CoNxNect API v0.1

HappyGnome

March 2019

Contents

1 Introduction

TODO

2 AI Player Library

2.1 Overview

A player library defines an ai player class, a player factory class, and a module file which is used to load the rest. For definiteness, we will call these `aiPlayer`, `aiFactory` and `aiModule` (i.e. in the file `aiModule.py`), although any names may be used. In order that the library can be found by CoNxNect, `aiModule.py` should be placed in `<CoNxNect_root>/AI_Players`. Any additional files should be placed in `<CoNxNect_root>/AI_Players/<aiModule>/`

2.2 `aiModule`

An AI player module, must implement the following in global scope:

`getFactory()` **Args:** None
Desc:

Return: An instance of `aiFactory`. This may be static within the module. It will only be used in the calling thread.

2.3 `aiPlayer` Reference

The following methods should be implemented:

`makeMove(self)` **Args:** self
Desc: The player will not be notified of its own moves, and so should update itself accordingly.
Return: Index of the column in which this player plays next.

`notifyMove(self, playerTurn, column)` **Args:** *playerTurn* - the turn position of the player making the move.
column - the column in which the player played
Desc: Called each time another player makes a move.
Return: None

`beginGame(self, playerTurn)` **Args:** *playerTurn* - the position of this player in turn order (1= first,...)
Desc: Initialise resources, threads etc. when this is called.
Return: None

`endGame(self)` **Args:** self
Desc: Free resources, threads etc. that were created at `beginGame`.
Return: None

2.4 `aiFactory` Reference

The following methods should be implemented:

`newPlayer(self, spec, allowed_depth)` **Args:** *spec* - an instance of gameSpec, defining the grid size, player count and victory conditions.
allowed_depth - A complexity hint. The returned player should compute at most this many levels in the decision tree.
Desc: Generate a new instance of `aiPlayer` based on the game specification.
Return: The new `aiPlayer` instance.

`newPlayers(self, spec, allowed_depth, n)` **Args:** *spec* - an instance of gameSpec
allowed_depth - A complexity hint. The returned player should compute at most this many levels in the decision tree.
n - the number of players to generate
Desc: Generate `n` new `aiPlayer` instances, based on `spec` and the current factory config.
Return: A list containing the new `aiPlayer` instances.

`loadDefaultConfig(self)` **Args:** self

Desc: Load the default config/training data for this factory. This will be called before players are generated.

Return: None

`updateConfig(self, player_rank)` **Args:** *player_rank* - A list of ordered pairs `(player, score)`. Each `player` is an instance generated by this factory, and `score` is the corresponding rating (higher is better), calculated from playing several games.

Desc: Update the parameters of the factory when generating new players. This routine may or may not save the new config/ learning data.

Return: None

`saveConfig(self)` **Args:** self

Desc: Save any config/learning data, as necessary. This may be called, for example before Default config is reloaded, or the program is closed.

Return: None

3 Structures

3.1 `gameSpec`

Public attributes:

`rows` - Number of rows in the grid.

`cols` - Number of columns in the grid.

`playerCount` - Number of players in the game.

`victoryN` - Length of streak required for victory.