Basic application documentation

1- The user enters his **username** or register a new one (once created the user is selected automatically). He can access it as a guest.

2- The user must set the number of rows, the number of columns, and the number of mines.

3- A board is presented, waiting for the first click.

4- The user clicks in a cell:

- There is a generation of mines in the board excluding the first selected cell. Positioning the mines in a random place in the board.

- The time tracking starts

- The system shows game options:

+ pause the tracking

+ save current game session if the user is not a guest

5- If the user Clicks with the main button over a clear status cell, the system processes the action doing the exploration. If he clicks over one mine the game ends and all the mines are discovered

6- If the user Clicks using the alternative mouse button the cell is marked with the following status

clear >> flagged >> Suspicious >> clear

7- If there are no more cells to discover and all the mines have been flagged the game has been won by the user. The reset board button shows an image with glasses. Time tracking ends and presets some data stats to the user.

8. Save the board if the user is already identified uses this user, otherwise ask for the name and register the user before saving.

# Types of cells

For me, the key solution of this problem is an efficient resolution of the closer cells.

So, I Identified two different types of cells.

* Cells: The center ones which are surrounded by other cells.
* Border Cells: Do not have closer cells to one side or two these cells are delimited by the 0 and the number of columns and rows.

Based on the position (x, y) where **x** means columns and **y** means rows

**For cells,** the closer are

[

(x-1, y-1), (x, y-1), (x+1, y-1),

(x-1, y), (x+1, y),

(x-1, y+1), (x, y+1), (x+1, y+1)

]

**For Border Cells,** the closer cells are the same but removing all these that are outside the borders

For columns (x >= 0 && x <= board.cols)

For rows (y >= 0 && y<= board.rows)

E.g.: **x-1 < 0** The red ones will be filtered out

[

(x-1, y-1), (x, y-1), (x+1, y-1),

(x-1, y), (x+1, y),

(x-1, y+1), (x, y+1), (x+1, y+1)

]

E.g.: **x+1 > board.cols** The red ones will be filtered out

[

(x-1, y-1), (x, y-1), (x+1, y-1),

(x-1, y), (x+1, y),

(x-1, y+1), (x, y+1), (x+1, y+1)

]

# System Objects

* User
  + Id: int
  + Username: String
  + FirstName: String
  + LastName: String
  + Birthdate: date
  + Record: int64 (expressed in milliseconds)
* GameStatus (enumeration): short
  + NotStarted
  + Active
  + GameOver
* Board
  + Id: int
  + Owner: User
  + Name: string
  + Columns: Int
  + Rows: int
  + millisecond: lnt64
  + Cells: [cells]
  + Mines: [cells]
  + GameStatus: GameStatus
* PersistibleBoard
  + UserId: int
  + BoardName: String
  + BoardDefinition: JsonObject
* CellStatus (enumeration): short
  + Clear
  + Flagged
  + Suspicious
  + Revealed
* Cell
  + Id: int
  + Columns: Int
  + Rows: int
  + Status: CellStatus
  + ItIsAMine; boolean
  + CloserMinesNumber(): int
  + GetCloserCells(): [cells]
* Border Cell: Cell
  + GetCloserCells(): [cells]

# Api Response Body definition

Success Response

{

“status”: “success”

“data”: {

“board”: {

“id”: 1,

“Owner”: {

“id”: 1,

“firstName”: “ASenna”,

…

“Birthdate”: “1984-01-24T00:00Z”,

“Record”: 02121321321564

},

…

}

},

“message”: “The current board”

}

Error Response

{

“status”: “error”,

“data”: null

“message”: “The server encountered an internal error”

}

Used default Json Date Time format 2012-03-19T07:22Z

# Routes Specification

Route: /user/register

Action: Register a new user

* Method: Put
* Parameters:
  + username: string
  + firstName: string
  + lastName: string
  + birthdate: datetime
* Response body data
  + user

Errors

* DuplicatedUsername
* InvalidUsername
* InvalidBirthdate

Route: / user /getBoards

Action: List all the saved boards for a user

* Method: Get
* Parameters:
  + username: string
* Response body data

Boards array

Errors

* InvalidUsername

Notes: In case of no boards for the user the array is empty

Route: / board /save

Action: Save the board

* Method: Put
* Parameters:
  + Board: Board
* Response body data
  + Boolean

Errors

* InvalidBoard

Route: / board /resume

Action: Resume a saved board

Method: Post

* Parameters:
  + boardId: int
  + currentUserName: string
* Response body data

Board

Errors

* InvalidBoardId
* InvalidBoardForCurrentUser

Notes: invalid board for user is in case of the user does not match with the user defined in board

Route: / board /initialize

Action: Start a new board for a user, no username means guest.

Method: Put

* Parameters:
  + username: string
  + columns: int
  + rows: int
  + mines: int
* Response body data

Board

Errors

* InvalidBoardId
* InvalidBoardForUser

Route: /cell/flag

Action: Sets a flag in a cell.

Method: Post

* Parameters:
  + board: board
  + cellColumn
  + cellRow
* Response body data

Board

Errors

* InvalidCell
* GameNotStarted

Route: /cell/check

Action: Evaluate the clicked cell and perform the action of reveal, or game over

Method: Post

* Parameters:
  + board: board
  + cellColumn
  + cellRow
* Response body data

Board

Errors

* InvalidCell
* GameNotStarted