UNIVERSITY OF DERBY

DEPARTMENT OF COMPUTING AND MATHEMATICS

A PROJECT COMPLETED AS PART OF THE REQUIREMENTS FOR

BSC(HONS) COMPUTER GAMES PROGRAMMING

ENTITLED

The Mystery Machine: An Answer Set Programming Approach to Generating Plots for Games

---

Author:
Michael IORIZZO

in the years 2012 - 2016

UNIVERSITY OF DERBY

Abstract

Department of Computing and Mathematics
BSc(Hons) Computer Games Programming

The Mystery Machine:
An Answer Set Programming Approach to Generating Plots for Games

by Michael IORIZZO

> "... There's not a cause for every effect," Otto said. "Life's a crap game."
>
> "Partner," said Sidney Blackpool, "you have to *make believe* there's a cause and effect at work or you'll never solve a whodunit."
>
> - Joseph Wambaugh, *The Secrets of Harry Bright* (2013)

The applications of Procedural Content Generation (PCG) are widespread, and their success has been demonstrated in both commercial games and research alike. Despite this, there are many areas that PCG is yet to cover comprehensively. This paper covers the possibilities of applying the paradigm to generating story plots, and the difficulties associated with such content. An existing technique which takes advantage of the power of Answer Set Programming (ASP) is implemented. The resulting plots are examined and tested in order to evaluate the use of ASP in the context of automated story generation for games. This implementation is built using the design space approach – as coined by Adam Smith and Michael Mateas – and it's components make up The Mystery Machine (TMM). This project finds that ASP and the design space approach provides great potential in application to the procedural generation of plots for games, ensuring that each plot remains a high degree of coherency without sacrificing variety. The implementation confirms Smith and Mateas' claims that the approach provides a powerful means for designers and story writers to become more heavily involved in the development of generators. Additionally, several weaknesses of the technique are found, and potential solutions are suggested.

Contents

Chapter 1

Introduction

1.1 Rationale

The application of Procedural Content Generation (PCG) – the algorithmic generation of content (Togelius et al. 2010) – to games has been an extremely prominent trend in recent years. However it is evident that neither current research nor commercial undertakings have even scratched the surface with regards to its full potential. PCG research takes many forms, and one of these forms covers exploration of the ability to generate creative content such as stories (Louchart and Aylett 2004; Aylett et al. 2005), artwork (Hello Games 2016) and even fully functioning games (Smith et al. 2011a; Cook, Colton and Gow 2014).

The paradigm of Answer Set Programming (ASP) has been creeping its way into games-driven PCG research with works by the likes of Smith and Mateas (2011), Thielscher (2009) etc. However it has yet to be fully applied to many of the areas covered by conventional generation techniques. The application of ASP to story generation has been addressed by Chen et al. in their paper: 'RoleModel: towards a formal model of dramatic roles for story generation'. Their implementation focuses on assigning a predefined *roles* to a line-up of characters in order to generate stories where each character performs a coherent series of actions that ultimately culminate in a conclusion. Perhaps contradictory to popular belief, using a computer's naturally logic-oriented operation – in the form of ASP programs – has already been proven to be an incredibly powerful tool in the generation of creative content.

1.2 Answer Set Programming

ASP is a paradigm originating from years of research in knowledge representation, logic programming and constraint satisfaction; it borrows key features from all of the above, all the while maintaining a balance between ease of use, expressivity, and computational efficiency (Brewka, Eiter and Truszczynski 2011). ASP, along with other methods of declarative programming, is oriented towards the solving of (primarily) *NP-hard* search problems. The paradigm and it's potential application to plot generation is covered in detail in section 2.6 of this study.

1.3 Plot/Story Generation

Plot or story generation is a concept often covered in research geared towards the achievement of *emergent narrative* (Chauvin et al. 2015). While this paper is not intended to focus on emergent or interactive narrative, it is a potential stepping stone towards the goal, and thus the concept is covered in section 2.5.

Plot generation in the context of this study refers to a content generator's ability to construct unique plots that reflect coherently in the eyes of a player, with the real end goal of achieving a system that could potentially create strong and complex plots to the likeness of popular story-driven games such as Bioware's Mass Effect (2007) and others.

The above goal is what separates this project from the *RoleModel* system developed by Chen et al. (2010). Instead of the comprehensive character-based approach taken by *RoleModel*, the implementation resulting from this study aims to provide a direct line of communication between a game's story-writer or designer and the resulting story generator; this provides the potential to achieve generators that provide varied content whilst maintaining the desired structure and coherency of similar, progressive game plots.

1.4 The Design Space Approach

Smith and Mateas have brought to the foreground a technique for applying ASP to the development of content generators that consists of three simple steps: identifying a problem's immediate design/generative space, building an interpreter to parse the resulting answer sets of the problem, and refining the original design/generative space via an on-going iterative process (until the desired design space is accomplished) (Smith and Mateas 2011). Their design space approach is referred to throughout this study, and an in depth coverage of the technique is covered in section 2.7.

1.5 Aims and Objectives

Smith and Mateas (2011) have presented a promising approach to effectively using ASP to design efficient and effective generative spaces for procedural content generation in games. This work has yet to be applied to the areas of game-focused story generation and interactive narrative.

The aim of this project is therefore:

To evaluate Smith and Mateas' design space approach to creating procedural content generators when applied to the generation of coherent plots in two respects: effect on the development process and the expressive range of resulting plots.

In order to achieve this aim, the project is split into the following objectives:

1. Conduct a broad review of existing procedural generation techniques for games.

2. Create an answer set program that defines a plot in logical terms, documenting the development process simultaneously.

3. Create an interpreter for the artefacts generated by the result of objective 2, in order for artefacts to be represented as data-structures that can be used effectively in a game environment.

4. Refine the logical program resulting from objective 2 by analysing output from the interpreter resulting from objective 3 to create a coherent generative space for creating plots, as adheres to Smith and Mateas' design space approach.

5. Evaluate the success of the developed product and the given hypothesis using results based on the expressivity and coherency of plots generated by the application developed.

6. Evaluate the development process in order to assess Smith and Mateas' design space approach as a practical choice for future plot generation projects.

1.6 Hypothesis

Based on the undertaking of objectives given in Section 1.5, the hypothesis of this study states that:

Applying a design-space approach to creating procedural content generators – using Answer Set Programming as presented by Adam Smith and Michael Mateas – will provide an effective means of generating both varied and coherent plots for use in games.

Chapter 2

Literature Review

2.1 Introduction

2.1.1 Review Aim

The aim of this literature review is to assess the value of applying the practice of ASP to the procedural generation of story elements in games, using Smith and Mateas' (2011) design space approach. In order to reach a conclusion, the review is structured in such a way as to provide the necessary background knowledge on the subject of narrative and procedurally generating game content first; thereafter, the technical background and a reasoning behind the hypothesis (see Section 1.6) is given.

2.1.2 Review Structure

- **2.2: Narrative in Games** - The review starts with a background and examination of what makes narrative an important focal point in commercial games, providing a basis for this study.
- **2.3: Malmgren's Anatomy of Murder** - This section provides an examination of what defines the Murder Mystery narrative and what it is that makes the structure an ideal candidate for PCG research in games. We focus on The Anatomy of Murder, an analysis by Malmgren on works written in the genre. Section 2.3.1 provides a definition that will be applied to the implementation covered in Chapter 3: Methodology.
- **2.4: Procedural Content Generation for Games** - This section provides an analysis of common PCG techniques used in games and demonstrates the typical layout of a procedural content generator.
- **2.5: Emergent Games** - Once the basic background information has been covered, section 2.5 covers in detail the challenges inherent in using PCG to replace or supplement the typical story writer for games.
- **2.6: Applying Answer Set Programming** - Once concepts of inherent challenges and the relevant generation techniques have been covered, section 2.6 will contextualize Smith's work on using ASP in PCG applications, and how a similar

approach could be applied to game narrative to solve some of the problems introduced in section 2.5.

2.2 Narrative in Games

Narrative has existed in games since the early days of the industry, however the method in which the narrative is portrayed has evolved significantly since the 1980s. In classic titles such as Super Mario Bros. (Nintendo 1985) for the Nintendo Entertainment System (NES), narrative was commonly provided as a supplement to the game itself, exclusively being told through mediums such as the instruction manual or additional literature. This often left much of a game's narrative up to the player's imagination, with little or often no guidance on the context of the events in the digital media. Plots were simple. In the case of Super Mario Bros., it becomes quite evident through playing the game that the plot is about the hero, Mario, saving a princess who has been kidnapped by a mutant tortoise. The player would only know that the tortoise is actually the king of the Koopas, King Bowser, and that the princess rules over the Mushroom Kingdom, if they had read the background information in the manual. Not knowing this information is not detrimental to the plot, however; it allows room for the player to create their own narrative whilst they play.

Over the years, narrative has taken much more of a front seat in many games. A common - albeit light-hearted - criticism of classic games is that they made little to no sense and lacked context. As a result, developers began to provide narrative context as a game mechanic. Super Mario World (Nintendo 1990), released years later for the Super Nintendo Entertainment System (SNES), provided players with simple cut-scenes and dialogue to help tell the story so that they did not have to look elsewhere. Overall, it is a trend that caught on and continues to be prominent today. Providing a side-by-side narrative has been shown to add significant value to a player's experience of a game. The importance can be demonstrated by comparing two games of the same genre, one which provides next to no narrative, and another who's world and plot is rich with information: Doom (id Software 1993) and Bioshock: Infinite (Irrational Games 2013) respectively.

Both Doom and Bioshock: Infinite are first person shooter titles, with an emphasis on the *Run and Gun* mechanic. The difference between the two games comes down to the plot. Doom has next to no plot involved, save for the set up: demons are pouring out of hell, kill them. The game uses level design to push the player forward. Other than that, the simple act of killing everything that stands in your way is enough to keep you involved as a player. In the case of Bioshock: Infinite, the plot offers an entirely new level of interaction. Remove the plot, and you've essentially got the same game with different levels and a different environment. The plot provides Bioshock with certain stimuli that make the player feel more involved; through emotion and curiosity.

2.2.1 Defining Plot and Narrative

Up until this point this study's reference to both *plot* and *narrative* may seem ambiguous. However, these two concepts are quite different in application. Plot refers to what happens along the course of a story. Narrative, on the other hand, covers what the end user sees and hears of what happens, and how it is portrayed (Brooks 1992). The purpose of the *Mystery Machine* is not to focus on one or the other, rather to be used as a means to produce data that can be shaped by would-be game designers in the application of games.

This project applies itself to the specifics of a typical *murder mystery* plot, due to their often logical and well structured design – the need for such construction will become apparent further into the study. The following section focuses on abstracting the structure of a murder mystery scenario so that we might later cover how to apply it to a logical answer set script.

2.3 Malmgren's Anatomy of Murder

The *murder mystery* genre has been prominent in western culture consistently throughout the last century (Kelly 1998; Haycraft 1968). The *Golden Age* of the genre brought a boom in its popularity throughout the 1920s and 1930s (Symons 1985), but within the last two decades there has been a resurgence in this popularity in literature (Trecker 2013; Worsley 2013).

Throughout his essay, 'The Simple Art of Murder', Chandler (1950) compares the literature of English and American authors of the genre. He states that: *"The English may not always be the best writers in the world, but they are incomparably the best dull writers."* Prior to this comment, he brings to our attention a clear polarization of the genre in this point in time between the two cultures. Here we are able to distinguish two sub-genres of the murder mystery: *Mystery* and *Detective* fiction, associated with both cultures respectively.

According to Malmgren (2001), what sets mystery fiction aside from its counterpart is the world in which the narrative takes place. Whether it take place in a mansion, a small estate, or a railway car, the mystery fiction is defined by a *centred* world; detective fiction, a *decentred* world. The centred world provides the narrative with a solid anchor around which the investigation is based. Effects can be connected to causes, and any external signs can be linked to internal conditions. This *centredness* provides great potential to develop the use of PCG in emergent narratives, because it allows room to eliminate a significant area of the problem space typically associated with large, complicated plots with many settings.

The next section will analyse exactly what makes a given scenario into a coherent story for the reader.

2.3.1 Means, Motive, and Opportunity

According to Grella (1970), the plot of a murder mystery – often also referred to as a "pure puzzle" or "whodunit" – is defined by a group of suspects who each have a motive, and have had the means and opportunity to murder a given victim. In the words of Chandler (1950), the genre has *"learned nothing and forgotten nothing"*. Grella confirms this statement by saying *"it subscribes to a rigidly uniform, virtually changeless combination of characters, setting, and events familiar to every reader in the English speaking world"*.

For the purpose of this study, we extract the three keywords of Grella's definition: means, motive, and opportunity. These keywords provide us with the foundations required to develop a content generator that follows the core principles of the *whodunit*, with which we hope to generate artefacts that reflect the coherence and variation of human-written plots. Our key focus will be on means – the killer must have had means to murder, i.e. access to a weapon that the killer would wield – and opportunity – the killer must have had an opportunity to strike, where he and the victim were alone (Mackall 2011). Motive is more subjective and will provide a less important role; it can be applied to any character with the correct narrative. This study aims only to create data for use in computer generated plots for games, as opposed to fully authored stories; therefore motive can be applied as an implication rather than an explicit point of data.

With the structure of our murder mystery data defined, the study will now move on to an entirely different topic: the use of PCG in games. It will become apparent further into the study how these two topics fit together; for now it is necessary to provide some background information on the more technical concerns of the project.

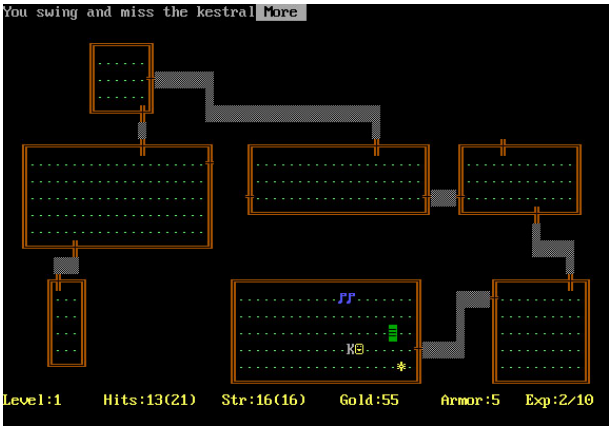
2.4 Procedural Content Generation for Games

PCG refers to the algorithmic generation of game content with limited or no human contribution (Togelius et al. 2013a); it is used to generate a variety of content types, such as art assets, levels, and even narratives. PCG can provide players with a vast stream of new content to extend the lifetime of a game. PCG is applicable beyond the physical details of a game world, and can be used effectively for more abstract content such as the detailed history and culture of multiple civilisations in Ultima Ratio Regum (Johnson 2015), and even mechanics such as in Michael Cook's Mechanic Miner (2013). The reasoning behind taking advantage of PCG in games is typically narrowed down into one of two limitations: technological and design space. Technological uses cover the use of PCG to create complicated or large applications on a budget, technologically speaking. On the other hand, design space uses cover the need to match an applications specifications while cutting down on development time and costs. Rogue (Toy and Wichman 1980) used PCG because of a technological limitation, whereas No Man's Sky (Hello Games 2016) uses it because of design space limitations; i.e. designing and creating millions of planets and their inhabitants would probably be a feat decades in the making.

2.4.1 Technological Limitations

In the early days of PCG the driving factor behind research was the desire to create more comprehensive and varied games under a very strict memory budget (Smith et al. 2011b). Early adopters such as Rogue (Toy and Wichman 1980) and Elite (Braben and Bell 1984) would simply have not been achievable on their target systems had PCG not been applied in development, due to the vast amount of content required in their design specifications. Even with today's modern computing technology, limitations in memory push researchers to develop methods further. The type of scale that can be achieved in games by adopting modern PCG approaches is apparent in No Man's Sky (Hello Games 2016). This game deals with the generation of entire worlds and star systems as opposed to the simple dungeons seen in Rogue and many of its predecessors. It also generates many of its own assets to help quell memory issues; storing each world and all of their assets on disc would be a difficult feat to accomplish even on many commercially viable computers even today.

(A) A planets surface in No Man's Sky. Each of the art assets seen in this image are generated by a PCG algorithm; everything from the fauna models and textures, to the wildlife roaming about (McKendrick 2014)



(B) A typical generated dungeon layout in Rogue. As opposed to No Man's Sky, PCG was mainly used in Rogue to overcome hardware limitations.

FIGURE 2.1: Two visual examples of PCG applications in games. Both games (No Man's Sky and Rogue respectively) use PCG to generate levels. In the case of No Man's Sky, this is to overcome design limitations, whereas in Rogue, it was used to overcome technological limitations.

2.4.2 Design Space Limitations

A common reason for developing PCG with modern technology is to develop Artificial Intelligence (AI) systems that can demonstrate creative *expression* in the process of game making. No Man's Sky achieves this to a degree with the systematic generation of its art assets. Michael Cook's *ANGELINA* algorithm is also being used to create entire games by itself (Cook, Colton and Gow 2014; Smith and Mateas 2010). Given

that the limits of memory with access to modern hardware in games is often negligible today, this concept of *artificial game design* has become the core driving force behind continued developments in PCG (Shaker et al. 2015). Achieving this concept to an applicable level in industry could help to alleviate some of the stress put on developers due to a continued rise in both time and monetary demand in the rapidly growing games industry (Togelius et al. 2010; UKIE 2015).

When applying a PCG system to a game, developers are required to surrender a part of the creative process to a logic-driven machine. This is a complicated problem to solve, because it involves creating algorithms that inform the generative process on what makes something *fun* or *boring*; both entirely subjective and *irrational* concepts.

2.4.3 Common Generative Techniques

The problem set associated with developing PCG games is typically dominated by the question of how to represent and interpret generative data in a way that creates interesting and engaging content for the player to interact with. The solution to this problem is typically achieved in three steps, which have been broken down in Figure 2.2 and covered throughout this section.

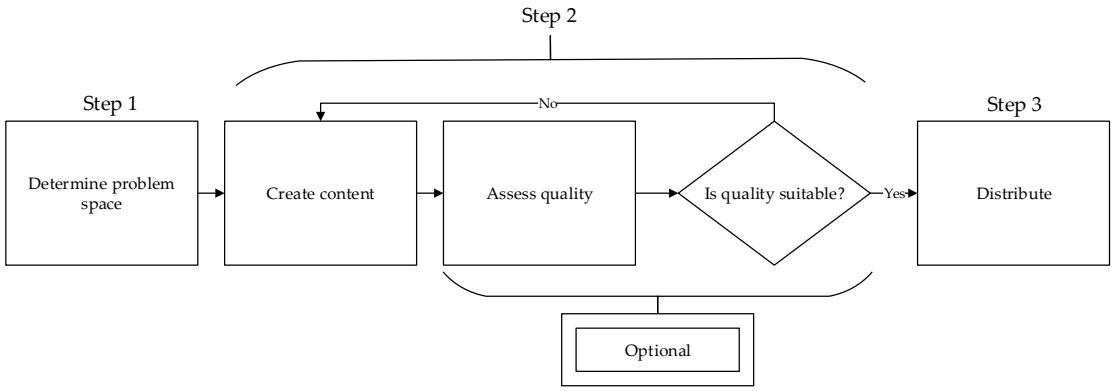


FIGURE 2.2: The typical process of creating a PCG system. Creating content and testing all fall under a single step because many commercial applications opt out of testing. In many such applications, the problem space is simple enough to not need testing; any content that fits the problem space will do.

2.4.3.1 Modelling the Problem Space

Problems in PCG must first be translated into a format that a computer can understand and work from, i.e. a rational format. For example, one can rationally describe a Roguelike dungeon to a computer program as a series of nodes that connect to one another; each node *may* contain a number of characters, decorations, and other objects (Shaker et al. 2015). Abstractions such as this are necessary so that the problem can be

presented in code; code abstractions of problem spaces are used to generate artefacts – the information that we require to use in the game. In many cases, these artefacts can be represented by something as simple as an array of characters; as in Spelunky (Mossmouth 2012) (see Figure 2.3).

1	1	0	0	0	0	0	0	0	0
4	0	L	0	1	1	1	0	0	0
1	1	P	0	0	1	1	0	0	0
1	1	L	0	0	0	0	0	0	0
1	1	L	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	1	1	1

(A) A data representation of a room in Spelunky

(B) A graphical representation of the room laid out in Figure 2.3a

FIGURE 2.3: The data representation of a room template in Spelunky is a simple 10×8 table of characters. Each character and its position in the table corresponds to a terrain or obstacle tile that should be placed in the room. For example an 'L' corresponds to a ladder tile, a '0' corresponds to air and a '1' corresponds to a solid block (Kazemi 2013). Here we have a side-by-side comparison of a rooms data and graphical representations.

2.4.3.2 Creating the Content

After a suitable data structure has been determined, the next step is to generate the data to fill it out. The three main approaches to this are outlined in Figure 2.4. The most commonly used in industry is the simple *constructive* approach. This is used with the fore-mentioned Spelunky and many other successful Roguelikes. As mentioned in Figure 2.2, this is down to the relatively simple requirements of a Roguelike where it is often deemed not necessary to have to ask the question *what makes a level good*, when *what makes a level beatable* will suffice.

Many popular and successful commercial Roguelikes - like Spelunky and The Binding of Isaac - use a template-based constructive approach (Johnson, Yannakakis and Togelius 2010). Using this method, individual assets and rooms are pre-created by developers and are then selected at the time of generation according to values in the data structure. These pre-fabricated templates will fit together as a larger whole into a level that appears original each time. One of the drawbacks to this method is the generated content can lead to a lack in variety between iterations.

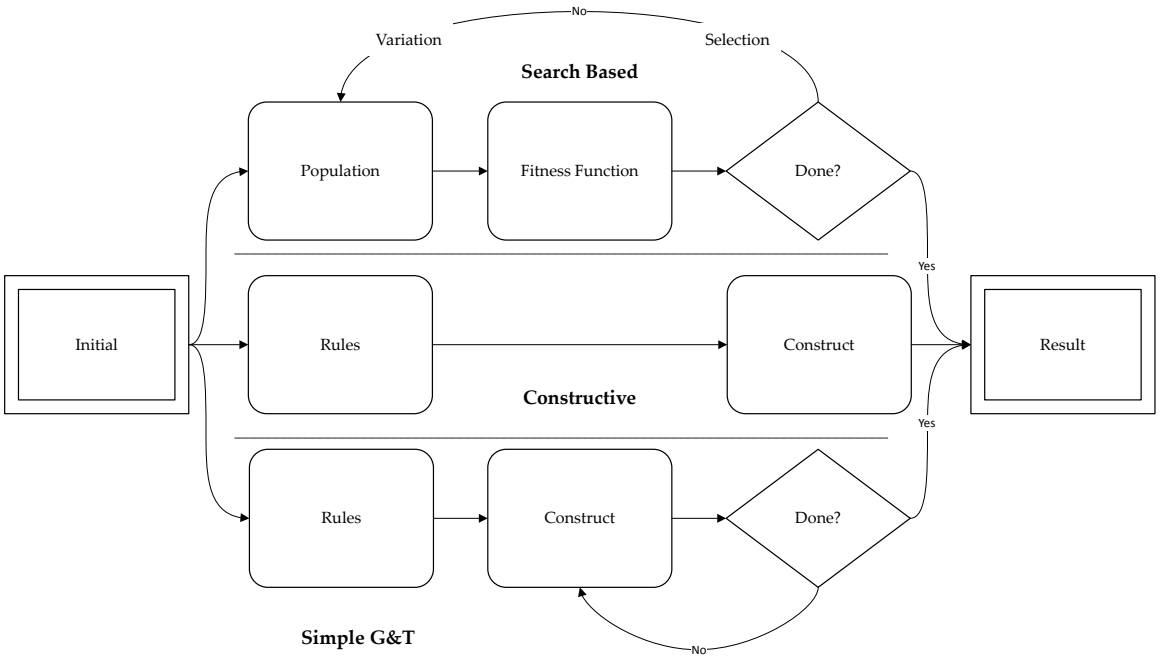


FIGURE 2.4: Three approaches to procedural content generation; **Constructive**, **Simple Generate-and-Test** and **Search Based**. Note that not all search/optimization algorithms suitable for PCG keep a population of candidate content, but the most commonly used ones do (Togelius et al. 2010).

2.4.3.3 Assessing Quality

Generating content to fit a set of criteria does not always result in *good* content being generated. As previously touched on, in many cases this isn't an issue because the desired output may be simple enough to avoid cases where it might be undesirable or *unplayable*; also mentioned previously, many applications use a template based approach which to some degree negates the possibility of generating undesirable content. However, when it is required that generated content be tested – such as with the generation of story content – Constructive approaches will not suffice.

Figure 2.4 provides an overview of **Constructive**, **Simple Generate-and-Test (SG&T)**, and **Search Based** approaches. Both SG&T and Search Based methods are implemented in cases where testing content is a necessity. Once content has been generated it is assessed by one or multiple *fitness functions* that determine whether or not the particular iteration is suitable for the application (Togelius et al. 2010). If the case matches *true*, it is accepted as the result. If it matches *false*, a new iteration is generated. **Search-Based** approaches will keep a population of content deemed *acceptable* or *good*, and create slight variations of successful candidates when generating new iterations; an application of this approach can be seen in *Galactic Arms Race* (Evolutionary Games

2010), which keeps a population of projectile types that a player regularly uses in order to create similar ones, deeming that if a player uses them often, they must be *good*. **SG&T**, on the other hand, does not keep a population of content. It can be seen as the halfway point between **Constructive** and **Search-Based**, whereby if an iteration fails in its fitness function, a completely new iteration is generated.

2.5 Emergent Games

An *emergent* system in a game works contrary to the typical *progressive* system; that is to say: situations in an emergent game *emerge* as a player progresses, as opposed to a progressive game where everything happens in a pre-designed time-line (Chauvin et al. 2015). Emergent games are created through an approach to game design that emphasizes the use of rules-based systems rather than pre-scripted or hard-coded paths (Sweetser and Wiles 2005), hence the employment of PCG techniques. Emergent games represent some of the most successful game products released each year, such as Minecraft (Mojang 2009), Spore (Maxis 2008), Terraria (Re-Logic and Engine Software 2011), and Don't Starve (Klei Entertainment 2013).

FIGURE 2.5: A sample of what a block of generated terrain might look like in Mojang's Minecraft (AotF 2013).

In Minecraft, the physical world emerges around you as you explore. As a player reaches the end of the world in its current state, another 16x16x128 *chunk* of terrain is generated – such as the one depicted in Figure 2.5 (Persson 2011).

2.5.1 Challenges in Interactive and Emergent Story Writing

Interactive and Emergent storytelling in games introduces a complex new issue to the practice of applying PCG systems. While issues in other areas of emergent games typically cover the areas of data representation and content testing – as outlined in

Section 2.4.3 – diverging from a progressive storyline introduces the new issue of *non-deterministic consequence*. In a progressive story such as in Mass Effect (Bioware 2007), at certain points in the game, players are presented with the *illusion of choice* (Jenkins 2004). At these points, players will be made to make a decision that is pre-fabricated to slightly alter the *narrative* process of the game, without causing a significant impact on the plot (see Figure 2.6). These branching points provide *deterministic consequence*; consequences that can always be foreseen by the story's author. Using deterministic consequence allows game writers to have complete control over a story in order to maintain a coherent story progression, but can diminish a players sense of *agency* (Riedl and Bulitko 2012). On the other hand, using non-deterministic consequence makes it much more difficult for the writer to maintain coherent story progression, but can bolster a players sense of agency.

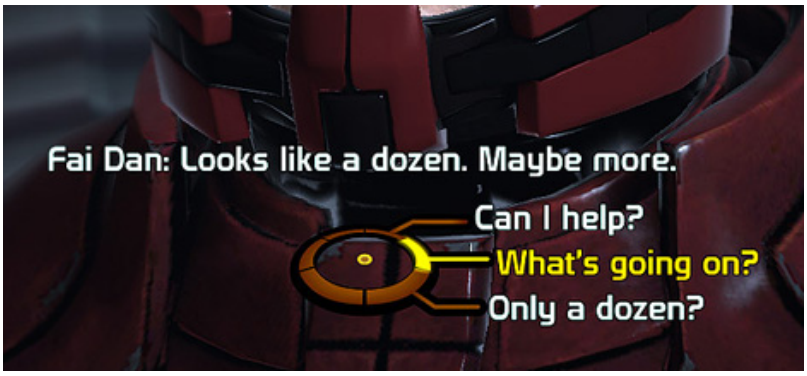


FIGURE 2.6: An example of the choices presented to the player throughout the course of Mass Effect. The player's decision will only impact the narrative process, but the plot will remain completely unaffected (Wikia n.d.).

In current research terms, emergent narrative (Aylett 1999) refers to the complete automation of an undefined story (Riedl and Bulitko 2012). The term goes hand-in-hand with *interactive* narrative; which refers to the management of a plot by giving consequence to the player agent's actions.

One method of how we might achieve an emergent story is to apply a *drama manager* to a game such as Minecraft. In Minecraft there is no set goal for the player to achieve, rather a setting and various NPCs to interact with and avoid. Typically the first thing a player will do in a game of Minecraft is to build a home; once they have a home they have a means to store food and put a wall between themselves and an angry mob of enemy NPCs. Chauvin et al. (2015) suggest that given the correct, formalized data representation of this player safe in his/her new home, a drama manager could begin to introduce interesting situations from which a player could build their own story. For example, the drama manager may introduce a new friendly NPC that might become a neighbour to the player, or instigate an enemy assault on the player's home. Such a system puts heavy emphasis on *co-direction*, which Chauvin et al. acknowledge as a fundamental aspect of interactive storytelling.

It is possible that co-direction could be achieved via plot generation. Given a plot defined logically and entirely, the player could truly be put in charge of the narrative process. An alternative covered by this study is to generate a single, static and comprehensive plot; from the data generated this way, a separate AI system could be applied that presents pre-defined narrative points to the player according to their actions.

2.5.1.1 The Whodunit in Games

The whodunit plot has already been applied in various games. Noir Syndrome (Glass Knuckle Games 2014a) and The Inquisitor (Brown 2014) stand out in this study due to their applications of PCG . These two games correspond to both detective and mystery fiction respectively (the distinction is covered in section 2.3).

(A) Explore various locations to uncover clues (Glass Knuckle Games 2014b).

(B) A player prepares to make the arrest (Glass Knuckle Games 2014b).

FIGURE 2.7: Two screenshots from Noir Syndrome. Players take control of a 1920s detective to track down a rampant murderer and eventually make the arrest.

Noir Syndrome takes place throughout a 1920s-esque city, where the player has to visit various locations to collect clues that help them track down a rampant murderer. The content generator works to populate a static city setting by random assignment. Several NPCs are created, and each assigned a group within the story, either: Civilian, Police, or Mobster. From the list of NPCs, one is picked at random to become the murderer. Finally, 10 *clue* items are generated which correspond to the selected murderer by their group and hobby (also randomly selected) (Gedarovich 2016).

The Inquisitor takes a different approach. Like Noir Syndrome, it generates a roster of characters (8 in total); each character is assigned a series of quirks, taken from templates. The characters are each placed in a random room throughout the mansion setting shown in figure 2.8. What makes The Inquisitor particularly interesting is the way in which it generates the murder; it won't simply apply a random character to the role, but it will take the character through a step-by-step process as to how the murder is committed. The generator considers which rooms the character moved through on their way to the victim, what weapon was used, and who witnessed their movements. This provides the game with the features of means and opportunity, which have been flagged as key features of the whodunit in section 2.3. The game also assigns each

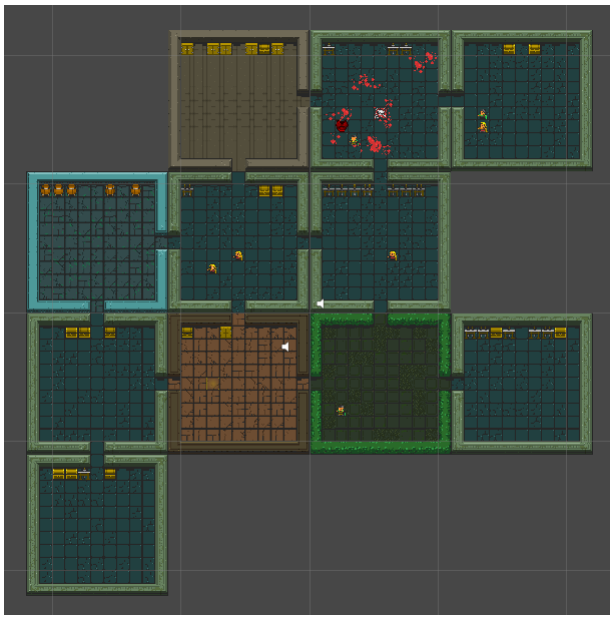


FIGURE 2.8: The mansion setting in The Inquisitor. It corresponds to the *centredness* referred to in section 2.3 (Brown 2016).

suspect character (4 from the list of innocent characters) and the murderer a motive, covering the third and final feature (Brown 2016).

The plots in both of these games are generated by using simple template-based constructive techniques, referred to in section 2.4.3; while The Inquisitor has never been available for commercial review, common criticisms from Steam user reviews for Noir Syndrome involve a lack of depth and variety (Glass Knuckle Games 2014b). Many widely used approaches have commonly received criticism in these two areas (Togelius et al. 2013b). However, by studying these existing titles, we can begin to visualise how a logical system might generate and represent artefacts which represent similar games whilst reliably maintaining plot integrity and providing higher levels of variety.

Before the idea of a logical content generator can be explored any further, however, it is necessary to provide a background on the paradigm of logic and answer set programming.

2.6 Applying Answer Set Programming

At this point in the study, the reader should have a sound understanding of traditional PCG techniques, the importance of plots in games, and should be aware of current research into AI driven storytelling. The last piece of background required before we move on to Smith and Mateas' ASP for PCG implementation concerns logical programming and its applications. This is necessary before talking in depth about ASP, as the two share many similarities.

2.6.1 Logical and Answer Set Programming

Logical programming is a form of programming used to solve specific logic problems; unlike the classes and functions you would typically find in a procedural language such as Java or C++, code in a logical language is expressed as a series of rules (predicates). Prolog is one such language. In Prolog, each rule is expressed as:

```
head :- body.
```

This syntax states that a head *must* be connected to a body. A query to the rule is considered true if the conditions are satisfied. An example given by Wong (1993) expresses the rule "X is a grandparent of Y if X is a parent of A and A is a parent of Y" as:

```
is_grandparent_of(X,Y) :- is_parent_of(X,A), is_parent_of(A,Y).
```

In a similar fashion, one could describe the basics of a whodunit as:

```
suspect(X) :- motive(X).
guilty(X) :- suspect(X), killer(X).
innocent(X) :- suspect(X), not guilty(X).
```

This set of rules simply declares that any candidate with a perceived motive must be a suspect, any suspect who has killed must be guilty, and any suspect who is not guilty must be innocent. The killer also *has* to be a suspect or else the whodunit is inherently unfair and goes against the rules (see section 2.3).

2.6.1.1 Finding Answer Sets

ASP (Baral 2003; Lifschitz 2002) is a derivative of traditional logic programming, and provides a similar declarative approach for modelling and solving search problems, represented as logic programs (Gebser et al. 2008). An ASP system consists of a grounder and solver. The grounder is used to translate a problem description – represented as the logic program – into a propositional program. This program is then processed by the second component, the solver. A visual representation of the process is illustrated in Figure 2.9

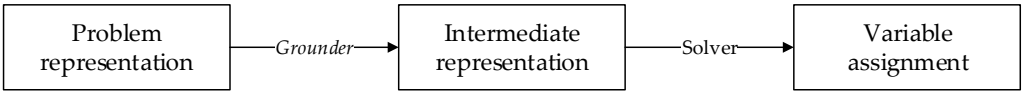


FIGURE 2.9: Architecture of an ASP system (Gebser et al. 2008).

ASP refers to the programming paradigm, much as one would refer to the paradigm of object-oriented programming. Examples given in this section will be written in the

Clingo 4 input language according to the ASP-Core-2 standard (University of Calabria 2012; Potassco 2008). Gringo refers to a concrete syntax used to write answer set programs; although it shares common syntax features with its parent for the description of logical terms, answer set solvers are not Prolog interpreters (Smith and Mateas 2011).

Using ASP, collections of logical terms can readily represent any data-structure. If you were to represent properties of some game content artefacts, for example, you might use some of the following logical terms:

```
weapon_type(butcher_knife,slashing).
valid_move(kitchen,cellar),
valid_move(kitchen,dining_room),
valid_move(dining_room,lounge),
valid_move(dining_room,hallway),
valid_move(lounge,hallway)
phase(1,murder),
phase(2,find_evidence),
phase(3,make_arrest).
```

This still appears very similar to what we might find in a typical logic program. Logical terms are used to describe to the logic program what *must* be true. What makes ASP unique is the ability to provide descriptions of what *might* be true; ASP introduces the use of *choice rules* and *integrity constraints*. A choice rule is defined in code as a collection of terms surrounded in braces where any number of them might be true as facts in the logical world. An example of reasoning with choice rules follows:

```
{stabbed;beaten;shot}.
bloody :- stabbed.
bloody :- shot.
bruised :- beaten.
clean :- not bloody, not bruised.
```

The above snippet says that a victim may have been stabbed, smothered, beaten, or shot; or a combination of any or none of the above. The following rules tell us that if the victim was either shot or stabbed (or both), then the corpse will be bloody; likewise if the victim was beaten, the corpse will be covered in bruises. The final rule denotes the expectation that if the corpse is neither bloody or bruised, then it is clean.

ASP is focused on answer *sets*: collections of ground facts that are consistent with the logical world described in an answer set program. The program above admits eight possible answer sets; three boolean values with two possible states each is equal to: $2^3 = 8$. Integrity constraints allow programmers to specify what *must not* be true in the logical world, regardless of what other rules state (Smith and Mateas 2011); they are used to *"imply contradiction"*. In the given murder example, we can use an integrity constraint to enforce that only one murder method can have been used, by adding the following line:

```
:- stabbed, beaten, shot.
```

Integrity constraints give a programmer the ability to filter out undesirable answer sets without having to understand the process by which those undesirables might arise in the first place (Smith and Mateas 2011). This capability implies an invaluable use for ASP in PCG, given that a lot of sophisticated generation methods – such as the search-based and SG&T methods covered in Section 2.4.3.3 – require the use of fitness functions after generation to do the same job; integrity constraints do not only hide undesirable answer sets, they prevent them from ever being generated in the first place.

2.6.2 Answer Set Programming for Procedural Content Generation

Procedural content generators create artefacts based on a *generative* or *design* space. Due to the *nondeterministic* nature of many generators, it is appropriate to view the success and failure of artefacts from a design space point of view. According to Smith and Mateas (2011), PCG inherits a default "*ill-definedness*" from a design perspective. ASP provides a logical alternative to defining design spaces within PCG applications. Thomas and Carroll (1979) suggest that the uncertain requirements of a design problem are best resolved in iterative steps by proposing candidate solutions; deterministic answer sets afforded by ASP provide an ideal platform for iterative refinement of a design space by a designer.

2.6.2.1 Designing Plot Structures as Answer Set Programs

Provided a plot structure can be defined in a definitive manner – such as the whodunit (see Section 2.3.1) – Smith and Mateas' design space approach should be applicable for use in generating said plots for games. This approach is ideal because if we can define the plot as an answer set through tweaks to the design space, the question of testing for a coherent plot will no longer be an obstacle. We know this because an answer set provides deterministic artefacts; traditional nondeterministic generators require artefacts to be tested to assess their relevance. This would also afford story writers a much more significant role in the development of generators, offering them more control over the resulting plots than traditional generators would. Albeit, the writer would no longer be writing stories, but crafting story-structures; much like the whodunit, or the famous *"hero's journey"* (Campbell's 1987).

2.7 Smith and Mateas' Design Space Approach

With the relevant background and technical information now covered, we can look into Smith and Mateas' design space approach in detail. Following an analysis of their implementation, we will take a look into further work that has stemmed from their research, and determine whether applying this approach to story generation and interactive narrative is constructive in the context of current PCG research.

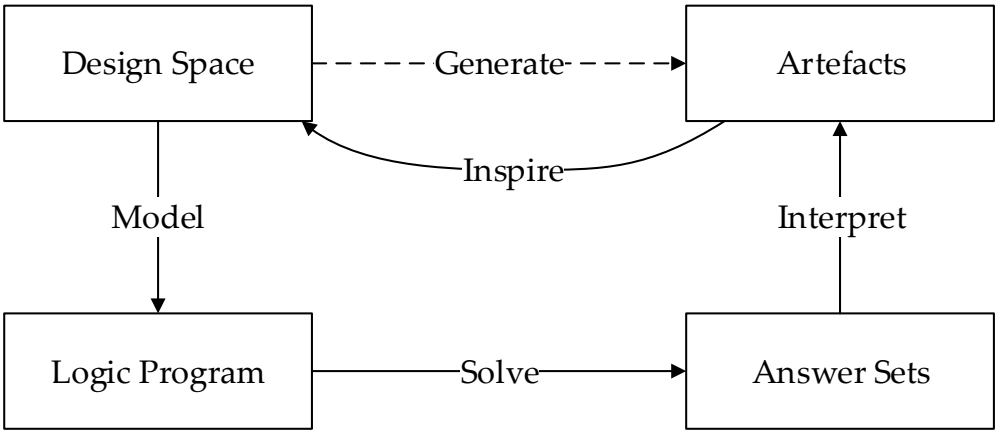


FIGURE 2.10: With Smith and Mateas' method, the intent to generate artefacts from a conceptual design space is carried out by modelling the design space with a logic program. A solver produces answer sets which are interprited as descriptions of the desired artefacts. Experience with generated artefacts inspires redefinition of the design space until the desired solution is found (Smith and Mateas 2011).

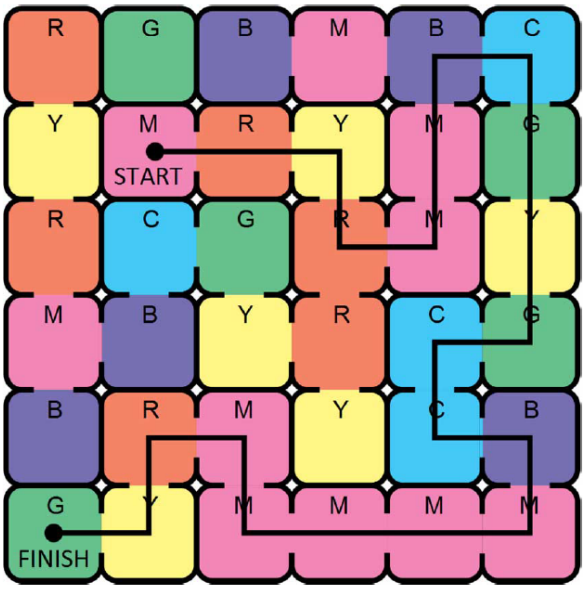


FIGURE 2.11: A sample chromatic maze created with Smith and Mateas' ASP-based generator. Valid moves consist of single steps on a red-yellow-green-cyan-blue-magenta colour wheel (repeats allowed). The dark line represents the shortest solution to the puzzle (Smith and Mateas 2011).

2.7.1 Implementation Details

Smith and Mateas' implementation of an ASP driven content generator covers the generation of a chromatic maze (Smith and Mateas 2011). A chromatic maze is a grid-based puzzle similar to a traditional maze, where movement is restricted by the colours of the current tile and adjacent tiles as opposed to explicit walls blocking access. Passage between tiles is by the adjacency of the tile colours on a colour wheel as shown in Figure 2.11. The first step in creating the generator is to organise the problem into a set of logical terms that describe a chromatic maze.

2.7.1.1 Defining the Logical Terms

In ASP, artefacts are represented by logical terms that describe their in-game properties; there is always a minimal set of terms needed in order to construct an artefact in the context of a game (Smith and Mateas 2011). For the chromatic maze – or any maze for that matter – the minimal set consists of:

- The start location;
- The end location;
- The maze's *"traversability"*;

As well as the minimal set, a good design space will define artefacts with additional annotations about properties which are derivable and analysable. Though not required in the running of a game, these annotations provide a vocabulary for describing the shape of a design space (Smith and Mateas 2011). In the case of a maze we might want to consider:

- The reachability of a given location;
- The length of the shortest solution;
- The number of dead ends;

Each item in the previous two lists denotes a *fact* or a series of *facts*. An answer set is always made up of a collection of facts that lead to an outcome; it is the unique combinations of these facts that lead to a particular answer set and provide the generator's artefacts. With our logical terms defined, the design space can start to be represented; Smith and Mateas represent the design space with some assertions about what properties individual artefacts might have and other complimentary assertions about what properties artefacts either *must* or *must not* have – using choice rules and integrity constraints respectively. When this design space model is passed on to an ASP solver, we see the solver produce the collections of facts we can use to construct the in-game artefact (Smith and Mateas 2011).

2.7.1.2 Modelling, Interpreting, and Refining

The design space approach is all about iteration; with a design space defined, the next step is to take samples of artefacts that the current design space produces in the game it is intended to be used in.

Smith and Mateas (2011) suggest the bulk of the modelling is done first and foremost; this is partially covered in Section 2.7.1.1 as defining logical terms. Logical terms must be represented in code, and provide the schema for properties that will define an artefact. We must know this schema in order to interpret the artefacts that the solver outputs. With this structure in place, the designer now requires a loader in place with which they can interpret the artefacts in some representation of the game world.

The interpreter is required in order to analyse what an artefact will look when observed by the user. In the case of the chromatic maze: how will a room or tile look, and how will the maze as a whole appear? According to Smith and Mateas (2011) the fixed grammar of logical terms makes them relatively easy to parse and convert into data structures required by a game engine. When run through an ASP solver, the artefacts should be displayed by the interpreter as they would appear in the game. This provides a means for the designer to analyse the artefacts and determine any potential changes that could create more desirable results, as dictated by the design goal.

With the interpreter and basic generative space defined, the greater process of *sculpting* the design space can begin. The basic generative space achieved in step one is liable to include many obviously undesirable artefacts due to lack of constraints, but it is the presence of these undesirable results that provide the feedback to drive the refinement of the model (Smith and Mateas 2011). This refinement might include building support for new data structures or carving away unwanted interactions between artefacts. For example, in the chromatic maze generator, noticing unreachable nodes might suggest an addition of a constraint that enforces a viable path between all nodes in the maze.

The purpose of this study is to determine the viability of applying this approach to the generation of game plots through defining plot structures as series of logical terms and constraints. The next section will conclude the analysis of Smith and Mateas' design space approach with a look into how this could be achieved.

2.7.2 The Potential for Generating Plots

The design procedures covered above in Smith and Mateas' approach shows great potential in application to plot generation for games. Each plot follows a given structure, such as the structure of a whodunit (see Section 2.3), and from observing trends in modern games we can see how much a game's story experience can differ despite using the same – or a very similar – plot structure. An example that comes to mind here is Bioware's Dragon Age (2009) and Mass Effect 3 (2012); the player's experience of the plots differ greatly, but they are very reminiscent of each other and clearly follow the same structure:

- Plot opens with some catastrophe;
- The hero is left alone with a small party;

- The hero is given leave to gather allies by some given authority;
- The hero and his party embark on a quest to gather a force of allies from a variety of backgrounds;
- The hero and his gathered force makes a final assault against the forces behind the initial catastrophe and prevail;

As an answer set generator, this plot structure and its prerequisites could theoretically be defined in logical terms to allow a unique experience for each player whilst maintaining the plot coherence demanded by the given type of game.

2.8 Discussion and Conclusions

In order to evaluate whether or not the proposed project will be valuable, ASP and the design space approach must be assessed on how well suited it is to the application of generating plots, and how it compares to other approaches presented in the review.

2.8.1 Discussion

Smith and Mateas (2011) state that the design space approach is '''rooted in the recognition that PCG is really concerned about two problems: the reliable generation of desirable content artefacts from a design space, and the design of content generation processes which are *fast*, produce *fancy* artefacts, and do so in a *flexible* way". They conclude that ASP provides a reliable and proven platform for creating '''trustable, search-intensive solutions to rich content generation problems in a way that minimizes commitment to procedural details". While the application of ASP to PCG problems has been relatively limited in comparison to other approaches, it has undoubtedly shown its potential with regards to how quickly it is able to produce a variety of content, in terms of both computing time and designer effort (Smith and Mateas 2010; Smith and Mateas 2011; Neufeld, Mostaghim and Perez-Liebana 2015).

2.8.1.1 Comparison to Alternative Approaches

Constructive approaches are by far the most commonly use in commercial games, whether they are applied to the generation of maps/dungeons (Mossmouth 2012; Macmillan and Himsl 2011), or weapon generation (Gearbox Software 2009). The use of this approach in many commercial products can be seen as success through simplicity, as the systems themselves take very little effort to implement while providing a game with novel variety. The weakness of the approach is its common reliance on template content; it is often applied in the form of random number generators selecting different combinations of hand-authored components. Examples of this can be seen in section 2.4.3. When applied to games with a story-centric nature, it has also shown to provide little in the way of interesting and engaging content (see section 2.5.1.1).

The coherency of a plot cannot be guaranteed by the constructive approach due to its inherent lack of testing facilities (Togelius et al. 2010).

On the other hand, an ASP powered generator would be designed in such a way that absolutely guarantees plot coherency, strictly according to the design space laid out by the designer (Smith and Mateas 2011). The design space approach also has no need to rely on a designer creating large selections of template content.

SG&T and Search-Based approaches (see section 2.4.3.3) have proven comprehensive methods for generating desirable content; particularly generators made using the latter often employ sophisticated genetic algorithms to create content that has proven successful by some measurable means. The development process for generators of this type have been geared towards the generation of simple content, such as the generation of weapon types in Galactic Arms Race (Evolutionary Games 2010) and terrains in Chapas (Frade, Vega and Cotta 2010). Their use of fitness functions to ensure the coherency of generated content – and the associated re-generation of content following a fitness function failure – also have significantly negative implications on computational time if applied to something as complicated and abstract as a story (Togelius et al. 2010). When compared to ASP there is perceivably less potential with these approaches to directly involve a designer or story-writer due to the complex procedural details inherent in the involved algorithms and fitness functions. ASP consists of a relatively simple syntax in up to date languages, and the recent establishment of the ASP standard (University of Calabria 2012) of the paradigm supports its use going forward. Its application to solving broad and complex search problems in comparatively fast computational times (see section 1.2) also suggests it to be an ideal candidate for use in the betterment of generating largely varied creative content such as stories and quests.

Programming complexity aside, the design space approach boasts great potential in more closely involving story writers in the process of developing generators (see section 2.7). Interpreters that are able to display answer sets that a writer can personally analyse and draw undesirable results from could provide a time efficient method of bringing varied and coherent stories to games.

2.8.1.2 Why the Whodunit?

The whodunit has been a commonly used plot structure used in PCG research, possibly due to its static nature and ability to be presented effectively as a logic puzzle. It has been used in commercial and other games (see section 2.5.1.1), and is currently being covered in research by academics such as Cook and Colton (2015). A common genre covered by quest generation and interactive narrative research is the fantasy RPG (Lima, Feijó and Furtado 2014), but this presents a much more ambiguous plot structure that would be potentially difficult to replicate effectively without the aid of a specialist story writer.

2.8.2 Conclusions

The aim of this literature review was to determine whether applying ASP and the design space approach – as explained by Smith and Mateas (2011) – to the generation of story elements for games will have a positive contribution to research in the area of PCG (see section 2.1.1). Conclusions which can be drawn from the review in regards to that aim are as follows:

2.8.2.1 There is value in generating plots for games

Plot generation for games is not an area that has been widely studied; many research projects focus on either standalone story generation, or the application of interactive narrative and procedurally generated quests to game environments. Plot generation could offer an encapsulating field in which quest and narrative research could draw ideas from in future projects.

2.8.2.2 There is value in further exploring Answer Set Programming as a means to generating creative content

With the likes of *RoleModel* (Chen et al. 2010) (covered in section 1.1), ASP has proven to be an effective method of generating creative content in the form of stories. Drawing from this and applying the paradigm specifically to games could add to current conclusions and work toward a more global game coverage of ASP within the community of PCG research.

2.8.2.3 There is value in exploring the design space approach as a story writing tool for use in game content generators

The design space approach demonstrates a comparatively straight-forward method of developing content generators that is geared towards directly involving designers of games and their components. Reinforcing this as an approach by introducing it to new applications such as story writing will open doors to new procedural generation research using ASP and involving real-world designers, as opposed to veteran academics and AI-oriented programmers.

Chapter 3

Methodology

3.1 Overview

The project implementation is split into three steps: define the logical terms associated with a whodunit plot structure in order to represent the plot within the constraints of a logic program; design and develop an interpreter for answer sets outputted by the generator so that artefacts can be viewed and analysed in a game environment; and refine the design space developed as a part of step one in order to achieve the generation of desirable artefacts according to the plot's requirements (the requirements of a coherent whodunit). These three steps follow Smith and Mateas' approach detailed in Section 2.7. The system created through combining the three steps will subsequently be referred to as The Mystery Machine (TMM).

Section 3.2 addresses project objective 2: Defining plot structures in logic programs. The logical plot structure was built to reflect the whodunit as described in Section 2.3. This structure was chosen because it is a well established structure used by many mystery fiction authors; it represented a plot structure that is has a logical beginning and end by design and thus was deemed a great fit for the project.

Section 3.3 addresses project objective 3: Interpreting logical plot data. The interpreter was built to mimic the scenarios created in the game The Inquisitor, a procedurally generated murder mystery game developed by Brown (2014) for PROCJAM 2014. Scenarios generated in The Inquisitor focus mainly on *means* and *opportunity* (Brown 2016), coinciding with the determined structure of a whodunit (see section 2.3.1). As the creation of an original game was not an objective of this project, it was deemed fitting to create artefacts that could viably be used in an existing game. Complications with data representation – covered in Section 2.4.3 – are successfully resolved and documented here.

Section 3.4 addresses project objective 4: Iteration of logical data to solidify a design space. The design space was solidified by a process of analysing a sample of artefacts generated by each iteration of the logic program through use of the interpreter. The development process for this step is documented here, and covered in section 2.7.1.2.

Figure 3.1 illustrates the structure of TMM as a finished product. In order to achieve the project aim of evaluating Smith and Mateas' design space approach, the development process and product were evaluated based on the factors of expressivity and

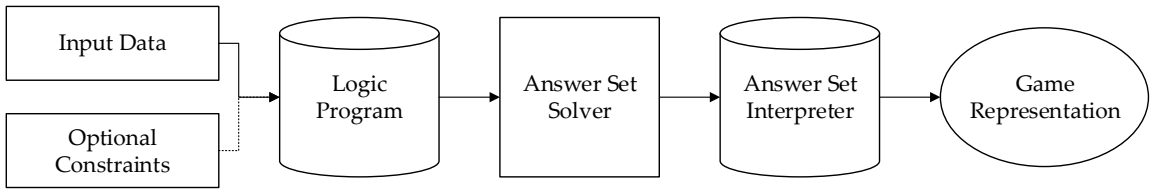


FIGURE 3.1: The aims of this project require the output of data interpretable in a game environment; as such, the structure of the application follows the work of Smith and Mateas (2011) and couples an answer set driven logic program with an interpreter to represent outputted artefacts into data structures that can be displayed in a game.

functionality. The generated plots were evaluated on whether or not they provide sufficient data for a coherent whodunit scenario. Methods of evaluation are covered in Section 3.5.

3.2 Defining the Whodunit in Logical Terms

The whodunit – or murder mystery – is covered in Section 2.3, while the process of defining problems with logical terms is covered in Section 2.7.1.1. This implementation follows the steps described in Section 2.7.1.1.

3.2.1 Tools

The answer set tools used in this project are provided by the Potsdam Answer Set Solving Collection (Potassco) (2008), and include Gringo (the grounder), Clasp (the solver) and Clingo (a combination of the two previous tools). Gringo is a grounder capable of translating user-defined logic programs into equivalent ground programs (ground meaning variable free). Clingo works by piping the output of Gringo into the solver – Clasp – which produces answer sets (Gebser et al. 2015). These tools were deemed ideal for the study by recommendation of Adam Smith (Smith 2016) – whose work this study is based on – and for their adherence to the latest ASP standard (University of Calabria 2012; Gebser et al. 2015). This was viewed as imperative to future work stemming from TMM, as it assured that the code could be re-used.

3.2.2 The Minimal Set of Terms

As per conclusions derived from Section 2.8.2, the minimal set of terms used to describe the whodunit for use in TMM are as follows:

1. A group of rooms to be described logically as simple nodes;
2. The *traversability* of the node map derived from point 1;

3. A number of characters in place to serve as suspects or information brokers;
4. A number of characters derived from point 3 who have a motive to kill;
5. A single character derived from point 4 to be the murderer;
6. A number of potential murder weapons to be presented as evidence;
7. A single murder weapon derived from point 6 that was used to kill;
8. The location of each character at a given point in time;
9. A murder to be initiated when *only* the victim and murderer are present in the same room;
10. The time and location of the murder;
11. A clue as to the murder weapon used to be present on the murder victim;

The above list provides information for five required data structures within the interpreter/game engine: *Character*, *Alibi*, *Weapon*, *Map*, *Room*. The usage of these data structures with regards to how artefacts are parsed into them will be covered in section 3.3.

The initial design space defines each of these minimal terms as a series of facts and choice rules. The weapons, characters, setting, and the traversability of the setting are declared as fact for the scope of this study; future work could involve expanding TMM to create its own setting, as map generation is something that ASP has been commonly used for. What TMM hopes to achieve is the generation of actual murders through character actions. Facts are simple declarations as covered in section 2.6, and look a lot like these:

```
person(miss_scarlett;
    colonel_mustard;
    mrs_white;
    reverend_green;
    mrs_peacock;
    professor_plum).
```

In the above code snippet, the semi-colons are just a syntactical sugar available in Gringo for shorthand declaration of facts such as:

```
person(miss_scarlett).
person(colonel_mustard).
person(mrs_white).
...
```

The setting is represented as a grid; to be imagined as a mansion whereby each adjacent tile (room) is connected by a doorway. This is akin to the generated maps used in The Inquisitor (Brown 2014). Each valid move is declared as a fact, taking two locations from the grid representation; this ensures that when creating *person_at* facts, we don't see people teleporting from one side of the mansion to the other.

The minimal terms state that a number of people will have a motive, and a single person with a motive will be the murderer. On the other hand, a single person without a motive will be the victim. These three rules were declared as such:

```
innocent(P) :- person(P), not guilty(P).
num_motives {motive(P):person(P)} num_motives.
1 {guilty(P) : motive(P)} 1.
1 {victim(P) : person} 1.
:- victim(P), motive(P).
```

The unfamiliar syntax covered above is a method of telling the solver that we only want a given number of results from the provided choice rule. The integer value to either side of the braces in the third rule states that *"only 1 person with a motive will be guilty"*. The third and fourth rules together state that *"only 1 person will be a victim and a victim must not have a motive"*.

To give the generated plot some dynamic properties, the movement of characters is simulated based on their current location, which locations could be reached from their current location in a single move, and the time for each person and location. To achieve this, a fact which holds the properties *person, time, and location* is used. A new *person_at* fact is declared for each value of time; the character can either move or stay put according the facts previously stated for the mansion's traversability. To achieve the movement of characters, the same syntax used in the previous code snippet is employed:

```
1 {person_at(P,T,L) : valid_move(Lp,L)} 1 :-
    person(P), time(T), person_at(P,T-1,Lp).
```

The above choice will produce a single *person_at* fact for each available *valid_move* at a given point in time. This prevents a *person_at* fact from being created for each possible movement a character can make at each time interval, which would produce results such as:

```
person_at(colonel_mustard, 0, foyer).
person_at(colonel_mustard, 1, foyer).
person_at(colonel_mustard, 1, dining_room).
person_at(colonel_mustard, 1, billiard_room).
person_at(colonel_mustard, 1, ballroom).
person_at(colonel_mustard, 1, library).
```

This is obviously undesirable and does not adhere to the initial design space, as we cannot expect that Colonel Mustard will be in more than one place at the same time. Instead, using the choice constraint we see results like:

```
person_at(colonel_mustard, 0, foyer).
person_at(colonel_mustard, 1, foyer).
person_at(colonel_mustard, 2, dining_room).
```

stating something along the lines of: *"Colonel Mustard was in the foyer, he decided to stay in the foyer for a while longer before moving along to the dining room to try out the free buffet."*

Another problem presented by the characters moving about the mansion is the need to ensure that a murder only takes place when both the victim and murderer are alone in a room together as shown below:

```
1 {murder(M,V,T,W,L) :
    guilty(M),
    victim(V),
    person_at(M,T,L),
    person_at(V,T,L),
    murder_method(W)} 1.
```

The first rule here states that: *"exactly one murder shall take place when a guilty character and a victim character are present in the same room and at the same time"*. This works by itself, but it does not enforce that the victim and murderer are the only two people in the room; there could be any number of clear witnesses to the crime. In order to enforce that they are the *only* two characters in the room, the following, unfamiliar syntax can be used:

```
1 {murder(M,V,T,W,L) :
    guilty(M),
    victim(V),
    person_at(M,T,L),
    person_at(V,T,L),
    murder_method(W)} 1 :- room_population(L,T,N); N<3.
room_population(L,T,N) :- N =
    #count{P : person_at(P,T,L)},
    person_at(_,T,L).
```

A body is added to the initial choice rule that enforces a fact will only be considered if the population of the given room is less than three. In order to get the population of a room at a given time, the *room_population* rule creates a fact for each room, at each point in time; for each *person_at* rule that matches the location (L) and time (T), the population (N) is incremented by one. *#count* is a function built into the Gringo language and is intended for uses such as this (Gebser et al. 2008).

3.2.3 Additional Terms

In order to more easily analyse the desirability of generated scenarios, the following additional terms – as covered in section 2.7.1.1 – are also considered when designing the initial generative space:

1. The number of moves taken to commit the murder (the shortest solution);
2. The *"mobility"* of each character;
3. The number of witness events that occur;

The above terms can always be analysed in an answer set output by TMM, although they are not explicit facts or conditions in the logic program. The logic program is implemented so that each move a character makes is assigned a time, and using the interpreter's results of an answer set we can see exactly what each character's moves are, and where and with whom each character was at a given point in time. Using this information, the interpreter is able to determine *"witness events"* where one character witnesses another change rooms. For example, if:

```
person_at(colonel_mustard, 20, hall).
person_at(mrs_white, 20, hall).
person_at(colonel_mustard, 21, hall).
person_at(mrs_white, 21, foyer).
```

Colonel Mustard's alibi at times 20 and 21 are to be interpreted as something along the lines of: *"At 8:00pm I was in the hall with Mrs White. At 8:30pm I was in the hall; I saw Mrs White leave towards the foyer."* These alibi statements are intended for use as evidence as to the murderer; by analysing each character's alibi, the guilty party can be determined by means of deduction. If the murder takes place at 9:00pm in the ballroom and a character witnesses Colonel Mustard moving toward the ballroom at 8:30pm, things would not be looking good for Colonel Mustard.

3.3 Building the Interpreter

The interpreter is a simple Python program that uses Clingo's Python module. This particular module allows Clingo to call a Python script from a commandline interface, providing the script is surrounded with the appropriate tags, as follows:

```
#script (Python).
print("This␣is␣a␣Python␣script␣run␣through␣Clingo.")
#end.
```

This provides us with the means to gather and interpret answer sets through a dedicated library. Processing the answer sets through Clingo's Python module allows

TMM to easily sort through each outputted atom (fact) and sort them into their respective data structures; avoiding the alternative option which is to read and parse an answer set's text form from *stdout* after running vanilla Clingo on the commandline.

Finding the answer sets is achieved through use of a Gringo *Control* object in Python. An object of this type allows TMM to load in an external answer set program, ground it, and then solve it. It also has support for solving multiple answer sets consecutively. The syntax is as follows for solving and sorting through an answer set program with a Gringo Control object:

```
import gringo

ctl = gringo.Control()
ctl.load("the_answer_set_program.lp")
ctl.ground([(base, [])])
ctl.solve(on_model=__on_model)
```

In the above snippet, the *on_model* parameter allows the specification of a callback function which the Control object will call once a model has been determined. The model will be passed as a parameter to the callback function, which allows TMM to iterate through each of the model's individual atoms and sort them accordingly:

```
def __on_model(model):
    for atom in model.atoms(gringo.Model.ATOMS):
        if atom.name() == "person" and len(atom.args()) == 1:
            ...
        if atom.name() == "person_at" and len(atom.args()) == 3:
            ...
```

The data structures themselves are represented as a series of very simple Python classes. These classes and the process with which they are outputted are covered in the next section.

3.3.1 Data Representation

As mentioned in section 2.4.3.1, determining how best to model a generators outputted data is always an integral stage to developing any procedural content generator. Using raw atoms from an answer set is not an ideal solution to presenting a generated plot within a game environment.

A series of simple classes were developed to capture all of the necessary data once an answer set had been gathered. The classes that were used and their function are illustrated via UML in figure 3.2.

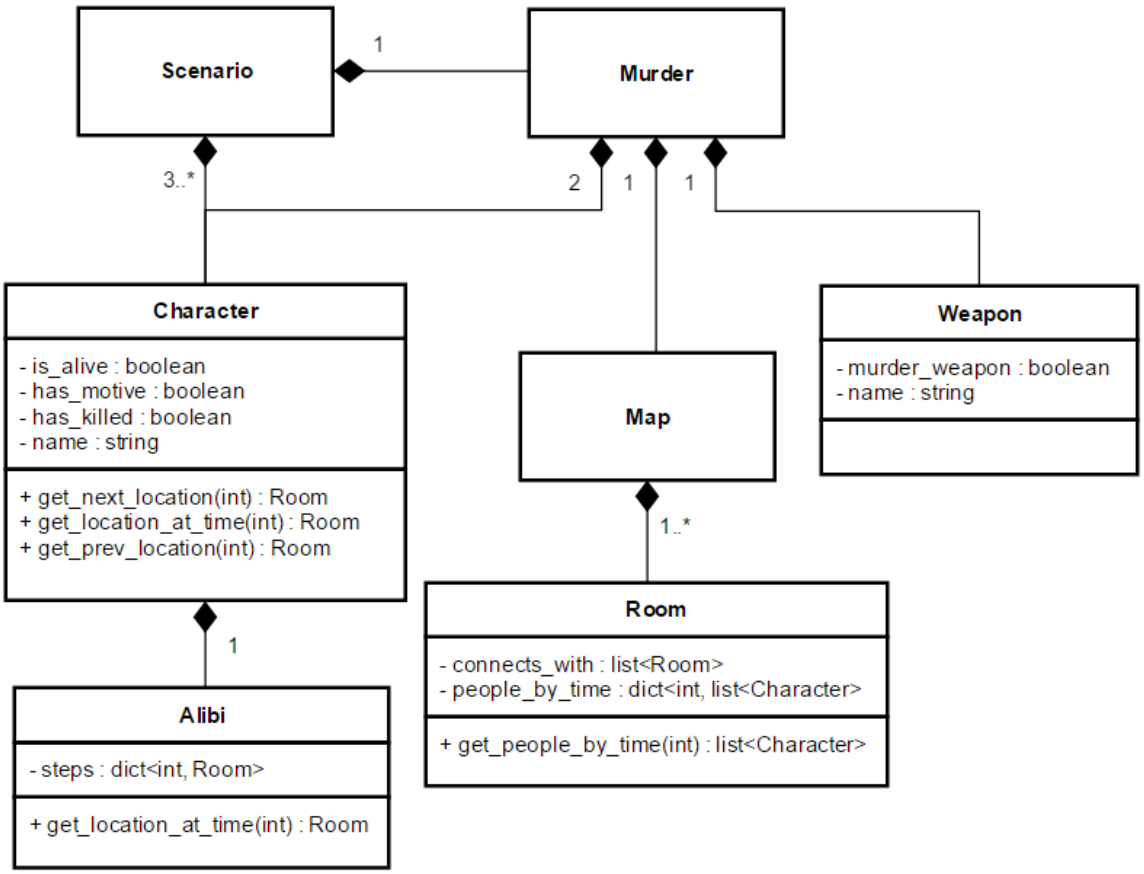


FIGURE 3.2: UML diagram of class structure in the Mystery Machine's interpreter.

These classes are populated by answer sets as follows:

- **Map**: A **Map** object is not directly effected by any atoms, but provides a container for all available **Room** objects, and asserts that no **Room** object is added more than once.
- **Room**: The composition of a **Room** object is dictated by *valid_move* and *person_at* atoms provided by an answer set.
- **Character**: The composition of a **Character** object is dictated by *person*, *motive*, *victim*, *guilty*, and *person_at* atoms provided by an answer set. Important functionality of this class includes the ability to check the character's current location, previous location, and next location, given an integer value for time.
- **Alibi**: The composition of a **Alibi** object is dictated by *person_at* atoms provided by an answer set.

- **Weapon**: The composition of a **Weapon** object is dictated by *weapon* and *murder_method* atoms provided by an answer set.

The classes described above are enough to capture all of the data required by any given answer set and develop it into a plot potentially usable in a game environment. The next step after determining a data structure is to figure out how to represent those data structures to a designer, so that the data can be analysed and the design space refined.

Interpreted data is represented in XML format. XML provides an ideal middle ground between readability for the designer and usability from the perspective of a game. Once an answer set is collected and sorted into the appropriate data structures, the interpreter is designed to sort through the constructed data and compile a *"case file"*. Case files consist of a number of XML data files named after each character; each of these XML files houses the alibi of the named character in the format:

```
<Alibi>
    <Character name="Colonel␣Mustard">
        <Entry time="2000" location="hall">
            <With name="Professor␣Plum" />
            <With name="Mrs␣White" />
            <With name="Miss␣Scarlett" />
            <Left name="Mrs␣Peacock" to="foyer" />
            <Entered name="Reverend␣Green" from="lounge" />
        <\Entry>
        ..
        <Entry time="2230" location="ballroom" />
    </Character>
</Alibi>
```

A final XML file is created by name of *"conclusion"*, which simply holds the scenario's conclusion, written in the pattern:

```
<Murder
    guilty="Colonel␣Mustard"
    victim="Mrs␣White"
    weapon="revolver"
    location="billiard␣room"
    time="2330"
/>
```

The final step after achieving the initial design space and the interpreter is to analyse and refine the design space, as covered in Smith and Mateas' approach (see section 2.7). According to this approach, the bulk of the work has now been done. Provided the initial design space and interpreter are to standard, any additional refinements made will be simple improvements.

3.4 Refining the Generative Space

Using the initial design space and interpreter implementations, a collection of 100 answer sets is gathered. The outputted XML files are examined and undesirable behaviours determined. This begins the process of refinement that makes up the final steps of a design space focussed procedural content generator. Due to the nature of ASP, altering the behaviour of the generator to remove undesirables is often just a matter of either adding or altering a single line in the logic program.

With this implementation, results from the initial design space note that more than one murder can occur in each answer set. Unfortunately in these cases the murderer and victim are always the same two people; not only are the characters having to deal with the trauma of a murder, they are having to deal with the trauma of seeing their friend murdered multiple times as he repeatedly returns to life as a zombie. This particular issue is solved with the following statements:

```
completed_at(T) :- murder(M,V,T,W,L).
complete :- completed_at(T).
:- completed_at(T), murder(_,_,T2,_,_), T2 > T.
```

The first line in this snippet creates a new atom `completed_at(T)` whenever a murder atom is produced, where the value `T` corresponds to the value `T` in the `murder(M,V,T,W,L)` atom. The final line introduces an integrity constraint that states *"No murder shall occur where the value T2 is greater than the value T in an existing `completed_at(T)` atom."*

The design space is further refined to ensure that we don't continue to develop a character's alibi once the murder has been completed, which involves one more integrity constraint:

```
:- person_at(P,T,L), completed_at(T2), T > T2.
```

The refinement process continues as such until the design space contains only desirable results (see section 2.7.1.2). To help with this process, another simple Python script is built that works through all of the answer sets of each iteration and compiles an XML document of statistics. This script keeps track of each murderer, each victim, the average time taken for a murderer to kill, the average grounding time, the average solving time, the number of times a murder took place in each room, the popularity of each murder weapon, etc. From this point in the study, each iteration of this process will be referred to in the format *refinement iteration #x*; the initial design space will still be referred to as such.

Each time a change is made to the logic program, it is run until another 100 answer sets are generated, and the process begins again. Instead of adding multiple changes at the same time it, changes are made individually so as to better represent the refinement process in chapter 4. This also ensures that changes remove more undesirable artefacts than they create, and essentially makes debugging much simpler.

3.5 Analysing the Mystery Machine

A common method of analysis for procedural content generators – particularly map generators – is determining expressive range (Smith and Whitehead 2010; Lavender and Thompson 2010); this approach typically focuses on evaluating a generator in respects to: constraints inherent to the generation algorithm, and analysis of the range of content produced. It was decided not to take this route with TMM, since artefacts generated by an answer set program will always adhere to the logical rules and constraints set out by the designer. Hence, the constraints are at the whim of the designer during the refinement step, and the range of content produced can always be determined exactly by the problem's state space.

In order to analyse TMM's success, the data collected in the Python test script is studied for each iteration. The success is determined on each iteration by the data's coherence to a set of rules which need to be present in order for a generated plot to be defined as a whodunit. If all of these rules are met, then the success of the generator's output is self evident.

To analyse the success of applying a design space approach and ASP to creating generators for plots, the development process itself is analysed by use of the data outputted by the test script. This data will be plotted to visualise how the generator evolves over each iteration of refinement.

Chapter 4

Results, Analysis and Discussion

4.1 Overview

The aim of the project was to evaluate Smith and Mateas' design space approach to developing procedural content generators when applied to generating story plots for games. This is achieved by considering:

1. The representation of a whodunit in data form;
2. The expressive range of the data generated by TMM;
3. The impact that the design space iterations had on the development process and resulting plots;

Section 4.2 analyses how successful the data generated by TMM is in representing the logical definition and required steps of a whodunit, discussing ASP's potential in the development of procedural plot generation. Section 4.3 extends this analysis onto the expressive range of TMM as an ASP powered content generator, evaluating the spread of data across the generator's state space. These two sections together cover objective 5.

Section 4.4 focuses on analysing TMM's implementation of Smith and Mateas' design space approach and its effect on the development process, thus covering objective 6.

It should be noted early on that gathering meaningful data for analysis has been of significant difficulty in this project. The interpreter was built to gather statistics on murders generated; these can be seen in Appendix A. However, these statistics were deemed irrelevant for the product's analysis due to its logical nature. The products expressive capabilities can be proven by means of a mathematical approach, and as it turns out it is nigh impossible to analyse the success of a story plot by means of studying solve times plotted on a graph.

4.2 Plot Representation Analysis

The coherence of a generated plot is measured by its ability to replicate the three core features of a whodunit as defined in section 2.3.1: means, motive and opportunity. The initial design space of TMM was built to honour these components (as covered in

section 3.2.2), and therefore represent the structure of a whodunit in its output data. The binary nature of logic programs – and by extension, answer set programs – dictates that the success of a representation will either be true or false. Table 4.1 shows each iteration's success in implementing the core features; the final column *"Clarity"* states whether generated plots provided enough evidence that pointed towards the true murderer.

Iteration	Means	Motive	Opportunity	Clarity
Initial	yes	yes	yes	yes
1	yes	yes	yes	yes
2	yes	yes	yes	yes
3	yes	yes	yes	yes
4	yes	yes	yes	yes

TABLE 4.1: This table provides a check-list of the necessary features of a whodunit as determined in section 2.3.1, and each refinement iteration's success in including the features. The very nature of ASP and the design space approach dictates that, given a sufficient initial design space, each iteration of refinement is bound to follow the same base set of rules; thus when the initial design space matches all of the required criteria, it is highly unlikely that subsequent iterations will not.

The values in table 4.1 were determined by analysis of the factual rules stated in TMM answer set program component and the inclusion of the relevant data in the outputted data files. The given definitions and the logical representations for each of the features are explained in the following sub-sections; the combination of each of these factors provides justification for the values plotted in table 4.1.

4.2.1 Determining Means

TMM treats means (the feature) as a character's access to several potential methods of murder. These methods of murder are represented by a number of `weapon(W)` facts throughout the logical representation of the mansion setting. Murder methods (weapons) were chosen as the representation of means by influence of the popular and established murder/mystery-themed board-game Cluedo (Pratt 1949), alongside the similarly structured computer game, The Inquisitor (Brown 2014).

The successful implementation of the means feature is determined by the presence of a number of `weapon(W)` facts in conjunction with a single `murder(_,_,W,_,_)` atom in any generated answer set. We can be absolutely sure that both of these conditions are met in each answer set, thanks to the following code:

```
weapon(revolver; candlestick; dagger; lead_pipe; rope; spanner;
poison).
..
1 {murder_method(W):weapon(W)} 1.
..
1 {murder(M,V,T,W,L) :
    guilty(M),
    victim(V),
    person_at(M,T,L),
    person_at(V,T,L),
    murder_method(W)} 1
    :- room_population(L,T,N); N<3.
```

4.2.2 Determining Motive

As covered in section 2.3.1, the explicit inclusion of a written motive was deemed out of scope for the purpose of this study. In mystery fiction, a character's motive is portrayed to the reader through *narrative*. Therefore it is sufficient, logically speaking, to simply state that a character *has* a motive in the construction of a plot, without explicitly defining what that motive *is*. This is not the same as foregoing the explicit inclusion of means or opportunity, because both of those features *are* things that need to be logically defined in order for a plot to be justifiable and coherent. To provide an example: to justify the presence of several lacerations on a victim's corpse, there logically *has* to be something sharp in the area that could cause such lacerations; to justify the opportunity a murderer had to strike, we *have* to know that there was a point in time where he could commit the crime with no witnesses; to justify a motive, we need only know it to be true that a character has that motive. Exactly *what* a motive is does not alter anything in the logical structure of a plot.

The successful implementation of the motive feature is determined by the presence of any number of `motive(P)` atoms in a generated answer set. The following code assures that at five of these atoms will be present in any answer set:

```
#const num_players=5.
..
num_players {motive(P):person(P)} num_players.
```

4.2.3 Determining Opportunity

Opportunity presents itself in TMM as a point in time where a `guilty(P)` and a `victim(P)` atom are the only two atoms with an associated `person_at(P,T,L)` atom, where the two respective values for `T` and `L` match. This implies that the two

characters are alone in a room. An associated `room_population(L,T,N)` atom with an N value of less than 3 assures that the murder has no witnesses.

The successful implementation of the opportunity feature is determined by the inclusion of a single `murder(M,V,T,W,L)` atom in any given answer set. The following code assures the inclusion of exactly one of these atoms in any answer set:

```
1 {person_at(P,T,L):valid_move(Lp, L)} 1
    :- person(P),
    time(T),
    person_at(P, T-1, Lp),
    not completed_at(T..t_max).
..
1 {murder(M,V,T,W,L) :
    guilty(M),
    victim(V),
    person_at(M,T,L),
    person_at(V,T,L),
    murder_method(W)} 1
    :- room_population(L,T,N); N<3.
```

4.2.4 Determining Clarity

Clarity refers to the certainty of a generated answer set. It is provided as an integral feature which ensures that it is *only* possible, given the provided evidence, that one character has committed the murder. Clarity is required to ensure that a whodunit as a whole is justifiable and adheres to *"the science of deduction"* (Doyle 2010). To quote Sherlock Holmes in *The Sign of the Four* (Doyle 1890):

> How often have I said that when you have excluded the impossible whatever remains, however improbable, must be the truth.

Clarity is achieved by means of supplying each character in each answer set a comprehensive alibi. Each set of `person_at(P,T,L)` atoms associated with a character is compiled into data that details their location at each point in time, as well as who they were with, and who they witnessed leave and enter their location. The presence of a full set of these atoms for each character is assured by the following code:

```
1 {person_at(P,T,L):valid_move(Lp, L)} 1
    :- person(P),
    time(T),
    person_at(P, T-1, Lp),
    not completed_at(T..t_max).
```

4.2.5 Perceived Strengths and Weaknesses of Plot Representation

Although the implementation succeeded in its representation of the whodunit plot structure, there are several clear areas in which it could be improved. Logically speaking, the plots generated by TMM are sound, but stories and plot cannot be evaluated purely in logical terms; they are creative works. This is considered a clear weakness of the project as opposed to the implementation, which could have been solved by considering human evaluation method (qualitative or quantitative) of several samples of the generated data. It would also require the data to be more formally presented, by means of a written story or simple game.

The generated data changes with each answer set, meaning that each plot will be different but still follow the same structure; further improvements that could be made include: generate the setting rather than have it set as fact, allow for additional or less characters whose names and backgrounds could be determined by the interpreter, and implement a system in the likes of `person_at(P,T,L)` atoms for weapons; with this a murderer would have to acquire the weapon in secret before committing the murder.

4.3 Product Expressivity Evaluation

4.3.1 Method of Evaluation

In order to gauge the expressivity of an answer set generator, we only need to determine the problem's state space. The expressivity of an answer set program is determined by this alone, similarly to how the expressivity of a logic program is dictated by whether a problem is deemed true or false. To determine the number of possible states (answer sets) we look at each choice rule in the final iteration of TMM and combine them into a final number. Determining the number of possible permutations for the values in a `murder(M,V,T,W,L)` atom, and the (approximate) possible number of combinations of a collection of `person_at(P,T,L)` atoms, and then finally multiplying the two answers together will give us a reliable approximation of the complete state space of TMM.

4.3.2 Murder Outcome Permutations

In TMM, there are 6 characters involved in each answer set; these are facts. Of these 6, a single character will become the victim, and another the murderer. The murderer and victim will not be the same character, so once a murderer has been determined, the victim will be from a choice of 5. Given the formula for the number of permutations of a set (figure 4.1), we can figure out the number of permutations for each required value:

$$ {}_nP_k = \frac{n!}{(n-k)!} = n(n-1)(n-2)...(n-k+1) \tag{4.1} $$

Using this – where n is the set and k is the number of required inversions – we can apply n the value of 6 (a set of 6 people) and k the value of 2 (2 people required from the set each time) to determine the state space associated with the selection of characters. Murder method and murder location are much simpler to figure out, since k would have the value 1 for each of these, meaning that the value for n would be our state space; 7 and 10 respectively. Murder time is just as simple, though a little different. This implementation has a maximum of 25 time values (treated like turns). Integrity constraints enforce that the murder always takes place between turn 10 and turn 25; this gives n the value 15.

All of the determined values are mutually exclusive; i.e. given the same value for murderer, victim, murder method and murder location, the murder time *must* be different in order for it to be a separate permutation. Therefore, the number of permutations for murder outcomes can be expressed by: $30 \times 7 \times 10 \times 15 = 31500$.

4.3.3 Approximate Alibi Permutations

A key aspect of TMM is how it executes murders according to logic, ensuring that a murder can always be solved. The alibis generated for each character are a core component of the data generated; given two identical values for a `murder(M,V,T,W,L)` atom, it is highly probable that our character's alibis differ.

A character's alibi is determined by each `person_at(P,T,L)` atom associated with them. With each value for time, a new atom of this type is generated based on the choice:

```
1 {person_at(P,T,L):valid_move(Lp, L)} 1...
```

This rule generates a single atom for time T from the choice of `valid_move(Lp,L)` atoms associated with location L as its second parameter. Since in TMM the map is always the same, we can use this to determine an approximate value for the number of combinations this will give us.

The number of valid moves is determined by the room the character is currently in; the kitchen only connects to the ballroom and dining room; the foyer connects to the hall, ballroom, dining room, billiard room, and library. Remaining stationary is always considered a valid move. With this information we see that the number of possible states of this choice is between 3 and 6. We can approximate the actual state space by taking an average of the values from each room.

The mode and median of the states combined gives us on average 4 possible states in each room. Given that one atom is generated for each point in time (each turn), and it takes between 10 and 25 turns for a murder to be committed, we'll work with the median/mean number of turns: 17.5. With 6 characters and 17.5 turns, this gives us the average total number of atoms produced: $6 \times 17.5 = 105$. The total number of possible states can be gauged by the formula $n = x^y$, where x is the number of atoms produced and y is the number of possible states per atom; totalling $1.215\,51 \times 10^8$.

4.3.4 Total Approximate Range of Expression

Given the results gathered from sections 4.3.2 and 4.3.3, we can now approximate the total possible number of plots that can be generated by TMM, giving an idea as to its expressive range.

Since the murder and the alibi are mutually exclusive, this approximation can be achieved by the formula $n = x \times y$, where x and y are the number of murder permutations and number of alibi combinations respectively. Therefore the final equation is: $31500 \times (1.215\,51 \times 10^{8}) = 3.828\,84 \times 10^{12}$.

From the above equation we can derive that there are *approximately* $3.828\,84 \times 10^{12}$ possible scenarios that TMM can generate. Of course the actual number would vary based on other specific constraints such as the need for a murder and victim to be alone in the same room when the killing blow is struck; however, this approximation gives us a reliable insight into the expressive range of the generator.

The seed provided to Clingo that determines random decisions is supplied by Python's *random* module; all random numbers generated using this method are based on the system time (Python Software Foundation 2016). This should be taken into account; the program might generate identical results if two or more copies are run in the same millisecond.

4.4 Process Evaluation and Discussion

This section evaluates the success the project process and the application of Smith and Mateas' design space approach to building procedural content generators. Section 4.4.1 focuses on how well the methodology was carried out, taking into account both successful and unsuccessful decisions. Section 4.4.2 covers the impact the design space approach had on the development process.

Room	**Number of States**
Kitchen	3
Conservatory	3
Study	3
Lounge	3
Hall	4
Dining Room	4
Ballroom	4
Billiard Room	4
Library	4
Foyer	6

TABLE 4.2: Possible states associated with each room in the mansion.

4.4.1 Success of Methodology/Implementation

4.4.1.1 Defining a Logically Sound Plot

Possibly down to the specific plot structure chosen for this project – the whodunit – defining the plot structure as logical terms was a relatively painless process. The key features of the whodunit are determined in section 2.3.1, and translating those features into data was simple. How they were represented in the data is covered in section 3.2.2. The defined terms provided a solid starting point for applying design space refinement. Early answer sets generated by this initial design space adhered to the rules that were set out and provided a variety of plots, demonstrating success in defining a plot structure as a logic program. It would be interesting to see how this process translates to a more complex plot structure.

4.4.1.2 Working with Clingo

Clingo turned out to be a wise choice thanks to comprehensive documentation provided by Potassco (Gebser et al. 2015). Installing the software initially took some work; The tools provided by Potassco are written specifically for Linux platforms, although they note support for Windows; originally the attempt was made to compile Clingo for Windows and make use of the C++ integration new to version 4.5 (Potassco 2015), but this proved to be an arduous task. While the initial idea was to utilise Windows (and C++), with it being a more widely used platform for games, it was decided that the project would best be suited to Clingo's native environment. After setting up a Linux (Ubuntu) virtual machine, the installation process was comparatively easy. Python was also chosen as a more appropriate language for which to build the interpreter, given its established relationship within Clingo (Potassco 2015), when compared to the new and undocumented C++ support. Making this informed decision earlier on would have significantly reduced the development time of the product.

4.4.1.3 The Interpreter

As covered in section 3.2.2, the data was designed around the existing game, The Inquisitor (Brown 2014). Following suggestions made by Smith and Mateas, the interpreter was built alongside the initial design space as covered in section 4.4.1.1. This proved fruitful, as both influenced each other. It is assumable that defining the logical plot would have been much more difficult if it was not clear how this data would be displayed; using The Inquisitor as a basis allowed the plot to be built around an existing skeleton structure whilst ensuring it held true to the required whodunit features.

The main weakness of the interpreter implementation was its inability to gather data that corresponded to the aim and hypothesis. This is covered fully in section 4.4.1.5.

4.4.1.4 The Refinement Process

The decision to keep refinements small proved successful in the implementation. While it is highly likely – and indeed true for this project – that iterations from a single refinement can reveal a number of clearly undesirable results, taking each problem on one at a time helped to ensure that refinements were actually improving the results. Figure 4.2 illustrates an example of how refinements can adversely effect answer sets. The graph plots the murder times (in narrative minutes) of 100 answer sets of refinement iteration #2. We're after variety in a procedural content generator; as we can see, in comparison to the graph in Figure 4.1, refinement #2 was counter-productive and required fixing before it was safe to make further refinements. This fix was implemented in refinement iteration #3, so the process could continue in refinement iteration #4.

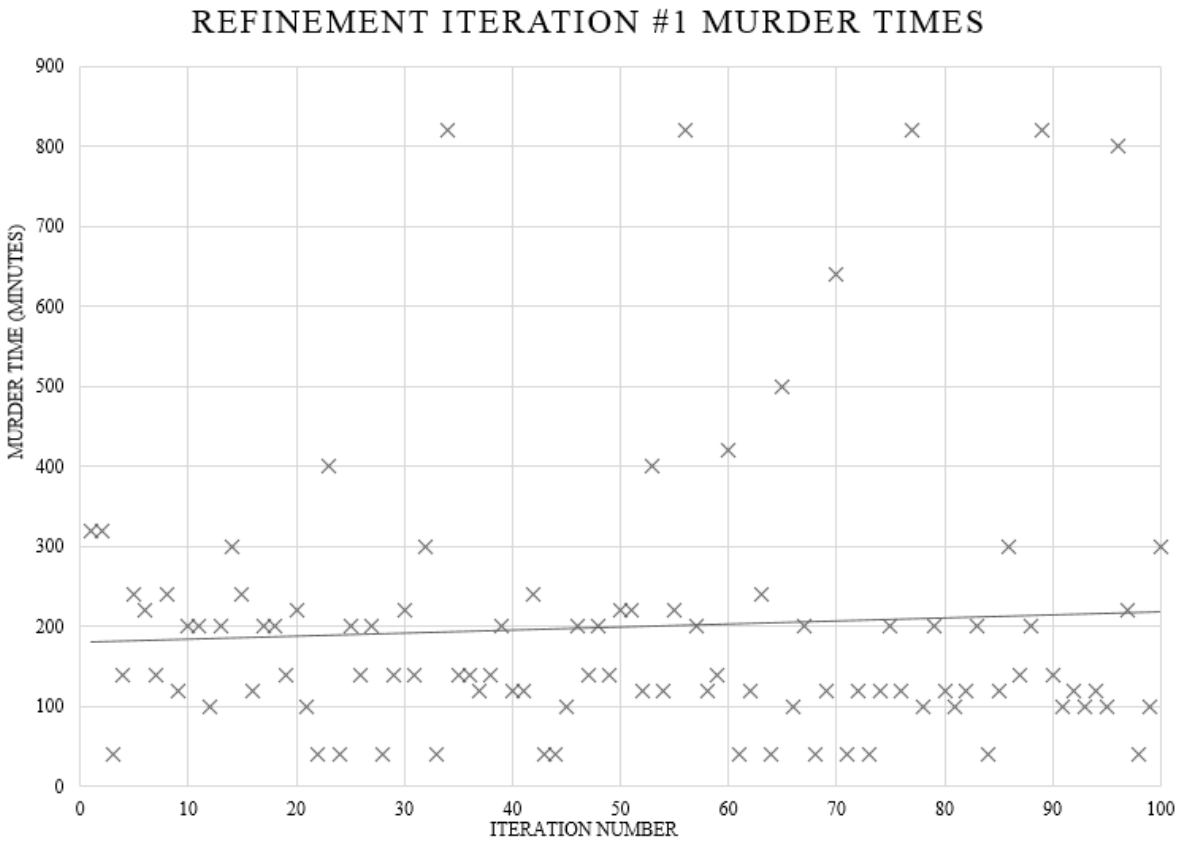


FIGURE 4.1: Graph showing the spread of murder times in case files within refinement iteration #1.

The fact that only a single change happened on each refinement made it extremely easy to determine the cause for a change in resulting answer sets. If all of the clear undesirables were covered in a single refinement, debugging the results would have potentially been made much more difficult.

By definition of Smith and Mateas (2011), the refinement process should go on until a satisfactory product is achieved. The product developed as a result of this project

FIGURE 4.2: Graph showing the spread of murder times in case files within refinement iteration #2.

undertaking has achieved the goals we set out for (objectives 2, 3 and 4) in as little as four refinement iterations, not including the initial design space. This can likely be put down to the success/solidity of the initial design space. Refinements themselves proved to make a real impact on the resulting answer sets, as the state space changed to match them. Despite this, there are still several refinements that could be made to improve the resulting answer sets; the only thing stopping them from being introduced is time constraints. More on this will be covered in section 5.4.

4.4.1.5 Product Evaluation

Gathering meaningful data from the product proved to be rather unsuccessful. Other than the mathematical/logical analysis undertaken in this chapter, there is little available information given by the interpreter that is constructive toward proving the applicability of the design space approach – or answer set programming – to generating plots in comparison to other methods. A core failure/oversight of the project is that the success of a plot is determined strictly by math and logic, where the very nature of a good plot is how it connects creatively with the reader or user. It would have helped immensely in the analysis of the product if a playable game or demo was available to

Group	Murder Time	Solve Time
Mean	482.33333333	14.09833998
SD	261.14980303	9.22740311
SEM	15.07749091	0.53274437
N	300	300

TABLE 4.3: Unpaired t-test results from comparing murder and solve times from multiple refinement iterations. The two-tailed P value is less than 0.0001; by conventional criteria, this difference is considered to be extremely statistically significant.

test the plots qualitatively with feedback from actual players. The lack of a full game is mainly down to time constraints; it was decided early on that the time required to construct the generator and a functional game to go with it would be too vast for the scope of the project.

4.4.2 Development with the Design Space Approach

The biggest perceived benefit of the design space approach, as stated earlier on in section 2.6.2.1, is that it can provide a reliable platform for designers and story writers to become more involved in the actual development of a game's content generator(s). The project has shown promising results toward this goal. The following paragraphs demonstrate how.

Table 4.3 shows the result of a t-test that directly demonstrates a notable correlation between the generator's solve time and time taken for a character to commit a murder. Values from the initial design space and refinement iteration #2 were excluded from this test as they were deemed clear outliers. The initial design space could commit multiple murders per answer set, and refinement iteration #2 enforced that a murder always take place at the last moment, meaning that a considerable area of the problem space was removed.

The correlation shows how solve times are directly affected by the state space the solver explores; earlier murder times explore less state space because once the `complete` atom is generated, nothing else is required. This demonstrates a considerable power that the designer has over the generator in comparison to conventional generators. The end state space is determined solely by the designer in this product, whereas conventional methods – particularly search-based as covered by Togelius et al. (2010) (see section 2.4.3.2) – would rely on fitness functions to determine the viable state space at run time. To reiterate a point made in section 2.6.1.1, in refining an answer set program's design space, we stop undesirable artefacts from ever being generated (Smith and Mateas 2011); in conventional methods, these artefacts can still be generated, but will get discarded afterwards.

The solve times associated with this data directly correspond to the design space that the designer has achieved, and show this direct impact in cement without having to consider re-runs potentially made by other generator's fitness functions.

4.5 Discussion and Proof of Hypothesis

The logical analysis of generated plots can be seen as successful as it proves the inclusion of all of the required features of the plot structure, as set out the the Literature Review and Methodology. The plot structure's representation has been explained and justified using existing materials and companion code that proves each feature's inclusion.

The expressivity evaluation can be seen as successful to a certain degree. It proved difficult to evaluate given the lack of documented methods of reviewing an answer set generator's creative range, but proving TMM's vast possible state space was a step in the right direction. Given the calculated approximate state space acquired in section 4.3, it would have been impossible to gather a reliable enough sample of data to express these findings with raw data, mainly due to time constraints; 1000 or even 10'000 samples would not have even made a dent in the number of possible permutations available.

The project hypothesis claimed that using the design space approach and ASP, put forward by Smith and Mateas (2011), would provide an effective means of generating both varied and coherent plots; given the evaluation undertaken in section 4.2, we can confidently say that the representation of plot data in logical format has proven successful. We can also see success in generating variation between plots using the same structure is clear given the state space analysis undertaken in section 4.3. However, we cannot speak for the coherency of generated plots in a conclusive manner without undertaking a qualitative study and exposing them to human interaction. Though the logical analysis points towards success, it is imperative to consider the creative nature of stories; the success of a story cannot be determined by logical analysis alone. A new project which addresses the question of whether generated plots prove successful when exposed to players through a game would provide a good follow-up to this one.

The hypothesis focussed on the specific application of a design space approach to the development of the product. Given the nature of ASP, this approach proved largely successful; each refinement made a clear impact on the achieved results. Through the evaluation it is clear how adopting this approach in the development of future procedural content generators could be beneficial. The control the approach offers to designers of generators has been covered, alongside the possible benefits of reduced solving times due to lack of required fitness functions.

Chapter 5

Conclusions and Recommendations

5.1 Success of Aim, Objectives and Hypothesis

This section revisits the criteria of the objectives, hypothesis, and aim, which were formulated early in the project's execution and are identified in section 1; each criteria is assessed on whether of not it has been met over the course of the work undertaken.

5.1.1 Objectives

The original project objectives are detailed in section 1.5.

1. The Literature Review consisted of a thorough background of the use of both narrative and procedural content generation in games, an investigation into the plot structure of a typical whodunit novel, a study of issues typically inherent in well-versed PCG approaches, an overview defining the meaning behind "emergent games" and issues typically involved with these areas of research, and a comprehensive background and explanation of the paradigm of ASP.

2. An ASP powered content generator that defines and generates plot data – compiled as a collection of logical terms – was successfully implemented. The development process was simultaneously documented, but not in any immense depth.

3. An interpreter was successfully developed, allowing the processing of atoms generated by the above mentioned content generator. Generated content was successfully parsed and represented in XML, a common data format used in the development and running of games.

4. The refinement of the answer set generator was executed according to the approach detailed by Smith and Mateas, successfully demonstrating the power the design space approach gives to the designer of a game's plot/content generator(s).

5. The success of the implementation was evaluated in technical detail, proving the stated hypothesis in section 1.6. The evaluation was not undertaken strictly using data generated from the implementation but included relevant data. Weaknesses were identified and potential methods of resolving these issues were presented.

6. The development process was evaluated from the perspective of a designer creating plots through code. Weaknesses and strengths of the design space approach were acknowledged.

By demonstration of the above aspects covered in this study, we can establish that all objectives were successfully covered.

5.1.2 Aim

As originally presented in section 1.5, the aim of this project was:

> *To evaluate Smith and Mateas' design space approach to creating procedural content generators when applied to the generation of coherent plots in two respects: effect on the development process and the expressive range of resulting plots.*

In order to evaluate the design space approach with regards to plot generation, several steps of implementation first had to be undertaken. These steps are covered in section 5.1.1. An investigation was then carried out on the expressive capabilities of the achieved product by means of logical analysis. Following this investigation, the effect of the design space approach on the product's development process was evaluated by means of assessing statistics from the product refinement stage detailed in section 3.4.

Implications on the development process are covered in section 4.4. Information was gathered on the process throughout implementation and reflected throughout section 3. Weaknesses and strengths of the process as a whole are documented here and an analysis of the larger process is given. Section 4.4.2 covers the specific implementations of applying the design space approach to the development of the design space approach, namely the benefits it provides over the development of conventional generators. The process proved to provide much more control over the development process, splitting what can typically be a long a complicated process into manageable smaller refinement steps. ASP has proven to be an invaluable tool in the development of content generators due to its logical nature and ability to represent problems and design spaces in a completely literal format. The implementation has shown that ASP provides a viable means to create generators that provide a comprehensive representation of design spaces *and* largely varied results, without the need to program complicated fitness functions or genetic algorithms. The biggest limitation listed is TMM's lack of a data for use in qualitative testing, which is a problem specific to TMM, and not the overall process. As mentioned previously, applying TMM to a game would help to eliminated this weakness by allowing us to present the stories to users.

The expressive range – or expressivity – of the product is assessed in section 4.3 by use of logical/mathematical means. The issues of evaluating the expressive range of an answer set generator by conventional means is also covered here. Other issues are separated between those inherent with answer set programs and those specific to TMM. Raw data gathered from the generator which was deemed unfit for purpose in the analysis is provided in Appendix A; this data simply illustrates the ability of the generator to provide multiple answer sets.

Both aspects of the aim were evaluated thoroughly, and implications were drawn from them respectively, meaning the aim of the project was completed with success.

5.1.3 Hypothesis

As originally presented in section 1.6, the hypothesis for this project was:

> *Applying a design-space approach to creating procedural content generators – using Answer Set Programming as presented by Adam Smith and Michael Mateas – will provide an effective means of generating both varied and coherent plots for use in games.*

The design space approach was successfully applied to an implementation which covers the generation of story plot data which corresponds to the definition of a whodunit. Evidence of this is provided in section 5.1.1 and in the implementation itself.

An answer set program has been successfully developed using the proposed techniques and approach. A companion interpreter has also been developed – covered in section 3.3 – which is capable of parsing resulting answer sets into comprehensive game data structures. The data outputted from the interpreter was successfully analysed in order to achieve the final requirement of the design space approach: the refinement step. The refinement step proved an effective means of both debugging and realising the desired final development of the initial answer set program. The development process has been covered in detail throughout section 3, and the success of the resulting application is evaluated thoroughly throughout section 4.

All plots generated by the developed application have been proven logically to work as desired, and the expressivity has been analysed with mathematical evidence that proves their variety with regards to their data.

Therefore we can successfully conclude that the application of ASP and the design space approach to plot generation was able to provide an effective means of achieving the desired outcome: creating a varied array of plots that follow a coherent structure. However, we cannot conclude that these plots can be portrayed effectively to players with regards to how they reflect creatively and qualitatively. Despite this, evidence has been provided towards this end in section 4.5, and potential means of clearly proving this have been covered in sections 4.2.5, 4.5 and 5.4.2 respectively.

5.2 Limitations

This section details the limitations inherent in the project's coverage. Section 5.2.1 addresses the lack of involvement of a designer or story writer in the development process; the potential short-comings of an ASP generator implementation made using the design space approach in a real life scenario, while section 5.2.5 covers the limitations specific to the implementation of TMM.

5.2.1 Lack of a Designer/Story Writer

The implementation covered in this project is lacking the input of an actual designer or story writer. It is likely that real-world applications of this technique would require some middle ground between the designer, answer set program, and a programmer; alternatively we would need to expect designer to become proficient in the paradigm of ASP.

5.2.2 Lack of Resources for Evaluation

The lack of a plot designer referred to in section 5.2.1 also had significant impact on the aspects of TMM which could be evaluated and how. Without involving such designers in the development of the answer set program, the approach cannot be realistically evaluated on aspects such as ease of use, success of plot representation etc.

There is also the factor of lack of documentation of evaluation procedures for answer set programs. This project covers the evaluation as best it could, given a lack of established methods, by using mathematical means. Due to lack of evaluation procedures, the resulting interpreter fell short on delivering data that could constructively be used for analysis.

5.2.3 Platform Compatibility

This implementation was build for Linux operating systems exclusively. Although theoretically the Clingo toolset with Python support can be built for other operating systems this represents an issue with the project that cannot realistically be ignored. Since the application of this project is geared towards use in games, the implementation should ideally be available on at least Windows.

Current public releases of Clingo on the Windows platform come without built support for Python, but with support for Lua. Perhaps with this in mind it would have been a much better and more applicable choice to have developed the interpreter as a Lua script.

5.2.4 Simplicity of the Interpreter

There is currently a lot of room for improvement within TMM's answer set interpreter. Currently it only covers the bare minimum, directly translating answer sets into data. However, if the system were to be applied to a game, or forwarded for qualitative testing, this could require some important additions or a complete overhaul. A good story may sometimes require that certain information is kept from the player, but in the interpreter's current state, this is not an option. There would need to be some sort of system in place that can determine when to alter or *hide* certain data before TMM could realistically be applied to a game. For example, the implementation of the interpreter in this project gives full alibis for each character, including the murderer. This is obviously undesirable as in a game environment it would essentially mean the murderer gives himself/herself up each time a plot is generated.

5.2.5 Specific Implementation

As is the nature of ASP, the implementation is very specific in the nature of the problem that it is designed to solve, and the games it could potentially be applied to. In order to apply a similar system to other plot structures/games, a vast overhaul of the implementation would be required for each specific application. It has also been covered that the current implementation does not directly demonstrate the generated content's use in a game environment; in order to further prove the applicability of a system of this type, it is recommended that any further work stemming from this project tests the application of answer set data as such (see section 5.4.2).

5.3 Contributions

5.3.1 A Practical Demonstration of Novel Applications to Answer Set Programming in Games

At the time of writing, ASP has had comparatively little involvement in the world of games-focused AI research. TMM demonstrates its potential in the field with a practical application that could realistically be applied to larger games in place of favoured methods such as constructive (Mossmouth 2012; Macmillan and Himsl 2011; Brown 2014) and search-based (Togelius et al. 2010) procedural content generators.

Conventional research into ASP has been more focussed on as an area of interest amongst logic-oriented AI researchers (Smith 2016). However, it has fairly recently begun creeping its way into the world of games with works by the likes of Smith and Mateas (2011), Thielscher (2009), and others; this study – given its demonstration – provides further argument for solidifying its relevance in application to games.

5.3.2 A Method of Defining Plots in Games

The implementation of TMM successfully demonstrates how creative works such as stories and plots can be represented in the logical confines of a game. TMM shows what can be achieved when we think of a plot as a structure rather that a solid piece of creative writing by successfully demonstrating the creation of data that corresponds to these structures. Furthermore, generated plots has been proven – though not yet qualitatively – to follow the defined structure without fail. Though we cannot yet prove the creative success of generated plots, this is certainly a step in the right direction.

5.3.3 Demonstration of a Method which Deals With Murder-Mystery Generation

The Murder/Mystery genres have been applied to games before (Glass Knuckle Games 2014a; Brown 2014), but with simple template based approaches that often exhibit issues with repetition and expressive range. This project has demonstrated an alternative method of generating plots for the genre that could be applied to future games

of a similar nature. It provides evidence supporting the use of ASP in the generation of these plots and provides a proven method of development that paves the way for similar work that could either expand the scope of TMM or be used to solve other problems with plot generation.

Within a larger scope, the project proves ASP to have great potential in the development of generators for plots of other genres, demonstrating that if they can be defined in logical steps, they can probably be represented as logic programs (as covered in section 5.3.2).

5.4 Future Work

5.4.1 Further Refine the Mystery Machine's Design Space

The development of TMM was completed in as little as four refinement steps following the initial design space implementation. It has been brought to the foreground in section 4.5 that there is significant room in the project for further refinements to be made toward the goal of improving the generated plots. There is room within these potential refinements to alter/re-implement the given interpreter to include new means of supplying evidence, and the selective exclusion of certain alibi points. Weaknesses of the current implementation in this regard are covered in section 5.2.4.

This would contribute to research by means of further proving the viability of the use of ASP and the design space approach for plot generation, by providing evidence of an even more comprehensive generated plot.

5.4.2 Develop a Game Around the Mystery Machine

One of the key limitations of this project – as covered in section 5.2.5 – is that the data is not directly represented in an actual game environment. Building a game around TMM – potentially in the likes of The Inquisitor (Brown 2014) – would help to prove the viability of the proposed methods in a creative and qualitative environment. The idea of generating plots for games is a novel one, and though proven logically in this project, proof provided by actual players and human testers by qualitative means would speak volumes on the actual applicability of the study.

5.4.3 Apply The Mystery Machine to Quest Generation

Quest generation for RPGs and similar games has been a significant area of study within the PCG academic community, demonstrated in works by Doran and Parberry (2011), and Ashmore and Nitsche (2007). Theoretically, given the success of generating plots using the proposed methods, the same methods could be applied to the structuring and generation of a multitude of quest types within existing and new games.

5.4.4 Apply Findings to Interactive Narrative

Interactive narrative (covered in section 2.5) is another popular area of study in PCG as demonstrated in works by Meadows (2002), Young (2000), and Louchart and Aylett (2004). By developing an ASP program which takes input as data generated by player actions, a system could potentially be developed in place of a drama management system that is able to respond to this data by providing logical next steps in the narrative of a game.

5.5 Summary

TMM demonstrates effective use of Smith and Mateas' ASP-powered design space approach to developing procedural content generators for use in games. It achieves this by applying the three-step development process to a novel application that generates plot data, that has proven to be coherent and follow the respective plot structure in a reliable and varied manner. The structure of the generated plot data has been built to mirror that of an existing game which makes use of plots from the same genre: murder/mystery. All of the steps to achieving this have been completed successfully and documented within the confines of this study. Analysis of the generated plot data has proven the generator's ability to produce both varied and coherent content by mathematical means; the development process has also been analysed to evaluate the success of applying the paradigm of ASP to content generators of this type, proving the hypothesis to be true. Despite these successes, the study leaves much room for improvement and further development. Potential future projects have been suggested that either expand the product achieved in this study, or branch out the given research into new and interesting areas in which ASP might prove successful. Particularly, the lack of a fully-fledged game is a big weakness of this study, and an obvious use of the conclusions in this paper would be to apply the TMM system – or a similar one – to the development of one such game. Ultimately, the information and demonstrations provided by this study introduce a new and exciting potential to applying ASP to content generators of all kinds for use in games.

Education never ends, Watson. It is a series of lessons with the greatest for the last.

- Arthur Conan Doyle, *The Adventure of the Red Circle* (1911)

Appendix A

Collection of Data Gathered From the Mystery Machine

This appendix consists of a collection of data gathered from TMM that was originally intended for use in section 4, but was deemed irrelevant.

A.1 Murder Times

The time taken for a murder to be committed in scenarios, tested for the initial design space and each refinement iteration. The initial design space proved to provide outlier results due to its lack of a constraint disallowing a murder from committing multiple crimes.

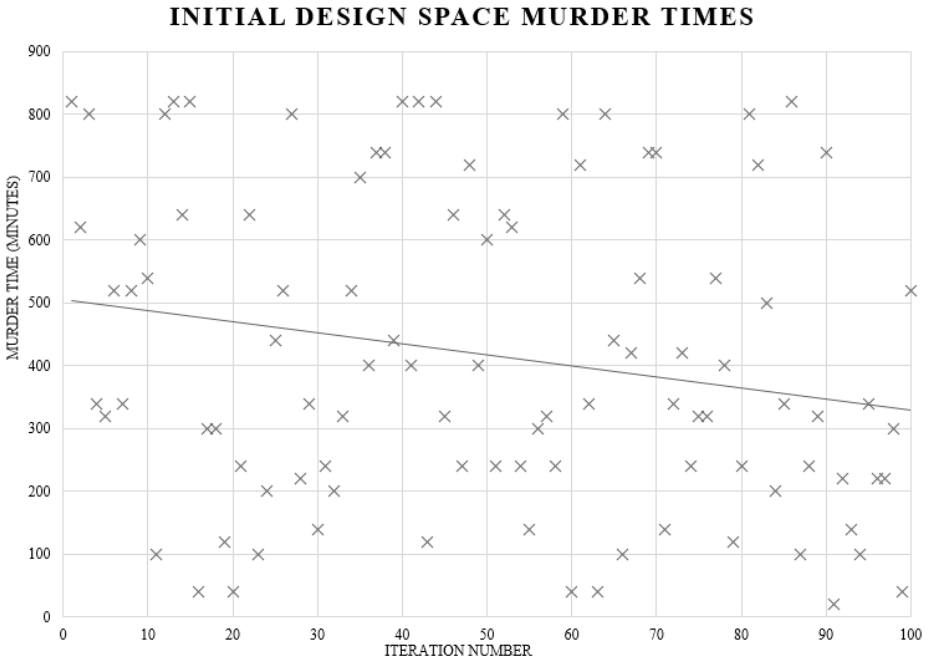


FIGURE A.1: Scatter graph of murder times gauged from a 100 scenarios generated by the initial design space implementation of TMM

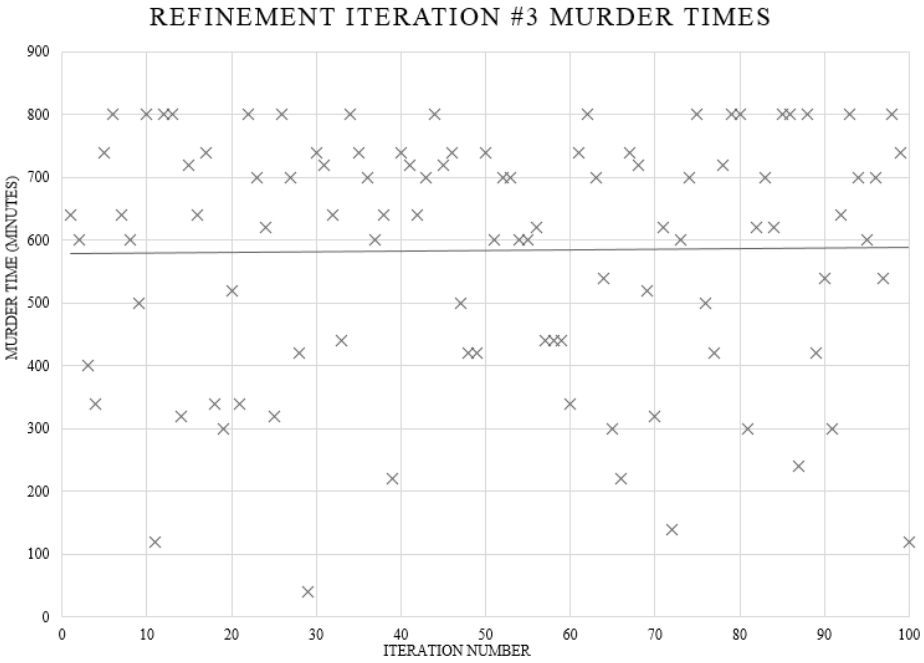


FIGURE A.2: Scatter graph of murder times gauged from a 100 scenarios generated by the refinement iteration #3 implementation of TMM

FIGURE A.3: Scatter graph of murder times gauged from a 100 scenarios generated by the refinement iteration #4 implementation of TMM

A.2 Solve Times

The following graphs demonstrate how solve times change over refinement iterations. This ended up being covered rather easily with a t-test in section 4. It is interesting to see how many of the solve times equate to less than 15 seconds, given the possible state space determined in section 4.3.4!

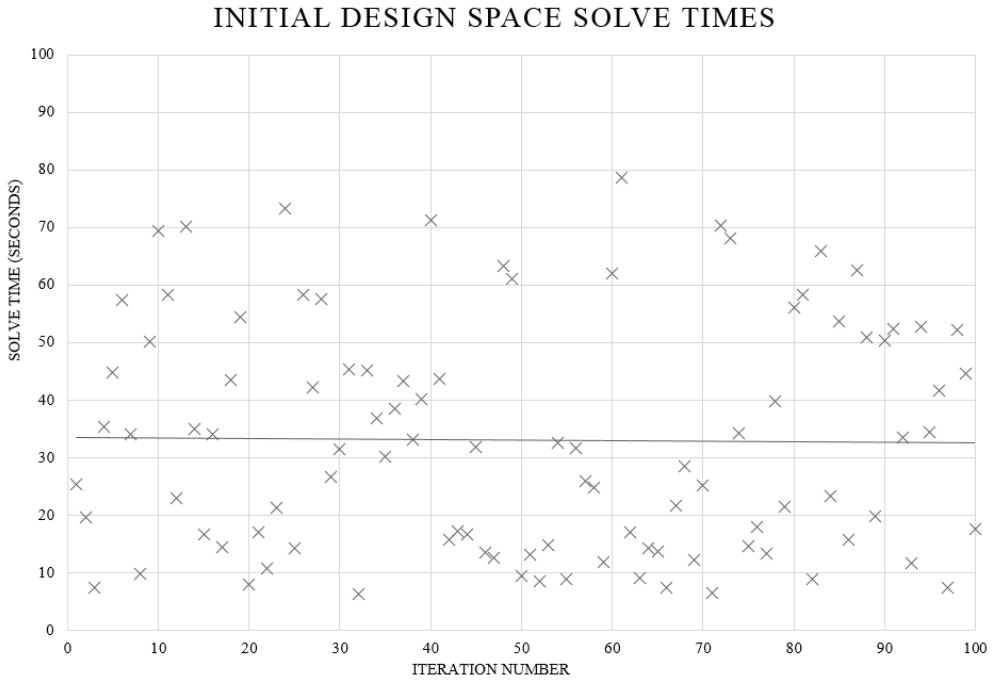


FIGURE A.4: Scatter graph of solve times gauged from a 100 scenarios generated by the initial design space implementation of TMM

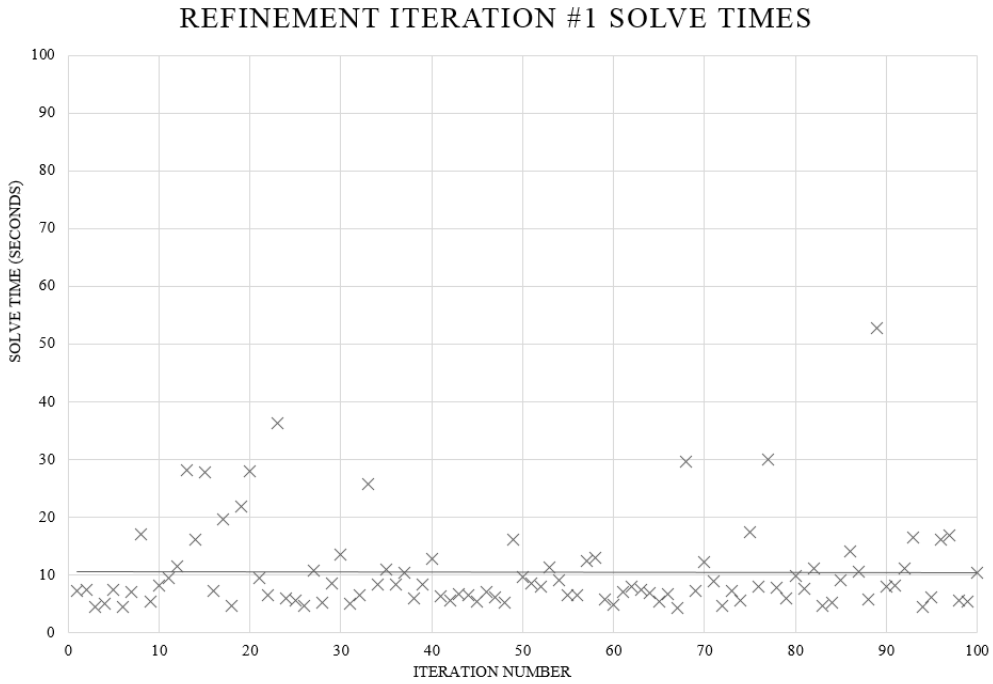


FIGURE A.5: Scatter graph of solve times gauged from a 100 scenarios generated by the refinement iteration #1 implementation of TMM

FIGURE A.6: Scatter graph of solve times gauged from a 100 scenarios generated by the refinement iteration #2 implementation of TMM

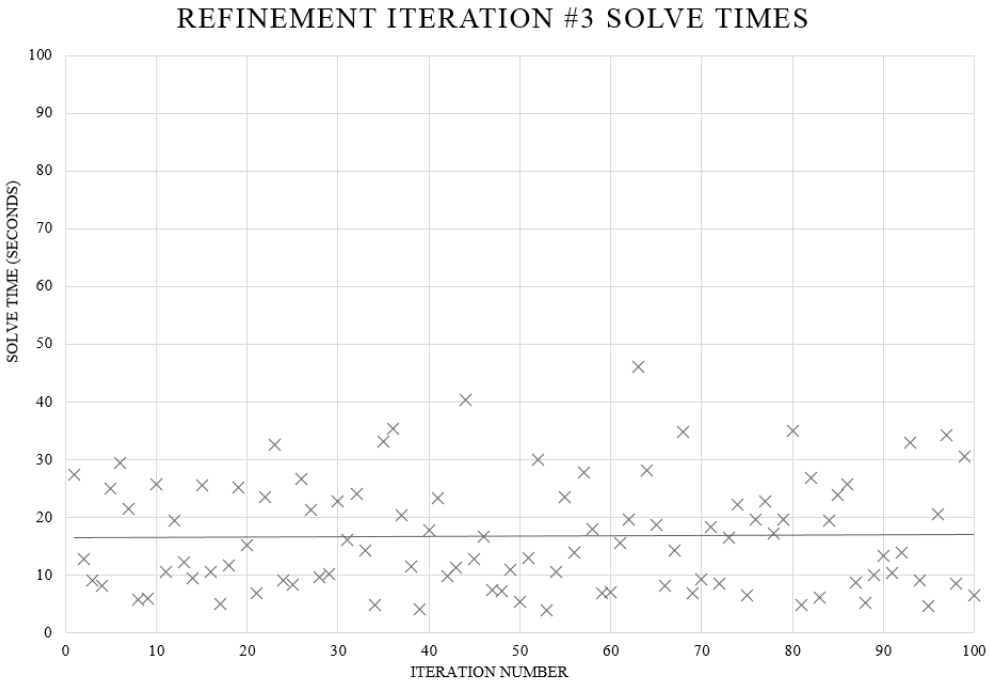


FIGURE A.7: Scatter graph of solve times gauged from a 100 scenarios generated by the refinement iteration #3 implementation of TMM

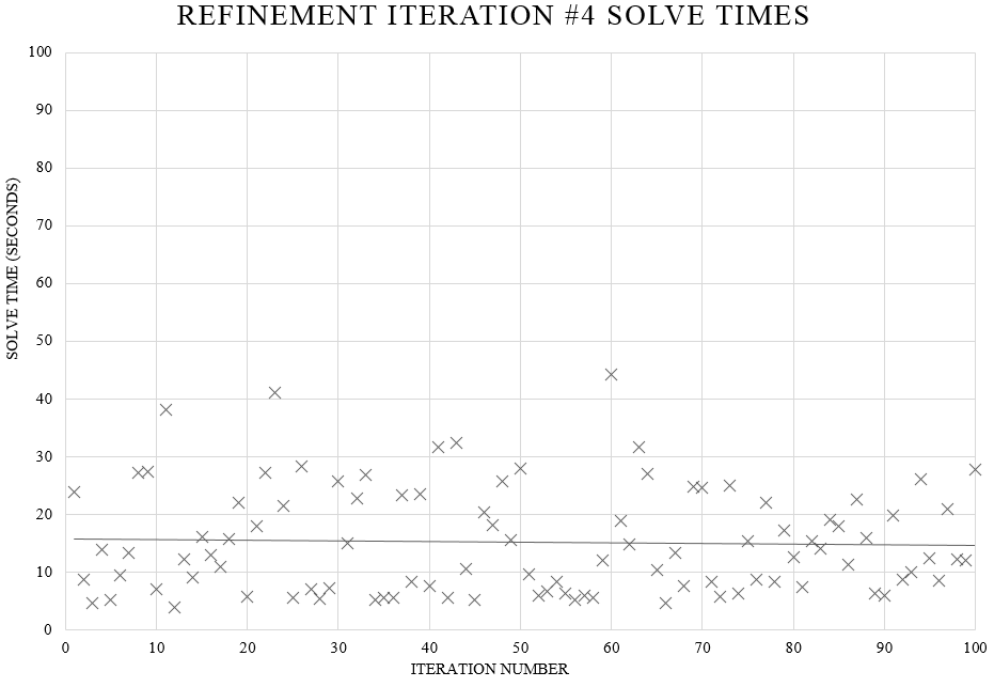


FIGURE A.8: Scatter graph of solve times gauged from a 100 scenarios generated by the refinement iteration #4 implementation of TMM

A.3 Average Murder vs. Solve Times

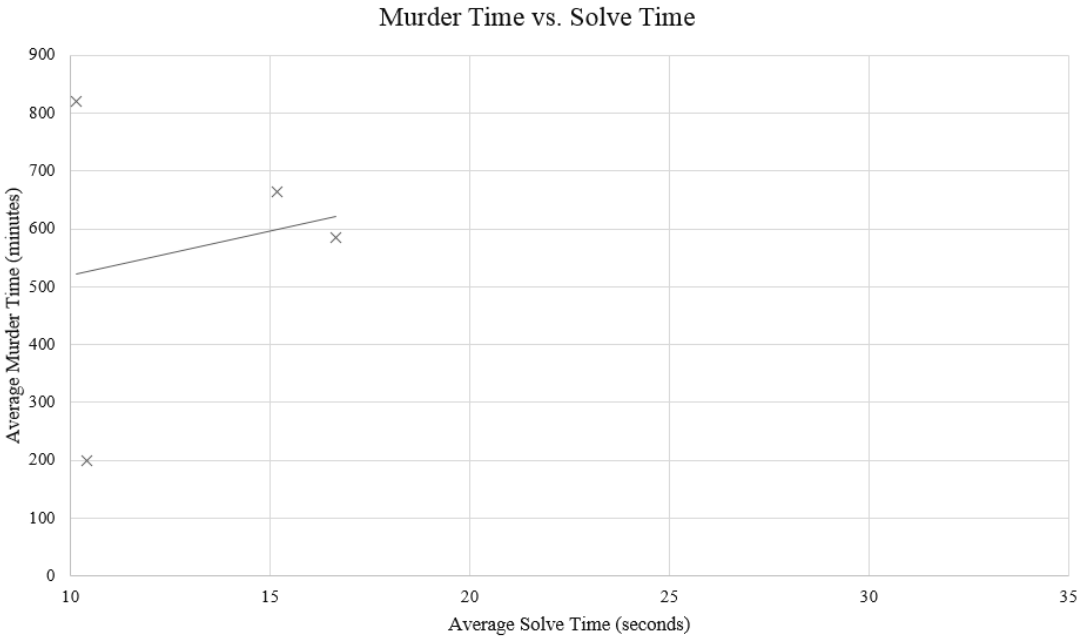


FIGURE A.9: Plot of average solve versus murder time of each refinement iteration

Bibliography

AotF (2013). *Minecraft Render*. URL: http://attackofthefanboy.com/wp-content/uploads/2013/04/minecraft-xbox-360-edition-map.jpg (visited on 14/03/2016).

Ashmore, Calvin and Michael Nitsche (2007). 'The quest in a generated world'. In: *Proc. 2007 Digital Games Research Assoc.(DiGRA) Conference: Situated Play*, pp. 503–509.

Aylett, Ruth (1999). 'Narrative in virtual environments-towards emergent narrative'. In: *Working notes of the Narrative Intelligence Symposium*. Vol. 1.

Aylett, Ruth S et al. (2005). 'FearNot!–an experiment in emergent narrative'. In: *Intelligent virtual agents*. Springer, pp. 305–316.

Baral, Chitta (2003). *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press.

Bioware (2007). *Mass Effect*. [PS3, X360, Microsoft Windows].

— (2009). *Dragon Age: Origins*. [PS3, X360, Microsoft Windows, OSX].

— (2012). *Mass Effect*. [PS3, X360, Microsoft Windows].

Braben, David and Ian Bell (1984). *Elite*. [BBC Micro, Acorn Electron].

Brewka, Gerhard, Thomas Eiter and Miroslaw Truszczynski (2011). 'Answer Set Programming at a Glance'. In: *Commun. ACM* 54.12, pp. 92–103. ISSN: 0001-0782. DOI: 10.1145/2043174.2043195. URL: http://doi.acm.org.ezproxy.derby.ac.uk/10.1145/2043174.2043195.

Brooks, Peter (1992). *Reading for the plot: Design and intention in narrative*. Harvard University Press.

Brown, Malcolm (2014). *The Inquisitor*. [Microsoft Windows, OSX, Linux].

— (2016). personal communication.

Campbell's, Joseph (1987). 'The hero's journey'. In:

Chandler, Raymond (1950). 'The Simple Art of Murder'. In: Haycraft, pp. 222–37.

Chauvin, S. et al. (2015). 'Making sense of emergent narratives: An architecture supporting player-triggered narrative processes'. In: *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 91–98. DOI: 10.1109/CIG.2015.7317936.

Chen, Sherol et al. (2010). 'RoleModel: towards a formal model of dramatic roles for story generation'. In: *Proceedings of the Intelligent Narrative Technologies III Workshop*. ACM, p. 17.

Cook, Michael and Simon Colton (2015). 'Hybrid Procedural Content Generation: A Proposal'. A proposal for Hybrid Procedural Content Generation using two games, Murder and Mystery, to create narrative through user actions.

Cook, Michael, Simon Colton and Jeremy Gow (2014). 'Automating game design in three dimensions'. In: *Proceedings of the AISB Symposium on AI and Games*, pp. 20–24.

Cook, Michael et al. (2013). *Mechanic miner: Reflection-driven game mechanic discovery and level design*. Springer.

Doran, Jonathon and Ian Parberry (2011). 'A prototype quest generator based on a structural analysis of quests from four MMORPGs'. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, p. 1.

Doyle, Arthur Conan (1890). *The Sign of the Four*. Spencer Blackett.

— (1911). *The Adventure of the Red Circle*. His Last Bow.

— (2010). *The sign of four*. Broadview Press.

Evolutionary Games (2010). *Galactic Arms Race*. [Microsoft Windows].

Frade, Miguel, Francisco Fernandez de Vega and Carlos Cotta (2010). 'Evolution of artificial terrains for video games based on accessibility'. In: *Applications of evolutionary computation*. Springer, pp. 90–99.

Gearbox Software (2009). *Borderlands*. [PS3, X360, Microsoft Windows, OSX].

Gebser, M et al. (2015). 'Potassco User Guide'. In: *Institute for Informatics, University of Potsdam, second edition edition*.

Gebser, Martin et al. (2008). 'A user's guide to gringo, clasp, clingo, and iclingo'. In:

Gedarovich, Dave (2016). personal communication.

Glass Knuckle Games (2014a). *Noir Syndrome*. [Microsoft Windows].

— (2014b). *Steam - Noir Syndrome*. URL: http://store.steampowered.com/app/299780/ (visited on 12/05/2016).

Grella, George (1970). 'Murder and Manners: The Formal Detective Novel'. In: *NOVEL: A Forum on Fiction* 4.1, pp. 30–48. ISSN: 00295132, 19458509. URL: http://www.jstor.org/stable/1345250.

Haycraft, Howard (1968). *Murder for pleasure: the life and times of the detective story*. Biblo & Tannen Booksellers & Publishers.

Hello Games (2016). *No Man's Sky*. [Microsoft Windows, PS4].

id Software (1993). *Doom*. [MS-DOS].

Irrational Games (2013). *Bioshock: Infinite*. [PS3, X360, Microsoft Windows, Linux, OSX].

Jenkins, Henry (2004). 'Game design as narrative architecture'. In: *Computer* 44.3, pp. 118–130.

Johnson, Lawrence, Georgios N. Yannakakis and Julian Togelius (2010). 'Cellular Automata for Real-time Generation of Infinite Cave Levels'. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. PCGames '10. Monterey, California: ACM, 10:1–10:4. ISBN: 978-1-4503-0023-0. DOI: 10.1145/1814256.1814266. URL: http://doi.acm.org.ezproxy.derby.ac.uk/10.1145/1814256.1814266.

Johnson, Mark R. (2015). *Ultima Ratio Regum*. [Microsoft Windows].

Kazemi, Darius (2013). *Spelunky Generator Lessons*. URL: http://tinysubversions.com/spelunkyGen/ (visited on 08/11/2015).

Kelly, R Gordon (1998). *Mystery fiction and modern life*. Univ. Press of Mississippi.

Klei Entertainment (2013). *Don't Starve*. [PS3, PS4, Xbox One, Microsoft Windows, etc.]

Lavender, Becky and Tommy Thompson. 'Adventures in Hyrule: Generating Missions & Maps For Action Adventure Games'. In:

Lifschitz, Vladimir (2002). 'Answer set programming and plan generation'. In: *Artificial Intelligence* 138.1, pp. 39–54.

Lima, Edirlei Soares de, Bruno Feijó and Antonio L Furtado (2014). 'Hierarchical generation of dynamic and nondeterministic quests in games'. In: *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*. ACM, p. 24.

Louchart, Sandy and Ruth Aylett (2004). 'Narrative theory and emergent interactive narrative'. In: *International Journal of Continuing Engineering Education and Life Long Learning* 14.6, pp. 506–518.

Mackall, Dandi Daley (2011). *Means, Motive, and Opportunity*. URL: http://www.silenceofmurder.com/index.php/plot-vs-character/11-means-motive-a-opportunity (visited on 28/04/2016).

Macmillan, Edmund and Florian Himsl (2011). *The Binding of Isaac*. [Microsoft Windows, Mac OS, Linux].

Malmgren, Carl Darryl (2001). *Anatomy of murder: mystery, detective, and crime fiction*. Popular Press.

Maxis (2008). *Spore*. [Microsoft Windows, OSX].

McKendrick, Hazel (2014). *Procedural Doesn't Mean Random: Generating Interesting Content*. URL: https://www.youtube.com/watch?v=xJRXNesI9f4&feature=youtu.be (visited on 14/12/2015).

Meadows, Mark Stephen (2002). *Pause & effect: the art of interactive narrative*. Pearson Education.

Mojang (2009). *Minecraft*. [Android, PS3, PS4, PSVita, etc.]

Mossmouth (2012). *Spelunky*. [XBLA].

Neufeld, X., S. Mostaghim and D. Perez-Liebana (2015). 'Procedural level generation with answer set programming for general Video Game playing'. In: *Computer Science and Electronic Engineering Conference (CEEC), 2015 7th*, pp. 207–212. DOI: 10.1109/CEEC.2015.7332726.

Nintendo (1985). *Super Mario Bros.* [NES].

— (1990). *Super Mario World*. [SNES].

Persson, Markus (2011). *Terrain generation: Part 1*. URL: http://notch.tumblr.com/post/3746989361/terrain-generation-part-1 (visited on 17/04/2016).

Potassco (2008). *The Potsdam Answer Set Solving Collection*. URL: http://potassco.sourceforge.net/ (visited on 30/04/2016).

Potassco (2015). *Python: module gringo*. URL: http://potassco.sourceforge.net/gringo.html (visited on 03/05/2016).

Potassco (2015). *Sourceforge - Potassco*. URL: https://sourceforge.net/p/potassco/mailman/message/34281337/ (visited on 10/05/2016).

Pratt, Anthony E. (1949). *Cluedo*. [Tabletop].

Python Software Foundation (2016). *Python Documentation – Random*. URL: https://docs.python.org/2/library/random.html (visited on 11/05/2016).

Re-Logic and Engine Software (2011). *Terraria*. [Android, iOS, PS3, PS4, etc.]

Riedl, Mark Owen and Vadim Bulitko (2012). 'Interactive narrative: An intelligent systems approach'. In: *AI Magazine* 34.1, p. 67.

Shaker, Noor et al. (2015). 'Constructive generation methods for dungeons and levels'. In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Ed. by Noor Shaker et al. Springer.

Smith, Adam (2016). personal communication.

Smith, Adam M and Michael Mateas (2010). 'Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games'. In: *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, pp. 273–280.

— (2011). 'Answer set programming for procedural content generation: A design space approach'. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 3.3, pp. 187–200.

Smith, Gillian and Jim Whitehead (2010). 'Analyzing the expressive range of a level generator'. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, p. 4.

Smith, Gillian et al. (2011a). 'PCG-based game design: enabling new play experiences through procedural content generation'. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, p. 7.

— (2011b). 'PCG-based Game Design: Enabling New Play Experiences Through Procedural Content Generation'. In: *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*. PCGames '11. Bordeaux, France: ACM, 7:1–7:4. ISBN: 978-1-4503-0872-4. DOI: 10.1145/2000919.2000926. URL: http://doi.acm.org.ezproxy.derby.ac.uk/10.1145/2000919.2000926.

Sweetser, Penelope and Janet Wiles (2005). 'Scripting versus emergence: issues for game developers and players in game environment design'. In: *International Journal of Intelligent Games and Simulations* 4.1, pp. 1–9.

Symons, Julian (1985). *Bloody murder: from the detective story to the crime novel: a history*. Viking Pr.

Thielscher, Michael (2009). 'Answer set programming for single-player games in general game playing'. In: *Logic Programming*. Springer, pp. 327–341.

Thomas, John C and John M Carroll (1979). 'The psychological study of design'. In: *Design Studies* 1.1, pp. 5–11.

Togelius, Julian et al. (2010). 'Search-based procedural content generation'. In: *Applications of Evolutionary Computation*. Springer, pp. 141–150.

Togelius, Julian et al. (2013a). 'Procedural content generation: Goals, challenges and actionable steps'. In: *Dagstuhl Follow-Ups* 6.

Togelius, Julian et al. (2013b). 'Procedural Content Generation: Goals, Challenges and Actionable Steps'. In: *Artificial and Computational Intelligence in Games*. Ed. by Simon M. Lucas et al. Vol. 6. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 61–75. ISBN: 978-3-939897-62-0. DOI: http://dx.doi.org/10.4230/DFU.Vol6.12191.61. URL: http://drops.dagstuhl.de/opus/volltexte/2013/4336.

Toy, Michael and Glenn Wichman (1980). *Rogue*. [DOS].

Trecker, Janice Law (2013). 'Wilkie Collins's Sleuths and the Consolations of Detection'. In: *The Midwest Quarterly* 54.4, p. 337.

UKIE (2015). *The Games Industry in Numbers*. URL: http://ukie.org.uk/research (visited on 13/12/2015).

University of Calabria (2012). *ASP Standardization Activity*. URL: https://www.mat.unical.it/aspcomp2013/ASPStandardization (visited on 30/04/2016).

Wambaugh, Joseph (2013). *The Secrets of Harry Bright*. Bantam.

Wikia (n.d.). *Mass Effect Dialogue Image*. URL: http://vignette3.wikia.nocookie.net/masseffect/images/9/98/Conversation_Wheel_Pic.png/revision/latest?cb=20090805012108 (visited on 14/03/2016).

Wong, W. K. C. (1993). 'Logic programming and deductive databases for genomic computations: a comparison between Prolog and LDL'. In: *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*. Vol. i, 834–843 vol.1. DOI: 10.1109/HICSS.1993.270607.

Worsley, Lucy (2013). *A Very British Murder with Lucy Worsley*. URL: http://www.bbc.co.uk/programmes/p01gyptp (visited on 22/02/2015).

Young, R Michael (2000). 'Creating interactive narrative structures: The potential for AI approaches'. In: *Psychology* 13, pp. 1–26.