Electronics and Computer Science
Faculty of Physical and Applied Sciences
University of Southampton

Thomas A. E. Smith
taes1g09@ecs.soton.ac.uk
April 16, 2012

Intelligent Procedural Content Generation for Computer Games

Project supervisor: E. Gerding – eg@ecs.soton.ac.uk
Second examiner: C. Cirstea – cc2@ecs.soton.ac.uk

A project report submitted for the award of
MEng Computer Science with Artificial Intelligence

Abstract

Increasingly, as the demand for ever larger and more varied computer game environments grows, procedural content generation (PCG) is used to ensure that content remains 'fresh'. However, many of the opportunities to use these systems to generate truly personalised content have so far been largely overlooked. When content is generated manually or algorithmically during the design phase of a game, it can only be created according to the designers' expectations of the players' needs. By instead generating content during the execution of the game, and using information about the player(s) as one of the system's inputs, PCG systems should be able to produce more varied content that can be far more tailored to enhance individual players' experiences than anything manually created. In a related field, much has been written about the generation of player models from observed data, including for the purposes of adaptivity or dynamic difficulty adjustment (DDA), and literature exists examining the problem of generating satisfying game environments via challenge adjustment. This project looks at combining these fields to create a prototype 'intelligent' PCG system (IPCG) that is capable of monitoring players' progress and fully dynamically generating upcoming challenges to best suit their abilities. [TODO: specify that this is a prototype, investigating a specific aspect of the many that IPCG is able to cover.]

Contents

Statement of Originality

- You are strongly encouraged to include a one or two paragraph statement of originality this is all my own work is rarely true you should acknowledge the help you have received - Was the idea for the project yours, or was it based on an earlier project, or your supervisors research? - The examiners will assume that the analysis, design, implementation, testing, ... are your own work - So tell them where this is not true the design of component X follows a standard technique/pattern described in [source] this is my own code except for [package/class/method] which I have copied from [Internet site/author]

1 Project Description

1.1 Introduction

The aim of this project is to investigate the use of intelligent procedural content generation (IPCG) in computer games, by looking at existing products, research in related areas and constructing a minimal prototype. [TODO: explanation of IPCG] Due to the increasing demand for both detail and variety within computer game environments, various aspects of in-game content are now often generated procedurally (that is, algorithmically rather than manually), using techniques that are frequently just refinements of algorithms used in the early days of computing, for games such as Elite [2] or Nethack [**?**]. However, one of the strengths of modern procedurally generated content (PCG) is that (within reasonable limits) it may be performed at runtime, allowing it to also use information about the player in order to dynamically generate content on-the-fly in response to the player's actions. In general, this will involve making use of algorithms from the field of Artificial Intelligence in order to evaluate the information available and condense it to a form suitable for input to a PCG system; hence such systems might reasonably be termed 'intelligent' procedural content generators (IPCG). As shown later, IPCG techniques can be applied to many different scenarios, for many reasons. For the purposes of this project, the prototype will be restricted to using IPCG for dynamic difficulty adjustment (DDA) in basic 2D platformer levels.

1.1.1 Definitions

- **Procedural Content Generation** Given the variety of mechanisms used by games to serve content to players, it can be difficult to define exactly what PCG is and is not. Togelius *et al.* define PCG as "*the algorithmical creation of game content with limited or indirect user input*" [29]. This definition deliberately does not specify whether randomness is required in a PCG system, as examples of both random (stochastic) and deterministic PCG systems exist. It also does not distinguish between 'offline' PCG, and adaptive 'online' PCG, as both have their uses.
- **Dynamic Difficulty Adjustment** DDA is another broad term that could be applied to wildy differing systems in various games. In general, DDA is often viewed as the process of altering aspects of a game based on some part of the state of the game world - often a model of the performance of the player. Normally this is done with the intention of maintaining the player's ability to remain within a state of 'flow' [4]. It can range from the simple 'rubber-banding' used in basic racing games, to the more subtle alterations of timing, item placement and enemy frequency used by FPS games such as Resident Evil 5 [**?**].
- **Intelligent Procedural Content Generation** Intelligent Procedural Content Generation is an extension of typical PCG in that it forms a model of some aspect of the player or game state, and then uses this as an input to an adaptive

(parameterised) PCG system that modifies the generated output appropriately. Often this is done for the purposes of DDA (in contrast to typical non-PCG DDA, which generally makes minor adjustments to existing content, rather than generating new content), though it can also be applied for a range of other purposes, from providing more of the type of content a player favours [10], to directing the player towards unexplored areas [18].

1.2 Literature and Existing Systems

Though the focus of this project centres on the topic of IPCG, it can be seen from the definitions above that the fields of PCG and to a lesser extent DDA are closely related. An IPCG system cannot work without some mechanism existing to generate content appropriate for the model of the player that it has constructed, and even if the purpose of the system is customisation of a game aspect completely unrelated to challenge, the techniques and systems developed for the field of DDA are often relevant when attempting to model some aspect of the player. Both PCG and DDA are mature fields with plenty of academic literature and example systems; many commercial game products use these techniques.

1.3 Prototype

One of the aims of this project is to construct a minimal prototype demonstrating the use of an IPCG system. By combining ideas from existing literature on PCG and DDA systems, it should be possible to construct a limited system that performs all of the stages necessary for it to function as an example of IPCG. That is, it should monitor the game state, form a conceptual model of some aspect of the player, and then use that model in order to generate content tailored to the particular player in some way. For the purposes of this project, the prototype will be necessarily basic, performing only the functions sufficient to display working IPCG. It should also provide some facility for player-provided evaluation of the system, to aid in final analysis.

1.4 Goals

The aim of this project is to investigate the use of intelligent procedural content generation (IPCG) in computer games, by looking at existing products, research in related areas and constructing a minimal prototype. [TODO: fill out, listitems]

- **Investigate the use of IPCG in computer games** To date, a remarkably small number of commercial games have contained systems that could reasonably classified as IPCG. In order to successfully research the current use of IPCG then, it will be important to view the field in the context of both PCG and DDA, more mature systems with more extensive bodies of existing literature and successful systems.
  - **Existing commercial products**
  - **Existing academic literature**
- **Construct a minimal prototype** – **Specify requirements**

- **Design and Implement**
- **Test and Evaluate**

2 Project Background

Lot of academic work in related fields, Paucity of actual shipping systems

2.1 Procedural Content Generation

Procedural Content Generators have been used since the early days of gaming. Well-known games such as Elite and Rogue made extensive use of PCG in order to present the player with expansive game worlds far larger than could have been fully stored on the distribution media that was available at the time. In the case of Elite, this was done using a fully deterministic PCG system, and storing only the seeds used to generate the desired content - resulting in a game world that was identical each time it was generated, but that took very little memory to store. For Rogue, environments were generated pseudo-randomly, meaning a different play experience each time, but following strict constraints that ensured that levels were completable [29]. As technologies improved, focus shifted more towards hand-crafted environments as it was easier to ensure that these provided value and did not feel sparse [19]. However, with the further progress of technology attention has returned to procedural generation. Modern game worlds contain vast amounts of detail, and procedural content generation algorithms are ideally suited to producing large numbers of variations on a theme, be that trees, clouds, textures, or even sounds. Producing each of these items individually by hand would take many hours of labour and much disk space, but by defining specific sub elements and assembly rules, variation can be almost endlessly reused. As PGC mechanisms have matured, they are once again being used for the provision of entire play environments. Commenting on the use of PCG in a successful commercial title (Borderlands, [9]), A. Doull claims that "it points the way forward to a time where the current role of the level designer will be as obsolete as punch cards" [8].

, as in the game Infinite Mario.[20]. TODO: mention occupancy regulated expansion / extension

2.1.1 Existing PCG Systems

Many interesting PCG systems have been developed, too numerous to mention. Of particular note:

- **Charbitat** [TODO: write description for charbitat. academic paper /example system. arguably IPCG. . .]
- **Infinite Mario** A Java reimplementation of the original Mario, uses PCG to create an endless variety of potential levels [20]. Considered something of a standard in academic PCG implementations, the codebase is often used as a launching point for competitions of various kinds.
- **Speedtree** Possibly the most widely-used commercial PCG system, Speedtree is a middleware application that generates trees and other vegetation for use in games and some movies [6].

– **Borderlands** A commercially successful videogame, Borderlands was one of the first to extensively use and publicise a PCG system; in this instance to generate millions of different varied weapons [21] - a selling point for the game.

2.2 Dynamic Difficulty Adjustment

Another game design concept receiving increasing attention is dynamic difficulty adjustment (DDA). Typically, challenge adjustment within video games has consisted of user choice between one or more discrete challenge settings that have been painstakingly balanced at production time. However, this solution is far from ideal - typically, if a game is begun with a certain difficulty it is difficult to later change; and this upfront decision also alienates players that are unfamiliar with the terminology or expectations, or uncertain how to classify themselves[15]. Furthermore, since game difficulty is typically a continuous function of multiple parameters, it should be possible to precisely match each player to their ideal level of challenge rather than enforcing adherence to low-resolution skill profiles. By monitoring and then modelling the players' ability in some fashion, it can be possible to make informed changes to the play environment that satisfyingly help or hinder their progress. Typically, DDA is achieved by altering values that are hidden from the player, such as enemy health, accuracy, or the amount of ammo and health-kits available in the world [13]. Often, the intention is to do this invisibly, and merely ensure that the player remains optimally challenged. By manipulating values behind the scenes, it is possible to ensure that the player is neither over-challenged (leading to frustration / anxiety), or under-challenged (leading to boredom)[4]. As DDA systems are given more control over additional aspects of the game environment, they can begin to cross the line and enter the realm of PCG, fundamentally altering the structure and pacing of the player's experience. Much DDA literature is relevant to the field of IPCG, as the the data-collection and model-forming portions of IPCG systems have existing parallels in DDA research.

2.2.1 Existing DDA systems

– **Resident Evil 5** A successful commercial game, Resident Evil 5 contained a [TODO: write about the adaptive difficulty where players pick a difficulty rating as usual, which limits them to a window on the 1 to 10 rating. The system monitors: x, the system adjusts: y [**?**].]
– **Hamlet** Hamlet is a system designed by R. Hunicke to examine "basic design requirements for effective dynamic difficulty adjustment" [12]. It integrates with an SDK for an existing game engine in order to monitor aspects of the player's status, and modifies upcoming encounters based on a historically-trained map from player evaluation result to desired game world adjustments.
– **Polymorph** [TODO: write about Polymorph. Academic paper / example system. Also arguably IPCG, especially since it's paper is in that section...] [14]

2.3 Intelligent Procedural Content Generation

Though mechanisms fulfilling the definitions of IPCG systems have already started appearing in games, little has so far been written specifically on the subject - "personalized and player-adaptive PCG [...] is a new research direction"[22]. However, existing literature in related areas borders on the topic: in some cases DDA algorithms are being used to generate entire levels; thus qualifying as IPCG. One of the most thorough papers on this area is by Jennings-Teats *et al.* [14], who developed Polymorph - a system that generates 2D platformer levels on-the-fly. Approaching the topic of IPCG from another direction, Lopes' and Bidarra's survey of adaptivity challenges in games and simulations investigates the use of adaptivity in general in order to combat static and predictable content[15], including via PCG.

2.3.1 Existing IPCG Systems

IPCG can be (and has been) used for a wide range of purposes, almost as varied as PCG itself. Three very different such uses in commercial games are detailed below. It is unsurprising that many of the existing applications of IPCG are used to tackle some of the current key challenges in game design: maintaining players' engagement with the game via enhancing immersion and controlling 'flow'[4]. [TODO: link to earlier description of intentions for flow maintenance]

- **Valve's 'AI Director'** One of the most well-known such applications is used in Valve's games Left 4 Dead and Left 4 Dead 2. Known as the 'AI Director', the system monitors the "emotional intensity" of each players' gameplay experience, by tracking factors such as each player's current health and recent kills, proximity and deadliness of visible enemies, and separation from the main group. It then dynamically alters the placement of supplies and the generation of enemies of various types in order to control pacing and maintain flow. It follows a policy of encouraging intensity to build up to a peak, sustaining threat for a short time, and then allowing intensity to fade during a 'relax' period, thereby creating somewhat-unpredictable peaks and valleys during gameplay. In Left 4 Dead 2, the Director has additional control over the structure of the level [1].
- **Bethesda's Radiant Story** Another recent example of IPCG is the Radiant Story system used in Bethesda's game Skyrim. Rather than monitor the player's performance and aptitude, it evaluates their progress and history. When the player receives a quest, the system looks for viable locations that the player has not yet explored, and customises details of the task so that it will take the player to this new location. This avoids requiring the player to return to already-completed areas, and instead force exploration of previously unknown regions, helping to increase immersion by avoiding repetition of content [18].
- **GAR's Weapon Evolution** Finally, the weapon evolution mechanism in the game Galactic Arms Race[10] is an unconventional application of an IPCG system. All of the weapons in the game are represented by procedurally generated particle systems, with a small collection of variables that control their behaviour [11]. The IPCG system tracks only which weapons the player spends most time using, and then uses small neural networks to evolve new weapons

that are variations on the player's favourite weapons so far. This allows the player to experience more of the type of content that they prefer.

2.3.2 IPCG Academic Literature

Polymorph: DDA Through Level Generation

Polymorph is a progression of prior work in both DDA and PCG: like previous DDA systems it alters challenge during play, and as with traditional PCG for 2D platformers it generates levels algorithmically. However, the DDA is effected via structural differences in the level design rather than 'numerical tweaking', and the level is generated 'online' out of small rhythm segments[24], rather than fully ahead of time. The authors present an interesting statistical model of difficulty, along with some of the issues and solutions found while evaluating the system [14].

Adaptivity Challenges in Games and Simulations: A Survey

The survey is an investigation of present research into and existing commercial implementations of adaptivity in games. The topic is broken down into three areas: purpose, target and method, with a wide range of examples given for each point raised. In contrast to other papers on the subject, the authors look beyond challenge as the sole steering purpose for adaptivity, and then discuss the wide range of types of content that may be adapted or generated algorithmically. Finally, they look at the methods by which content may be adapted or generated, and conclude that PCG is one of the most promising for offline generation, and also may increasingly be suited to online adaptation. [15]

2.4 Problem Specification

[An analysis and specification of the solution to the problem.] By looking at [the existing literature], it can be seen that not many [demonstrations of successful IPCG] exist. An attempt will be made to construct a minimal working prototype that demonstrates a successful IPCG system. Existing approaches, both in PCG and IPCG, have focused on simple 2D platforming systems, and so in order to benefit from developing the ideas presented in current literature, and also to aid in later comparative evaluation, it seems sensible to follow this approach also. [Following the work of Hunicke who worked to examine "basic design requirements for effective dynamic difficulty adjustment" [12]. Attempt a similar thing for IPCG]

3 Proposed System

3.1 Overview

The project is intended to provide the minimum functionality necessary to demonstrate a working IPCG system. This means that it will require the ability to monitor game state as the player progresses, form a model of some aspect of the state, and then generate further content based on the input from the model. Following the separation of concerns (SoC) design practice, the project may easily be modularised into three principle components: a 'host' or base module, an adaptive PCG system, and a means of taking data about the player and converting it into a model usable by the PCG. The proposed flow of information is shown in Figure 3.1.

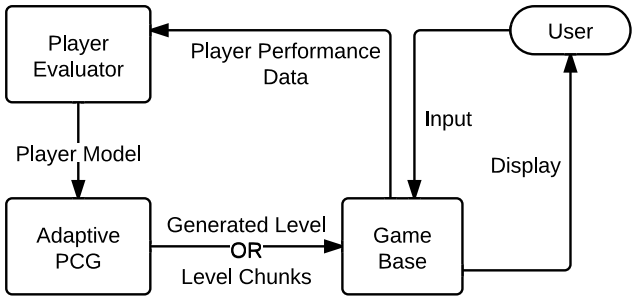

Figure 3.1: Data flow in proposed system.

3.1.1 Modules

Game Base

The game base need be nothing more than a simple 2D platformer engine. This module should handle all input and rendering activity, and should follow the Model-View-Controller architecture in order to facilitate monitoring and live updating. In addition to presenting the user with the output of the IPCG system, the Base module should provide the basic input handling, physics and game functionality necessary to play the platformer, while also logging multiple types of information about the players performance for the evaluator module.

Adaptive PCG

Forming the first portion of the IPCG system, this should be an adaptive (parameterised) 2D platforming level generator. Building on the work of Compton *et al.*[5], this module should maintain a context-free grammar (CFG) of obstacles available, along with weights representing the estimated challenge of each element (terminal obstacle or combination).[TODO: explain this more fully previously] By taking these weights into account when deriving a string of obstacles from the CFG, sequences of a desired difficulty can be produced - or alternatively, the estimated challenge of existing sequences may be evaluated. A PCG system designed in this way should be able to generate entire levels 'offline', by maintaining pre-determined maxima and

local variations in difficulty, but should also be capable of generating levels on-the-fly, by ensuring that short-term future difficulty levels match those requested by the Evaluator.

Player Evaluator

The second portion of the IPCG system should be essentially a multi-class classifier. Given the varied inputs from the Base module, the Evaluator should form a belief about the player's skill relative to the current challenge of the level. By running the player's data through a previously-trained classifier, this module should obtain a model that can be passed to the PCG module and acted upon.

3.1.2 Approach

The modules above are presented in logical order of development: none of the IPCG system will be testable without the Base program (which can be tested standalone if given a hand-crafted level), but the PCG may be run and tested using specimen models, and finally the Evaluator can be tested once the other systems are in place.

Classifier Training

The development of the classifier will involve initially collecting data on as many potentially relevant features of the player's performance as possible, and then performing principle component analysis (PCA) upon the data-set in order to identify the maximally variant features. These can then be retained and used as the input to a K-means discretisation algorithm, which can finally be used to train a One-vs-All SVM classifier.

Evaluation

In order to perform retrospective evaluation of the system, it would be useful to integrate some facility for users to provide deliberate feedback outside of the player modelling system.

3.1.3 Justification

Justify: 2D platformer as simplest possible approach. Use of Java, as it is my most well known language. Custom-built base module, as will allow to be kept simple. An alternative approach to the problem of generating a 2D platforming environment for use with DDA is presented by Sorenson *et al.*[27], who detail a more general top-down approach using genetic algorithms. However, their system is also more complex and provides an unneeded degree of generality for this project. This system as proposed should be able to fulfil the requirements given, and demonstrate intelligent variation in output based upon the skill of the player. [TODO: mention the intelligent DDA. justify different genre [17]. However, didn't know about this at design time.]

3.2 Requirements

The main aim of the project requirements will be to constrain the problem to an achievable scale, and inform future evaluation of the final solution. The 'functional' requirements that follow specify functionality that the system must provide. The majority of them are generic to any IPCG system, as this project intends to cover that alone, and refinements specific to this project are specified where necessary. The 'non-functional' requirements define qualities that the system must adhere to, and in general are constraints that should serve to encourage feasibility and quality.

3.2.1 Functional

In order to properly implement IPCG, the system should:

- **Present the user with an interactive game environment.** Without at least a basic game environment in place, there will be no player interaction to collect data on, and nothing to generate content for. The system should provide a simple 2D platforming environment, with enough complexity to demonstrate working IPCG. Typical game mechanisms such as score or powerups are unnecessary in this context.
- **Record data on the game state and player's behaviour.** The system will need to be able to monitor and record data on many aspects of the game environment. Statistics such as number of mistakes, number and average width of gaps jumped, and time to completion of level will all be needed for the player evaluator. In addition, non-player data such as the length of the level may also need to be taken into account.
- **Evaluate this data according to specific criteria.** [a trained classifier]
- **Form a model of some aspect of the player.** [skill relative to current difficulty]
- **Use this model to inform further PCG activities.** Finally, the system will need to be able to make use of this model in order to generate future level chunks at a difficulty suitable for the player. To do this, the PCG must be able to evaluate the difficulty of its own output, and ensure that this matches the desired difficulty indicated by the evaluator.

In addition, in order to facilitate evaluation of the system itself, it should:

- **Keep a usable record of the data used to generate a level.** This serves two purposes: it should allow re-generation of a previous level when given the same input, for inspection of the generation process. It will also allow more accurate re-calibration of the classifier, should that be necessary.
- **Request feedback from the player.** In order to evaluate the effectiveness of the system, it would be useful to have feedback from the actual users. Rather than require the use of an external platform, an inbuilt feedback facility could store responses alongside the in-game data stored above.

3.2.2 Non-Functional

In order to remain at a manageable scale, the system should:

- **be written in Java.**
- **be presented as a basic platformer.** Though IPCG could be applied in some fashion to most genres, adjusting jump and obstacle placements is likely to be one of the most basic applications.
- **be confined to 2D.** As the degree of complexity of the generated content grows, so too does the complexity of the PCG required to generate it. A simple 3D terrain is relatively easy to generate, but in order to keep the complexity of a platforming game to a minimum it should be restricted to two dimensions.
- **limit the user to move and jump actions.** Many 2D platformers provide the player with novel interaction methods, which in this context would complicate the both the level generation and player evaluation systems. By restricting the player avatar's moveset, complexity is minimised.

In order to function satisfactorily as an interactive experience for the users, the system should also:

- **remain responsive**
- **properly maintain the challenge of generated content**

[TODO: make this far longer and more detailed. Write about requirements of/for evaluation method using swing]

4 System Design

- A detailed design
high level design or architecture
comparison of alternative technologies

o Which process did you follow: waterfall, iterative, evolutionary?
o How does this show in your plan?
o Why did you choose this process?
o How well did it work out for you?

During development, I chose to use an agile development programme, in order to increase flexibility in case of unforeseen difficulties. Did research into 'Scrum-for-one' [cite the workflow paper]

4.1 Overview

The system should be written as a java applet. this has the advantage that it will be able to be run online, making receiving feedback easier. There will be two primary types of interface: game segments drawn on Canvas objects using active rendering, and JPanels containing feedback/instructive segments created using Swing. The Java [object management system] facilitates swapping out these two types of elements

4.2 Game Segments

4.2.1 Game Base

Discuss the functions that all objects must have (rateable etc)

- final objects such as platforms and jumps, these are 'parts'
- grouped into obstacles, or 'components'
- components are built up into repeating 'patterns'
- multiple patterns make a level.

this is the structure as recommended by [5]

In terms of implementation, a level owns a canvas, which can be added to the parent application (this is no longer true, but was at design time) More accurately, a level is the highest item in the model, and the canvas is a game segment that also has a view and a controller.

4.2.2 Generator

Ties into the data structure discussed above. Each part or collection can return its difficulty. This can be queried at any point in the hierarchy. Parts return a

difficulty that is a function of their size (quote formulas here). The larger collections all calculate their difficulties according to some function of the objects that they contain, and occasionally how late in the level they are. Need to also specify the probabilistic grammar that is used, and explain how the probabilities are weighted based upon the desired difficulty. [26], which uses a grammar based generator too.

4.2.3 Evaluator

Monitors this list of features about the player's behaviour:

– **stuff**

and also these features of the current level:

– **more stuff**

Then, do use k-means discretisation with these features.

4.2.4 Interactions

[TODO: detail callbacks, between components/systems.] [Describe the entire quintuple model view controller system. I've already done this once somewhere, If I can find it. Don't know whether to put this in design, or as a realisation...]

4.3 Feedback Interface

One of the advantages of using Java is that it provides the Swing GUI widget toolkit, which makes creating a simple interface allowing player feedback relatively simple. Four custom layouts are needed:

– **Introductory text.** Used for the welcome screen with a brief overview of the experiment. Re-used for the interaction instructions. Re-used again for the final submission confirmation and thanks screen.
– **Introductory questions.** Initial mini-questionnaire containing two labelled five-point Likert items to query the participants' familiarity with games and 2D platform games.
– **Level evaluation questions.** Used after each of the four levels to request player feedback. A single seven-point Likert item with label and explanatory text, possibly level / progress statistics.
– **Final review and submission.** Penultimate screen. Provides a non-editable textbox containing all data to be submitted, and a final 'Submit' button for confirmation.

[TODO: discuss justification for using likert items, advantages of symmetry. Cite ideal use of likert items in surveys, justifying neutral presentation.]

4.3.1 Logging and Submission

All of the player feeback and evaluator input and conclusions will need to be logged [if user permissions permit, to a file-backed stream of some kind]. The system itself

can provide a global function for this. A YAML-like simple parseable format should be sufficient to represent the types of data needed. On the penultimate screen, users will be given a chance to review the data that has been logged, before submission. On submission, the applet will attempt to POST the data to a php script hosted on my personal webspace on ECS systems, which will save the data to a file. In case of error, the system will also attempt to save the data to a local file, and provide instructions on how to upload the file anonymously.

4.3.2 Web Service

In order to receive data from participants running the system remotely and anonymously, a simple .php script will be running at a hard-coded location on the ECS servers. This will be able to receive POST data from the system, and after sanitising it save it to a file in a directory on the server. In case POST submission has failed, the script will also be linked to by a page that allows data file upload though a simple HTML form.

4.3.3 Analysis

In order to make use of the data collected, simple scripting can be used to parse the files into participant feedback data and participant interaction data. The feedback data can then be analysed using a mathematical tool such as Octave in order to evaluate the effectiveness of the system. Time permitting, the interaction data can be used to update the calibration data of the evaluator in order to improve its performance in preparation for a potential second survey. If necessary, it can also be inspected to determine the cause of any suspected anomalous feedback, or to give further insight into the veracity or cause of the results of analysis.

5 System Implementation

see testing Discuss use of java

o Which process did you follow: waterfall, iterative, evolutionary? o How does this show in your plan? o Why did you choose this process? o How well did it work out for you?

5.1 Approach

I used an agile methodology, with regular milestones and refactoring, and an aim of always having a working prototype at the end of each coding session. The overall system development intention was to follow a waterfall approach with regards to completing each module before moving onto the next, this proved to be fairly optimistic and nave, as for that to work the modules would have to be completely independent. While generality of that kind is desirable, there were practical limits to the amount of time available, and higher-priority features, and so development of the three modules each started independently but ended up being developed together roughly in parallel, particularly the generator ant the game base. In MVC terminology, the model is shared through the system, and then there are two views and two controllers. The game base's view renders the level to the screen, through the GameObject's `render()` function. In response, the player provides input to the system, handled by the game base's controller which applies gravity and collision checking, and occasionally calls upon the IPCG system to populate the model with more content. Within the IPCG system, the evaluator can be seen as a pure-data non-visual view, which queries the model using the GameObject's `rate()` function. Based on its classification, it passes input to the aPCG, which can be seen as a controller that modifies the model.

5.2 Game Base

Started with a similar base to previous Java projects: an applet containing a Canvas used for active rendering [7]. Looking ahead to the need to link in with the PCG, began by implementing the hierarchical model described in [5] using a pseudo-component model, using interfaces such as `Drawable`, `Rateable` and `Collideable` to represent components, each containing a static collection that kept track of all of its members. When it came to implementing collision and debug rendering, I noticed the similarities between the hierarchical model and the traditional scene graph pattern, and ended up refactoring the internal data structures to follow a more traditional inheritance model. Almost every object within the game inherits from the abstract `GameObject` base class, which provides the standard interface as shown in Listing 5.1.

```
import java.awt.BasicStroke;
import java.awt.Graphics2D;
import java.awt.Point;
```

```
public abstract class GameObject {
        public abstract void render(Graphics2D g2d);

        public void debugRender(Graphics2D g2d, int inset) {
                g2d.setStroke(new BasicStroke(2*inset));
                g2d.setColor(Constants.DEBUG_COLOURS[inset]);
                g2d.drawRect(getStartPoint().x, 0, getEndPoint().x-
                    getStartPoint().x, Constants.WINDOW_HEIGHT);
        }

        public abstract Point collide(Player p);

        public abstract void tweak();

        public abstract void translate(Point delta);

        public abstract Object clone();

        public abstract float rate();

        public abstract Point getStartPoint();

        public abstract Point getEndPoint();
}
```

Listing 5.1: GameObject.java

This approach is augmented by the abstract child `GameCollection` class (Listing 5.2), which uses generics to support arbitrary collections of game objects, and provides default implementations for most of the required functions that simply call the same function on each of its children, in some cases culling by collision with the camera or player.

```
public abstract class GameCollection<T extends GameObject> extends
    GameObject {

        ArrayList<T> elements;
```

Listing 5.2: Excerpt from GameCollection.java

Each of the concrete items within the world, such as `FlatPlatformPart`, inherit from `GameObject`, often via `Part`. Each of the collections inherit from `GameCollection`, specifying generics to restrict the type of object they can deal with, as in `Component.java`: `public class Component extends GameCollection<Part> .`

discuss physics, sonic, simplification

Had significant problems throughout development when making things too general: specifically collision, among others. Initially worked on a general collision algorithm for colliding the player (a circle) with a platform of arbitrary angle. This turned out to be superfluous, as for simplicity the control physics does not deal differently with movement on non-horizontal surfaces, and the generator does not generate them to reduce complexity. This could be the subject of a future extension however. Confining all platforms to be horizontal greatly simplified the collision algorithms, though there are still difficult cases there the player is potentially colliding simultaneously

with two contiguous platforms: in these cases the results of the two collisions are compared and whichever results in a greater y-displacement is accepted: due to the horizontality restriction on the platforms, this guarantees the player is no longer intersecting either possible collision (TODO no it doesn't, will have to look into this... well, it does for contiguous platforms, but not for disjoint ones: _- gah.)

Used test-driven-development for the collisions, as these were tricky and I wanted to be certain of getting it right. Throughout development, had to use regular integration testing in order to ensure that new features and functionality was compatible with existing code. [TODO: Write about agile development and continuous unit and integration testing.] [TODO: mention Java strong-typing and generics] [TODO: write about]

6 System Evaluation

o Comparative evaluation(cf.competition)
- performance graphs, feature lists
o Critical evaluation
- with respect to your project goals and plan

Personal evaluation of system: Simpler than I would have desired, due to time constraints. Features that I would have liked to have included:
investigation into whether (and how) placement of the player within the play surface affects difficulty, based on look-ahead time. i.e |_o__ —— vs. |__o__|
angled platforms
loopback cells (major effect on difficulty, don't die just return)
death penalty: return to start of component, pattern or level
auto-movement of camera

Wider evaluation of system: look at whether participant feedback matches expected: items:
Levels 1 & 2: familiar players should report that they experienced less challenge than unfamiliar players
Levels 1 & 2: all (most) players should report that 2 was more difficult than 1
Levels 3 & 4: there should be less variance in the reported difficulties than for 1 & 2
Levels 3 & 4: all (most) players should report that 4 was more difficult than 3

Comparative evaluation with competition:
difficult. closest competition is Polymorph, possibly Launchpad; can look at similarities and differences

Critical evaluation:
has fulfilled all (most?) requirements. However, many desired features missing Important question: has it demonstrated working IPCG? If so, are there interesting comments that can be made 'even with an environment this simple/restricted, it it possible to make meaningful changes that impact the player's experience of challenge'. If not, why not? was the system too simple, or is it purely bad calibration. In either case, is it possible to improve the calibration? if there's time, can the wider evaluation be run again with a better-calibrated version using the first survey's data, and does this give interesting results to be used in the viva?

7 Project Evaluation

One or more Gantt charts showing the planned schedule and the actual progress

o Critical evaluation
with respect to your project goals and plan
o Reflection
in hindsight, did you use the right tools, techniques, metrics and methods?
what did you learn?
were your goals and plan sensible?
how could you have done it better/differently?

7.1 Critical Evaluation

Have achieved all (most) of the project goals. Have conducted a thorough review of academic literature relating to systems that could be described as IPCG, as well as [background] technologies that are important to the development, such as PCG and DDA. Have constructed a system that provides something akin to a minimum working demonstration of IPCG. However: there are many interesting questions left unanswered. Even within the relatively simple 2D platforming environment, restricted to jumps, obstacles and movement, there are many gameplay variations left uninvestigated. The original requirements were not restrictive enough; c.f loopback, moving at varied speeds, varied lookahead, non-horizontal platforms, death penalties etc Though the plan turned out to be nave in many respects, it did provide a useful starting point for guiding the overall structure of the implementation process.

7.1.1 Reflection

- **Tool use:** Tools used were generally sensible. Mercurial for version control and eclipse as an IDE were both fairly invisible. One major advantage of Eclipse is that it also provides LaTeXauthoring functionality through a plugin, allowing development of both system and report in a single familiar environment. Online diagram and flowchart webapp LucidChart (`http://www.lucidchart.com/`) proved invaluable for general design work.
- **Techniques:** Constant refactoring was a definite boon. Not being afraid to rewrite entire sections made the code more flexible, and I had to ensure I didn't become too attached/committed to a particular way of doing things.
- **Metrics:** largely gut instinct. Actually attempted to minimise LOC as much as possible, improves maintainability. Had initial false start where rather than using a class hierarchy for GameObject was using an interface- and static collection-based pseudo-component system. This might have worked in a language with multiple-inheritance, but in Java I discovered that I was near-duplicating too much code. The current scene-graph-like hierarchical approach results in on average deeper (and therefore more numerous) function calls, as to

render individual parts the call digs down through the level, pattern and component. However, the approach is more flexible in that it allows render- and collision-culling at any granularity, and is also easier to debug and maintain. A vestige of the old collection-based approach is apparent in the obstacle (or is it active?) interface with static collection, where the independent collection of obstacles for pre-collision checking was too valuable to lose. CRC was largely instinct also: when it became obvious that the Level class which owned the drawing Canvas had too many responsibilities, refactored it into the GameSector class that contained the level, and split out the view and controller classes explicitly. [TODO: worth using analysis tool possibly?]

– **Methods:** Using generics in the GameCollection class allowed very easy extension. Following the MVC architecture has simplified the separation of concerns, naturally leading to code that is more cohesive and less coupled. A certain degree of coupling has been unavoidable, as in the GameObject hierarchy, but it has transpired that the complete generality and independence called for by the schedule was both unnecessary and counterproductive.

Things I learnt: don't be afraid to throw out the current approach and rewrite it. Similarly, don't be afraid to ignore/rewrite the plan when it becomes apparent that it doesn't match the actuality of development. Don't reinvent the wheel! see engine mistake later. Don't plan for the ideal; more important to accept the practicalities of the project at hand.

Goals and plan: Goals were in the main sensible, though probably not well enough defined. 'Investigate' is too vague: research was done with an eye specifically towards papers, examples and techniques that would aid in developing a demonstration system. This resulted initially in a very broad search, c.f Perlin noise generators. Thereafter honed in on PCG and DDA for 2D platformers, was forced to remain more general for IPCG as there is comparatively little literature. However (back on topic) there is scope for a more detailed review and comparison of any particulart technique in the literature than that provided here; particularly in investigating ways that various approaches could work together. In this instance, was forced to remain more abstract in order to also serve as an introduction to the field, and provide background for the prototype. Plan was initially sensible, but suffered from becoming to broad and general throughout. More on this later. How could I have done it better differently? Could write loads on this alone. Brief list: Earlier investigation on the difficulties of sending POST data from Java. Have done a small amount of work with this in the past, but partly owing to time taken on other items, did not allow properly enough time for this. Pick a simpler testbed. Initial research showed a lot of PCG, DDA and even some IPCG literature applicable to the problem of 2D platforming. As this is generally considered fairly basic, seemed appropriate to follow suit. However, the goal was to create a minimal working demonstration. Partway through implementation discovered a paper by Missura *et al.* on the use of machine learning algorithms in player modeling for intelligent difficulty adjustment [17]. The example system in this case was a very simple 'arcade' style 2D shooter, a genre which would possibly have been an even simpler testbed than a 2D platformer. However, due to the nature of the genre it would have been more difficult to affect the game environment in manners that were more obviously IPCG than DDA. Given that the choice of a 2D platformer was probably a correct one, then rather than cre-

ating engine from scratch should have picked mature code. The IPCG system is quite demanding in that it must be tied into the engine at a level that allows it to monitor game state and provide upcoming content. At an early stage in the project, I investigated the availability of open-source engines, particularly in Java, but found none that were simple enough to not require a large time investment to learn the codebase, and that were structured in a way that would not prevent the IPCG system from working well. Had I found one, it would have saved time designing and implementing the game base module, albeit at the cost of having to integrate the design and implementation of the IPCG system with potentially complex existing code. However, none of the open-source engines that I found were usable, and as the game base was intended to be deliberately simple it appeared it would be quicker to write it myself [see the physics mistake]. However, I later discovered that by looking simply for an open source engine, I had missed Infinite Mario by Markus 'Notch' Persson - an open source 2D platformer engine with inbuilt PCG system. If I had used that as an original code base, I would have been saved from many non-relevant project difficulties. [TODO discuss: Be more specific with requirements. Too much ambiguity leads to ambition and then disappointment.]

Generality mistakes: Modules too general: designed and intended to operate fully independently. Unrealistic and unnecessary Initial research too general: many PCG techniques not directly applicable to 2D platformer Collision too general: arbitrary angles Location too general: ended up constricted to tiles Requirements unintentionally general: vague.

The physics mistake: writing the physics for a 2D platformer, even one as restricted as this has ended up being, turns out to be decidedly non-trivial. There are so many possibly variations in terms of realism, amplitude and sensitivity. Initially followed the Sonic Physics Guide by Mercury (`http://info.sonicretro.org/Sonic_Physics_Guide`), which was recommended as a good starting point. In terms of the game base alone, this was fine as it resulted in solid, satisfying physics. However, turned out to be taking separation of concerns too far, as the nature of the physics and freedom of movement has a significant impact on the evaluation of challenge of a particular jump or obstacle. Ended up stripping out a lot of the more advanced features such as acceleration and variable jump responsiveness, in order to simplify the challenge evaluation.

Things could have done differently: used the infinite mario codebase. Been more specific with requirements

Project Management and Planning
o You should account for your time
this is the major expense for most projects
o Compare your initial plan with how things actually went
perhaps include project diary as an appendix
o If you fell behind how did you catch up or decide which features to drop
o Did you consider and allow for risks illness, equipment failure or delays

Majority of the time spent in the initial reading phase on very broad subject background.

First semester was more directed research, design work, interim report and what could loosely be called preparatory activities.
Second semester moved onto implementation, to some extent according to the second gantt chart
Did not allow time for: ethics submission. Player feedback implementation. Submission system implementation and difficulties. Navely separated modules. Timetabled testing as a separate activity: due to nature of agile methodology, testing was a continuous and integrated process.
Definitely fell behind. Did not *exactly* end up dropping features: due to the unhelpful vagueness of the initial requirements, all of the functionality that was cut were investigatory topics that were not initially specified. The worst result from the lack of time was that the participant feedback is necessarily also fairly simple, reducing the number of interesting conclusions that can be drawn from it to simply whether or not the system works. If I had allowed more time for the wider testing phase, it would also have been interesting to perform a second study with a more accurately-calibrated system using the feedback from the first study, although possibly I will still be able to do this in time for the viva. At the time of writing, I thought that the plan was fairly generous. I was also fortunate in that I had no other academic coursework commitments during the second semester, and so believed that I would be able to stick to schedule uninterrupted. However, this proved difficult in the face of non-academic commitments, and was further hindered by the fact that the direction[] of actual development did not match the clean separation indicated in the original plan, leading to the use of ad-hoc milestones under the agile system. I believe that this flexibility was ultimately necessary, as making the individual modules general and robust enough to match the separate implementation indicated in the schedule would actually have taken far more time than the schedule allowed for. However, this same flexibility made it more difficult to reconcile the [actual] milestones with those indicated by the schedule, making it difficult to judge progress. Scheduling aside, equipment failure would have been mitigated by the use of a cross-platform programming language, IDE and report-writing environment, and having all files (including bibtex file for references) backed up and under version control, allowing development to continue close-to-seamlessly on another machine. In retrospect, one weakness of this approach is the number of bookmarked resources and potential references that would have been lost if equipment failed. Scheduling allowed for a sensible mix of project work and other non-academic commitments, however expectations in this area proved unrealistic. During the research phase, ongoing reading took place alongside other commitments. During the implementation phase, it proved much more difficult to maintain continuous application to the project, partially due to the necessary wind-up time needed for effective coding, in comparison to the relative ease of quickly reading one or more potentially relevant papers. This resulted in the most effective coding taking place during irregular sprints - ideally suited to the agile methodology, but resulting in time lost to re-acquaintance overheads.

Learned how different this report/project is from typically available academic literature. Almost all papers abstract away the development work (apart from key algorithms and techniques) and write instead only about results and conclusions. This leads to inaccurate beliefs about the areas of challenge in the project.

7.2 Gantt Charts

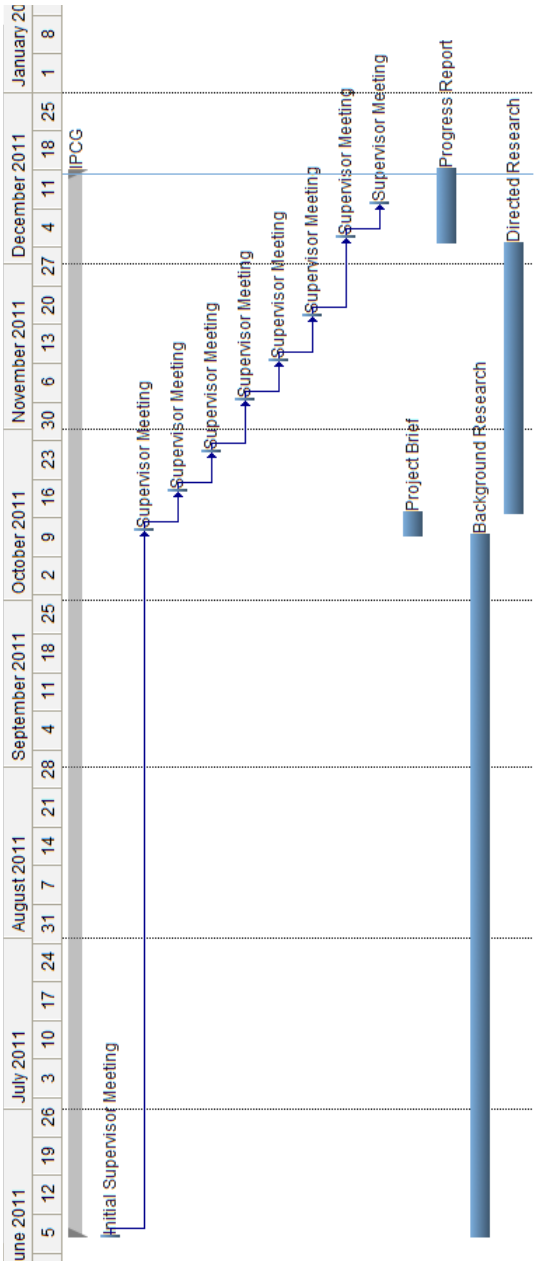

Figure 7.1: Work Completed

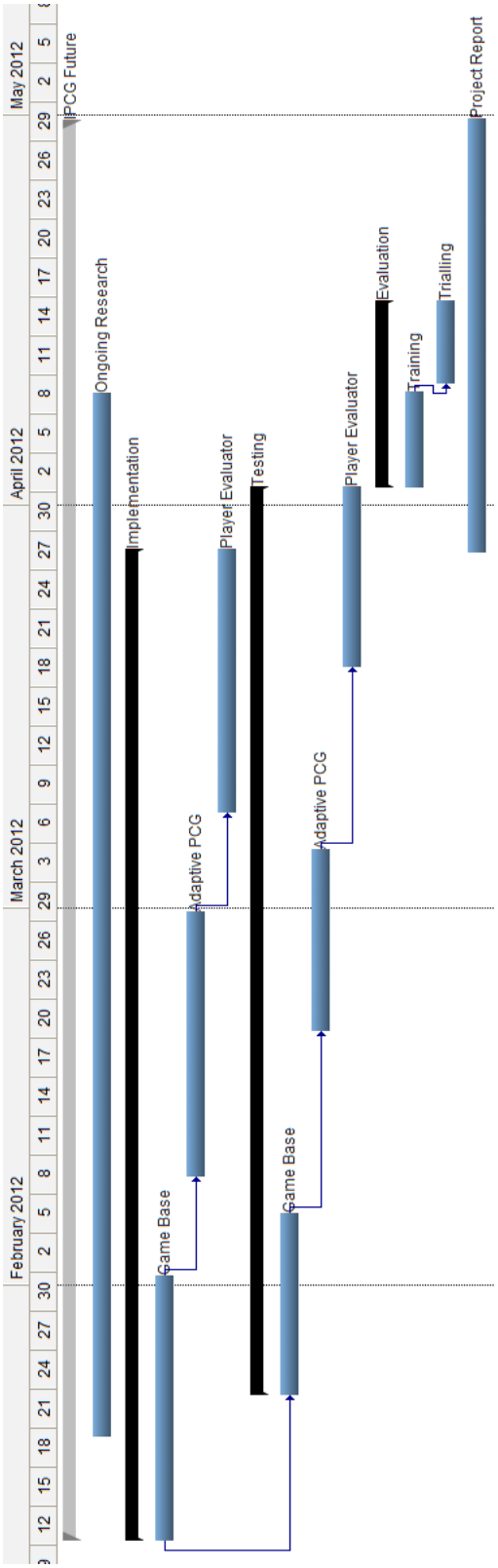

Figure 7.2: Work Remaining

8 Conclusion

Overall, the project has met (the majority of) its stated goals, and so can be considered to some extent successful. However, in many ways more valuable have been the lessons learned from the mistakes made during the project. Certainly the most damaging mistake, made repeatedly, though in different contexts, was the desire for, specification of and implementation of overly general systems, without regards to the practicalities of the problem at hand

8.1 Future Work

8.1.1 Possible Extensions

One drawback of the system as proposed is that it must still reduce all of the data about the player's performance into a single discrete decision: [TODO: make this clear earlier when talking about the evaluator] whether to continue generating the level at the current difficulty, increase the difficulty, or decrease it. There is no opportunity for granularity representing player aptitude at a particular type of challenge. One possible extension would be to divide the obstacles by type (stationary hazard, timed hazard, projectile etc.), and evaluate the player's skill on particular classes individually, then use this more detailed model to inform a slightly more sophisticated PCG module. This approach would be an ideal candidate for a collaborative filtering algorithm, which with a large enough dataset would further allow the system to predict a player's aptitude at obstacle types that had not yet been seen by that player. [IGNORE: citations testing] [30]

[28, 15, 13, 23, 10, 7, 9, 19, 3, 2, 4, 21, 20, 11, 26, 17, 14, 5, 18, 24, 30, 31, 25, 1, 12, 8, 6, 27, 22, 29][16]

8.1.2 Related Areas

8.2 Summary

References

[1] M. Booth. The AI systems of Left 4 Dead. In *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '09)*, Stanford, CA, October 14 - 16, 2009.

[2] D. Braben and I Bell. *Elite*. (Armstrad CPC), Acornsoft, 1984.

[3] Darryl Charles and Michaela Black. Dynamic player modelling: A framework for Player-Centered digital games. In *Proceedings of International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, November 2004.

[4] J. Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, April 2007.

[5] K. Compton and M. Mateas. Procedural level design for platform games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE)*, 2006.

[6] A. de la Re, F. Abad, E. Camahort, and M. Juan. Tools for procedural generation of plants in virtual scenes. In *Computational Science – ICCS 2009*, pages 801–810. Springer Berlin / Heidelberg, 2009.

[7] A. Denault and J. Kienzle. Avoid common pitfalls when programming 2D graphics in Java: lessons learnt from implementing the Minueto toolkit. *Crossroads*, 13(3):7–7, 2007.

[8] A. Doull. The death of the level designer: Procedural content generation in games. *ASCII Dreams*, Jan 2008.

[9] Gearbox Software. *Borderlands*. (Xbox Game), 2K Games, 2009.

[10] E.J. Hastings, R.K. Guha, and K.O. Stanley. Automatic content generation in the galactic arms race video game. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(4):245–263, 2009.

[11] E.J. Hastings, R.K. Guha, and K.O. Stanley. Interactive evolution of particle systems for computer graphics and animation. *Evolutionary Computation, IEEE Transactions on*, 13(2):418–432, 2009.

[12] R. Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM, 2005.

[13] R. Hunicke and V. Chapman. AI for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI Workshop*, pages 91–96, 2004.

[14] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin. Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 11. ACM, 2010.

[15] R. Lopes and R. Bidarra. Adaptivity challenges in games and simulations: a survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):85–99, June 2011.

[16] P. Mawhorter and M. Mateas. Procedural level generation using occupancy-regulated extension. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 351–358. IEEE, 2010.

[17] O. Missura and T. Gärtner. Player modeling for intelligent difficulty adjustment. In *Discovery Science*, pages 197–211. Springer, 2009.

[18] B. Nesmith. Radiant Story: Dynamically created content in The Elder Scrolls V: Skyrim. Available: http://www.gamedesignexpo.com/panel/2011/radiant-story-dynamically-created-content-in-the-elder-scrolls-v-skyrim/, Jan 21 2012.

[19] M. Nitsche, C. Ashmore, W. Hankinson, R. Fitzpatrick, J. Kelly, and K. Margenau. Designing procedural game spaces: A case study. *Proceedings of FuturePlay*, pages 10–12, 2006.

[20] M. Persson. *Infinite Mario Bros!* (Online Game), Mojang, 2008.

[21] A. Robinson. Gearbox Interview: Randy Pitchford on Borderlands' 17 Million Guns. *Computer and Video Games*. Available: http://www.computerandvideogames.com/220328/interviews/gearbox-interview/, July 28 2009.

[22] N. Shaker, G.N. Yannakakis, and J. Togelius. Towards automatic personalized content generation for platform games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press*, 2010.

[23] A.M. Smith, C. Lewis, K. Hullett, G. Smith, and A. Sullivan. An inclusive view of player modeling. In *Proceedings of the 6th International Conference on Foundations of Digital Games (FDG 2011)*, 2011.

[24] G. Smith, M. Treanor, J. Whitehead, and M. Mateas. Rhythm-based level generation for 2D platformers. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 175–182. ACM, 2009.

[25] G. Smith, J. Whitehead, and M. Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.

[26] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha. Launchpad: A rhythm-based level generator for 2D platformers. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(1):1–16, March 2011.

[27] N. Sorenson and P. Pasquier. Towards a generic framework for automated video game level creation. *Applications of Evolutionary Computation*, pages 131–140, 2010.

[28] N. Sorenson, P. Pasquier, and S. DiPaola. A generic approach to challenge modeling for the procedural creation of video game levels. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):229–244, 2011.

[29] J. Togelius, E. Kastbjerg, D. Schedl, and G.N. Yannakakis. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, page 3. ACM, 2011.

[30] Wichman G. Arnold K. Toy, M. and J Lane. *Rogue.* (PC Game), A. I. Design, 1980.

[31] D Yu. *Spelunky.* (PC Game), Mossmouth LLC, 2009.

Appendix A: Project Brief

Intelligent Procedural Content Generation for Computer Games

Thomas Smith
Supervisor: Enrico Gerding

Problem: In modern computer game development, content production accounts for a large proportion of the initial (and in some cases, ongoing) outlay. As both budgets and in-game worlds get larger, there is increasing demand to offload some of these production efforts to automated systems. The concept of procedural content generators (PCG) has been around for some time, and they have been used in many successful games, but many of the advantages made available by these systems have so far been largely overlooked. When content is generated manually or algorithmically during the design phase of a game, it can only be created according to the designer's expectations of the players' needs. By instead generating content during the execution of the game, and using information about the player(s) as one of the system's inputs, PCG systems should be able to produce more dynamic experiences that can be far more tailored to enhance individual player's experiences than anything manually created. An intelligent procedural content generator (IPCG) should therefore consist of two parts: some means of evaluating (some aspect of) the player and generating a model, and a PCG system that is able to accept this model as an input and dynamically generate variants on its standard output based on the contents of the model.

Goals: As specified above, an intelligent PCG should consist of two subsystems: an evaluator and its companion generator. The aim of the project will be to create a simple game-like application that uses an IPCG system to produce dynamically variable content based on the player's behaviour. I will begin by creating a variable PCG that is able to produce content based on specimen player models, and then use environments created in this way to create and tune a player evaluator for further generation.

Scope: In order to attempt to ensure that the project goals remain achievable, the scope should be restricted to the simplest possible system. Based on initial inspection of the problem space and existing literature, it appears that this would be adjustment due to player skill in a 2D platforming environment. The project will be coded in Java, as that comprises the majority of my recent coding experience, and it has a wealth of 2D graphics drawing support which will simplify the less-relevant areas of coding. Similarly, many of the peripheral components traditionally included in computer games are irrelevant to the project and will not be needed.

Appendix B: Questionnaire Script

These questions will be presented to participants during the course of the experiment:

1) At the start of the experiment, rating agreement on a 5-point Likert item:

I am familiar with videogames in general.

strongly disagree		neutral		strongly agree
○	○	○	○	○

I am familiar with 2D platforming games.

strongly disagree		neutral		strongly agree
○	○	○	○	○

2 - 5) After each of the four interactive sections, evaluating the previous section on a 7-point Likert item:

The previous level presented a level of challenge that was:

too easy			about right			too hard
○	○	○	○	○	○	○

6) At the end of the experiment:

This data has been collected during the course of this session:

```
< collected data >
```

Submit >>

Appendix C: DVD Contents

- A software project, or a project with a significant software component should include a CD or DVD at the back of the report with the sources and executables. A short table of contents for the CD/DVD should be included as a printed appendix. TODO: once files are finalised. Possibly create a python script for this?