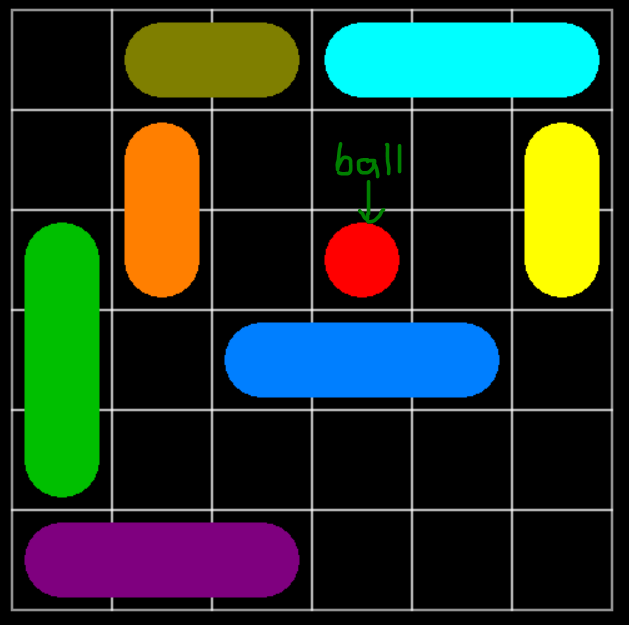
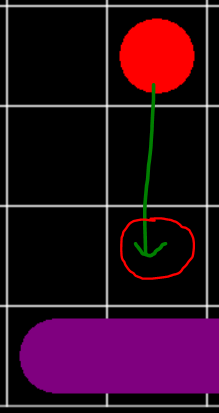
Description: A puzzle game where you have to move obstacles so a ball can drop to the bottom.



When there are no obstacles below the red ball, the red ball falls until there is an obstacle directly under it.

Engine: Love2D

Language: Lua

# Reference

Key:

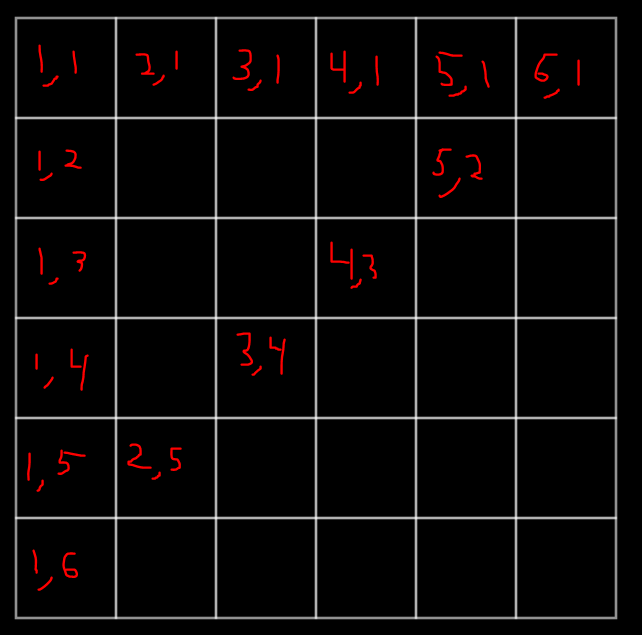
* Red: Bugs in the code
* Orange: No bugs present, but there can be a better way to code it
* Green: Fine for now
* Blue: Not finished, should be ready for use in the future

main.lua:

* love.load(): Initializes the variables
* love.update(dt): Constantly updates the game
* love.draw(): Constantly called and calls the :draw() method in the level that is being solved

scales.lua:

* int getHeightFromDecimal(h): Returns a number for the height position on the screen for a float h
* int getWidthFromDecimal(w): Returns a number for the width position on the screen for a float w
* float getDecimalFromHeight(d): Returns a float for the height position on the screen for d
* float getDecimalFromWidth(d): Returns a float for the width position on the screen for d



The coordinate system for a grid

grid.lua: (object)

* grid.new(x, y, d, r, c):

x = upper-left corner x-value

y = upper-left corner y-value

d = length in both directions (a grid is a square, but a cell is a rectangle)

r = number of rows

c = number of columns

* Grid:draw(): draws the grid
* number[2] Grid:getColumn(x): returns an array representing the column that the x parameter lies in

[1] = x-value in pixels

[2] = x-value as a float

* number[2] Grid:getRow(y): returns an array representing the row that the y parameter lies in

[1] = y-value in pixels

[2] = y-value as a float

* int[2] Grid:getID(): returns the x and y value in the grid array:

[1] = x-value from 1 through 6

[2] = y-value from 1 through 6

* int[2] Grid:getID2(): does the exact same thing as Grid:getID() but has a faster computation run-time
* float Grid:getDecimalFromGridWidth(p): self-explanatory
* float Grid:getDecimalFromGridHeight(p): self-explanatory

obstacle.lua: (object)

* Obstacle.new(l, c, iX, iY, h, g):

l = length of obstacle

c = color of obstacle

iX = initial x-value of obstacle (1-6)

iY = initial y-value of obstacle (1-6)

h = is horizontal (boolean)

g = grid the obstacle lies on

* Obstacle:draw(): Draws an obstacle

**BUG:** When an obstacle is drawn off the grid, the length becomes 1 when the obstacle should be reverted back to where it was before it was drawn off the grid

ball.lua: (object)

* Ball.new(iX, iY, g):

iX = initial x-value of ball (1-6)

iY = initial y-value of ball (1-6)

g = grid the ball lies in

x = the current x-value of ball (1-6)

y = the current y-value of ball (1-6)

* Ball:draw(): draws the ball
* Ball:update(): checks if the ball should be moved down
* Ball:regenerate(): moves the ball down until either it hits another obstacle or falls off the grid

levels.lua: (object)

* love.mousereleased(x, y, button): An event that is triggered when a mouse button is released
* love.mousepressed(x, y, button): An event that is triggered when a mouse button is pressed
* love.mousemoved(x, y): An event that is triggered when the mouse is moved
* Level.new(grid, ball, obstacles)

grid = grid the level uses

ball = ball in the level

obstacles = an array of obstacles in the level

* Level:draw(): draws everything in the level
* Level:update(): updates everything in the level
* boolean Level:isRoot(x, y): checks if an obstacle at the position x,y is the root of an obstacle
* number[2] OR false Level:findRoot(x, y): finds the root of the obstacle on position x,y

[1] = the x-value of the root (1-6)

[2] = the y-value of the root (1-6)

* obstacle Level:getRoot(x, y): returns the obstacle whose root is on position x,y
* number[2] Level:getMouseGrid(): returns the x-value and y-value of where the mouse is located in the game window

[1] = the x-value of the mouse

[2] = the y-value of the mouse

* Level:moveObstacle(x, y): moves the obstacle
* **BUG:** Literally the entire thing