Hex-a-Pawn

Due Monday Mar 28, 2016

Objectives

For this assignment, you will:

- Gain experience working with tree data structures
- Gain more experience with recursive algorithms
- Implement a basic Artificial Intelligence algorithm for a two-player zero-sum game

Description

This week's assignment uses a tree data structure to play a small chess-like game. You will build a tree representing all possible states that the game board can be in. You will then implement several different players that consult this tree to make moves as they play Hex-a-Pawn. The players include: a human player that asks the user for moves to make; a random player that picks possible moves at random; and a computer player that improves its strategy by learning from past mistakes. In the end, you will be able to run the different players against each other.

Before starting this assignment, read Section 12.11 in Bailey. This section describes the Hex-a-Pawn game as well as listing some steps to help you get started. This section also tells you that a complete game tree for 3x3 boards has 252 nodes. In the margin where it states this, you will find some helpful debugging advice (e.g. if your tree has 370 nodes then you have the wrong win test).

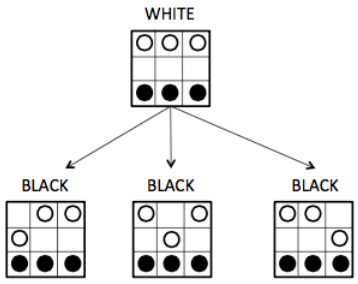
Classes

In this assignment, you must implement a `GameTree` class as well as three different classes corresponding to different types of players. Each of these player classes will implement the `Player` interface.

`GameTree` class

You are responsible for designing the `GameTree` class. This is a tree structure with potentially many children instead of just two. Think about the methods you will need for this class and how you can represent the structure.

The picture below shows the top two levels (i.e. the nodes at depth 0 and 1) of the game tree that results from a call to the constructor: `new GameTree(new HexBoard(3, 3), HexBoard.WHITE)`.



The root node of the tree contains the starting board position (`new HexBoard(3,3)`) and the player is white (`HexBoard.WHITE`). Note that this is not a binary tree since it is possible for a node in the tree to have multiple children – each child corresponding to a legal action by the player.

Player classes

You will construct three player classes: `HumanPlayer`, `RandPlayer`, and `CompPlayer`. Each player implements the `Player` interface. This interface has only one method:

```
public Player play(GameTree node, Player opponent);
```

The `play` method takes in a `GameTree` corresponding to some configuration of the board and the opponent player. The return value of this method is the player who won the game. Inside `play`, you should check if the board configuration is a win for the opponent. If not, the player makes a move based on what type of player it is. After making a move, you should then call the opponent's `play` method.

You will have to think carefully about how the `Player` classes interact with the `GameTree` class – this should inform what methods and instance variables you include in the `GameTree` class. You can also create a `main` method inside each `Player` class where you create two players and have them play a game – this will help you debug your code.

Getting Started

1. As usual, the starter files can be found in the `/common/cs/cs062/assignments/assignment08` directory. Familiarize yourself with these files *before* starting to work on this assignment. They are described briefly in the book, and the JavaDoc documentation is available at:

   `http://www.cs.williams.edu/~freund/cs136-073/javadoc/hexapawn/index.html`

2. **Please come to class on Wednesday prepared to talk about how you will implement the `GameTree` class.**

3. You can play a game of Hex-a-Pawn by running the command `java HexBoard` after compiling the starter files.

4. You can look at the `main` method in the `HexBoard` class to see an example of how to code up a game between two players.

5. The starter directory contains Gardner's original paper on Hex-a-Pawn. There are other learning algorithms described in that paper if you are interested in experimenting further.

Grading

You will be graded based on the following criteria:

criterion	points
Efficient implementation of `GameTree` class	3
Correct `HumanPlayer`	3
Correct `RandPlayer`	3
Correct `CompPlayer` that trims tree over time	3
General correctness	3
Appropriate comments (including JavaDoc)	2
Style and formatting	2

Submitting Your Work

As usual, export your project in Eclipse. This will create a directory on your Desktop. Please rename this directory **Assignment08_LastNameFirstName**. Drag this directory to the dropbox.