CoNxNect API v0.1

HappyGnome

March 2019

Contents

1 Introduction

TODO

2 AI Player Library

2.1 Overview

A player package defines an ai player class, a player factory class. For definiteness, we will call the package `aiPackage` In order that the package can be found by CoNxNect, `/<aiPackage>/__init.py__` should be a subdirectory of in `<CoNxNect_root>/PlayerModules`.

2.2 `aiPackage`

A player package in the directory `PlayerModules/<aiPackage>/`. This directory should contain `__init__.py`, `factory.py` and `player.py`.

2.3 `player` Reference

A class defined in `<aiPackage>/player.py`.

`player` should implemented the following methods:

`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, winner)` **Args:** *winner* - Turn position of the winning player, 0 for a draw or -1 for game ended early.
Desc: Free resources, threads etc. that were created at `beginGame`. This is also the place to log results of the game.
Return: None

`getFactory(self)` **Args:** self
Desc:
Return: The factory that generated this player.

2.4 `factory` Reference

A class defined in `<aiPackage>/factory.py`.

The methods below should be implemented. It must be safe to call any of them from any thread.

`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 `player` based on the game specification.
Return: The new `player` 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 `player` instances, based on `spec` and the current factory config.
Return: A list containing the new `player` 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

`debriefPlayer(self, player)` **Args:** *player* - A player previously generated by this factory.
Desc: Update the parameters of the factory for generating new players, based on the experience of passed player. 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.