2048

Documentation for "2048" in Ruby, by Rafał Leja, created using Mintlify (vscode extention), Yard and Pandoc.

Top Level Namespace

Defined Under Namespace

Classes: Blok, Grid, InputField, Leaderboard, Scene, ScoreCounter

Class: Blok

Inherits:
Object

- Object
- Blok

show all

Defined in:
Blok.rb

Overview

The Blok class is responsible for handling square blocks with numbers in a game. klasa odpowiedzialna za obsługę kwadracików z liczbami

Instance Method Summary collapse

- #**add** => Object

  The function creates a square and a text object with specific properties based on the value of a variable.

- #**initialize**(val, x, y, size) => Blok constructor

  size: The parameter “size” is referring to the size of a shape.

- #**size** => Object
- #**size=**(v) => Object
- #**val** => Object
- #**val=**(v) => Object

  This is a Ruby class with getter and setter methods for instance variables val, x, y, and size.

- **#x** => Object
- **#x=**(v) => Object
- **#y** => Object
- **#y=**(v) => Object

Constructor Details

#initialize(val, x, y, size) => [Blok]("Blok (class)")

size: The parameter “size” is referring to the size of a shape.

14 15 16 17 18 19

def initialize(val, x, y, size) @val = val @x = x @y = y @size = size end

Instance Method Details

#add => Object

The function creates a square and a text object with specific properties based on the value of a variable.

24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41

def add() Square.new(x: @x, y: @y, size: @size, color: if @val == 0 [0.5, 0.28, 0.1, 1] else [0.5 + (Math.log(@val, 2)/11), 0.28 + (Math.log(@val, 2)/11), 0.1, 1] end) if @val != 0 tekst = Text.new(@val, x: @x + @size/2, y: @y + @size/2) end end

#size => Object

77 78 79

def size() @size end

#size=(v) => Object

73 74 75

def size=(v) @size = v end

#val => Object

53 54 55

def val() @val end

#val=(v) => Object

This is a Ruby class with getter and setter methods for instance variables val, x, y, and size.

Args:

`v: This is a variable that is being passed as an argument to the `val=` method. It is used t`

the value of the instance variable '@val'.

49 50 51

def val=(v) @val = v end

#x => Object

61 62 63

def x() @x end

#x=(v) => Object

57 58 59

def x=(v) @x = v end

#y => Object

69 70 71

def y() @y end

#y=(v) => Object

65 66 67

def y=(v) @y = v end

Class: Grid

Inherits:
Object

- Object
- Grid

show all

Defined in:
Grid.rb

Overview

The Grid class contains methods for manipulating and checking a 4x4 grid of blocks in a game of 2048.

Instance Method Summary collapse

- #**add** => Object

  This function sets the position and size of blocks in a grid layout.

- #**check**(state) => Object

  The function checks if there are any empty blocks in a 4x4 grid and updates the state accordingly.

- #**down** => Object

- #**event**(e, state) => Object

  The function takes an event and a state as input, and based on the key pressed in the event, it calls different functions to move in different directions, checks the state, and randomizes the game board.

- #**initialize**(window) => Grid constructor

  window: The "window" parameter is an object representing the game window or screen.

- #**left** => Object

- #**randomize** => Object

  The function randomizes the placement of a new block with a value of 2 in a 4x4 grid.

- #**right** => Object

- #**sum** => Object

  This function calculates the sum of values in an array of blocks.

- #**up** => Object

  The above code defines four functions for moving blocks in a 2048 game board in the up, down, left, and right directions.

Constructor Details

#initialize(window) => [Grid]("Grid (class)")

window: The "window" parameter is an object representing the game window or screen.

10 11 12 13 14 15 16 17 18 19 20

def initialize(window) @window = window @w = @window.width @h = @window.height @size = @h * 0.2 @blocks = Array.new() for x in 0..15 do @blocks.push(Blok.new(0, 0, 0, 0)) end randomize() end

Instance Method Details

#add => Object

This function sets the position and size of blocks in a grid layout.

24 25 26 27 28 29 30 31 32 33 34 35

def add() @w = @window.width @h = @window.height @blocks.each { |n| n.add() } for x in 0..3 do for y in 0..3 do @blocks[4*x + y].x = (@w/2) + (x-2)*@size @blocks[4*x + y].y = (@h/2) + (y-2)*@size @blocks[4*x + y].size = @size*0.8 end end end

#check(state) => Object

The function checks if there are any empty blocks in a 4x4 grid and updates the state accordingly.

Args:

```
state: The parameter "state" is an array or hash that stores the current state of the game.
```

The method “check” seems to be checking if there are any empty blocks in the game board (represented by the instance variable “@blocks”). If there are no empty blocks, the method updates the “

168 169 170 171 172 173 174 175 176 177 178

def check(state) sum = 0 for i in 0..15 do
if @blocks[i].val == 0 sum += 1 end end if sum == 0 state[0] = 2 end end

#down => Object

131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146

def down() for x in 0..3 do for i in 0..3 do for y in [0, 1, 2] do if @blocks[x*4+y].val == 0 elsif @blocks[x*4+y].val == @blocks[x*4+(y+1)].val @blocks[x*4+(y+1)].val = 2 @blocks[x*4+y].val = 0 elsif @blocks[x*4+(y+1)].val == 0 @blocks[x*4+(y+1)].val = @blocks[x*4+y].val @blocks[x*4+y].val = 0 end end end end

#event(e, state) => Object

The function takes an event and a state as input, and based on the key pressed in the event, it calls different functions to move in different directions, checks the state, and randomizes the game board.

Args:

`e: This parameter is an event object that represents a user input event, such as a key`

press. The function is designed to respond to user input and update the game state accordingly.

`state: The state parameter is a variable or object that represents the current state of the`

game or program. It is being passed into the event function as an argument, suggesting that the function may be responsible for updating the state based on user input.

49 50 51 52 53 54 55 56 57 58 59 60 61 62

def event(e, state) c = e.key.to_s if c == "w" up() elsif c == "s" down() elsif c == "d" right() elsif c == "a" left() end check(state) randomize() end

#left => Object

97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112

def left() for y in 0..3 do for i in 0..3 do for x in [3, 2, 1] do if @blocks[x*4+y].val == 0 elsif @blocks[x*4+y].val == @blocks[(x-1)*4+y].val @blocks[(x-1)*4+y].val = 2 @blocks[x*4+y].val = 0 elsif @blocks[(x-1)*4+y].val == 0 @blocks[(x-1)*4+y].val = @blocks[x*4+y].val @blocks[x*4+y].val = 0 end end end end end

#randomize => Object

The function randomizes the placement of a new block with a value of 2 in a 4x4 grid.

150 151 152 153 154 155 156 157 158 159

def randomize() for i in 0..255 x = rand(4) y = rand(4) if @blocks[x*4+y].val == 0 @blocks[x*4+y].val = 2 break end end end

#right => Object

114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129

def right() for y in 0..3 do for i in 0..3 do for x in [0, 1, 2] do if @blocks[x*4+y].val == 0 elsif @blocks[x*4+y].val == @blocks[(x+1)*4+y].val @blocks[(x+1)*4+y].val = 2 @blocks[x*4+y].val = 0 elsif @blocks[(x+1)*4+y].val == 0 @blocks[(x+1)*4+y].val = @blocks[x*4+y].val @blocks[x*4+y].val = 0 end end end end end

#sum => Object

This function calculates the sum of values in an array of blocks.

Returns:

`The sum of the values of the first 16 elements in an array called` `` `@blocks` ``.

69 70 71 72 73 74 75

def sum() sum = 0 for i in 0..15 do sum += @blocks[i].val end return sum end

#up => Object

The above code defines four functions for moving blocks in a 2048 game board in the up, down, left, and right directions.

80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95

def up() for x in 0..3 do for i in 0..3 do for y in [3, 2, 1] do if @blocks[x*4+y].val == 0 elsif @blocks[x*4+y].val == @blocks[x*4+(y-1)].val @blocks[x*4+y].val = 2 @blocks[x*4+y].val = 0 elsif @blocks[x*4+(y-1)].val == 0 @blocks[x*4+y].val = @blocks[x*4+y].val @blocks[x*4+y].val = 0 end end end end end

Class: InputField

Inherits:
Object

- Object
- InputField

show all

Defined in:
InputField.rb

Overview

The InputField class creates a text input field that can receive keyboard input and return a state.

Instance Method Summary collapse

- #**add** => Object

  This is a Ruby function that creates a new Text object with specific properties.

- #**event**(e, state) => Object

  This function handles keyboard events for a text input field in Ruby.

- #**initialize**(text: [""], x: 0, y: 0, size: 20) => InputField constructor

  This is a Ruby constructor that initializes instance variables for text, x, y, and size.

Constructor Details

#initialize(text: [""], x: 0, y: 0, size: 20) => [InputField]("InputField (class)")

This is a Ruby constructor that initializes instance variables for text, x, y, and size.

5 6 7 8 9 10 11

def initialize(text: [""], x: 0, y: 0, size: 20) @state = 0 @x = x @y = y @text = text @size = size end

Instance Method Details

#add => Object

This is a Ruby function that creates a new Text object with specific properties.

16 17 18 19 20 21 22 23

def add() Text.new(@text[0], x: @x - ((@size * @text[0].length)/4), y: @y, size: @size) end

#event(e, state) => Object

This function handles keyboard events for a text input field in Ruby.

Args:

`e: This parameter represents the event that triggered the function. It is an object that`

contains information about the event, such as the key that was pressed.

`state: The state parameter is a variable that stores the current state of the program or`

application. It can be used to keep track of various variables, settings, or user inputs. In this specific code, the state parameter is an array that stores a single integer value.

34 35 36 37 38 39 40 41 42 43 44 45 46 47

def event(e, state) c = e.key.to_s if(c.match(/[a-zA-Z]/) && c.length ==1 && @text[0].length < 30) if @text[0].length == 0 @text[0] += c.upcase else @text[0] += c end elsif c == "backspace" @text[0] = @text[0].chop elsif c == "return" && @text[0].length != 0 state[0] = 1 end end

Class: Leaderboard

Inherits:
Object

- Object

- Leaderboard

show all

Defined in:
Leaderboard.rb

Overview

The Leaderboard class manages the saving and displaying of player scores in a game.

Instance Method Summary collapse

- #**add** => Object

  The function adds scores to a table and displays the top 8 scores in a text table.

- #**event**(e, s) => Object

  This is a Ruby function that checks if the key pressed is “return” and closes the event if it is.

- #**initialize**(playerScore, scoreTable, window) => Leaderboard constructor

  that the game is being displayed in.

- #**save** => Object

  This function saves the player’s score to a file and updates the score table.

Constructor Details

#initialize(playerScore, scoreTable, window) => [Leaderboard]("Leaderboard (class)")

that the game is being displayed in. It’s an object representing the game window.

14 15 16 17 18 19

def initialize(playerScore, scoreTable, window) @playerScore = playerScore @scoreTable = scoreTable @window = window @saved = false end

Instance Method Details

#add => Object

The function adds scores to a table and displays the top 8 scores in a text table.

40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

def add() if !@saved save() @saved = true end

tmpArray = @scoreTable.to_a

textTable = []

numberOfEntries = [8, tmpArray.length].min - 1

for i in 0..numberOfEntries do textTable.push(Text.new((i+1).to_s + ". " + tmpArray[i][0] + ": " + tmpArray[i][1].to_s, x: @window.width*0.3, y: @window.height * 0.2 + i*(@window.height * 0.1), size: 30)) end end

#event(e, s) => Object

This is a Ruby function that checks if the key pressed is “return” and closes the event if it is.

Args:

`e: This parameter refers to an event object, which could be triggered by a user action such`

as pressing a key on the keyboard or clicking a button. The code is checking for a specific key press event, as indicated by the line ‘c = e.key.to_s‘.

`s: The parameter "s" is not used in the given code snippet, so it is difficult to determine`

purpose without additional context.

71 72 73 74 75 76

def event(e, s) c = e.key.to_s if c == "return" close end end

#save => Object

This function saves the player’s score to a file and updates the score table.

23 24 25 26 27 28 29 30 31 32 33 34 35 36

def save()

if @scoreTable[@playerScore[0]] == -1 @scoreTable[@playerScore[0]] = @playerScore[1]
elsif @scoreTable[@playerScore[0]] < @playerScore[1] @scoreTable.delete(@playerScore[0])
@scoreTable[@playerScore[0]] = @playerScore[1]
end

@scoreTable = @scoreTable.sort_by {|k, v| -v}.to_h

tmpStr = Marshal.dump(@scoreTable) File.write("score", tmpStr) end

Class: Scene

Inherits:
Object

- Object

- Scene

show all

Defined in:
Scene.rb

Overview

The Scene class initializes with items and has methods to add and handle events for those items.

Instance Method Summary collapse

- #**add** => Object

  This is a Ruby function that iterates through a collection of items and calls the “add” method on each item.

- #**event**(e, state) => Object

  This function iterates through a collection of items and calls their event method with the given event and state parameters, ignoring any NoMethodError exceptions that may occur.

- #**initialize**(items) => Scene constructor

  within.

Constructor Details

#initialize(items) => [Scene]("Scene (class)")

within

12 13 14

def initialize(items) @items = items end

Instance Method Details

#add => Object

This is a Ruby function that iterates through a collection of items and calls the “add” method on each item.

19 20 21

def add() @items.each { |n| n.add() } end

#event(e, state) => Object

This function iterates through a collection of items and calls their event method with the given event and state parameters, ignoring any NoMethodError exceptions that may occur.

Args:

`e: The "e" parameter in the "event" method is an event object or event data that is being`

passed as an argument to the method. It is used to trigger some action or behavior in the items that are stored in the "@items" array.

`state: The "state" parameter is an object or data structure that represents the current`

state of the program or system. It is being passed as an argument to the "event" method, which is responsible for handling events or actions that occur within the program and updating the state accordingly.

35 36 37 38 39 40 41 42

def event(e, state) @items.each do |item| begin item.event(e, state) rescue NoMethodError end end
end

Class: ScoreCounter

Inherits:
Object

- Object
- ScoreCounter

show all

Defined in:
ScoreCounter.rb

Overview

The ScoreCounter class initializes with a grid, window, and player score, and has a method to add up the grid and display the score as text.

Instance Method Summary collapse

- #**add** => Object

  This Ruby function calculates the sum of elements in a grid and displays the result as a text string.

- #**initialize**(grid, window, playerScore) => ScoreCounter constructor

parameters.

Constructor Details

#initialize(grid, window, playerScore) => [ScoreCounter]("ScoreCounter (class)")

parameters.

16 17 18 19 20

def initialize(grid, window, playerScore) @grid = grid @window = window @score = playerScore end

Instance Method Details

#add => Object

This Ruby function calculates the sum of elements in a grid and displays the result as a text string.

25 26 27 28 29 30 31

def add() @score[1] = @grid.sum() @text = Text.new("Score = " + @score[1].to_s, x: @window.width * 0.1, y: @window.height * 0.1) end

Generated on Mon Jun 19 15:15:58 2023 by yard 0.9.34 (ruby-3.0.1).