Qwirkle Reboot

2023-11-21

Simply let the game be able to play against itself

Either text or WPF interface

Board representation is private, should implement needed functions to find whether a square is playable or not, or if a tile is compatible with a square, but not expose its internal representation.

Board representation should be chained, with a new version after some turn is added linked to a previous version for minimax strategy.

For minmax strategy, distribution of squares must be predefined for a game

Game principle, for one player:

* PossibleMoves = []
* Find all playable squares on the board
* For each playable square:
  + For each of the tiles in the dock compatible with current playable square:
    - CurrentMoves = []
    - ExploreMove(board, dock, currentMoves, new Move(square, tile), true, true)
    - First level of trimming of currentMoves, add best currentMoves to PossibleMoves
* Select best move from PossibleMoves using appropriate strategy (max, minmax, final turn)
* Add best move to the board
* Replenish user dock

A Tile is a plain object with two attributes, shape and color, and a third internal attribute instance (1..3) so that a set can contain up to 3 tiles with same shape and color

Dock is a simple set of Tile.

A Move is a struct of (int row, int col, Tile tile).

How to build a list of valid placements N/S with a starting square:

def ExploreMove(Board b, Dock d, list<Move> currentMoves, Move move, bool NS, bool EW):

Board newB = ChainedBoard(b, move)

Dock newD = d.clone – move.tile

List<Move> newCurrentMoves = currentMoves.clone + move

add newCurrentMoves to PossibleMoves

If NS:

TryExplore(newB, newD, newCurrentMoves, 1, 0)

TryExplore(newB, newD, newCurrentMoves, -1, 0)

If EW:

TryExplore(newB, newD, newCurrentMoves, 0, 1)

TryExplore(newB, newD, newCurrentMoves, 0, -1)

def TryExplore (Board b, Dock d, list<Move> currentMoves, detaRow, deltaCol):

If Find playable position in direction(deltaRow, deltaCol) in b:

For each tile t in d compatible with playable:

ExploreMove(b, d, currentMoves, new Move(playable, t), deltaRow!=0, deltaCol!=0)

Board services:

* Returns board extent (min/max on row/col)
* Retrieve cell(row, col) state: tile, empty isolated, empty playable, empty non-playable
* Returns all playable squares
* Get the constraints of a playable square (maybe simply provided by cell state)

Tile services:

* IsCompatible with constraints

Constraints of a playable cell:

* 4 sets of constraints, N, S, E, W
* A constraint can be:
  + Empty, no color of shape constraint
  + Color constraint:
    - Either a fixed color among the 6 possible
    - Or a set of forbidden colors (already played) in this direction
  + Shape constraint:
    - Either a fixed shape
    - Or a set of forbidden shapes (already played) in this direction