AutoChecker, Specification

Ricky Fan, HaoWei Chen

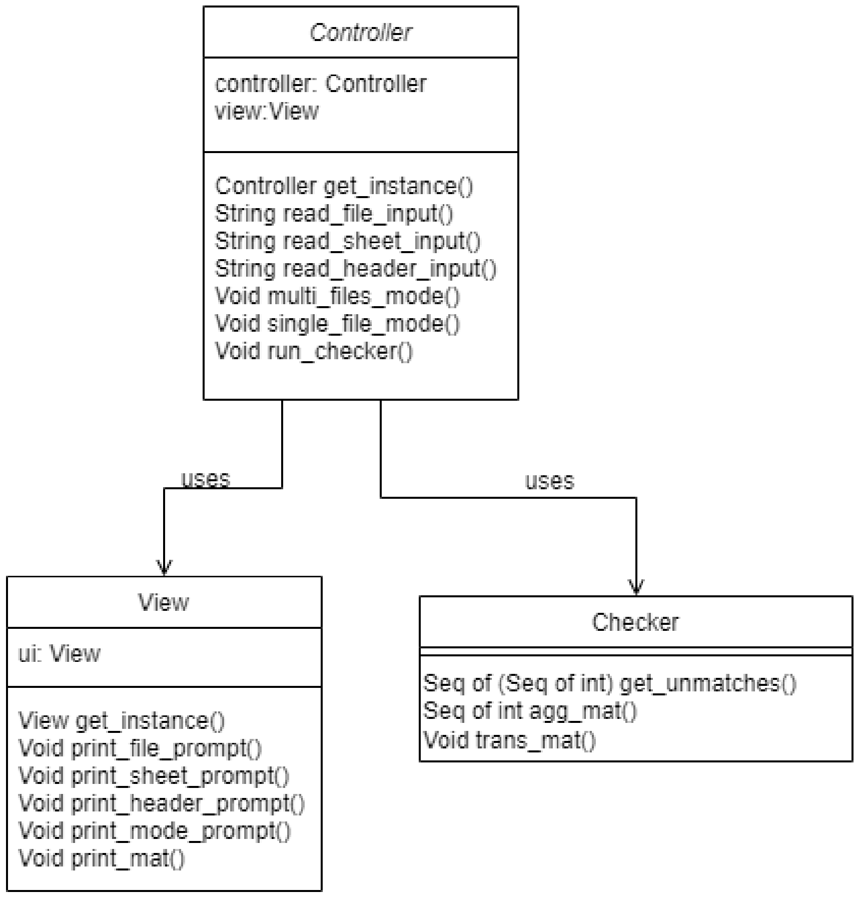
January 28, 2022

This Module Interface Specification (MIS) document contains modules, types and methods used to support the game of *2048*. At the start of the game, players are given the option to choose the size of the game board they want to play on; either 4x4, 5x5 or 6x6. Once the player has picked a size, a board with the user specified size will be displayed. Initially, the game start with two lowest possible numbers (2 or 4) at two random positions on the board. Then, players are required to move the cells left, right, up or down and everytime a player makes a move, a new number (2 or 4) will pop up at a random position on the board. While the players are moving the cells, when two cells with the same number on them collide, they will merge into one single cell with the sum of their original numbers. A player can beat the game by generating a cell with number 2048. However, the game is over when there are no empty cells and no adjacent cells with the same value. The game can be launched and play by typing `make demo` in terminal.

1 Overview of the design

This design applies Module View Specification (MVC) design pattern and Singleton design pattern. The MVC components are *Checker* (model module), *View* (view module), and *Controller* (controller module). Singleton pattern is specified and implemented for *View* and *Controller*

An UML diagram is provided below for visualizing the structure of this software architecture



The MVC design pattern are specified and implemented in the following way: the module *BoardT* stores the state of the game board, the abstract object *GameStatus* determines the status of the game.A view module *View* displays the game board and the state of the game using a text-based graphics. The controller *Controller* is responsibe for handling input actions

For *View* and *Controller*, use the get_instance() method to obtain the abstract object.

Checker Module (Abstract Object)

Module

Checker

Uses

None

Syntax

Exported Constants

None

Exported Types

Checker = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
get_unmatches	seq of $\mathbb{N}$, seq of $\mathbb{N}$	seq of (seq of $\mathbb{N}$)	
agg_cols	seq of (seq of $\mathbb{N}$), seq of $\mathbb{N} \to \mathbb{N}$	seq of $\mathbb{N}$	
trans_mat	seq of (seq of $\mathbb{N}$)		

Semantics

State Variables

None

State Invariant

None

Assumptions

- Every row that is being compared correspond to the same app prefix
- The number of rows that are being compared between columns are equal

Access Routine Semantics

get_unmatches(col1, col2):

- output: out := $[i : \mathbb{N} | i \in \{0..|\text{col1}| - 1\} : \text{col1}[i] \neq \text{col2}[i] \Rightarrow [i, \text{col1}[i], \text{col2}[i]]]$
- exception: none

agg_cols(cols, func):

- output: out := $[i : \mathbb{N} | i \in \{0..|\text{cols}| - 1\} : \text{func}(\text{cols}[i])]$
- exception: none

trans_mat(matrix):

- transition: $\forall i : \mathbb{N} \mid i < |\text{matrix}| \wedge (\forall j : \mathbb{N} \mid i \leq j < |\text{matrix}[i]| \wedge \text{tr\_swap}(i,\ j))$
- exception: none

Local Function:

tr_swap: seq of (seq of $\mathbb{N}$) $\times \mathbb{N} \times \mathbb{N} \rightarrow$ void
tr_swap(matrix, row, col):

```
  tmp := matrix[row][col]
  matrix[row][col] := matrix[col][row]
  matrix[col][row] := tmp
```

View Module

Module

View

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
get_instance		View	
print_file_prompt			
print_sheet_prompt			
print_header_prompt			
print_mode_prompt			
print_mat	seq of (seq of $\mathbb{N}$)		

Semantics

Environment Variables

window: A portion of computer screen to display the messages (i.e. the terminal)

State Variables

ui: View

State Invariant

None

Assumptions

- The View constructor is called for each object instance before any other access routine is called for that object.
- The constructor can only be called once.

Access Routine Semantics

get_instance( ):

- transition: ui := (ui = null ⇒ new View())
- output: *self*
- exception: none

print_file_prompt( ):

- transition: window := Displays a prompt message asking the user to enter a file directory

print_sheet_prompt( ):

- transition: window := Displays a prompt message asking the user to enter a sheet name

print_header_prompt( ):

- transition: window := Displays a prompt message asking the user to enter a header name

print_mode_prompt( ):

- transition: window := Displays a prompt message asking the user to select a checker mode

print_mat( ):

- transition: window := Displays the matrix row by row

Local Function:

__init__: void → View
__init__() ≡ new View()

Controller Module

Controller Module

Uses

Checker, View, pandas

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
get_instance	View	Controller	
read_file_input		String	
read_sheet_input		String	
read_header_input		String	
multi_file_mode	Map of String and String Pair of String and Map		
single_file_mode	Pair of String and Map Pair of String and Map		
run_checker			

Semantics

Environment Variables

None

State Variables

view: View
controller: Controller

State Invariant

None

Assumptions

- The Controller constructor is called for each object instance before any other access routine is called for that object.
- The constructor can only be called once.
- Assume that the view instances are already initialized before calling Controller constructor

Access Routine Semantics

get_instance(v):

- transition: controller := (controller = null $\Rightarrow$ new Controller (v))
- output: *self*
- exception: None

read_file_input():

- output: *input* : String, file directory entered by the User
- exception: none

read_sheet_input():

- output: *input* : String, sheet name entered by the User
- exception: none

read_header_input():

- output: *input* : String, column header entered by the User
- exception: none

read_mode_input():

- output: *input* : String, mode selected by the User

- exception: none

single_file_mode(file1, file2):

- transition: operational method

```
arr1 := get_col_arr(file1[0], file1[1])
arr2 := get_col_arr(file2[0], file2[1])
unmatches_rows := Checker.get_unmatches(arr1, arr2)
view.print_mat(unmatches_rows)
```

- output: none

multi_files_mode(f_map, file2):

- transition: operational method

```
f_arr := [$f$:String | $f \in$ f_map.keys() : get_col_arr($f$, f_map[$f$]) ]
Checker.trans_mat(f_arr)
arr1 := Checker.agg_cols(f_arr)
arr2 := get_col_arr(file2[0], file2[1])
unmatches_rows := Checker.get_unmatches(arr1, arr2)
view.print_mat(unmatches_rows)
```

- output: none

run_checker():

- transition: operational method for running the game.
  Start by prompting the user to select the checker mode (single file vs multi files)
  - If single file mode is selected:
    ```
    $f1$ := ()
    $f2$ := ()
    populate_pair($f1$)
    populate_pair($f2$)
    single_file_mode($f1$, $f2$)
    ```
  - If multi files mode is selected:
    ```
    $f\_map$ := {}
    ```

```
        $f1$ := ()
        populate_pair($f1$)
        inputs := get_inputs()
        f_map[inputs[0]] = {'sheet':inputs[1], 'header':inputs[2]}
        populate f_map by repeating step 3 - 4 five times
        multi_files_mode($f\_map$, $f1$)
```

- output: None

Local Function:

__init__: View → Controller
__init__(*view*) ≡ new Controller(*view*)

cal_sum: seq of $\mathbb{N} \rightarrow \mathbb{N}$
cal_sum(seq) ≡ $(+s : \mathbb{N} \mid s \in \text{seq} : s)$

cal_avg: seq of $\mathbb{N} \rightarrow \mathbb{N}$
cal_avg(seq) ≡ cal_sum(seq)/|seq|

get_col: String × Map of String and String → seq of $\mathbb{N}$
get_col_arr(file_dir, info_map):

```
  df := load_xlsx(file_dir, info_map['sheet'])

  return df[info_map['header']].values
```

get_inputs: seq of String
get_inputs(p):

```
  view.print_file_prompt()

  file_dir := read_file_input()

  view.print_sheet_input()

  sheet := read_sheet_input()

  view.print_header_input()

  header := read_header_input()
  return [file_dir, sheet, header]

populate_pair: Pair → void
populate_pair(p):

  inputs := get_inputs()

  $p$[0] := inputs[0]

  $p$[1] := 'sheet':inputs[1], 'header':inputs[2]

load_xlsx: String × String → DataFrame load_xlsx(file_dir, sheet_name) ≡ pandas.read_excel(file_dir,
sheet_name)
```