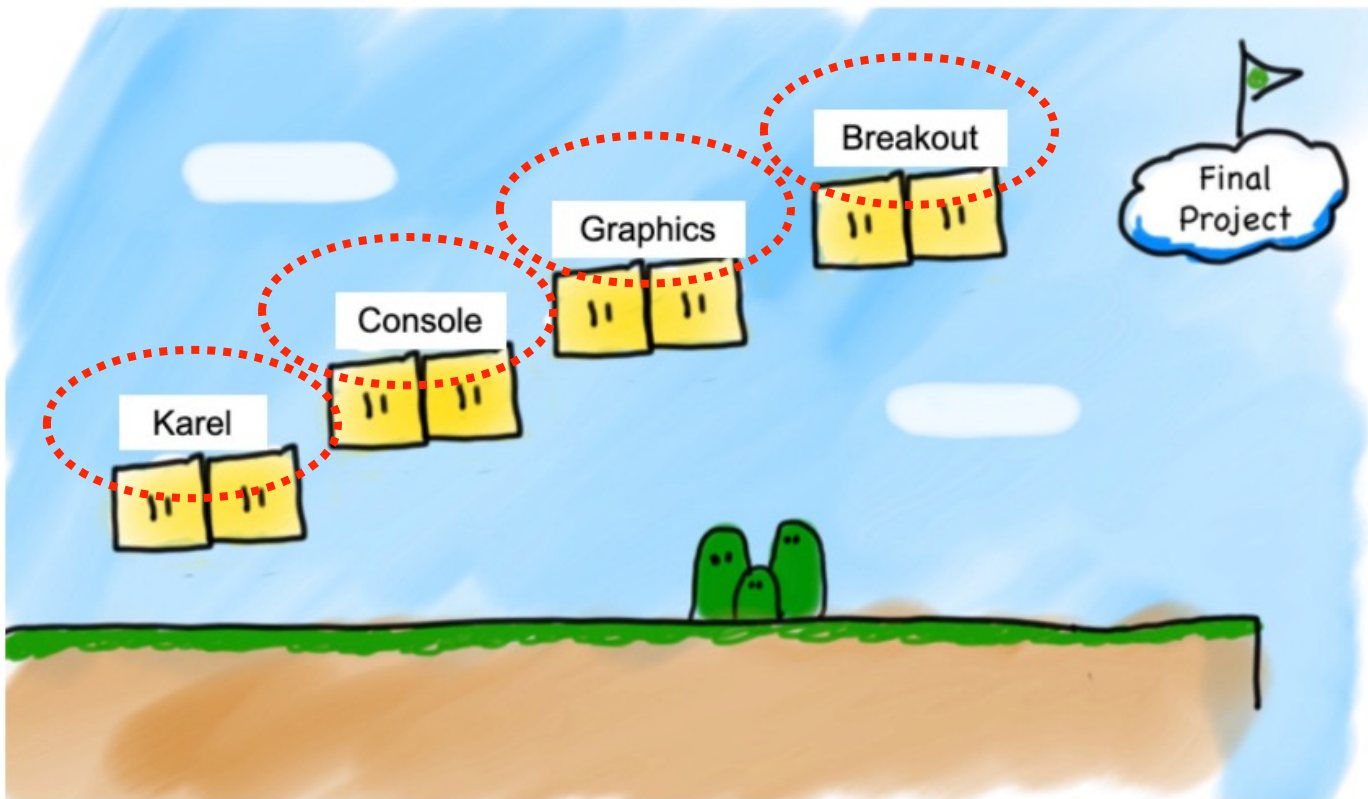
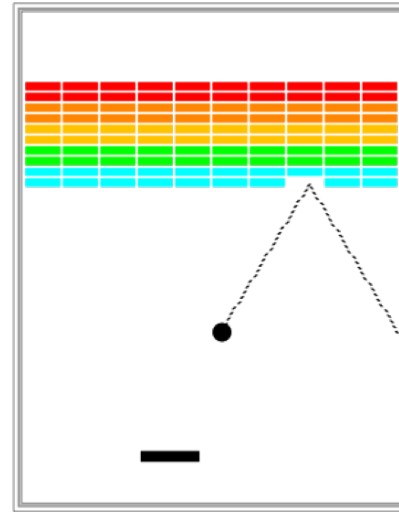
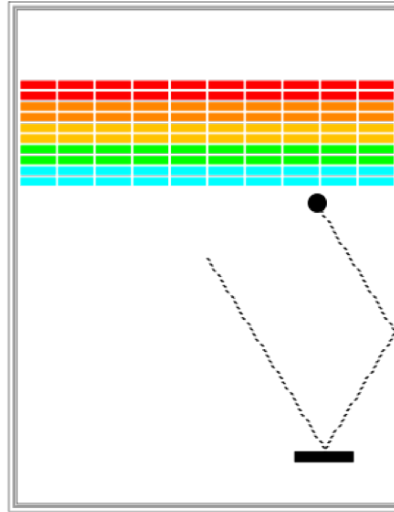
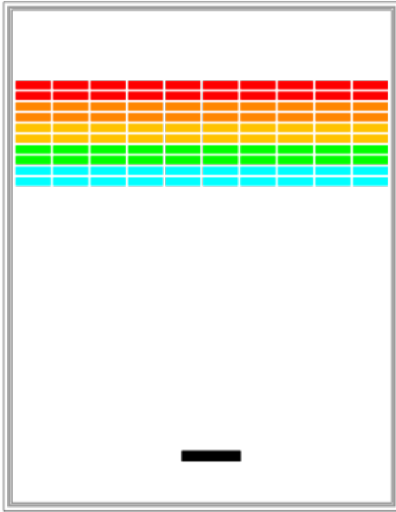


# CS Bridge, Lecture 13 Breakout



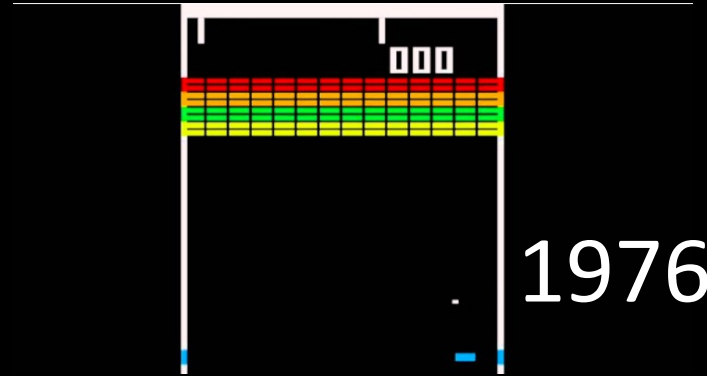
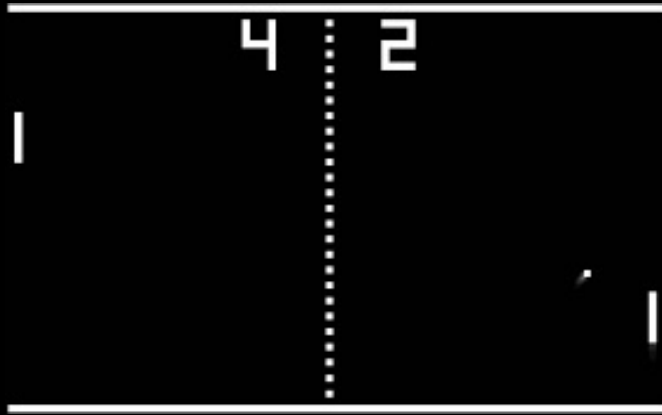




# Breakout

Submission deadline: Friday morning

# 1972: Pong by Atari



1976

Designer(s)

Nolan Bushnell

Steve Bristow

[https://en.wikipedia.org/wiki/Breakout\\_\(video\\_game\)](https://en.wikipedia.org/wiki/Breakout_(video_game))

Steve Wozniak

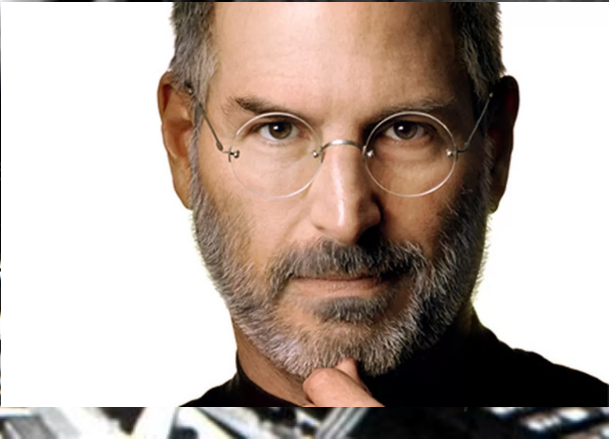




*Bushnell offered the little specialized know with a small number between them. Wozniak's deadline was met at his day job at Hewlett-Packard.*



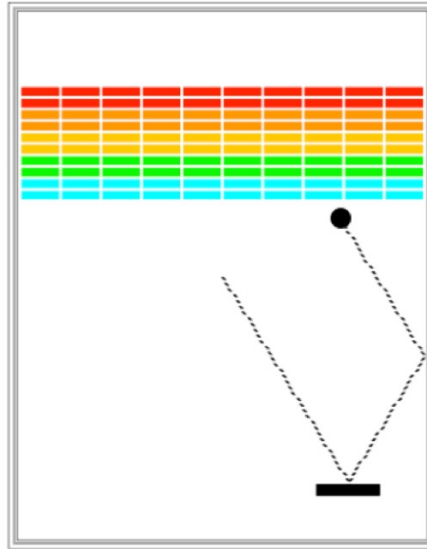
*Wozniak had no idea how to design but Jobs had a lot of experience and Jobs had a lot of time to spend on it. Wozniak had a lot of time to spend on it. Wozniak had a lot of time to spend on it.*



*Jobs had a lot of time to spend on it. Jobs had a lot of time to spend on it. Jobs had a lot of time to spend on it.*

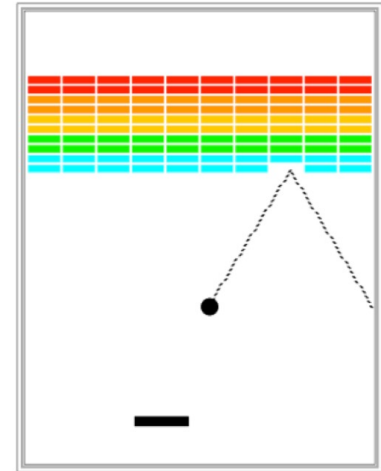
# Requirements

- ❖ Goal is to break all bricks
- ❖ User has 3 turns
- ❖ Ball in the center start moving towards bottom at random angle
- ❖ Ball bounces paddle, right wall and the bricks



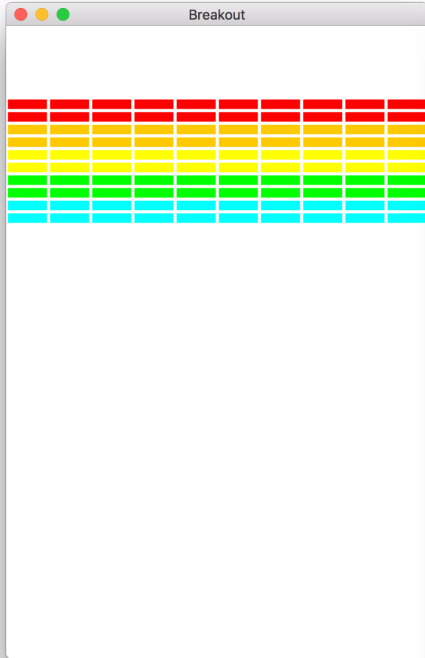
# Requirements

- ❖ When hit a brick, the brick disappears
- ❖ Ball moves down either hitting left wall, paddle or bottom wall
- ❖ The turn continues until
  - ❖ The ball hits the lower wall.
    - ❖ NEXT TURN or YOU LOOSE
  - ❖ The last brick is eliminated.
    - ❖ YOU WIN

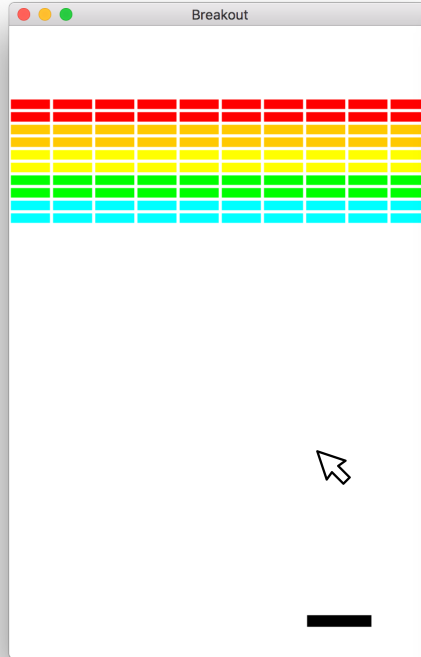


# Big program. Do it in parts

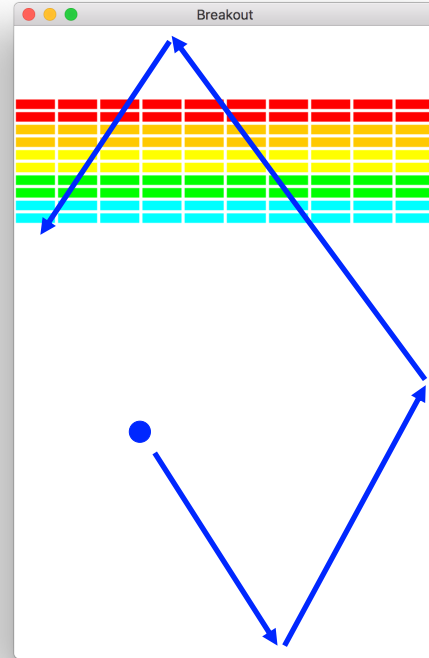
1



2



3





# Some suggestions before we start

**A task may include parts that look like other tasks:**

*Bricks in Breakout ~ Checkerboard project*

*Programming is not patch-work, re-think on the design*

*The first idea that pops-up may not be the best*

**This is an individual adventure:**

*Discuss concepts, ask questions about problems you face*

*Do not copy-paste someone else's code*

**Think about decomposition, write clean code, add comments:**

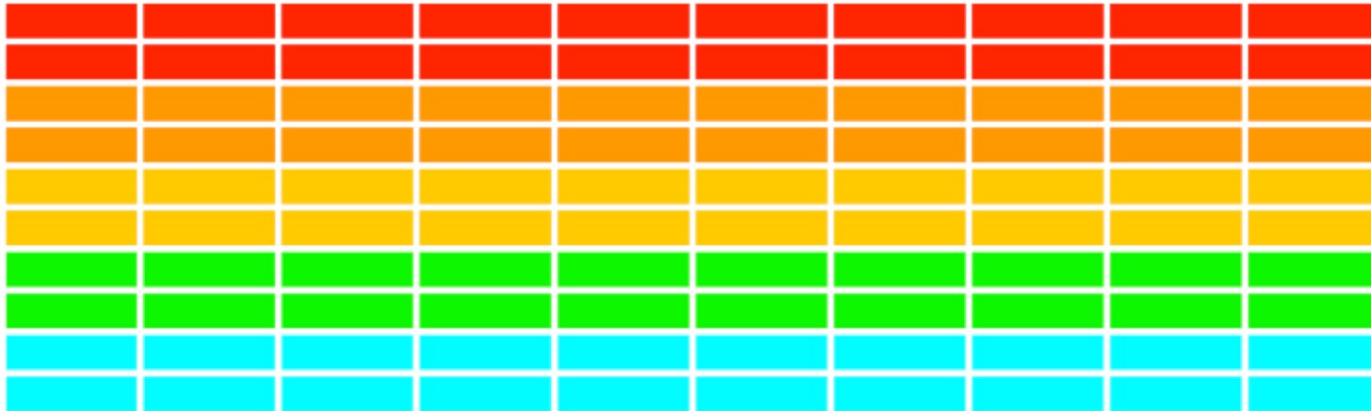
*Design on paper, apply several steps of decomposition*

*Use meaningful function names*

*Add comments to your code*

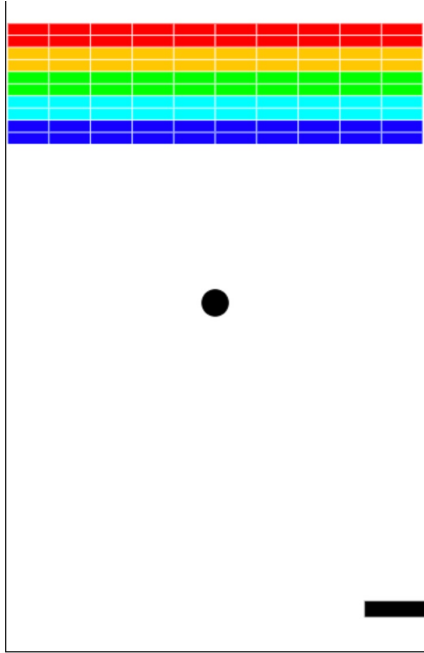
# Part 1 - Creating Bricks

- ❖ Number, dimensions and spacing of bricks as constants (define them at the beginning)
- ❖ Calculate x coordinate of the first column - so that bricks are centered
- ❖ Colors of bricks - red, orange, yellow, green and cyan



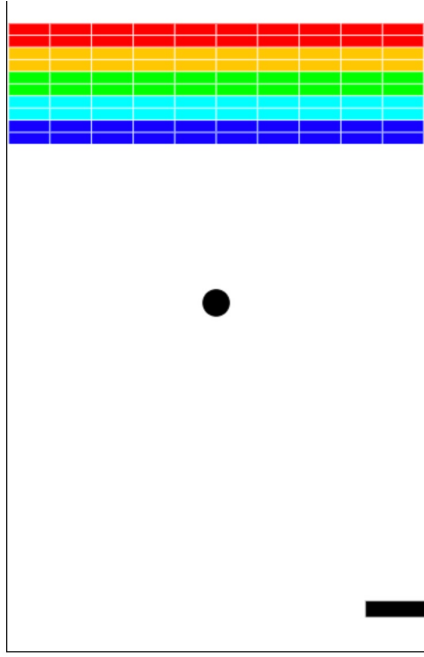
No need for lists here because we don't plan accessing and/or modifying parameters of the objects

## Part 2 - Add and move ball



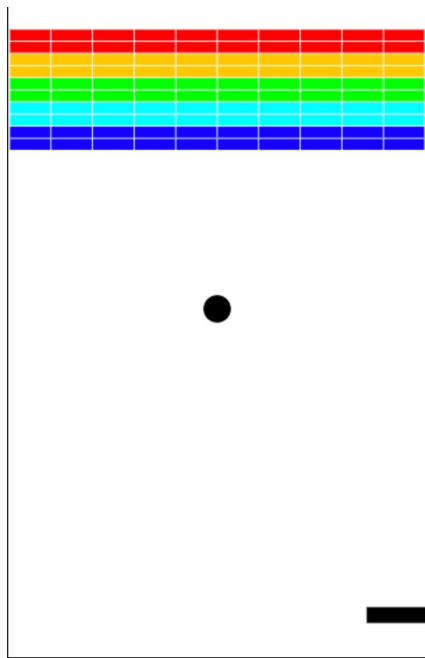
- ❖ Put filled ball at the center of the window
- ❖ Velocity of the ball - declared as variables (specify a max speed (constant))
- ❖ Pick random values for change in x and y
- ❖ Move the ball using
  - ❖ **canvas.move(object, change\_x, charge\_y)**
  - ❖ **canvas.moveto(object, new\_x, new\_y)**
- ❖ Move the ball - Initially - ball heading downwards

# Part 2 - Bouncing ball



- ❖ Animation loop - where ball is moving
  - ❖ Bounce from right, left and top walls
  - ❖ Bottom wall - start in the middle again
- ❖ When bounced from top wall
  - ❖ Inverse **change\_y**
- ❖ When bounced from left or right wall
  - ❖ Inverse **change\_x**
- ❖ Update the canvas

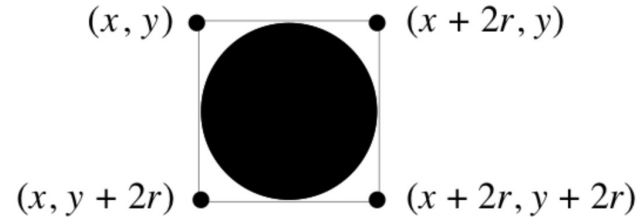
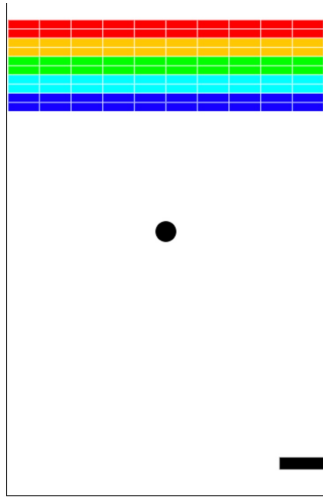
## Part 3 - Add Paddle



- ❖ Define some constants for the dimension and location of the paddle: width, height, y-offset from bottom
- ❖ Create Paddle - filled rectangle at a specific location
- ❖ Link paddle move with mouse move
  - ❖ Move paddle - track x coordinate of the mouse only
  - ❖ Use **`mouse_x = canvas.get_mouse_x()`**

# Part 4 - Check for collision

- ❖ Did ball collide with another object in the window
- ❖ **canvas.find\_overlapping** (x1, y1, x2, y2) which return list of objects overlying with rectangle whose upper left is (x1, y1) and bottom right is (x2, y2)



```
colliding_list = canvas.find_overlapping(.....)
for loop to get a collider object in each loop:
.... do something with collider object
```

Think about writing a *check\_collision* function that implements all listed above

# Part 4 - Check for collision

## ❖ Use of list

```
# this graphics function gets the location of the ball as a list
```

```
ball_coords = canvas.coords(ball)
```

```
# the list has four elements:
```

```
x_1 = ball_coords[0]
```

```
y_1 = ball_coords[1]
```

```
x_2 = ball_coords[2]
```

```
y_2 = ball_coords[3]
```

```
# we can then get a list of all objects in that area
```

```
colliding_list = canvas.find_overlapping(x_1, y_1, x_2, y_2)
```

## Part 4 - Check for collision

- ❖ Collide with a paddle
  - ❖ Bounce ball towards up
- ❖ Collide with a brick
  - ❖ Bounce ball towards down
  - ❖ Remove brick from the screen  
`canvas.delete(square) # deletes the object called square`
  - ❖ Count the number of removed bricks
    - ❖ That's how you know you hit the last brick



You  
can  
do it!

