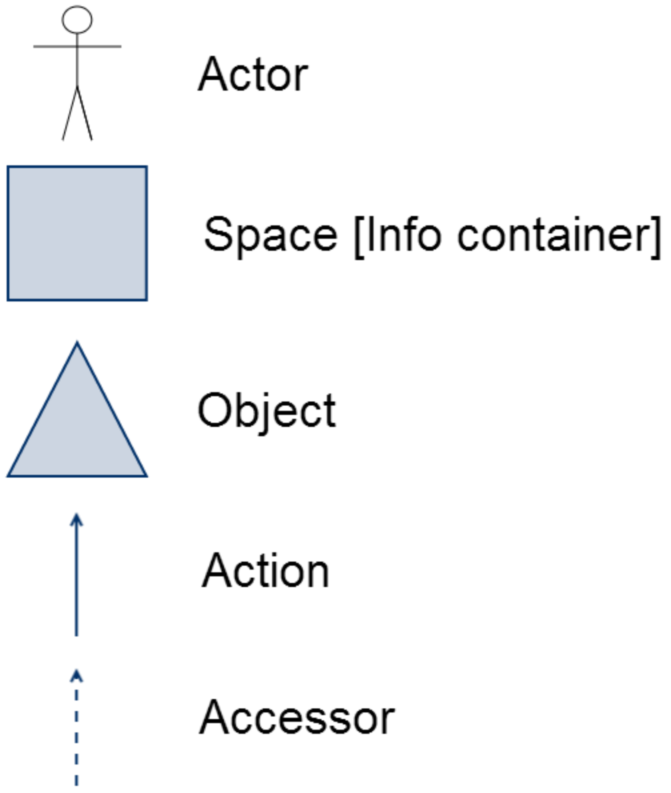


Figure 5.1: DML Symbols

should adhere to the *Checkhov's Gun* principle. This dramatic principle states that all objects used in a narrative should eventually be used. I quote: "*One must never place a loaded rifle on the stage if it isn't going to go off. It's wrong to make promises you don't mean to keep.*[1]" Objects have three properties; a `name`, a `ID` and a `type` The ID is a unique identifier dependant on the type. And the type is used with actions/accessors as seen in figure 5.2 (Futher discussed in section 5.1.2). The name is the noun given to an object within the story context. For example; The player receives the *Skeleton Key of Awesomeness*, but the type is just `key` giving it no special properties than any other key. It might make sense to name an object something else than it's type, but a name is not given it defaults to it's type.

Objects form the world, and all other entities derive from the Rouge object. This means that all entities have the same properties as the object, but can extend upon it.

5.1.2 Actors

An actor is the representation of any one object that can, as the noun implies, act. Examples are the store-clerk, a wandering adventurer or the player. The actor is

[1]`http://berlin.wolf.ox.ac.uk/lists/quotations/quotations_by_ib.html`

the only entity that can physically interact with the world, and by doing so the only that can change the world's state. By being able to change the world, the actors are the only entities that can ensure plot progression.

Just like the object, an actor has three properties; a `name`, a `ID` and a `type`. The type property is used in predefined actions as seen in figure 5.2a. This figure defines that the `Player` can **`speak`** to all actors of type `NPC`. A further glance at figure 5.2 shows some more actions that could be defined for the player actor. These predefined actions tell us that the player can trigger all events and enter all spaces. I will expand upon these actions in section 5.3.

5.2 Spaces

Information containers are entities that hold story information. A space holds information about its spacial children Information containers have the unique property that they are nestable. For example; a space that represents a city can hold several spaces that represents housing.

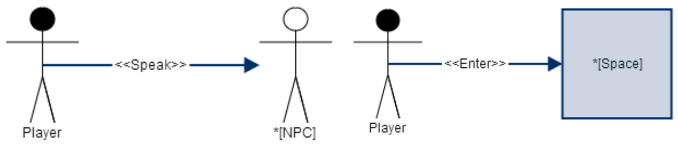
A space is the representation of any segment of the world or the world itself. As spaces are info containers they are nestable, as mentioned before, but they differ in the fact that they can hold every entity as a child. These children make up the spacial awareness of the space and tells us what information it can pass on to actors. Usually an actor gains all the information a space can give upon the moment it enters the space; when the actor becomes a child object to the space. This can be modified, and some parts of information maybe withheld from the actor, but this is where the events come in.

5.3 Actions and Accessors

Actions and accessors are the abstract connectors in Rouge. They convey what actors can do (actions) and what knowledge they possess (accessors). Some accessors are implied, due to the fact that an actor might be the child of a space, in other cases these connections are explicitly added to a Rouge model. If we review the diagram in figure 5.3 we see that the Mayor has no connections whatsoever. This would imply that the Mayor has no knowledge about what's or who's in the store. If we compare figure 5.3 to figure 5.5, we see that the Mayor now has a connection to the Shop, thus we can be sure that the Mayor has the knowledge it would have, if it had been in the shop space itself.

5.4 The narration

A DML diagram is a snapshot of the current narrative setting, and displays all relevant information to that setting. One look at the diagram should tell you what a given entity knows about its surroundings. This information can be conveyed whenever a entity interacts with an other entity.



(a) A standard action for NPC interaction

(b) A standard action for space interaction

Figure 5.2: Predefined actions

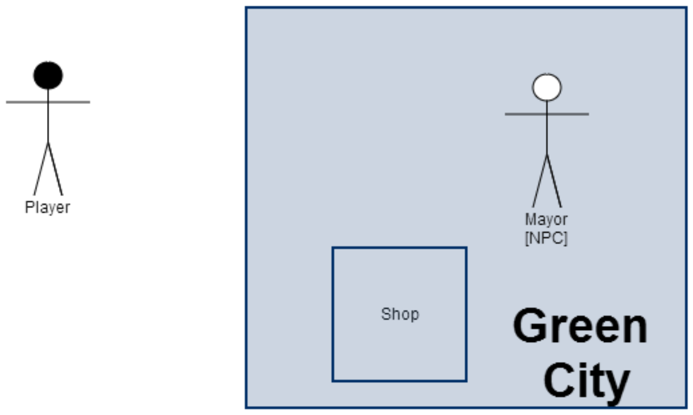


Figure 5.3: An example of a Diage diagram using predefined actions

As a rule, only actors are allowed to record information. Objects are still-lives and as such do not have spacial awareness, but may give information to the actor. The prime example is a treasure map that an actor picked up. The object in question contains the information on a possible treasure as located by the map. This information gets transferred the same way that an actor could learn new information from another actor.

As a snapshot of the given scene, DML loses the information of the scenario's before and after it. To gain a sense of story progression, one needs to look at several diagrams in quick succession to see what is really happening. Observe figures 5.6 and 5.7. These two diagrams display how a story segment (or quest in this instance) is modelled in DML. The examples in question are of the *Save the Princess* and *Fetch* quest variety.

The five diagrams in figure 5.6 shows us a *Save the Princess* quest. The Mayor is trapped, and the concerned citizen asks the player for help. To get into the dungeon the player needs the dungeon key, and helpfully the citizen told him where he could find that key. After unlocking and entering the dungeon, the player will have found the mayor in some kind of distressful state, and leads him outside. The point is, that we need the knowledge of previous story states, before we can assume that we know what is happening. DML has not been designed as a language that tells a story, but purely a way to display the state of one at a certain point of time.

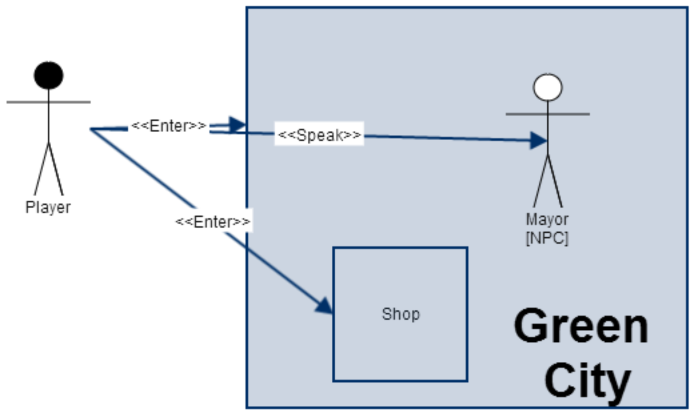


Figure 5.4: An example of a Diage diagram without using predefined actions

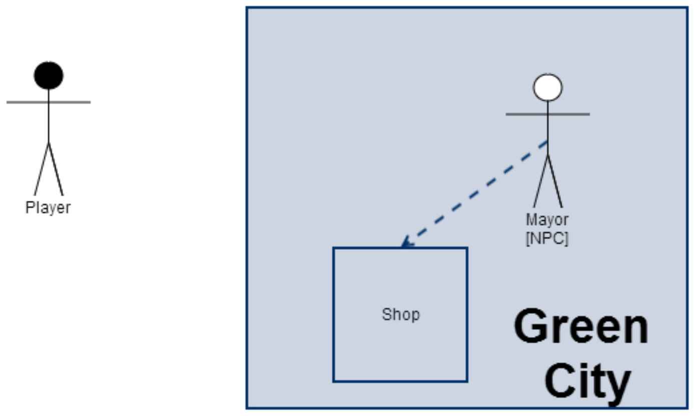


Figure 5.5: A Rouge diagram showing a explicit connection between the Mayor and the Shop

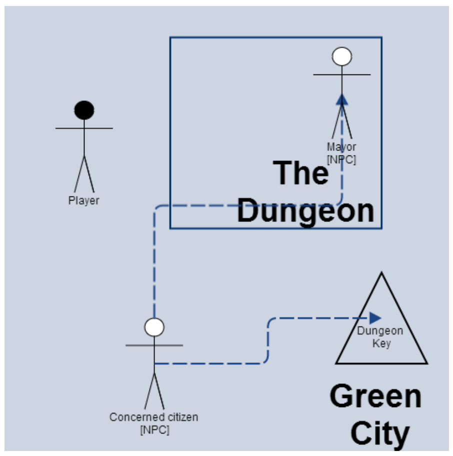
Figure 5.7 shows us a *Fetch* quest. In this quest, the player wants the hundred gold that the NPC has, but in order to gain that money, the player must locate and return an item that the NPC lost in the forest. Finding and returning said item gives the player the option of trading the lost item for the gold.

5.5 DML in Rouge

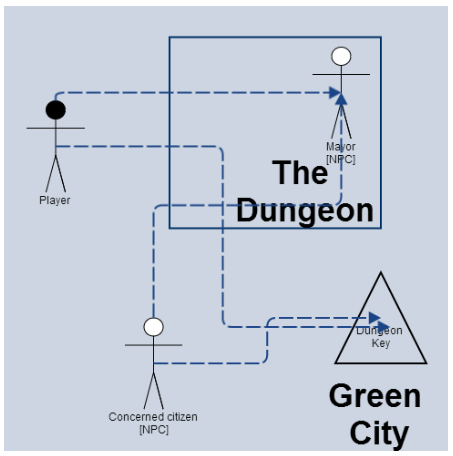
DML was created as tool to help designing and developing the narrative planner, not to be implemented within the game itself. However, the possibility of feeding Rouge the initial story state and reviewing the subsequent states that followed was a powerful addition to have. Enabling the designer to see if any changes made to the Narrator's constraints would have the desired effect, contrasted to outputting a debug log or text dump that could very well be filled with irrelevant information. The graphing software used up to this point, and indeed for this document, generates files that makes sense for it's type of software; but not to be intelligible for Rouge or the narrative planner. Hence, the DML Creator was developed. This tool, as seen in figure 5.8, allows a user to create DML diagrams, add attributes to actors and load the output file in to Rouge. When the player takes a snapshot of the story state, Rouge creates a similar DML file that the DML Creator can read.

5.6 Future work

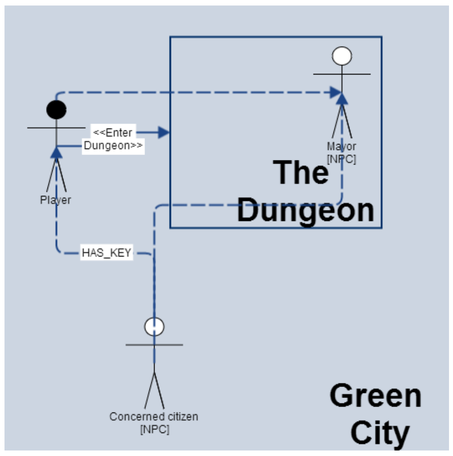
Rouge is normally expressed with a DML diagram, but the underlying structure is that of a graph. Due to time constraints there has been no work done in using this graph as a means to structure the direction of a narrative; however, DML is directly translatable to a graph. Figure 5.9 demonstrates this in relation to figure 5.7. More research in respect to graph translations in relation to Rouge could result in a better way to direct the narrative into a specific direction. As example; the designer could define an initial state, and an eventual state. Using graph translations Rouge could interpolate several *in-between* states that the narrative will take as seen in figure 5.10. A method could be developed in which Rouge could output all variants of narrative trees respectively to the choices available to the player, displaying all possible endings a discreet story arc could have.



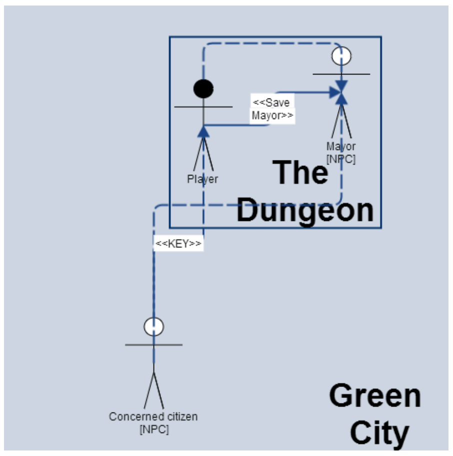
(a) Initial state.



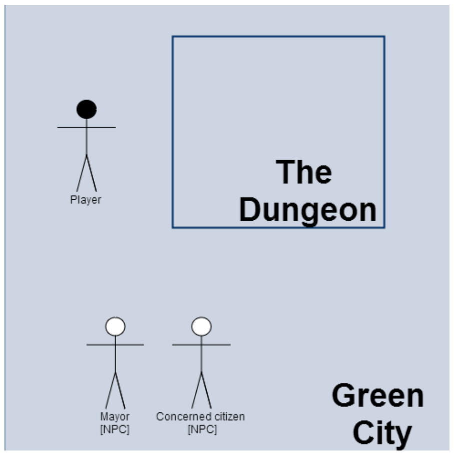
(b) Player speaks to the NPC



(c) Player has aquired the key

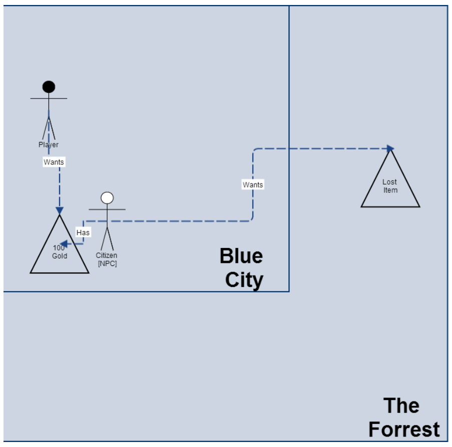


(d) The player entered the dungeon to save the *Mayor*

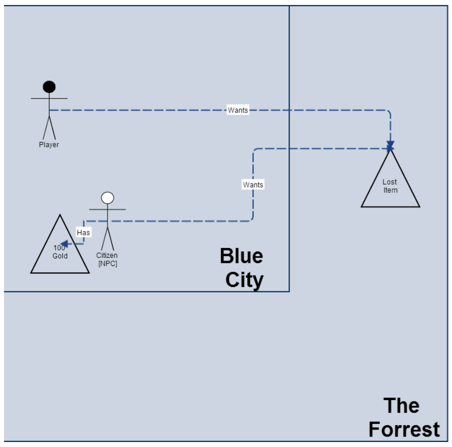


(e) The mayor is rescued from the dungeon! You stallwart hero of the land!

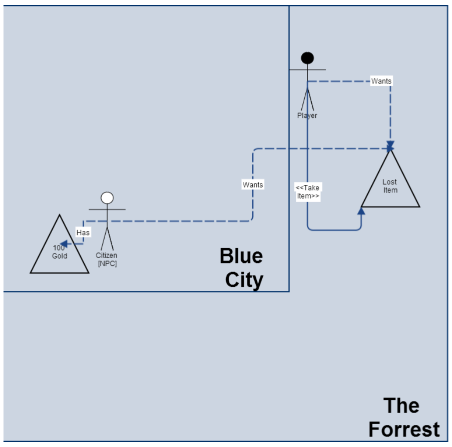
Figure 5.6: A *Save the Princess* quest modelled in DML



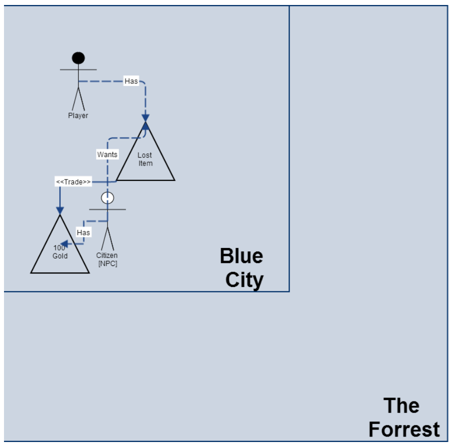
(a) Initial state.



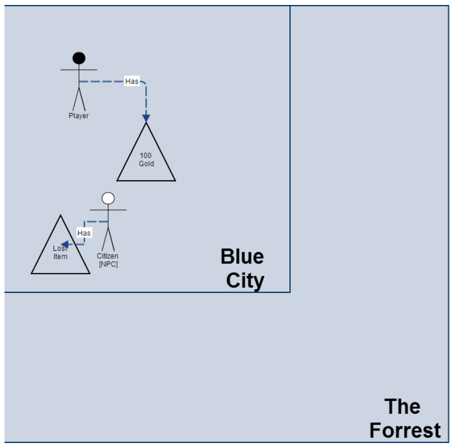
(b) The player speaks to the NPC



(c) The player enteres the forrest to retrieve the lost item.

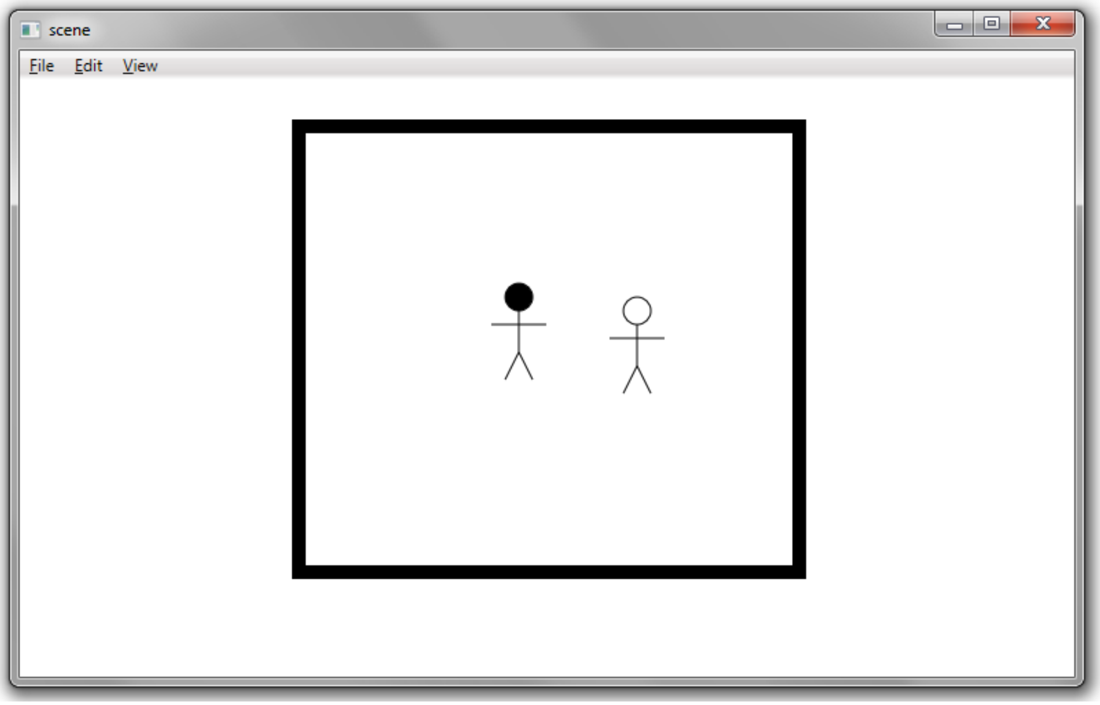


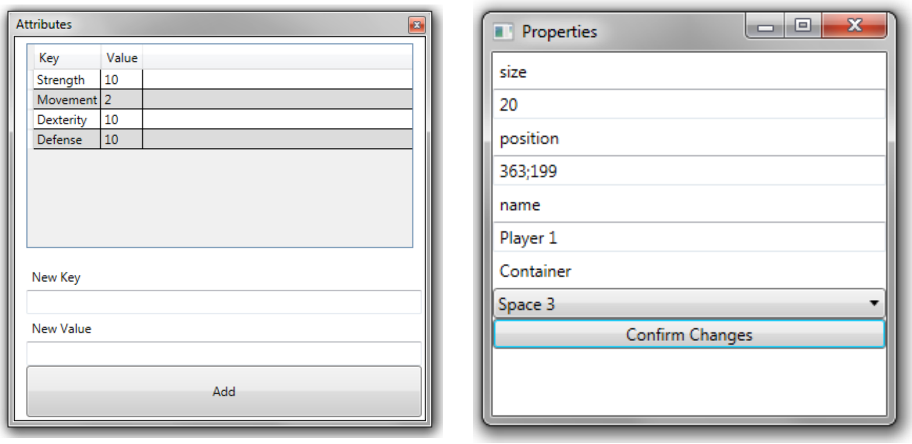
(d) The player returned to the city with the lost item.



(e) The player traded the item for 100 gold

Figure 5.7: A *Fetch* quest modelled in DML





Key	Value
Strength	10
Movement	2
Dexterity	10
Defense	10

New Key

New Value

Add

size

20

position

363;199

name

Player 1

Container

Space 3

Confirm Changes

Figure 5.8: The properties and attribute windows

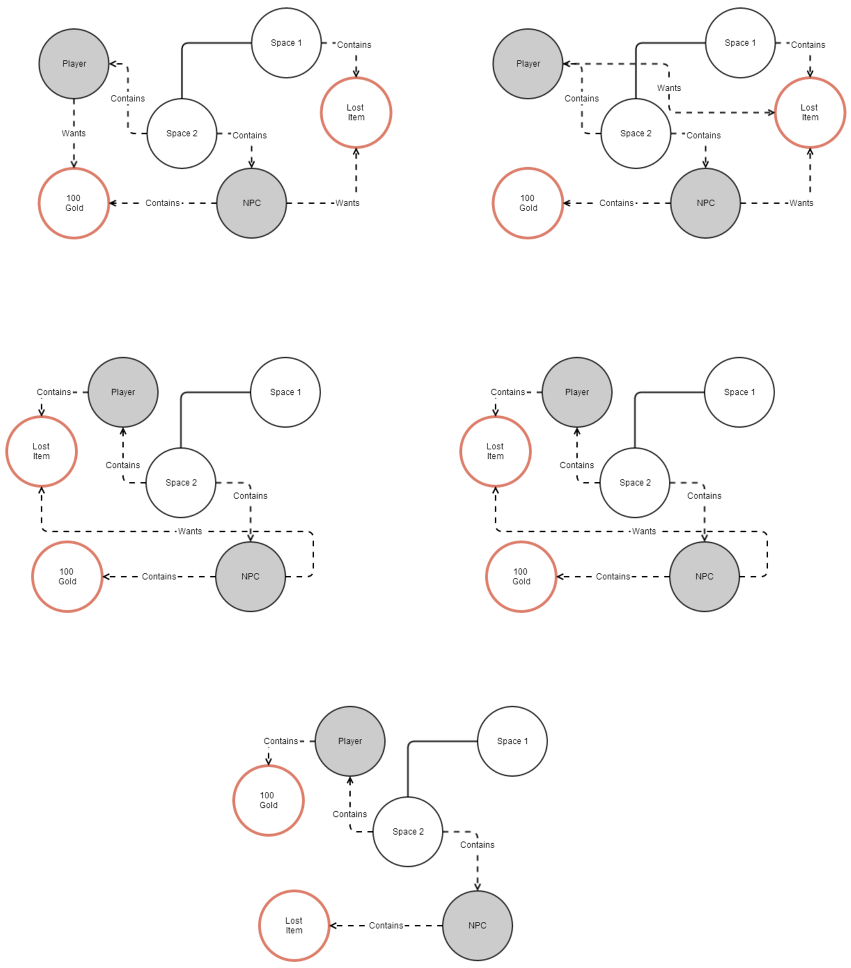


Figure 5.9: Figure 5.7 represented as a graph

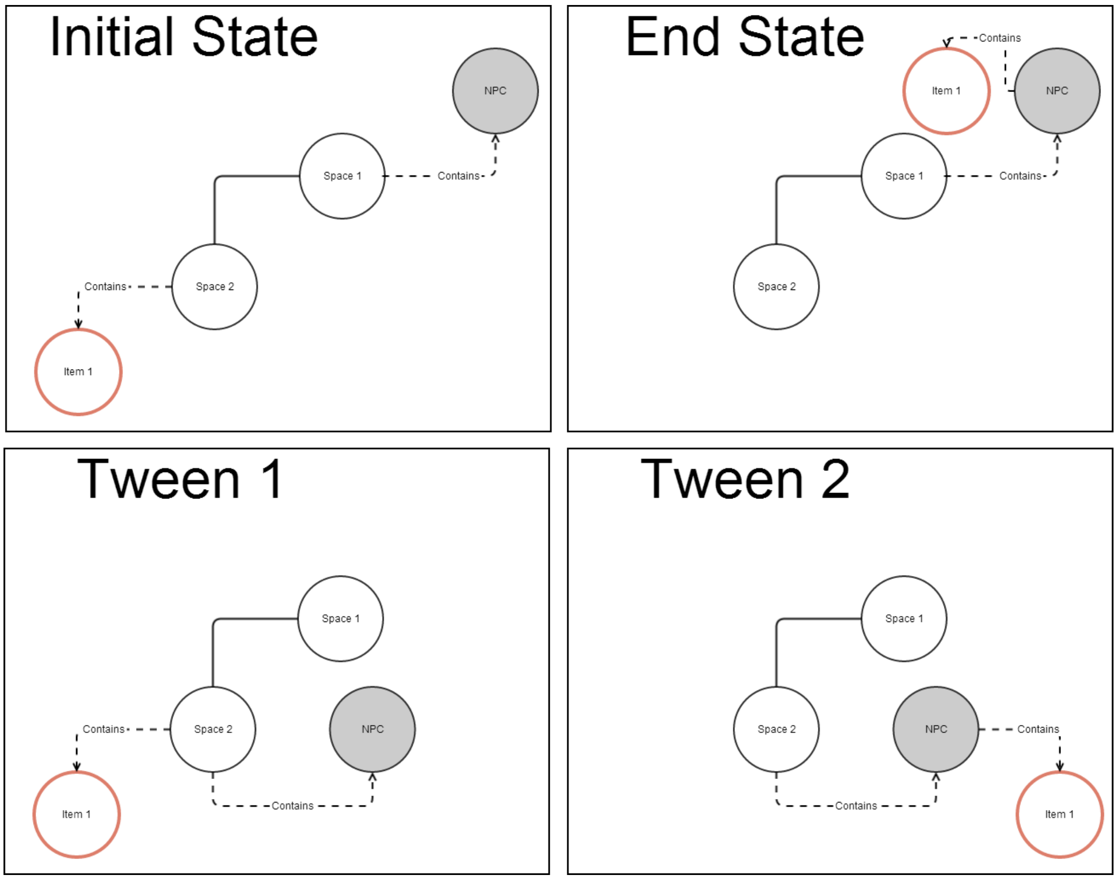


Figure 5.10: An example of an interpolated graph

Conclusion

When I started my research I asked the questions *How does the development process of a roguelike game benefit from using a generative narrative?*.

Procedural content is a large part of the roguelike experience. Without novel levels and a new way to traverse the world on every play through, the concept of turn-based dungeon crawling with perma-death loses its appeal.

In general terms; I hold that procedural narrative has value within video game development as a whole. In 20 weeks I've built a system that enables a developer to create games that generate an emergent narrative as a result of simple values given to simple representations. In chapter 4 I reasoned that for my scope and time going forward with a narrative planner was the way to go, in contrast to a BDI system. A narrative planner gives me much more control over the narrative system, whereas a BDI model would not. My conclusion is, however, that any further development of the narrative system should contain a form of the BDI model, with the Narrator as overall director of the actors. While true that my research focuses on one type of game, the field is open for further work into other genres. The Narrator itself can help developers create roguelike games within a smaller time-frame, due to the fact that the Narrator handles most of the story-writing. Using the Narrator a designer could design a game around the ambiguity of the context, with the inherent fact that every play through can give a player a sense of an actual new adventure. Even without the use of the Narrator, the attribute system allows designers to quickly design and test any game that has a implicit use of numbers, such as RPGs, Dungeon Crawlers and Tactics games.

As a summary; the development process gets the added benefit of a quick way to prototype stories in the form of the DML diagrams, a fast and efficient way of adding and perfecting game mechanics by the use of the narrative attribute system, and gains the ability to generate unique and novel narratives that is highly controllable, but requires relatively little input.

Reflection

I feel that a good grasp on agile software development strengthens any developer in his work. In any team effort I will always try to ensure that all team members have the same idea on how the development process should go, but not so much in any solo project. I have the impression that my efforts on solo projects are less disciplined than when in a team and as such decided to use a more agile approach in the form of a *design* - *develop* - *test* - *repeat* iterative process to try and structure my own projects.

What went wrong?

When we started this project, I had no idea of the field I was getting into. I had some previous knowledge on the world of Interactive Storytelling, but not to the extent of my own research goals. As a result, I spend a lot of time reading paper after paper on the idea of generating an interactive story. With half of the project time spend reading, there wasn't a lot of time left to build and test my theories on how to implement a storytelling algorithm for video games, resulting in me pressing for time to get the product and my thesis done. To make matters worse, during the last month of the project I decided to take on a completely different tract with my research, throwing away most of the work done already so I could do it properly. This enormous set-back was the result of figuring out what kind of product I wanted to make, with the consequence that I was just trying a lot of things out and became jumbled.

What went right?

I've always believed that good software comes from good processes, and this project was a great way to ingrain that in to my way of working. Instead of thinking of the product as a whole, I started to focus on what the functionalities where that I needed. Each functionality was first designed and developed to work alone, and only when this tested positive it was included within the greater whole. with no concrete idea of what the end result would be, I had no distractions as to work towards an actual product. We had a feature-set that stated what we wanted to accomplish, and that was my guidance for the project.

Conclusion

This method of working really suited me and helped me focus on the task at hand, without losing sight of the bigger picture. I am of the opinion that software development could benefit a lot with this form of looking at a product. Developers of a product can have a general idea for what the end result could be, but the spotlight should be on the requirements at hand, and solving them one by one until a product comes out of it.

Bibliography

[1] Marc Cavazza, Fred Charles, and Steven J. Mead. Character-based interactive storytelling. *IEEE Intelligent Systems*, 17(4):17–24, July 2002.

[2] Chris Crawford. *Chris Crawford on interactive storytelling*. New Riders, 2012.

[3] Marvin Katilius-Boydstun. The semiotics of A.J. Greimas: An introduction. *Lituanian quarterly journal of arts and sciences*, 36(3), 1990.

[4] Brian Magerko, John E. Laird, Mazin Assanie, Alex Kerfoot, and Devvan Stokes. Ai characters and directors for interactive computer games. In *Proceedings of the 16th conference on Innovative applications of artifical intelligence*, IAAI'04, pages 877–883. AAAI Press, 2004.

[5] Michael Mateas. An oz-centric review of interactive drama and believable agents. Technical report, 1997.

[6] Michael Mateas and Andrew Stern. Faade: An experiment in building a fully-realized interactive drama, 2003.

[7] James R. Meehan. Tale-spin, an interactive program that writes stories. In *In Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 91–98, 1977.

[8] Julie Porteous and Marc Cavazza. Controlling narrative generation with planning trajectories: The role of constraints. In *Proceedings of the 2nd Joint International Conference on Interactive Digital Storytelling: Interactive Storytelling*, ICIDS '09, pages 234–245, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] Mark O Riedl and R Michael Young. Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research*, 39(1):217–268, 2010.

[10] Mark Owen Riedl, C. J. Saretto, and R. Michael Young. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 741–748, New York, NY, USA, 2003. ACM.

[11] Mark Owen Riedl and R. Michael Young. Character-focused narrative generation for execution in virtual worlds. In *Proceedings of the International Conference on Virtual Storytelling*, pages 47–56, 2003.

[12] Mark Owen Riedl and R. Michael Young. An intent-driven planner for multi-agent story generation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pages 186–193, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Nikitas M. Sgouros. Dynamic generation, management and resolution of interactive plots. *Artificial Intelligence*, 107(1):29 – 62, 1999.

[14] Peter William Weyhrauch. *Guiding interactive drama.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1997. AAI9802566.

[15] R Michael Young and Mark Riedl. Towards an architecture for intelligent control of narrative in interactive virtual worlds. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 310–312. ACM, 2003.