Fortress generation for Dwarf Fortress

Procedural Content Generation in Games Autumn 2014

Jakob Melnyk, jmel@itu.dk
Jacob Claudius Grooss, jcgr@itu.dk

I. Introduction

This paper was written as part of a project for the "Procedural Content Generation in Games" course on the Games Technology track at the IT-University of Copenhagen.

For the project we could do one of two things: Develop a new PCG algorithm, or apply an existing algorithm (or combination of algorithms) to some game domain in a novel way.

A. Problem Statement

For our project, we decided to generate fortresses for the game Dwarf Fortress[1] by creating random map layouts and and then evolve the contents of the maps.

Our goal was to be able to generate fortresses that could be used in-game in practice, even though they may not be perfect fortresses[2]. We decided to not aim for perfect fortresses, as Dwarf Fortress is an amazing complex game and there are an immense number of variables to take into account in order to create a perfect fortress.

II. Background

Procedual content generation has been around for a long time, but it continues to evolve and become better. As it is now, it is mostly used by game creators to create content for their games. There are, however, some games where a PCG tool will be able to help the players learn more about the game and how to do better while playing.

Stuff about pcg/evolution

A. Dwarf Fortress

Dwarf Fortress is a game about leading an expedition of dwarves in order to create a new home for them. The game takes place on a procedurally generated map with multiple layers, where the player can direct dwarves to perform various tasks (mining, gathering plants, building furniture, crafting weapons, and so forth). The goal of the game is to build a huge fortress for the dwarves and to keep them alive for as long as possible.

Dwarf Fortress is often described as having a very steep learning curve, as it tells the player nothing about how they play or what they are supposed to do. All of it is something the player has to figure out by themself. This often means that a player's first fortresses will be of very low quality, as they discover more and more things they have to add to it that they did not plan for.

In order to help the novice player out, we intend to create a fortress generator that can give them some basic layouts from which they can choose which one they like the most. These layouts should also give the player an idea of how many different things they need to play for in any future fortresses they may want to make.

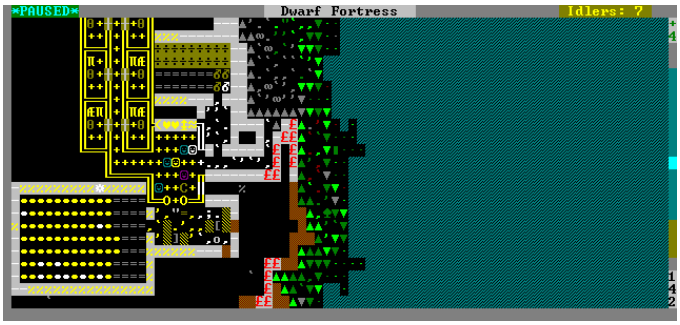

Fig. 1. A screenshot of a very basic fortress.

B. Evolutionary computing

Evolutionary computing[1, Chapter 2] is based on biological evolution. It works on the principle that if the program knows how to evaluate if an object is good and it knows how to change this object in order to affect how good/bad it is, then it should be able to create a good result given enough time.

Evolutionary computing (EC) is often used for optimization problems, as they are quite solid in what results they produce. Assuming that the evaluation function is well written, the evolution should keep moving towards better results, without the need for humans to constantly change things.

This also applies to fortresses in Dwarf Fortress. As creating a good fortress layout is, in essence, an optimization problem, evolutionary computing is a very suitable technique to use, as it is able to explore a lot of possibilities that would not be feasible by "hand".

III. Game Design

Whats the design of the game you are going to use? Why do you need PCG in this game?

Design: Tile-based map where you are the leader of a dwarven expedition. You tell the dwarves what to do and attempt to build a huge fortress in which they can live and thrive.

[1] http://www.bay12games.com/dwarves/

[2] "Perfect" in this context meaning a fortress that encapsulates everything a fortress can have, with a layout that optimizes everything.

PCG: Because coming up with a fortress layout can be very difficult, especially for new players. Having an idea of how to create a fortress would be an immense help.

Ikke sikker p dette kapitel er ndvendigt. Er ikke i eksempel rapporten vi har fet. Muligvis copy/pasta fra Modern AI, men str inde p PCG siden p Learnit

IV. Methods

There are various different ways to generate dungeons[2, Chapter 3], **more stuff/better intro**.

For our dungeon generation, we use two different algorithms: A map generation algorithm to generate the basic layout of the map and evolutionary algorithms[1, Chapter 2] to determine how the map layout can be used in the best way.

A. Evolution

Only write about how We do OUR evolution. Rest should be in background.

We chose to make our evolution a bit different. Instead of evolving a number of map layouts every generation, we create a number of map layouts which each have their own set of room assignments (candidates). Evolution is done individually for each map layout on their set of room assignment, instead of on the map layouts themselves. The best room assignment is then chosen as the candidate to mutate for the next generation. This way we avoid situations where a map layout is discarded because the mutations were less successful than mutations of other map layouts.

Note about genotypes

1) Create 10 different maps layout (using the map generation algorithm, see below) with empty rooms. When a map has been generated, find the distance from any room to all other rooms (see IV-C).
2) For each map layout, create 10 random room assignments.
3) For every map layout, do the following every generation until enough generations have passed:
   a) Use the current best room assignment as candidate for the generation.
   b) Create 10 mutations of the candidate. A mutation iterates over every room and has a 30% chance to transform the room into another random type of room.
   c) When the mutations have been created, calculate their fitness value (note about that).
   d) If a mutation is better than the current candidate for the map, replace the candidate with the best mutation.
4) At the end, for every map, copy its best room assignment into the actual map.

B. Map generation

For the map generation algorithm, we decided to write our own algorithm, as we could not find any other algorithm that could do specifically what we wanted. Our map generation algorithm works as follows:

1) Create a map of the user-specified dimensions and calculate the number of rooms we want, based on how many dwarves the user expects to have in their fortress.
2) Create an entrance to the fortress on the top layer along one side of the room.
3) For each layer in the map (starting at the top one), make a list that contains all positions that are not dug out and then do the following:
   a) Choose a random position from the list of open positions and use it as one corner of the room.
   b) Pick a random, diagonal direction (north/east, south/east, south/west or north/west), go 6 tiles that way and see if the positions in that direction are open.
   c) If they are open, build an empty room in the middle 4x4 square, set the tiles around that square to room walls, and remove the positions now occupied by the room from the list of open positions.
   d) Create a path from the room to the nearest entrance using Dijkstra (stairs on the lower levels count as entrances to that level).
   e) Once enough rooms have been created, or only 10% of the layer is open, create stairs to the next layer, and connect these stairs to the entrance of the current layer.
   f) Repeat steps a) to e) for each layer until enough rooms have been built.

C. Distances between rooms

V. Results

Did it work? How well? Provide some figures, and a table or two. How much time does it take? Remember to include significance values (remember the t-test?), variance bars Reread some of the papers from class and compare how they report their results.

PCG Book 2.4: Evaluation functions

PCG Book 12: Evaluating Generators

VI. Discussion

What are the strengths and shortcomings of your method? Why did you choose method X instead of Y? How well would it generalize to other game genres? How would you develop it further, if you had time?

References

[1] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2007.

[2] A. lot, *Procedural Content Generation in Games*. Not yet published, 2014.

[3] J. Orkin, "Three states and a plan: The A.I. of F.E.A.R." *Monolith Productions / M.I.T. Media Lab, Cognitive Machines Group*, 2006, [Online; accessed December 9, 2014]. [Online]. Available: http://alumni.media.mit.edu/ jorkin/gdc2006_orkin_jeff_fear.pdf