ZPR_LtL

Generated by Doxygen 1.9.5

Chapter 1

ZPR

Larger than Life

1.1 How to install and run

1.1.1 general instructions

1. clone pybind11 repository from github (`git clone https://github.com/pybind/pybind11.git`)
2. install requirements (`pip install -r requirements.txt`)
3. configure and build cmake
4. run main.py (`python main.py`) NOTE: In case there is a problem with import, try adding `__init__.py` file to build folder (`touch build/__init__.py`)

1.1.2 example using pip (preferred)

git clone https://github.com/pybind/pybind11.git pip install -r requirements.txt mkdir build cd build cmake .. make cd .. touch build/__init__.py python main.py `or` python3 main.py `or` python3.9 main.py

1.1.3 example using conda

git clone https://github.com/pybind/pybind11.git conda env create -f zpr_ltl_conda_env.yml -n ltl conda activate ltl mkdir build cd build cmake .. make cd .. touch build/__init__.py python main.py `or` python3 main.py `or` python3.9 main.py

1.2 How to give your own starting board as an argument

If you want, you can give your own starting board as a txt file.

- the file's name must be `starting_board.txt`
- `example_starting_board.txt` is a good example (if you want to use it, just change it's name to `starting_board.txt`)
- the file must have 50 lines with 50 integers seperated by spaces
- the integers can't be below 0 and can't be bigger than `count_of_states - 1`
- if the `starting_board.txt` file is invalid, or if there is no such file, the program will just run with a randomized board

1.3 Run tests

1.3.1 example using pip (preferred)

git clone https://github.com/pybind/pybind11.git pip install -r requirements.txt mkdir build cd build cmake .. make cd .. touch build/__init__.py pytest tests/

1.3.2 example using conda

git clone https://github.com/pybind/pybind11.git conda env create -f zpr_ltl_conda_env.yml -n ltl conda activate ltl mkdir build cd build cmake .. make cd .. touch build/__init__.py pytest tests/

1.4 Generate documentation

doxygen config_doxygen

1.5 Additional installs

for Ubuntu runs, `sudo apt-get install python3-tk` is required

1.6 Format python code

Python code formatter - Black:
Python linter - Pylint:
```
pylint ./GUI
```

```
pylint main.py
```

1.7 Format C++ code

chmod +x engine/run_clang_format.sh ./engine/run_clang_format.sh

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 GUI.board_display Namespace Reference

Classes

- class BoardWindow

Functions

- List[List[int]] randomize_board (BoardEngine engine, int num_of_random_cells)
- List[List[int]] calculate_next_state (BoardEngine engine)
- List[tuple] define_colors (int states)

Variables

- int **CELL_SIZE** = 12

5.1.1 Detailed Description

```
Module responsible for board display management and the class BoardWindow.
```

5.1.2 Function Documentation

5.1.2.1 calculate_next_state()

```
List[List[int]] GUI.board_display.calculate_next_state (
            BoardEngine engine )

Returns next state of the board.

Args:
    engine:     Game engine written in C++

Returns:
    A matrix with elements equivalent to given cell's state.
```

5.1.2.2 define_colors()

```
List[tuple] GUI.board_display.define_colors (
             int states )
```

```
Sets different colors for cells with different states.

Args:
    states:     Number of possible cells' states.

Returns:
    List of colors (in RGB notation) for different states.
```

5.1.2.3 randomize_board()

```
List[List[int]] GUI.board_display.randomize_board (
             BoardEngine engine,
             int num_of_random_cells )
```

```
Returns board with given number of randomized cells

Args:
    engine:     Game engine written in C++
    num_of_random_cells:    How many cells to randomize

Returns:
    A matrix with elements equivalent to given cell's state.
```

5.2 GUI.user_options Namespace Reference

Classes

- class UserOptions

Functions

- None error_msg (str title, str msg)
- None check_user_options ()
- def main ()

Variables

- int **WIDTH** = 100
- int **HEIGHT** = 100
- int **BOARD_SIZE** = 50
- float **SLEEP** = 0.5

5.2.1 Detailed Description

```
Module responsible for the GUI:
- menu with game's options
- gathering custom game options
- running main pygame window with board
```

5.2.2 Function Documentation

5.2.2.1 check_user_options()

```
None GUI.user_options.check_user_options ( )
```

```
Verifies values in options provided by the user.
In case of incorrect data, shows a pop up window with error.
```

5.2.2.2 error_msg()

```
None GUI.user_options.error_msg (
            str title,
            str msg )
```

```
Shows pop up window with error message.

Args:
    title:  Title of the error message.
    msg:    Main text of error message.
```

5.2.2.3 main()

```
def GUI.user_options.main ( )
```

```
Main function.
```

5.3 GUI.utils Namespace Reference

Classes

- class Params

Functions

- Params load_params (str path)

Variables

- **OPTIONS** = Params()

5.3.1 Detailed Description

```
Module responsible for a class with game's parameters.
```

5.3.2 Function Documentation

5.3.2.1 load_params()

```
Params GUI.utils.load_params (
            str path )
```

```
Loading parameters from a json file.

Args:
    path:       Path to the json file with parameters.

Returns:
    Object of type Params with parameters extracted from the file.
```

5.4 main Namespace Reference

Functions

- def print_board (board)

5.4.1 Detailed Description

```
Module responsible for invoking main function from user_options that starts user interface.
```

5.4.2 Function Documentation

5.4.2.1 print_board()

```
def main.print_board (
             board )
```

```
Function for printing provided board.
Useful for debugging.
```

Chapter 6

Class Documentation

6.1 BoardEngine Class Reference

Public Member Functions

- void **set_parameters** (const int Rr, const int Cc, const bool Mm, const int Smin, const int Smax, const int Bmin, const int Bmax, const std::string Nn)
- void **set_cell** (int row_num, int col_num, int new_value)
- void **print_current_board** () const
- Board **get_board** () const
- int **get_height** () const
- int **get_width** () const
- void **change_random_cell** ()
- void **randomize_board** (int num_of_random_cells)
- void **calculate_next_state** ()
- int **count_neighbours** (int row_num, int col_num, int max_num_of_neighbours) const

The documentation for this class was generated from the following files:

- engine/BoardEngine.hpp
- engine/BoardEngine.cpp

6.2 GUI.board_display.BoardWindow Class Reference

Public Member Functions

- def __init__ (self, int height, int width, int states, int cell_size=CELL_SIZE)
- None update (self, List[List[int]] new_board)
- None save_as_img (self, str name)

Public Attributes

- **height**
- **width**
- **states**
- **cell_size**
- **window**
- **colors**

6.2.1 Detailed Description

```
Class representing a window with board to be displayed.
```

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
def GUI.board_display.BoardWindow.__init__ (
              self,
             int height,
             int width,
             int states,
             int  cell_size = CELL_SIZE )
```

```
Args:
    height:     Height of the window.
    width:      Width of the window.
    states:     Number of cells' states.
    cell_size:  Size of one cell (one dimension is enough as cell is a square).
```

6.2.3 Member Function Documentation

6.2.3.1 save_as_img()

```
None GUI.board_display.BoardWindow.save_as_img (
              self,
             str name )
```

```
Saves displayed board as PNG image.

Args:
    name:   Desired name of the file.
```

6.2.3.2 update()

```
None GUI.board_display.BoardWindow.update (
              self,
             List[List[int]] new_board )


Updates displayed board with a new board.

Args:
    new_board:  A matrix with elements equivalent to given cell's state.
```

The documentation for this class was generated from the following file:

- GUI/board_display.py

6.3 GameParameters Struct Reference

Public Attributes

- int **range** = 1
- int **count_of_states** = 5
- bool **count_middle** = false
- int **alive_min** = 2
- int **alive_max** = 3
- int **be_born_min** = 3
- int **be_born_max** = 3
- std::string **neighb** = "NM"

The documentation for this struct was generated from the following file:

- engine/BoardEngine.hpp

6.4 GUI.utils.Params Class Reference

Public Member Functions

- def __init__ (self, int range=1, int states=5, bool mid=False, int s_min=2, int s_max=3, int b_min=3, int b_↩ max=3, str neighb="NM", int sleep_time=0.5)

Public Attributes

- **range**
- **states**
- **mid**
- **s_range**
- **b_range**
- **neighb**
- **sleep_time**

6.4.1 Detailed Description

```
Class representing parameters required to the game
```

6.4.2 Constructor & Destructor Documentation

6.4.2.1 __init__()

```
def GUI.utils.Params.__init__ (
              self,
             int  range = 1,
             int  states = 5,
             bool  mid = False,
             int  s_min = 2,
             int  s_max = 3,
             int  b_min = 3,
             int  b_max = 3,
             str  neighb = "NM",
             int  sleep_time = 0.5 )

Args:
    range:      Range of the cells.
    states:     Number of states.
    mid:        If middle cell is included in the neighbourhood count.
    s_min:      Minimum count limit for the cell to survive.
    s_max:      Maximum count limit for the cell to survive.
    b_min:      Minimum count limit for a dead cell to become a birth.
    b_max:      Maximum count limit for a dead cell to become a birth.
    neighb:     Extended neighborhood type.
                Possible values: NN - von Neumann, NM - Moore.
    sleep_time: Sleep time in seconds for changing game status.
```

The documentation for this class was generated from the following file:

- GUI/utils.py

6.5 GUI.user_options.UserOptions Class Reference

Public Member Functions

- def __init__ (self, int width=WIDTH, int height=HEIGHT)
- def options (self)
- def save_options (self)
- def file_options (self)
- def save_file_options (self)
- def save_board (self)
- def sleep_option (self)
- def save_sleep (self)
- def resleep (self)
- def board_in_file_has_correct_size (self, board_from_file)
- def board_in_file_is_correct_for_parameters (self, board_from_file)
- def set_starting_board_from_file (self, game_engine, file_name="starting_board.txt")
- def start (self)
- def run (self)
- None stop (self)

Public Attributes

- **root**
- **start_frame**
- **options_frame**
- **middle_opt**
- **neighb_type**
- **entry_range**
- **entry_states**
- **entry_s_min**
- **entry_s_max**
- **entry_b_min**
- **entry_b_max**
- **stop_btn**
- **file_frame**
- **entry_filepath**
- **path_btn**
- **sleep_opt_frame**
- **sleep_opt**
- **entry_sleep_time**
- **sleep_opt_btn**
- **board**

6.5.1 Detailed Description

```
Class representing main window that enables user to choose and specify options for the game.
```

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
def GUI.user_options.UserOptions.__init__ (
              self,
             int  width = WIDTH,
             int  height = HEIGHT )
```

```
Creates a window for choosing a way to specify game parameters or an option
to run with default parameters.
Possible choices:
    Start:  Starts a game if parameters are not provided, game runs with random parameters.
    Custom options: Redirects to a window for typing custom parameters.
    Custome file:   Redirects to a window for specifying path to json file with parameters.
    Save board:     Saves current board to a PNG file.
    Sleep:          Slows down the game.
    Close:  Closes all windows.
```

6.5.3 Member Function Documentation

6.5.3.1 board_in_file_has_correct_size()

```
def GUI.user_options.UserOptions.board_in_file_has_correct_size (
              self,
              board_from_file )
```

Returns whether board specified in given file has correct size.

6.5.3.2 board_in_file_is_correct_for_parameters()

```
def GUI.user_options.UserOptions.board_in_file_is_correct_for_parameters (
              self,
              board_from_file )
```

Returns whether board given in file is correct for given parameters
(e.g. all cells are inside the specified state range).

6.5.3.3 file_options()

```
def GUI.user_options.UserOptions.file_options (
              self )
```

Window for entering a path to json file with custom parameters.

6.5.3.4 options()

```
def GUI.user_options.UserOptions.options (
              self )
```

Window for manually specifying custom parameters.
Parameters to fill are: range, states,
count limits for a state to survive,
count limits for a dead cell to become a birth,
middle, neighbourhood type.

6.5.3.5 resleep()

```
def GUI.user_options.UserOptions.resleep (
              self )
```

Return to previous time setup.

6.5.3.6 run()

```
def GUI.user_options.UserOptions.run (
              self )
```

Runs the game. Updates window with game display each time new board becomes available.

6.5.3.7 save_board()

```
def GUI.user_options.UserOptions.save_board (
              self )
```

Saves currently displayed board as a PNG file.
Name of the file: board_{current_date_with_time}.

6.5.3.8 save_file_options()

```
def GUI.user_options.UserOptions.save_file_options (
              self )
```

Saves file path provided by the user.

6.5.3.9 save_options()

```
def GUI.user_options.UserOptions.save_options (
              self )
```

Saves custom options entered by the user.
If user provides incorrect values, an error message will be displayed.

Raises:
    ValueError: When one or more values are missing.

6.5.3.10 save_sleep()

```
def GUI.user_options.UserOptions.save_sleep (
              self )
```

Saves custom options entered by the user.

6.5.3.11 set_starting_board_from_file()

```
def GUI.user_options.UserOptions.set_starting_board_from_file (
              self,
              game_engine,
              file_name = "starting_board.txt" )
```

Sets starting board from a txt file named "starting_board.txt".
If there is no such file or the file is invalid, sets the board to a random state.

6.5.3.12 sleep_option()

```
def GUI.user_options.UserOptions.sleep_option (
              self )
```

Window for manually specifying custom parameters.
Parameters to fill are: range, states, middle, neighbourhood type.

6.5.3.13 start()

```
def GUI.user_options.UserOptions.start (
              self )
```

Starts game.

6.5.3.14 stop()

```
None GUI.user_options.UserOptions.stop (
              self )
```

Stops the game and closes all windows.

The documentation for this class was generated from the following file:

- GUI/user_options.py

Chapter 7

File Documentation

7.1 BoardEngine.hpp

```
1 #ifndef BoardEngine_H
2 #define BoardEngine_H
3 #include <array>
4 #include <iostream>
5 #include <string>
6
7 const int NUM_OF_ROWS = 50;
8 const int NUM_OF_COLS = NUM_OF_ROWS;
9
10 struct GameParameters {
11   int range = 1;              // range of neighborhood
12   int count_of_states = 5;    // number of possible states (starts from 0)
13   bool count_middle = false; // defines whether to count middle cell when
14                               // counting cells in neighborhood
15   int alive_min = 2;          // Smin
16   int alive_max = 3;          // Smax
17   int be_born_min = 3;        // Bmin
18   int be_born_max = 3;        // Bmax
19   std::string neighb = "NM"; // defines whether neighborhood type is NM or NN
20 };
21
22 typedef int CellValue;
23 typedef std::array<CellValue, NUM_OF_COLS> Row;
24 typedef std::array<Row, NUM_OF_ROWS> Board;
25
26 class BoardEngine {
27   GameParameters parameters;
28   Board current_board;
29   Board previous_board;
30
31 public:
32   BoardEngine();
33   void set_parameters(const int Rr, const int Cc, const bool Mm, const int Smin,
34                       const int Smax, const int Bmin, const int Bmax,
35                       const std::string Nn);
36   void set_cell(int row_num, int col_num, int new_value);
37   void print_current_board() const;
38   Board get_board() const;
39   int get_height() const;
40   int get_width() const;
41   void change_random_cell();
42   void randomize_board(int num_of_random_cells);
43   void calculate_next_state();
44   int count_neighbours(int row_num, int col_num,
45                        int max_num_of_neighbours) const;
46
47 private:
48   bool cell_is_dead(CellValue) const;
49   bool cell_is_alive(CellValue) const;
50   int add_bias_to_coordinate(int bias, int coordinate,
51                              int max_coordinate_value) const;
52   bool cell_in_neighbourhood(int current_row, int current_col, int center_row,
53                              int center_col) const;
54   bool dead_cell_should_be_born(int row_num, int col_num) const;
55   bool cell_should_be_incremented(int row_num, int col_num) const;
56   bool state_one_cell_should_survive(int row_num, int col_num) const;
57   int get_random_number_from_range(int min, int max) const;
58 };
59
60 #endif
```

Index