ChessMate Documentation

Release 1

Thomas Dahmen, Oscar Bouvier, Jean Forissier, Raphael Macque

Mar 16, 2018

Contents

Contents:

CHAPTER 1

Game engine

1.1 echecs.py

Game engine documentation Created on Tue Jan 30 14:44:38 2018

echecs.**alpha_beta_B**(*arb*, *profondeur*, *alpha*, *beta*)

Optimized minimax reccursive algorithm which doesn't consider useless branchs thanks to alpha and beta parameters

Parameters

- **arb** (*node*) – node algorithm is executing on
- **profondeur** (*int*) – height of the node algorithm is executing on
- **alpha** (*int*) – inferior value of cutting interval (initially negative infinite)
- **beta** (*int*) – superior value of cutting interval(initially positive infinite)

Returns

- tuple (x,k,l,g) - (x,y,k,l) is associated with the future move to do
- g is the result of the evaluation function of the chessboard configuration

Return type tuple

echecs.**alpha_beta_W**(*arb*, *profondeur*, *alpha*, *beta*)

Optimized minimax reccursive algorithm which doesn't consider useless branchs thanks to alpha and beta parameters

Parameters

- **arb** (*node*) – node algorithm is executing on
- **profondeur** (*int*) – height of the node algorithm is executing on
- **alpha** (*int*) – inferior value of cutting interval (initially negative infinite)

- **beta** (*int*) – superior value of cutting interval(initially positive infinite)

Returns

- tuple (x,k,l,g) - (x,y,k,l) is associated with the future move to do
- g is the result of the evaluation function of the chessboard configuration

Return type tuple

echecs.**chess_B**()
Tells if the black king is in a chess situation

Returns

- True -> Chess situation : the black king belongs to ensemble_move_possible_W
- False -> Not a chess situation

Return type boolean

echecs.**chess_Mate_B**()
Tells if the black king is in a chessmate situation

Returns True -> White player wins

Return type boolean

echecs.**chess_Mate_W**()
Tells if the white king is in a chessmate situation

Returns True -> Black player wins

Return type boolean

echecs.**chess_W**()
Tells if the white king is in a chess situation

Returns

- True -> Chess situation : the white king belongs to ensemble_move_possible_B
- False -> Not a chess situation

Return type boolean

echecs.**copy**(*tab*)

echecs.**create_tree_B**()

- If black player has to play, create a 3-height tree of all playing configurations (considering B-W-B)
- Each one is associated with an evaluation of the final chessboard configuration (evaluation function)
- This tree will then be crossed by minimax and alpha-beta algorithms.

echecs.**create_tree_W**()

- If white player has to play, create a 3-height tree of all playing configurations (considering W-B-W)
- Each one is associated with an evaluation of the chessboard configuration (evaluation function)
- This tree will then be crossed by minimax and alpha-beta algorithms.

echecs.**create_tree_W_viz**()

echecs.**ensemble_move_possible_B**()
Concatenates the possible moves of each black piece

Returns all the possible moves of black pieces

Return type tuple array

echecs.**ensemble_move_possible_W**()
Concatenates the possible moves of each white pieces

Returns all the possible moves of white pieces

Return type tuple array

echecs.**ensemble_valeurs_accessibles_B**()
Concatenates the accessible values of each black pieces

Returns all the accessibles values of black pieces

Return type tuple array

echecs.**ensemble_valeurs_accessibles_W**()
Concatenates the accessible values of each white pieces.

Returns all accessibles values of white pieces

Return type tuple array

echecs.**eval_denombrement**()
Part of the evaluation function useful for minimax and alpha-beta, that only considers taken pieces by each times (stored in wonB and wonW)

Returns gain of the current chessboard configuration

Return type int

echecs.**get_alpha_beta_B**()
Returns the tuple (x,y,k,l) that corresponds to the best move for black player considering the minimax algorithm optimized with alpha-beta method

Returns tuple (x,y,k,l) to implement in the move function which will finally simulate IA

Return type tuple

echecs.**get_alpha_beta_W**()
Returns the tuple (x,y,k,l) that corresponds to the best move for white player considering the minimax algorithm optimized with alpha-beta method

Returns tuple (x,y,k,l) to implement in the move function which will finally simulate IA

Return type tuple

echecs.**get_minimax_B**()
Returns the tuple (x,y,k,l) that corresponds to the best move for black player considering the minimax algorithm

Returns tuple (x,y,k,l) to implement in the move function which will finally simulate IA

Return type tuple

echecs.**get_minimax_W**()
Returns the tuple (x,y,k,l) that corresponds to the best move for white player considering the minimax algorithm

Returns tuple (x,y,k,l) to implement in the move function which will finally simulate IA

Return type tuple

echecs.**is_max_W**(*arb*)

- Epecially used by minimax and alpha-beta algorithms
- Tells if a node is a “max-node”

Parameters **arb** (*node*) – node

Return type boolean

echecs.**is_min_B**(*arb*)

- Epecially used by minimax and alpha-beta algorithms
- Tells if a node is a "max-node"

Parameters **arb** (*node*) – node

Return type boolean

echecs.**minimax_B**(*arb*, *profondeur*)

- Recursive algorithm which is supposed to cross a tree previously constructed by create_tree_W.
- Represents the best way to modelize zero sum games such as chess.

Returns

- tuple (x,k,l,g) - (x,y,k,l) is associated with the future move to do
- g is the result of the evaluation function of the chessboard configuration

Return type tuple

echecs.**minimax_W**(*arb*, *profondeur*)

- Recursive algorithm which is supposed to cross a tree previously constructed by create_tree_W.
- Represents the best way to modelize zero sum games such as chess.

Parameters

- **arb** (*node*) – node algorithm is executing on
- **profondeur** (*int*) – height of the node algorithm is executing on

Returns

- tuple (x,k,l,g) - (x,y,k,l) is associated to the future move to do
- g is the result of the evaluation function of the chessboard configuration

Return type tuple

echecs.**mouv_possible_chess_B**()

Concatenates the possible moves of black piece to avoid a chess situation

Returns

- list of possible moves (x,y,k,l) to avoid chess situation
- x,y : initial position of a black piece
- k,l : final position that avoid chess situation

Return type tuple array

echecs.**mouv_possible_chess_W**()
Concatenates the possible moves of white pieces to avoid a chess situation

Returns

- list of possible moves (x,y,k,l) to avoid chess situation
- x,y : initial position of a white piece
- k,l : final position that avoid chess situation

Return type tuple array

echecs.**move**(*a*, *b*, *c*, *d*)

- Moves a piece located on (a,b) to (c,d) if the movement is allowed by changing the values of plateau.
- Updates dico_position_W, dico_position_B, position_W, position_B,wonW,wonB
- Reverses the boolean value of tour_blanc to allow next player to play

Parameters

- **a** (*int*) – X axis of the piece we want to move
- **b** (*int*) – Y axis of the piece we want to move
- **c** (*int*) – X axis of the position we want to move the piece on
- **d** (*int*) – Y axis of the position we want to move the piece on

Returns None

Return type None

echecs.**move_IA_black**()

- **Complete IA black player game turn simulation**
  - Create black tree ready to be crossed
  - Execution of alpha-beta algorithm
- Used by Graphical User Interface

Returns best move parameters (x,y,k,l) for black player considering alpha-beta algorithm

Return type tuple

echecs.**move_chess**(*a*, *b*, *c*, *d*)

- Special goal of move(a,b,c,d) useful for chess_mate functions which doesn't take care about taken pieces.
- Doesn't update wonW,wonB

echecs.**movetest**(*a*, *b*, *c*, *d*)

- Same goal as move(a,b,c,d) but doesn't take care about rules (moving positions allowed, tour)
- Also updates dico_position_W, dico_position_B, position_W, position_B,wonW,wonB
- Essentially useful for tests

echecs.**opponent**(*a*, *b*)
Tells if piece a is an opponent of piece b.

Parameters

- **a** (*int*) – piece a (which could be a relative integer between -6 and 6 and couldn't be 0 , 0=empty piece)
- **b** (*int*) – piece b (which could be a relative integer between -6 and 6 and couldn't be 0, 0=empty piece)

Returns True -> a is an opponent of b

Return type boolean

echecs.**tour_Blanc**(*x*)

echecs.**valeurs_accessibles**(*x*, *y*)

Returns the array of accessible chessboard positions of a piece located on (x,y).

Parameters

- **x** (*int*) – X axis of initial piece's position
- **y** (*int*) – Y axis of initial piece's position

Returns array of accessible plateau positions of a piece located on (x,y)

Return type tuple array

echecs.**valeurs_accessibles_test**(*x*, *y*)

echecs.**valide**(*a*)

Tells if a piece is valid, that is to say belongs to the 8*8 square gamezone.

Parameters **a** (*int*) – exists x,y -> a=plateau[x][y]

Returns True -> piece is valid

Return type boolean

CHAPTER 2

Graphical User Interface

2.1 gui.py

GUI Documentation @author: Thomas Dahmen

class gui.**Ui_Dialog**
Bases: object

This class contains the GUI's window, in which pieces' locations are buttons Moving a piece is achieved by changing buttons' images The association between a button, the value of its piece and the piece's position on the chessboard is managed with two dictionaries, which are declared in retranslateUi

chk(*button*, *dict1*, *dict2*)
This is the action function called when a piece is selected. It interacts with the game engine echecs.py

Multiple cases must be managed:

Let's assume we play with whites, and it is the first time the player selects a piece

- **if we choose a white piece**
  - it is highlighted
  - the selected button is stored in pChecked, waiting for the second call of the function
  - the game engine is called in order to highlight accessible cases
- if we do anything else (choosing a black piece...), nothing happens

Let's now assume a white piece was previously selected (stil playing with whites)

- **if we choose a white piece**
  - the selection changes (new piece highlighted and stored in pChecked, waiting for another call of the function)
- **if we choose a black piece**
  - if it is not accessible, the selected white piece becomes unselected

  - if it is accessible, the selected white piece takes its place and the change is sent to the game engine
  - a new blank case is created
  - the next turn begins: blacks must play
- **if we choose a blank case**
  - if it is not accessible, the selected white piece becomes unselected
  - if it is accessible, the selected white piece takes the blank place and the change is sent to the game engine

Parameters

- **button** (*QtWidgets.QPushButton*) – The sender (user selected button)
- **dict1** (*dict*) – Dictionary button -> position on the chessboard
- **dict2** (*dict*) – Dictionary position on the chessboard -> button

Returns None

Return type None

retranslateUi(*Dialog*)

This function connects the GUI to user's actions. In this specific function, we define two dictionaries that are necessary to link the GUI to the game engine:

- **dict1**
  - key: a button of the GUI
  - value: the position of the piece in the game engine's chessboard
- **dict1**
  - key: the position of the piece in the game engine's chessboard
  - value: a button of the GUI

This enables the GUI and the game engine to interact with each other

setupUi(*Dialog*)

Python Module Index

e

g