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 Hand compatible with current playable square:
    - CurrentMoves = []
    - ExploreMove(board, Hand, 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 Hand

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

Hand 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, Hand h, list<Move> currentMoves, Move move, bool NS, bool EW):

Board newB = ChainedBoard(b)

newB.AddMove(move)

Hand newH = h.clone – move.tile

List<Move> newCurrentMoves = currentMoves.clone + move

add newCurrentMoves to PossibleMoves

If NS:

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

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

If EW:

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

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

def TryExplore (Board b, Hand 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
* IsCellCompatible with a tile
* ~~Get the constraints of a playable square (maybe simply provided by cell state)~~

Find if a cell is compatible with a tile

* Must be empty, at least 1 neighbor present
* Compatible = And(compatible for each neighbor in its direction)

Compatibility in a given direction, for each tile from a starting cell (not included):

* If the cell is empty, it’s compatible, return true.
* If the cell is (≠color and ≠ shape) or (=color and =shape), it’s not compatible, return false.

~~Constraints of a playable cell:~~

* ~~Applicable to an empty cell with at least 1 of the 4 neighbors~~
* ~~It’s the combination of constraints on all neighbor present~~
* ~~4 sets of constraints, N, S, E, W~~
* ~~A constraint from 1 direction can be:~~
  + ~~Empty, no color of shape constraint (neighbor is not present)~~
  + ~~Flexible, with 1 neighbor cell only, specifying shape or color to choose from~~
  + ~~Color constraint, when 2+ neighbor cells have same color, specifying color, and a set of shapes already played in this direction~~
  + ~~Shape constraint, when 2+ neighbor cells have the same shape, specifying shape, and a set of colors already played in this direction~~
  + ~~Not playable, when color or shape constraint have already 6 shapes or colors already played~~

~~Combination of directional constraints of a playable cell:~~

* ~~Empty constraint is ignored~~
* ~~Not playable from one direction makes the cell Not playable~~
* ~~2 color/shape constraints (same type):~~
  + ~~If the specified color/shape is different, not playable, because of conflicting constraints~~
  + ~~Merge with union of sets of shape/color, with the possible option of not playable if the union set contains all 6 options~~
* ~~1 color constraint and 1 shape constraint~~

# Model

## Objects

enum Shape, Color, CellState

class Board: Enumerable<Move>

record Tile: (Shape, Color, Instance)

record Move: (Row, Col, Tile)

record Play: (HashSet<move>, PointsBonus PB, Hand NewHand) returned by Board:Play

struct PointsBonus(Points, Bonus) tuple returned by Board:CountPoints

class Hand: HashSet<Tile>

class Bag a shuffled container for 108 tiles

# ViewModel

# View

UserControl UITile

bool GrayBackground

bool SelectionBorder

bool Hatched

string ShapeColor

Attached Properties: double Canvas.Left, double Canvas.Top

UserControl Hand represents a player hand