Documentation

Contents

Cublino Class

// Check if the encodings are well-formed.

isStateWellFormed

isPositionWellFormed

isDiceWellFormed

isMoveWellFormed

isStepWellFormed

bestMove(State state, int difficulty) // Give a Pur or Contra state and a difficulty value n, find the best move.

If n = 1, implement greedy AI. If n = 2 implement Minimax AI.

Game Tree Class

GameTree(State parent, int depth) // Given a depth (>=1), generate a game tree from a game state.

miniMax(boolean currentPlayer) // Evaluate a game tree.

miniMaxAB(int alpha, int beta, boolean currentPlayer) // Evaluate a game tree. Alpha = -9999 and Beta = 9999.

// Please see “stateEvaluate” function in “State” class for the use of the parameter “currentPlayer”.

State Class

copy() // Return a new identical state.

Checker Methods

isStateValid() // Determine whether a Pur or Contra state is valid.

containDice (Position position, boolean extraCondition) // Check if the state contains a dice at a given position.

// When the extra condition is enabled, check if the state contains a dice at a given position and the dice must belong to the current player.

isGameOver() // Determine whether a state represents a finished Pur or Contra game, and if so who the winner is.

// If the game is not over, return 0. If player 1 won, return 1. If player 2 won, return 2.

boolean isPur() // Check if it is a Pur state.

Setter Methods

setTurn(boolean isPlayer1Turn) // Given a player, set the turn.

changeTurn() // Change the turn.

applyMove(Move m) // Given a Pur or Contra game state and a move, update the state from the move.

// This method will actually change the state. If you want to keep the old state, please use copy() to make a copy first.

General Getter Methods

getDices() // Get a list of all dices.

ArrayList<Dice> getCurrentPlayerDices() // Gets a list of all dices belonging to the current player.

getPlayerTurn() // Get the current turn of the state.

legalMoves() // Get a set of legal moves of the Pur or Contra state.

legalJumpPur() // Get a list of all the legal jumps in a Pur state.

legalStepsPur(Dice dice) // Get a list of all the legal steps in for a given dice in a Pur state.

Heuristic Methods

stateEvaluate (boolean currentPlayer) // Evaluate a Pur or Contra state.

* The parameter “currentPlayer” indicates whether the AI is Player 1.
* If the AI is Player 1, the score of the state = Player 1 score - Player 2 score.
* If the AI is Player 2, the score of the state = Player 2 score - Player 1 score.
* The score = AI’s score – its opponent’s score.
* If the score is positive, the AI is winning the game.
* If the score is negative, the AI is losing the game.

Dice Class

Getter Methods

isPlayer1() // Get the player of the dice.

getPosition() // Get the position of the dice.

getFaces() // Get an array of number indicating the faces of a dice. {TOP, FORWARD, RIGHT, BEHIND, LEFT, BOTTOM}

getTopNumber() // Get the top number of a dice.

getEnemies (State state) // Get all the enemies of a dice in a state. (Used in Contra)

// Enemies are all the dice that are adjacent to the current dice but have different colours.

Setter Methods

jump(Position position) // Give a position, update the position of the dice.

tip(Step step) // Give a tip step, update the both position and direction of the dice.

Static Methods

adjacentDices (Position position, State state) // Give a state and a position, find all the dices adjacent to the position.

Move Class

copy() // Return a new identical move.

Setter and Getter Methods

getPositions() // Get a list of positions of the move.

getStart () // Get the start position of the move.

getEnd () // Get the end position of the move.

moveFurther (Position destination) // Add a new position to the current move object.

Checker Methods

isValidMovePur(State state) // Determine whether a move (sequence of steps) is valid for a given Pur game.

isValidMoveContra(State state, boolean quickCheck) // Determine whether a move is valid for a given Contra game.

// If quickCheck is disable, the function won't check the following conditions.

1. The starting position and the ending position are adjacent.
2. There is a current player's dice in the starting position.

Step Class

Setter and Getter Methods

setStep(Position start, Position end) // Update the starting position and the ending position of a step.

getStartPosition() // Get the starting position of a step.

getEndPosition() // Get the ending position of a step.

Checker Methods

isTip() // Check whether it is a tipping step

isValidStepPur(State state) // Determine whether a single step of a move is valid for a given Pur game.

Position Class

Setter and Getter Methods

getX(), getY(), setX(int x), setY(int y)

getPositionOrder() // Get the integer encoding (order) of a position. i.e., a1 = 0, b1 = 2, c1 = 3, a2 = 8, g7 = 48.

getAdjacentPositions() // Get all the positions adjacent to the current position.

getJumpPositions() // Get all the positions 2 unit away from the current position.

Checker Methods

equals(Position pos) // Check if two locations are equal.

isAdjacent(Position other) // Checks if two locations are adjacent.

isOnBoard() // Check if the current position is on the board.

Static Method

manhattanDistance(Position loc1, Position loc2) // Calculate the manhattanDistance between two locations