Graph-Based Search for Game Design

Daniel Ashlock, University of Guelph

Cameron McGuinness, University of Guelph

This article is the second in a tutorial series on the relationship between games, puzzles, and combinatorial graphs. It introduces some classical and modern algorithms for permitting a computer to play games and design games and puzzles using search-based procedural content generation. These algorithms are applied in the evolutionary design of level maps.

1 Introduction

This article introduces algorithms that are useful both for creating *artificial intelligence* (AI) players for games and for *procedural content generation* (PCG) [1]. AI algorithms for playing games must search the space of possible future moves, which could require large amounts of time if the algorithms are not clever about how these searches are performed. The process of devising better search methods is the basis for much of the research in this area.

Procedural content generation covers a very wide range of topics. It includes any algorithmic method for designing a game or part of a game. This article introduces the topic and presents *search-based procedural content generation* (SB-PCG) [2] as a motivating example. SB-PCG is demonstrated for creating maps for use in a fantasy role-playing game.

Graph algorithms[1] are algorithms that operate on or are structured by graphs. This article introduces two types of graph algorithms that are especially useful in game play and puzzle design. The first, tree search algorithms, operate on a structure called a *game tree*. The second, path finding algorithms, are used to search a game board or map to find the most efficient set of moves to reach an objective.

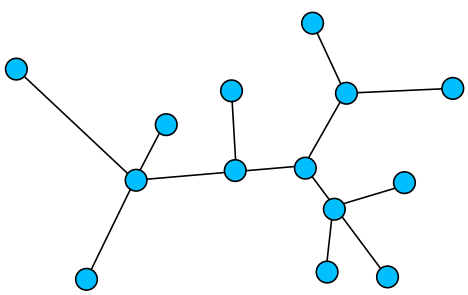

Figure 1. An example tree.

1.1 Graph Theory

The first article in this series [3] defined some basic graph theory terminology. We now introduce additional terms for this article. A *tree* is a *connected graph* with no closed paths. Figure 1 shows an example tree with nine leaves and five internal nodes. It is connected – it is possible to go from any node to any other node by following connections in the tree – but there are no closed loops in the structure.

A *rooted tree* is one that has a distinguished node called a *root*. Rooted trees are typically displayed in layers relative to the root. An example of a rooted tree is shown in Figure 2. Its root is the unique vertex on level 1. For both sorts of tree, a *leaf* is a vertex of degree 1 (with one adjacent neighbour) that is not the root. The vertices with degree greater than one are called *internal nodes*.

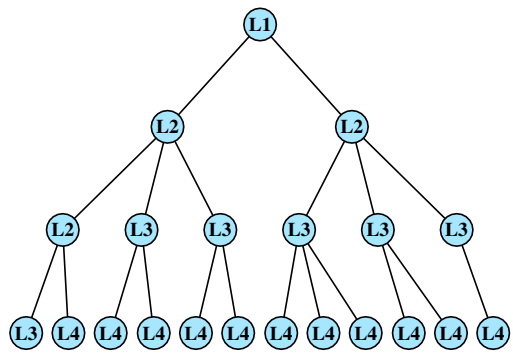

Figure 2. A rooted tree with four levels.

2 Tree Search

Consider a two-player game in which the players take turns moving. This situation can be naturally structured as a tree. The root of the tree is the starting (or current) position for the game. The second level of the tree contains all the positions that can be reached with one move by the

[1] For examples, see: https://en.wikipedia.org/wiki/Book:Graph_Algorithms

Ashlock, D. and McGuinness, C., 'Graph-Based Search for Game Design', *Game & Puzzle Design*, vol. 2, no. 2, 2016, pp. 68–75.