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

Thomas A. E. Smith
taes1g09@ecs.soton.ac.uk
December 13, 2011

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 progress 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 generated 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 systems inputs, PCG systems should be able to produce more dynamic experiences that can be far more tailored to enhance individual players' experiences than anything manually created. Much has been written about the generation of player models 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 two fields to create an intelligent PCG system (IPCG) that is capable of monitoring players' progress and dynamically generating upcoming challenges to best suit their abilities.

Contents

1 Project Description **1**
1.1 Requirements . . . 1
1.1.1 Functional . . . 1
1.1.2 Non-Functional . . . 2

2 Project Background **3**

3 Literature Review **5**
3.1 . . . 5

4 Proposed System **6**
4.1 Overview . . . 6
4.1.1 Game Base . . . 6
4.1.2 Adaptive PCG . . . 6
4.1.3 Player Evaluator . . . 7
4.2 Approach . . . 7
4.2.1 Classifier Training . . . 7
4.3 Justification . . . 7
4.4 Possible Extensions . . . 7

5 Plan of remaining work **8**
5.1 Gantt Chart . . . 8

Bibliography **9**

1 Project Description

The aim of this project is to investigate the use of IPGC 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 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 (Acornsoft 1984). However, one of the strengths of 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 in the Literature Review and Background Research sections, IPCG techniques can be applied to many different scenarios. For the purposes of this project, the problem will be restricted to using IPCG for DDA in basic 2D platformer levels.

1.1 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. In this instance, the functional requirements are generic to any IPCG system (with refinements specific to this problem in brackets, like so), while the non-functional requirements are constraints that should serve to encourage feasibility and quality.

1.1.1 Functional

In order to properly implement IPCG, the system should:

- present the user with an interactive game environment (basic 2D platformer)
- record data on the player's behaviour (in this case, performance)
- 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 (level chunk generation)

1.1.2 Non-Functional

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

- be written in Java
- be confined to 2D
- be presented as a basic platformer
- limit the user to move and jump actions

but also:

- remain responsive
- properly maintain the challenge of generated content

2 Project Background

procedural content for many purposes (music strucure enemies textures) DDA resident evil use in existing games (valve, infintie mario) IPCG for difficulty, preference(fun), exploration

Procedural Content Generators have been used since the early days of gaming. Elite ¡one of the first great Y¿ had ¡stats¿, all procedurally generated. These technique were required, as it was not possible to store the full data about that many unique planets on the distribution media that was availabe at the time. 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 [**?**]. 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 [**?**], ¡expand¿ clouds textures 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 assmbly rules, variation can be almost endlessly reused. Not only used for cosmetics now either. Minecraft, infinite Mario?

Another facet of game design that has benefited ¡introduction to DDA¿. Previously, games had been limited to specific discrete difficulties as defined at production time. However, these (can have been) considered to be overly restrictive - typically, if a games is begun with a certain difficulty it is difficult to later change; and this also alienates players that are unfamiliar with the standards or uncertain how to classify themselves. Furthermore, since game difficulty is typically a continuous function of multiple parameters ¡stuff¿. Typically, DDA is achieved by altering values that are hidden from the player, such as enemy health, accuracy, or the amount of ammo and healthkits available in the world [**?**]. 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 overchallenged (leading to frustration), or underchallenged, leading to boredom [**?**]. As DDA systems are given more control over additional aspects of the game environment, they can begin to enter the realm of PCG, fundamentally(?) altering the stucture and pacing of the player's experience. In the game Left 4 Dead, there is an 'AI Director' that is capable of estimating the ¡intensity¿ of each player's situation, and dynamically altering the generation of enemies of various types in order to ensure that the general flow of the game is kept exciting, ¡following build up, panic and calm phases¿ [1]. In Left 4 Dead 2, the director has additional control, and is able to vary the stucture of the level (something about the placement of ammo)(possible additional cite).

Typically, PCG is used in an offline manner, taking input only from a random seed. DDA, in contrast, uses information about the players' actions as an input,

IPCG can be (and has been) used for a wide range of purposes, almost as varied as PCG itself. As one of the key challenges in game design is maintaining players' engagement with the game via enhancing immersion and controlling flow[**?**], it is unsurprising that many of the existing applications of IPCG are used for these reasons.¡¡send to front¿ 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, and manipulates the game environment in order to control pacing and maintain flow[1].

3 Literature Review

3.1

4 Proposed System

4.1 Overview

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 4.1.

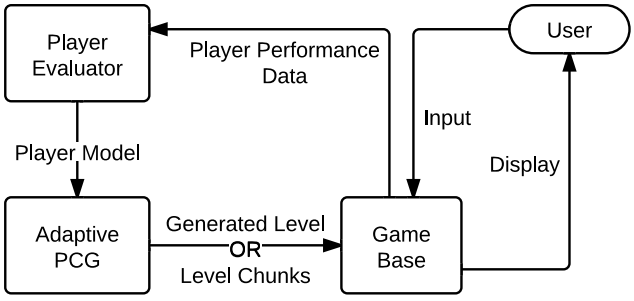

Figure 4.1: Data flow in proposed system.

4.1.1 Game Base

A simple 2D platformer engine. This module should handle all input and rendering activity, and ought to be created following the MVC architecture. 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 harvesting multiple types of information about the players performance for the Evaluator module.

4.1.2 Adaptive PCG

An adaptable 2D platforming level generator. Building on the work of ¡nop¿, this module should maintain a context-free grammar (CFG) of obstacles available, along with weights representing the estimated challenge of each element (terminal obsta-

cle 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. 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.

4.1.3 Player Evaluator

A ¡nop¿ 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.

4.2 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.

4.2.1 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 be used to train a One-vs-All SVM classifier.

4.3 Justification

4.4 Possible Extensions

[**?**] [2]

5 Plan of remaining work

5.1 Gantt Chart

Bibliography

[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. 2009.

[2] Michael Nitsche, Calvin Ashmore, Will Hankinson, Rob Fitzpatrick, John Kelly, and Kurt Margenau. Designing procedural game spaces: A case study. October 2006.