Statement of Originality

Aside from any part specified otherwise, this implementation and report is all my own work. It is based upon my own ideas born out of a particular interest in the field. Some elements of the design have been strongly influenced by existing work – specifically the use of a probabilistic grammar for level generation, and the use of a one-dimensional support vector machine to calculate a decision boundary between two non-linearly-separable groups – and this has been made clear in each respective section. I have also made extensive use of the standard Java packages.

1 Project Description

1.1 Introduction

Intelligent procedural content generation is a technique that may be used during the execution of a computer game. It involves using machine learning algorithms to intelligently distil information about the current state of the game into a form that may be used to affect ongoing procedural generation of content. 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.

Due to the increasing demand for both detail and variety within computer game environments, various aspects of in-game content are now increasingly generated procedurally (i.e. algorithmically rather than manually), using techniques that are frequently simply refinements of algorithms used in the early days of computing, by games such as Elite [2] or Nethack [20]. However, one of the strengths of modern procedurally generated content (PCG) is that within reasonable limits it may be performed during runtime, allowing it to also use information about the player or current game state 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), in contrast to merely 'adaptive' procedural content generators which respond to specific parameters, or non-interactive PCG systems. 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*" [31]. 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 are used widely for varying purposes.

IPCG is an extension of typical PCG in that it will form a model of some aspect of the player or game state, and then use this as an input to an adaptive (parameterised) PCG system that modifies the generated output appropriately. Often this is done for the purposes of dynamic difficulty adjustment (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 [19]. For the purposes of this project, a prototype will be developed which uses IPCG for DDA in basic 2D platformer levels.

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 [15].

1.2 Goals

The key goals of this project are to investigate the current state of intelligent procedural content generation (IPCG) in computer games and academia, and to develop an example prototype that demonstrates each of the major components of an IPCG system.

- **Investigate the use of IPCG in computer games:** To date, remarkably few 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 techniques with more extensive bodies of existing literature and successful systems.
  - **Existing commercial products:** A small number commercial products exist using IPCG systems, and many more are available that demonstrate use of PCG or DDA. Investigating these products will help to define the field of IPCG, and inform the development of the prototype.
  - **Existing academic literature:** Again, many academic papers have been written on the topics of PCG and DDA, and comparatively fewer about systems that could be described as IPCG. In general those that do cover the topic approach it as a more powerful form of DDA, overlooking the potential for other uses.
- **Construct a minimal prototype:** In order to demonstrate the use of an IPCG system, it should be possible to develop a basic prototype that implements all of the required functionality of an IPCG system, and provides results that it would be difficult to otherwise obtain without using IPCG.
  - **Specify requirements:** As with any software project, requirements will inform the development process and should serve to ensure the quality and achievability of the end product.
  - **Design and Implement:** Following a typical software development cycle, a system fulfilling the requirements of both the project and the wider IPCG definition should be developed.
  - **Test and Evaluate:** In order to evaluate the effectiveness of the system, one or more user studies will be necessary to investigate the general applicability of the solution, and highlight potential areas for further development.

1.3 Prototype

From investigation of existing literature, it can be seen that few demonstrations of successful IPCG exist. Following the efforts by Hunicke to establish the "basic

design requirements for effective dynamic difficulty adjustment" [12], this prototype will build upon those requirements and extend the concept into the field of IPCG. 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. 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. 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 Overview

This report will begin by investigating work in related fields, specifically PCG and DDA, as many of the techniques used for content generation and game state modelling are directly transferable to IPCG. Examples of both commercial systems and academic research will be used to illustrate certain features or ideas, and the concepts will be developed in a similar section on IPCG. This background will provide a starting point for specifying the requirements for the prototype system, which are necessary to define the scope of the desired system. These will then be developed into a detailed design, from which it should be possible to create the IPCG prototype. An account of the implementation will be followed by details of the system evaluation process and an analysis of the data collected. The report will close with an evaluation of the project as a whole, as well as the conclusions that can be drawn and suggestions for future work.

2 Project Background

Though Intelligent Procedural Content Generation is the main focus of this project, in order to understand the approaches taken towards developing IPCG systems it is important to investigate the supporting work in the fields of PCG, and to a lesser extent DDA, which are closely related. An IPCG system cannot work without some mechanism in place to generate content appropriate for the model of the player that it has constructed, and the techniques and systems developed for the field of DDA are often relevant when attempting to construct this model - even if the purpose of the system is customisation of a game aspect completely unrelated to challenge. In contrast to the minimal body of academic literature concerning IPCG, both PCG and DDA are mature fields with plenty of research papers and example systems, and many commercial game products use these techniques.

2.1 Procedural Content Generation

Procedural Content Generators in various forms 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 be 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 [31]. 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 [21]. However, with the further progress of technology attention has returned to procedural generation. Modern game worlds typically contain vast amounts of detail, and procedural content generation algorithms are ideally suited to producing this by generating 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 require large data files, but by defining specific sub elements and assembly rules, a small effort can lead to hundreds or thousands of variations. 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].

2.1.1 Existing PCG Systems

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

- **Infinite Mario:** A Java reimplementation of the original Super Mario, Infinite Mario uses PCG to create an endless variety of potential levels [23]. Considered something of a standard in academic PCG implementations, the codebase is notable as it is often used as a launching point for AI competitions of various kinds - both pathfinding and alternative generation.
- **Speedtree:** Possibly the most widely-used commercial PCG system in existence, 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 modern games to extensively use and publicise a non-decorative PCG system; in this instance to generate millions of different varied weapons [25] - 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 [16]. 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

From 'rubber-banding' – the simple modulation of maximum velocity by position used in racing games – to the more complex level-based adjustments used in games such as Fallout, Oblivion and Homeworld, DDA systems are widely used in modern games. The best such systems often go unnoticed by the player [18].

- **Resident Evil 5:** A successful commercial game, Resident Evil 5 contained both a traditional difficulty selection screen and a far less publicised DDA system that used information about the player's health, equipment and difficulty setting in order to adjust the properties of enemies that they faced [15].
- **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 3D game engine in order to monitor aspects of the player's status, and modifies upcoming encounters based on a historically-trained map between the result of player evaluation and the desired game world adjustments.
- **Cannon:** A system developed by Missura *et al.* in order to investigate the use of K-means clustering and SVM clustering for effective DDA. Cannon is a very simplified 2D shooter that provides minimal gameplay but a wealth of monitoring and statistics [17].

2.3 Intelligent Procedural Content Generation

Although 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" [26]. 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. Another academic IPCG system is Charbitat, which generates an entire 3D world dynamically based on the player's behaviour. Approaching the topic 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 [16], including via PCG.

2.3.1 Existing Commercial 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 the player's sense of 'flow' [4].

- **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 forces exploration of previously unknown regions, helping to increase immersion and interest by avoiding repetition of content [19].
- **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, leading to increased engagement with the game.

2.3.2 Existing IPCG Academic Literature

In contrast to the idiosyncratic uses of IPCG in games, academic examples of IPCG are less obviously differentiable from the DDA and PCG systems that they have developed from. Papers these fields are in some cases converging towards describing IPCG, but only as extensions to existing techniques, rather than entirely new systems.

- **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 [27], 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].
- **Charbitat:** Charbitat is a prototype game system that generates an entire play space procedurally, according to the play style of the gamer. Space generation is a deliberate feature of the player's interaction with the system, though in accordance with the definition of PCG [31] it is performed only indirectly thorough analysis of the player's behaviour and in-game choices.
- **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 has been 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 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 is increasingly suited to online adaptation [16].

3 Proposed System

This section of the report provides a specification for the prototype IPCG demonstration system. Necessary components are discussed, and an approach suggested which is intended to result in a working system within the timeframe. The choices and assumptions made are justified, and a set of functional and non-functional requirements are listed in order to guide the development and evaluation of the system, as detailed in later chapters.

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 IPCG system may easily be modularised into three principle components: a 'host' or base module, an adaptive PCG system, and an intelligent procedure for taking data about the game state and converting it into a model usable by the PCG. The proposed flow of information is shown in Figure 3.1.

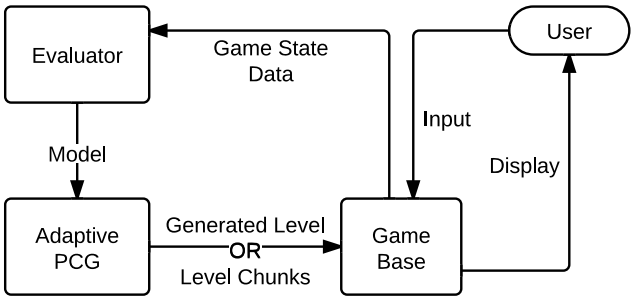

Figure 3.1: Data flow in proposed system.

3.2 Modules

Based on the divisions as defined above in Figure 3.1, each of the components may be developed as follows:

3.2.1 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 any information about the participant's performance necessary for the evaluator module.

3.2.2 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). 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 [28]. 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.

3.2.3 Evaluator

The second portion of the IPCG system should provide the capability to intelligently classify the player's performance in some way. 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 then acted upon in order to generate further suitable content.

3.3 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.

3.3.1 Testing

In order to ensure that the modules interact as expected, it will be important to define the interfaces between them and then use either test driven development (TDD) or unit testing in order to ensure that the modules conform to the specifications as expected. Integration testing will also be useful to check that the modules interact properly during execution of the final program.

3.3.2 Classifier Training

The development of the classifier will initially involve 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.

3.3.3 Evaluation

In order to perform retrospective evaluation of the system, some facility for users to provide deliberate feedback outside of the player modelling system will be necessary. Particularly after each period of interaction with the system, an opportunity to give feedback will provide relevant data, and it will be useful to also gather contextual information such as a participant's general level of familiarity with computer games.

3.4 Justification

The specification and approach as detailed should result in a working IPCG system, thereby fulfilling one of the goals of this project. Each aspect of the design has been chosen in order to produce the simplest possible demonstration prototype. For example, an alternative approach to the problem of generating a 2D platforming environment for use with DDA is presented by Sorenson *et al.*[29], 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 fulfill the requirements given, and demonstrate intelligent variation in output based upon the skill of the player. Initial research shows that much PCG, DDA and even some IPCG literature is applicable to the problem of 2D platforming. 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. However, even more basic game environments exist - for example 'Cannon', the very simple arcade style 2D shooter developed by Missura *et al.* [17] for the development of machine learning algorithms and player modeling for intelligent difficulty adjustment. Unfortunately, due to the natures of many of the simpler genres it becomes more difficult to modify the game environment in manners that are more obviously IPCG than DDA.

3.5 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 features that the system must provide. The majority of them are generic to any IPCG system, as this project intends to cover that area specifically, and refinements particular 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.

3.5.1 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.** Using machine learning algorithms, it should be possible to process the monitoring input in some reliable manner, in order to retain the most useful aspects of the data while minimising surplus information.
- **Form a model of some aspect of the player.** The final form of the collected data should be a representative model that provides useful information about the player, and gives a clear indication of the necessary next actions by the IPCG system.

– **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 will be useful to obtain 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.
– **Run on multiple operating systems and be widely available.** To ensure that the largest number of potential participants are able to use the system, it will need to be cross-platform and easy to access. Excessive size, acquisition difficulties or specific system requirements will all limit potential participation.
– **Return information to a central storage point.** In order to recover information collected by the system, it will need to be capable of sending data back to a server once each session is finished. This can then be stored, collated and analysed in order to aid in system evaluation.

3.5.2 Non-Functional

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

– **be written in Java.** The Java language has several properties which make it attractive for this project. It is supported on multiple platforms, and is capable both of launching from a website and then later communicating via http GET and POST requests. It also has a large number of built-in libraries that support basic drawing and user interface widgets, reducing the burden of these portions of the implementation.
– **be presented as a basic platformer.** Though IPCG can be applied in some fashion to most genres, adjusting jump and obstacle placements is likely to be one of the most basic applications that demonstrates convincing adaptability according to the output of the IPCG system.
– **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 minimise the complexity of a platforming game 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 reduced.

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

– **consistently maintain the challenge of generated content.** Rather than generate content precisely matched to the participant's abilities, the system should provide challenges within a narrow range of difficulties centred on the target challenge level. This will ensure that the user will experience a variety of levels of challenge, including also continual opportunities to attempt content harder than that completed so far.
– **conform to established gaming conventions.** In order to avoid confusing or alienating users that are already familiar with computer games of the same genre, the IPCG system should avoid breaking commonly understood conventions such as the goal existing at the far right of a level, or falling off the screen resulting in death.
– **remain responsive.** To avoid frustrating participants it is important that the system should not noticeably slow or become unresponsive during the execution of the program.

4 Detailed System Design

Following the problem specification and requirements given in the previous chapters, this section provides a more in-depth design for each of the components that will be necessary to the execution of the final system. After looking at the intended high-level structure and the development approach, each module is detailed with observations about its specific challenges and interactions with the rest of the system.

4.1 Overview

In order to facilitate both presenting an IPCG system to the user and also recording feedback about that same system, the final program will need two main classes of interface: game segments that allow real-time interaction while the evaluator and generator are running, and a selection of feedback and instruction interfaces that enclose the interactive sections. In terms of high-level Java architecture and object types, the system should be written as a Java Applet using the Swing interface toolkit. This has the advantage that it will be able to be run online, making both distribution to participants and collection of feedback easier. The game segments can be drawn on Canvas objects using active rendering [7], and JPanels containing Swing widgets can be used for the feedback and instructive segments. The Java Swing toolkit will facilitate swapping between these two types of elements.

4.2 Development Strategies

Several basic software development approaches will be useful during the course of the project. Primarily, the class of software engineering process models known as 'Agile' methodologies will help to keep the development process predictable and manageable. Agile methods follow an iterative cycle that promotes short sprints resulting in incrementally more functionality. They favour continual analysis of situation and workload, and help to ensure flexibility in planning. Another key component of the

Agile philosophy is the use of a robust version control system and issue tracking, as alongside continuous integration and regression testing, these can ensure that a runnable version of the deliverable is always available, and the next incremental version is thoroughly specified. Though the techniques are typically used by small development teams, there exist modifications geared to solo developers [22].

4.3 Game Segments

Interactive game segments form the link between the user and the IPCG system. While the user is playing, the system is continually monitoring the game state and generating new content as necessary. The components that make up the game segments each perform specific roles within the system, as detailed previously.

4.3.1 Game Base

The largest module within the system, the game base should be responsible for all user-facing aspects of the segment. Following the Model-View-Controller architecture, it can be further split into these separate components.

- **Model:** The model contains all of the data about the current game state, such as platform and player locations. In order to facilitate rendering, collision detection and challenge rating, this data can be stored within a scene-graph-like object hierarchy, as used in computer graphics. Individual platforms, gaps and obstructions are *parts*, and an obstacle (*component*) such as a jump or wall is a particular arrangement of a small group of parts. Components are built up into repeating *patterns*, and multiple patterns are owned by a single *level* object, which also owns the *player* object and various useful metadata - this structure is based upon that used by [5].
- **View:** The view performs all rendering within the system. The view class should own the Canvas that is displayed on screen, and use double-buffering to smoothly render the correct portions of the model for the player. Ideally it should also be capable of rendering useful debug information for testing purposes.
- **Controller:** The controller handles all interaction between the user and the rest of the system, and also all changes that occur to the model over time. it is also responsible for passing the game state to the evaluator when necessary, and updating the model with the output from the generator. Once the model has been altered appropriately, the view will render it to the screen for the player.

4.3.2 Generator

The generator module will rely heavily on the ability to calculate a specific difficulty rating for each obstacle based upon the properties of the physics system within the game. Using the kinematic equations of motion for particles with constant acceleration, each component will have a rating, and the *level* and *pattern* collections will each be able to produce a rating as a function of their children. When generating new content, the aPCG should be able to take a desired difficulty rating and produce a new component or pattern that is within some acceptable margin of the target rating. If the available components are ranked by difficulty then this can be done by selecting a random normally-distributed value, where the mean of the distribution

is the desired rating, and the variance is some appropriate value based upon the acceptable margins. The closest valid component to this new value can then be added to the existing model.

4.3.3 Evaluator

The evaluator module must continually monitor the game state, and use intelligent algorithms in order to process the information it receives into a form usable by the generator. The evaluator may perform regression or classification statically after being trained on a previously collected data set, or there are a number of incremental learning algorithms that would be capable of updating themselves with each new piece of data from the participant. A selection of metrics that the monitor could use:

- **The rating of each obstacle the player fails**
- **The rating of each obstacle the player completes**
- **The number of player successes**
- **The number of player deaths**
- **The overall velocity of the player**
- **The time taken to complete the level**
- **The length of the level**

If a sample data set is available, then one approach would be to discretise it into a number of groups using the K-means algorithm, and then train a multiclass SVM classifier on these groups. Classification during runtime would then consist of checking the current game state against the trained SVM.

4.4 Feedback Interface

In addition to the game segments presenting the IPCG system, participant feedback screens will be necessary in order to collect non-observational data concerning the participant's context and thoughts about the system. 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. Once the information has been collected, it will need to be sent back to a central location for later analysis.

4.4.1 Panel Layouts

Four distinct custom panel layouts will be needed:

- **Plain text panel.** Used for the welcome screen with a brief overview of the experiment. Re-used for the interaction instructions, and then again after the final submission confirmation, for the thanks screen.
- **Questionnaire panel.** Initial mini-questionnaire containing two labelled five-point Likert items to query the participants' familiarity with games and 2D platform games.
- **Level evaluation panel.** 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 submission panel.** The penultimate screen in the system. The submission panel should provide a non-editable textbox displaying all of the data to be submitted, and a final 'Submit' button for confirmation.

4.4.2 Logging and Submission

To collect data useful for evaluation and future calibration of the system, all of the player feedback and evaluator input and conclusions will need to be logged. A YAML-like simple parseable format should be sufficient to represent the types of data needed, in a human-readable format without excessive overheads. On the penultimate screen of the system, 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 a centralised server, which will save the data to a file. In case of error, the system may also attempt to save the data to a local file, and provide instructions on how to upload the file anonymously.

4.4.3 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 a centralised server. 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 sending POST data from the Java Applet has failed, the script will also be linked to by a page that allows data file upload though a plain HTML form.

4.4.4 Web Interface

To allow potential participants to access the applet, it will need to be available online. However, in order to comply with Ethics requirements the system must not be accessible to users that have not viewed the participant information sheet and completed a consent form. The simplest way to enforce these requirements will be to present the system via a .php script that restricts access until consent has been given.

4.4.5 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

The design, requirements and context detailed by the previous sections provided a useful starting point and guide for the implementation of the system; however during the coding phase of the project certain inadequacies in the original design became clear. This section presents the components of the system in the same order as previously, and notes for each any significant choices made or problems encountered during the creation of that aspect.

5.1 Overview

As the base visible component for the IPCG system (the game base's view's Canvas) already exists as a UI widget, placing it within the applet was done in the same manner as adding a JPanel or any other component. However, initialising the level required wrapping both the game segments and the feedback UIs in a minimal custom Screen interface. Switching between the two views also caused difficulties due to the way that Java handles contentPanes.

5.2 Development Strategies

An Agile methodology was used throughout the project, with regular milestones and refactoring, and the aim of always having a working prototype at the end of each coding session. The overall system development intention was initially to follow a waterfall approach with regards to completing each module before moving onto the next; this proved to be fairly optimistic and naïve, as the modules were not fully independent. Defining common interfaces for the modules to share also proved difficult, as in many cases it was initially unclear how best to pass data between the classes, and so the boundaries remained fluid. While greater independence between the modules would be desirable, there were practical limits to the amount of time available, and so development of the three modules occurred largely simultaneously. Fortunately, the flexibility provided by the use of Agile methods meant that the project schedule was able to accommodate this fairly major alteration to the plan and adjust to the parallel development method instead.

5.3 Game Segments

As a result of the way the modules in the game segments interact, the final architecture can be seen as an MVCVC system. That is, in MVC terminology, the model is shared through the system, and then there are two views and two controllers. Within the game base, the 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 controller within the game base, which applies gravity and collision checking

and – when necessary – 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 secondary view, which queries the model using the GameObject's `rate()` function, and calculates a classification for the current game state. When the generator is used, it accesses the classification and uses it as input to the aPCG, which can be seen as a secondary controller that modifies the model under certain conditions.

5.3.1 Game Base

The development of the game base module started along similar lines 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, the next step was to begin implementing the hierarchical model described in [5] using a pseudo-component model, via interfaces such as `Drawable`, `Rateable` and `Collideable`, each containing a static collection that kept track of all of its members. However, when it came to implementing collision and debug rendering, the similarities between the hierarchical model used in the paper and the traditional scene graph pattern as used in computer graphics became apparent. This led to refactoring the internal data structures to follow a more traditional inheritance model.

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.

```
package Model;

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_l);

        public abstract Object clone(boolean placeAtEnd);

        public abstract int 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 redirect the same function to 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> .`

Controller

One unfortunate mistake made during the creation of the controller module was the early implementation of a physics engine too realistic to work well with the IPCG system. Writing the physics for a 2D platformer, even one that is deliberately as simple as for this project, turned out to be more of a challenge than originally anticipated. Initial research led to the 'Sonic Physics Guide' by Mercury[1], which was broadly recommended as a good starting point. In terms of the game base alone, this resulted in solid, satisfying movement. However, the more complex nature of the physics had a significant impact on complexity of the evaluation of challenge for a particular jump or obstacle. As the project goal is to create a simple, minimal demonstration a lot of the more advanced features – such as horizontal acceleration and variable jump responsiveness – were removed, in order to simplify the challenge evaluation.

5.3.2 Generator

The generator module works largely as described in the design, with the exception that during the final stage rather than always picking the 'nearest' obstacle to the target, it picks between the two nearest randomly in proportion to their distance. One issue encountered was the non-uniqueness of an obstacle's rating; this necessitated a change of data structure and partial rewrite of the algorithm in order to allow for multiple obstacles with the same rating. The method for placing the level end platform also proved difficult to fine-tune, and so in the final version it is automatically generated whenever the level exceeds a certain length.

[1] `http://info.sonicretro.org/Sonic_Physics_Guide`

5.3.3 Evaluator

Of all of the program components, the design for the evaluator changed most during the implementation phase. The starting intention to train an SVM on a large dataset classified via K-means was dependent upon the existence of a large dataset; however without a calibrated evaluator it would not be possible to run the study needed to acquire the data needed. The obvious conclusion would be to run a pre-study using a different machine learning algorithm that did not need large-scale training. Further research into this area indicated the feasibility of using a one-dimensional SVM to classify a player based upon the difficulty of the obstacles that they succeed or fail at [30]. By plotting the difficulty of failed obstacles against the rating of successfully completed ones, build two classes that are likely to be non-linearly-separable. However by using an SVM with slack variables, it is possible to calculate an optimal separating plane. This represents the perceived upper boundary of the participant's ability, and so by using this as the target input for the aPCG content can be generated that should challenge but not frustrate the player.

One major disadvantage of using a one-dimensional SVM compared to the original trained SVM design is that the 1D SVM must be retrained with each new piece of data, and this requires both processing power and space to store all of the previous training examples from the same session. In comparison, a trained SVM would only require the learned support vector values and a single classification check. Fortunately, Cauwenberghs *et al.* provide an algorithm for incremental SVM leaning [3], aspects of which were easily applicable to the one-dimensional SVM in use by the evaluator. This meant that far less data needed to be retained by the evaluator, and training was potentially much quicker.

5.4 Feedback Interface

In contrast to the mathematically heavy implementations for the rendering, physics engine, generator and evaluator, the non-IPCG portions of the project were relatively simple.

5.4.1 Panel Layouts

After experimenting with alternative layouts, it became apparent that the optimal design for the Swing portions of the program reinforced the goals of the interactive segments. Typically 2D platform games are designed so that the player character moves rightwards from the beginning of the level on the left, to the end of it at the far rightmost edge. In accordance with the non-functional requirement to avoid breaking videogame conventions, the IPCG segments of the program follow this tradition, and by mirroring this left-to-right flow throughout the user interface the Principle of Least Astonishment is preserved[2].

5.4.2 Logging and Submission

A subtle error in the logging code resulted in all of the obstacle components recording a difficulty of either 622 or 568 in the submitted data during the study, rather than

[2]http://www.faqs.org/docs/artu/ch11s01.html

any one of the 7 specific lower values that ought to have been logged. Fortunately the values were sufficiently spread that it was possible to reconstruct the original ratings with a high degree of certainty, and hence repair the affected data files.

5.4.3 Web Service

A flaw in the manual submission service led to the two data files that were uploaded using this method to be stripped of all line-breaks. However, the simple format of the files meant that it was possible to restore the files using a basic regex replace.

5.4.4 Web Interface

The web interface caused the most problems out of all of the system components, due to the inconsistent support for embedding Java online across different web browsers. The final solution presented three different options for accessing the game after providing consent, to ensure that all potential participants were able to complete the study.

5.5 Testing

Test-driven-development proved useful for several important subsystems; most notably the simplified collisions, the gap difficulty rating and the generator difficulty cache, as each of these held potential border case problems. Throughout development, regular integration testing was also necessary in order to ensure that new features and functions were compatible with existing code.

6 Project Evaluation

There are two main ways to evaluate the success of this project: via the results of the user study, which indicate the effectiveness or otherwise of the IPCG system; and via critical comparison of the system and achievements to the initial requirements and goals.

6.1 User Study

In order to help evaluate the effectiveness of the IPCG system, the final program was made available online for a period of days. During that time 27 participants completed the study and submitted data, providing enough information to come to useful conclusions about the success of the system.

6.1.1 Study Design

The study consisted of questionnaire sections alternating with single levels, either pre-generated or generated by the system during operation. The first questionnaire asked about the participant's familiarity with games in general, and 2D platform

games specifically. After instructions on how to interact with the game, participants were presented with a simple pre-generated level with a specific target difficulty, and then asked to evaluate how challenged they were by the it. The process was then repeated with a more difficult pre-generated level. The system then generated a level suited to the observed ability of the player, and asked the participant to report how challenged they were. Another, more difficult level was generated, and a final evaluation performed. After a chance to review the collected data, the participants were thanked for taking part. A text copy of the interview questions is attached as Appendix B.

6.1.2 Study Expectations

If the IPCG system is effective, four statements can be made about the expected results from the user study, as follows:

- **1: Levels 1 & 2:** 'Familiar' players report less challenge than 'unfamiliar' ones. Participants that self-report a greater familiarity with games should find the same level comparatively easier. This result says nothing about the system.
- **2: Levels 1 & 2:** Most players report that 2 was more difficult than 1. If level 2 was correctly generated with a greater difficulty than 1 as intended, then this result would indicate the effectiveness of the generator module.
- **3: Levels 3 & 4:** Less variance in the reported difficulties than for 1 & 2. If the evaluator is effective then levels 3 & 4 should be closely tailored to each players' skill, leading to less variance in reported challenge.
- **4: Levels 3 & 4:** Most players should report that 4 was more difficult than 3. If items 2 and 3 hold, then the evaluator and generator are both effective.

6.1.3 Study Results

Using the participants' data, and the property of Likert items that two items on the same scale may be summed to ease comparison, the following comments can be made about the original statements:

- **1: Levels 1 & 2:** The Pearson's correlation between familiarity and challenge for the first two levels is close to 0. If greater familiarity led to lower reported challenge as expected then the coefficient ought to approach -1. However, this result is likely to be strongly affected by the fact that out of the 27 participants, only one reported a summed familiarity of less than 4, and in fact the average summed response was 6.8.
- **2: Levels 1 & 2:** The mean difficulty reported for level 2 was 25% greater than the mean for level 1; this is despite verbal feedback from several participants indicating that since they expected the second level of a game to be harder than the first, they would give the second level the same rating.
- **3: Levels 3 & 4:** The reported variance for levels 3 & 4 actually increased by 15%, indicating that the degree of fit worsened when the IPCG system was active. A number of possible causes for this result are discussed below.
- **4: Levels 3 & 4:** The mean difficulty reported for level 4 was 3% less than the mean for level 3.

6.1.4 Study Conclusions

Out of the four hypotheses that would indicate the effectiveness of the IPCG system, only one showed any positive evidence. Through observation of the results, three main conclusions have been reached:

- **1: The IPCG system is too aggressive.** In simplified terms, the system determines an optimal separation between obstacles that the player can manage, and obstacles that they cannot, and then repeatedly presents the player with obstacles from this region. However, in application, this causes problems as due to the nature of the algorithm, half of the time the player is presented with obstacles that the system believes are too hard – occasionally, considerably so. This has lead to an IPCG version of the Peter Principle [24]; participants are raised precisely to their level of failure, rather than positioned a little below it.
- **2: The pre-generated levels, and most of the obstacles, are too easy.** The majority of the obstacles within the expressive range of the aPCG seem to be too easy for most players, leaving little granularity within what might be termed the 'interesting' or useful range. Informal feedback included suggestions about altering non-obstacle aspects of the game – such as player speed or checkpoint density – in order to further increase the variety within the appropriate range of challenge.
- **3: A wider range of participants are needed to properly test the system.** The vast majority of potential study participants would have been students of Computer Science or related fields, biasing the probability that they would self-report as 'familiar' with computer games. This will have affected the results obtained from the study.

None of these problems are inherent to the IPCG system developed. If the study were to be run again after calibrating the system in light of these results, it is entirely possibly that supporting evidence could be obtained for all four original hypotheses.

6.2 Critical Evaluation

Despite the existence of calibration flaws discovered through the user study, in many other ways the system has succeed in fulfilling its requirements. The system correctly presents the user with and interactive game environment, records and evaluates game state data, and uses the resultant model to inform PCG activities. As demonstrated by the completion of the user study, it is also capable of recording playthrough and feedback data from multiple users on different systems, and returning this data for analysis. The system has also conformed to all of the non-functional requirements, though it could be argued that it pushes the challenge of generated content, rather than maintaining it as it ought.

However, despite technically fulfilling the requirements, it has not managed to demonstrate a successful IPCG system. Though it is entirely possible that the problem could be fixed through improved calibration, this does mean that one of the project's two main goals has not been fully attained. The investigation of IPCG in computer games is contained within this report, including a thorough review of academic literature relating to systems that could be described as IPCG, as well as related

technologies that are important to the development of IPCG, such as PCG and DDA.

6.2.1 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[1] 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 ensured the system was never committed to a particular way of doing things.
- **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.
- **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 the research direction honed in on PCG and DDA for 2D platformers, and was forced to remain more general for IPCG as there is comparatively little literature.

[1] `http://www.lucidchart.com/`

6.3 Gantt Charts

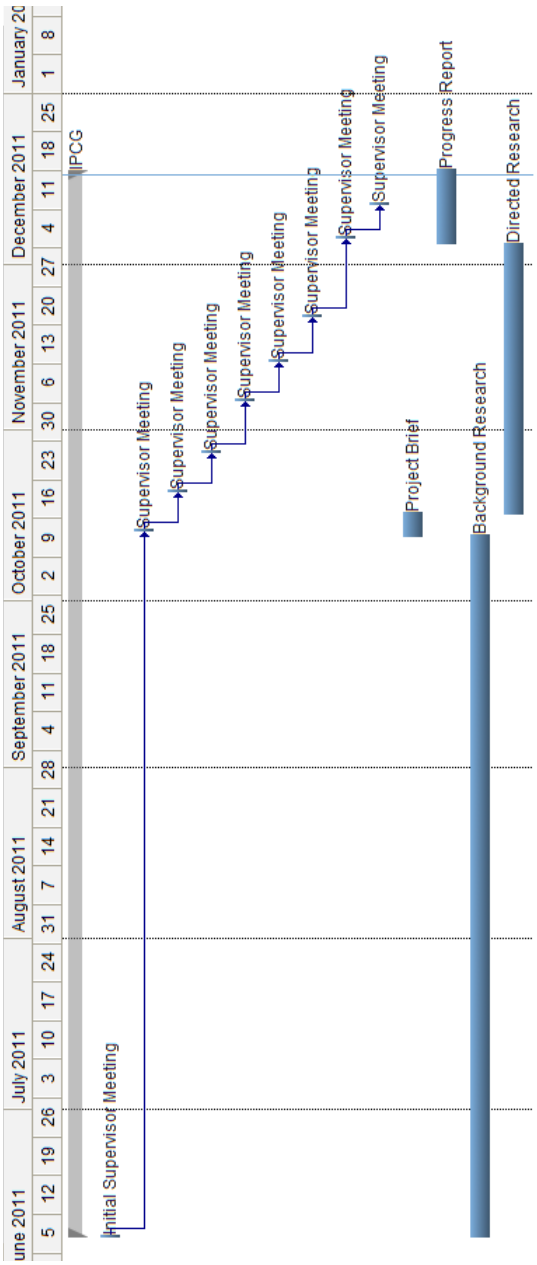

Figure 6.1: Semester 1 (actual)

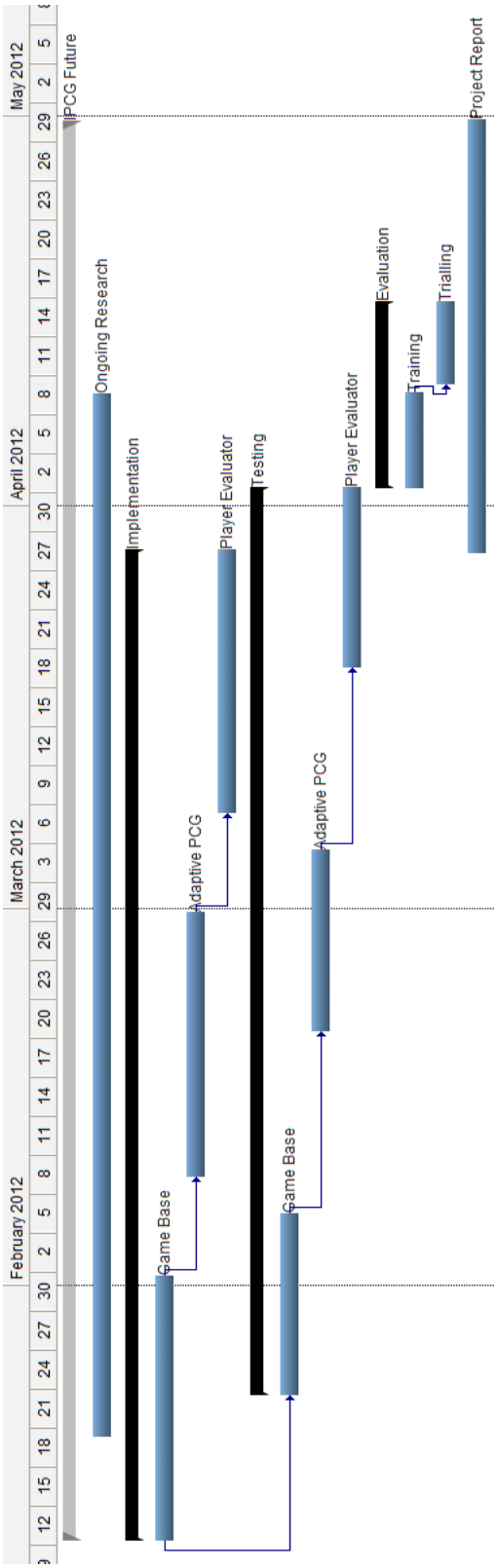

Figure 6.2: Semester 2 (planned)

7 Conclusions and Future Work

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

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: 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.

Further work on the existing system should definitely include efforts to provide more granularity at the more challenging end of the current expressive range. Further studies using the improved system will be necessary to ensure that the calibration is accurate.