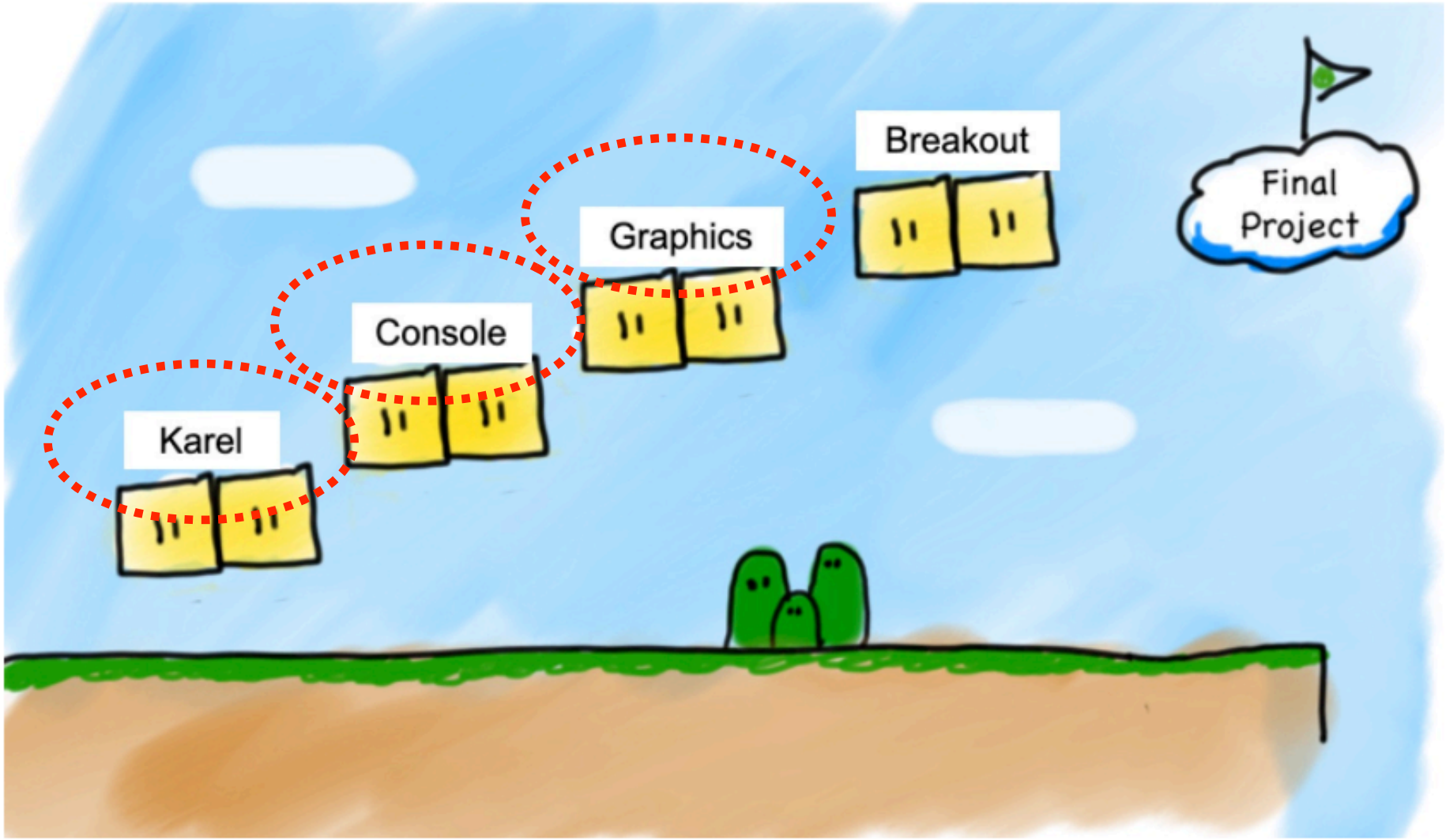
Intro to Computer Science

Summer 2020

August 4th to August 20th, online

Breakout

Adopted from Stanford Uni's CS106ap course slides by Kylie Jue and Sonja Johnson-Yu and Code in Place by Piech and Sahami; Koca Uni's Comp125 course by Ayca Tuzmen


Breakout
Graphics
Console
Karel
Final
Project

Today's topics

1. Review
    - Mouse
2. Breakout
    - Requirements
    - Stages in development

Mouse Event

At any time, we can ask the canvas for

- the current location of the mouse

```
mouse_x = canvas.get_mouse_x()
mouse_y = canvas.get_mouse_y()
```

- the clicks of the mouse

```
clicks = canvas.get_new_mouse_clicks()
```

Handle Mouse Events

- Ask to canvas for mouse location and handle it events
  - If there are, then we do something.
  - If there are not, we do nothing and check again later.

```
while True:
    # Handle any new mouse events
    # ...
    canvas.update()
```

Handle Mouse Movement

- Ask to canvas for mouse location and handle it
  - If there are, then we do something.
  - If there are not, we do nothing and check again later.

```
while True:
    # get mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # do_something
    DO_SOMETHING()

    # update canvas
    canvas.update()
```

Doodler

```
SQUARE_SIZE = 10
...

while True:
    # Get the mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # do_something
    # Create a black rectangle at this location
    rect = canvas.create_rectangle(mouse_x, mouse_y,
                                   mouse_x + SQUARE_SIZE,
                                   mouse_y + SQUARE_SIZE)
    canvas.set_color(rect, 'black')
    # update canvas
    canvas.update()
```

Handle Mouse Clicks

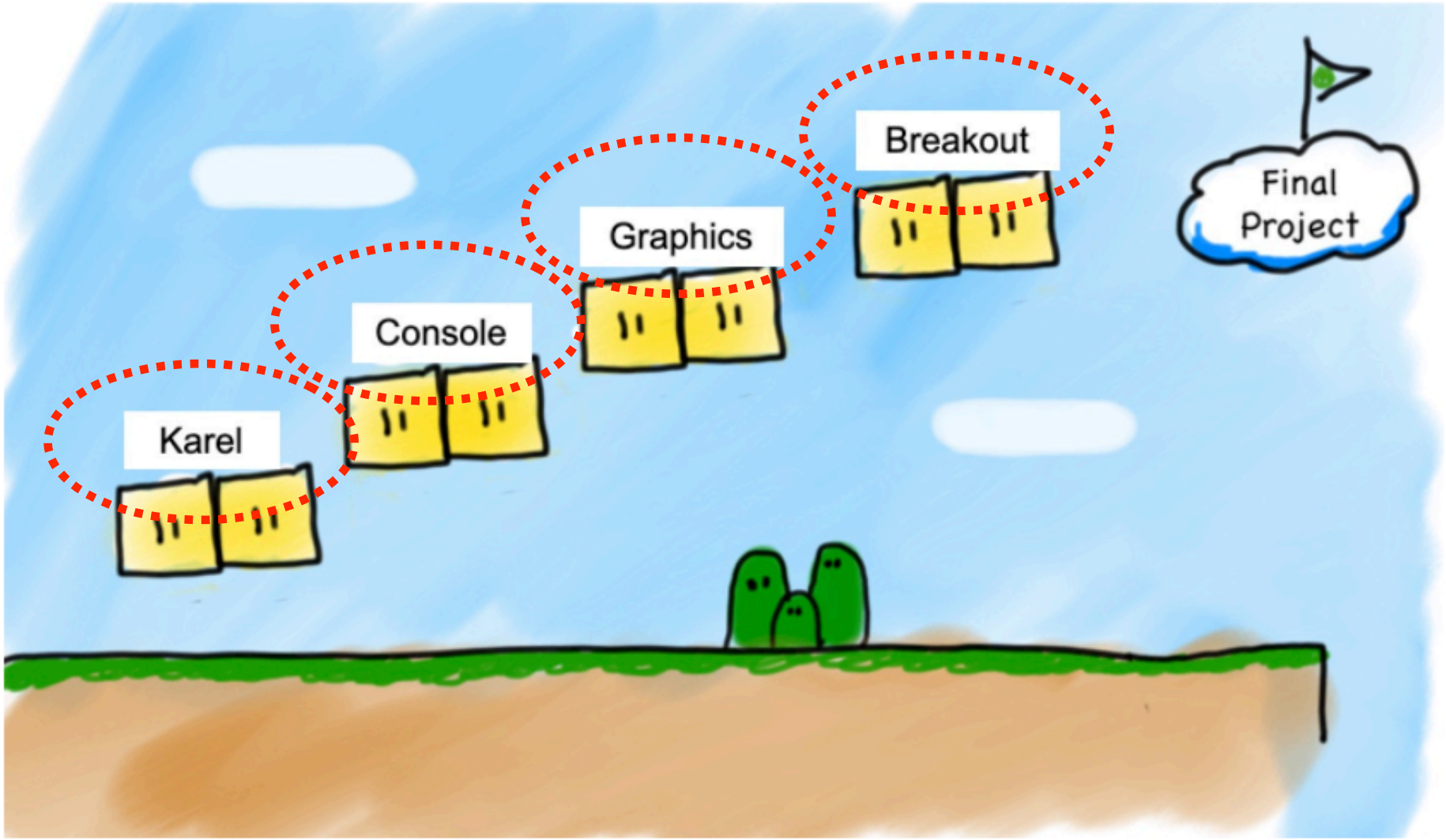
- Ask to canvas for mouse click and handle it
  - If there are, then we do something.
  - If there are not, we do nothing and check again later.

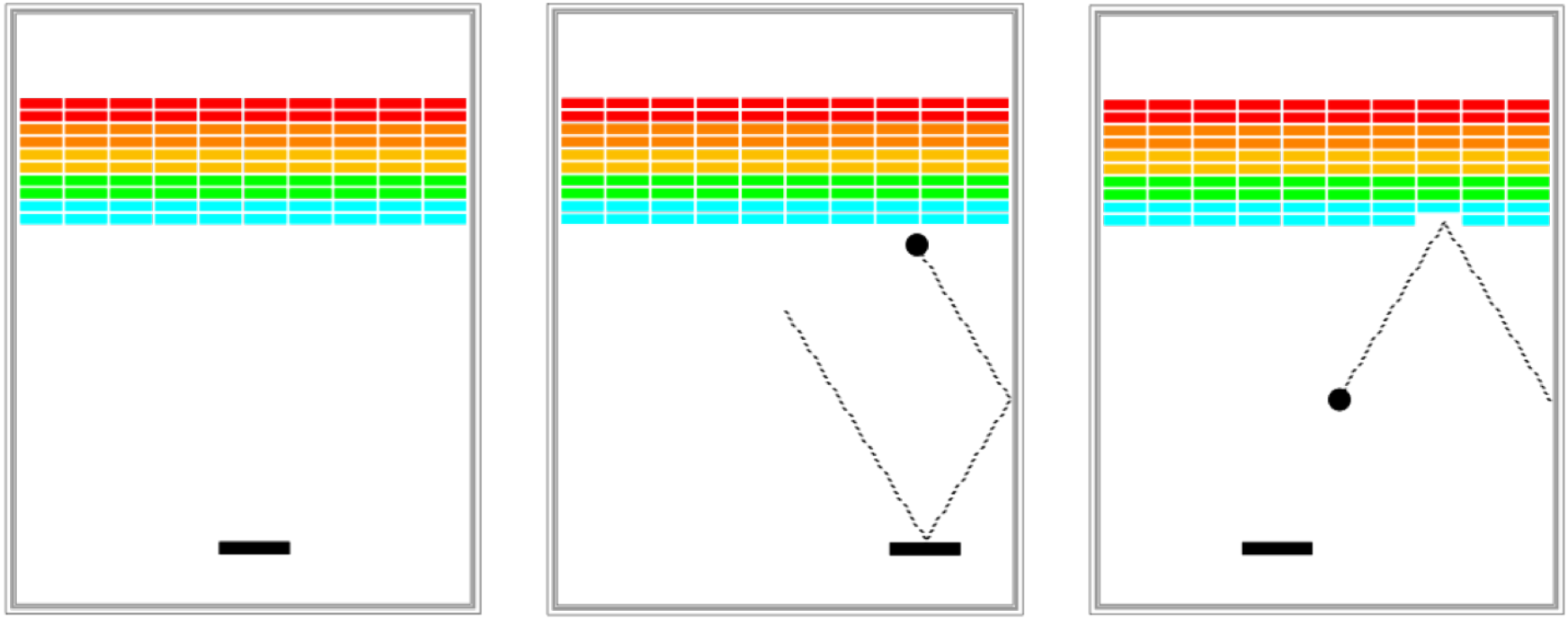
```
while True:
    clicks = canvas.get_new_mouse_clicks()

    # each time the user clicks do something
    for click in clicks:
        DO_SOMETHING()

    # update canvas
    canvas.update()
```

find_element_at

find_element_at returns the object at this location on the canvas.

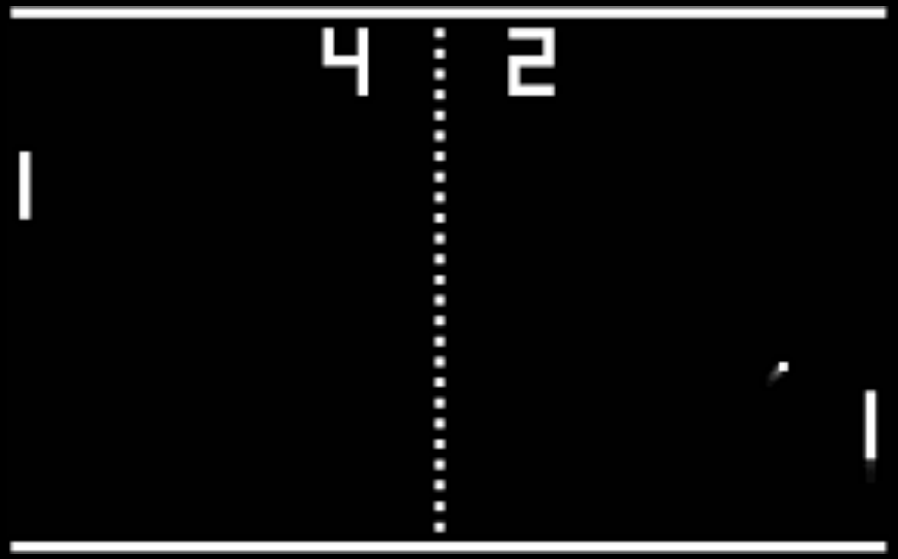
```
object_here = canvas.find_element_at(x, y)
if object_here:
      // do something with object_here
else:
      // nothing at that location
```

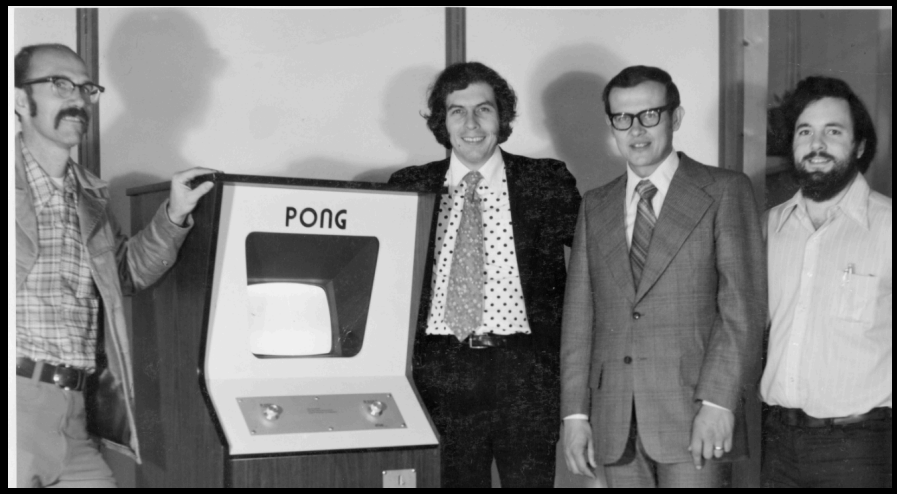

Karel
Console
Graphics
Breakout
Final Project

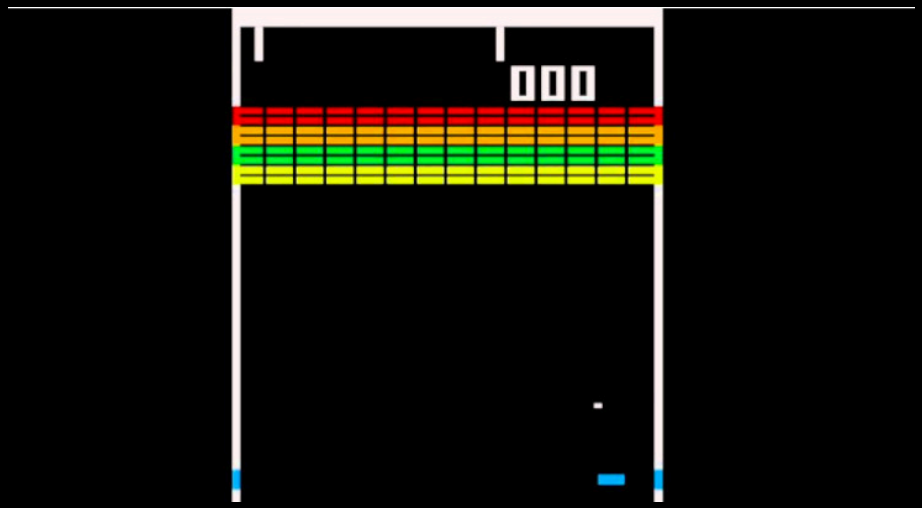


Breakout

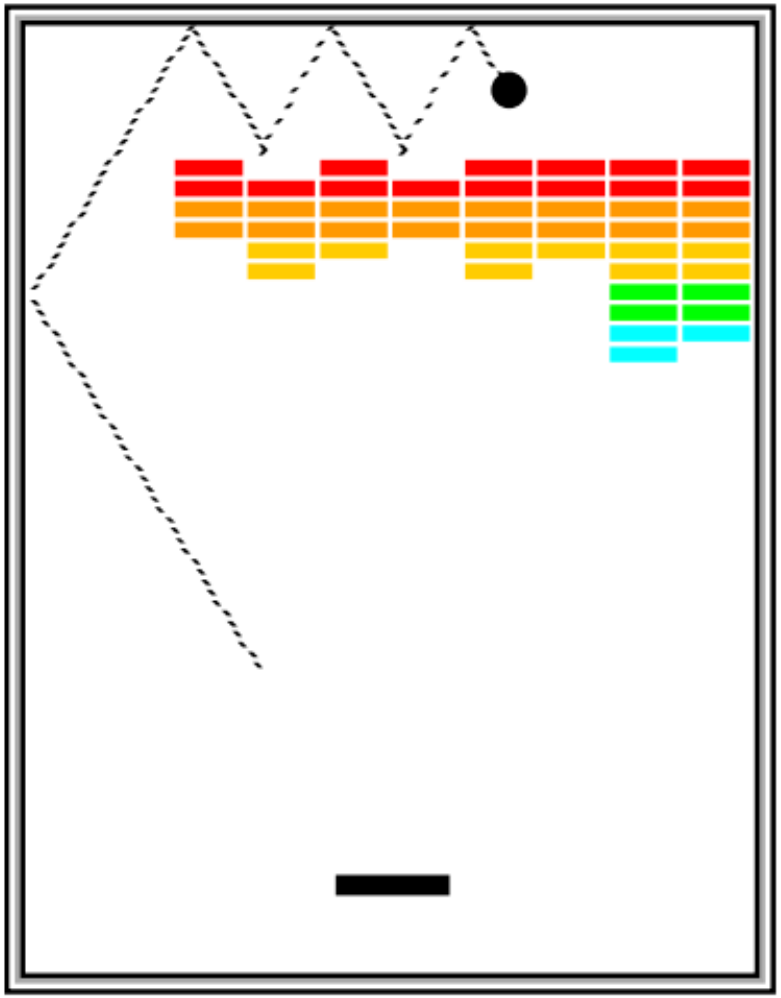
1972



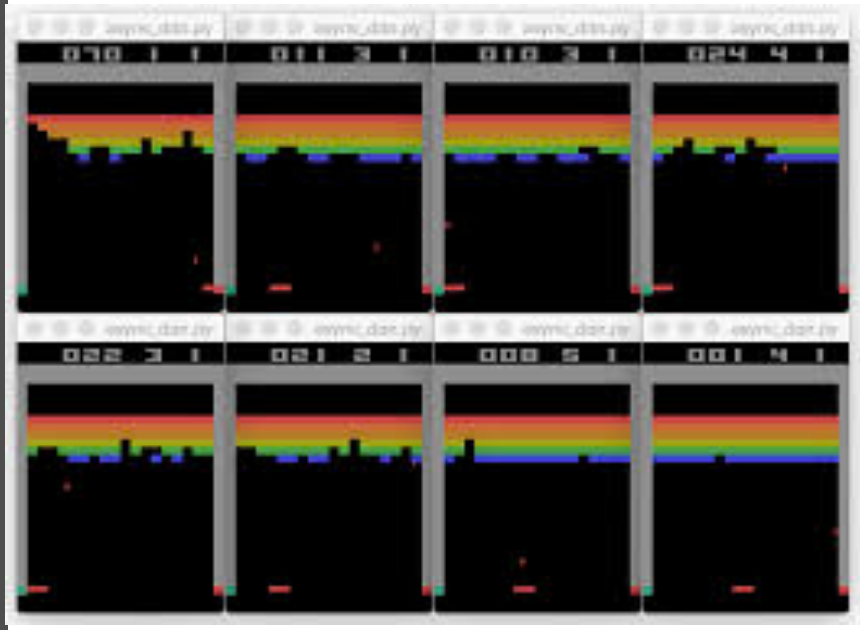




1976

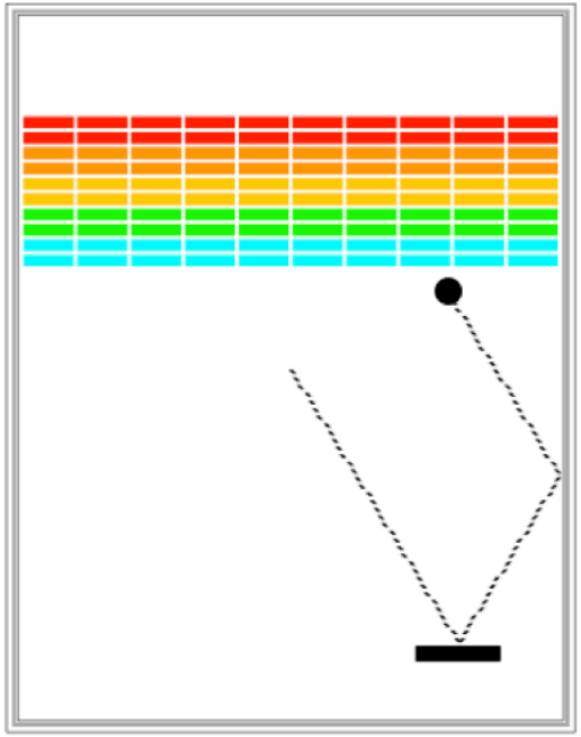



DQN
with tensorflow



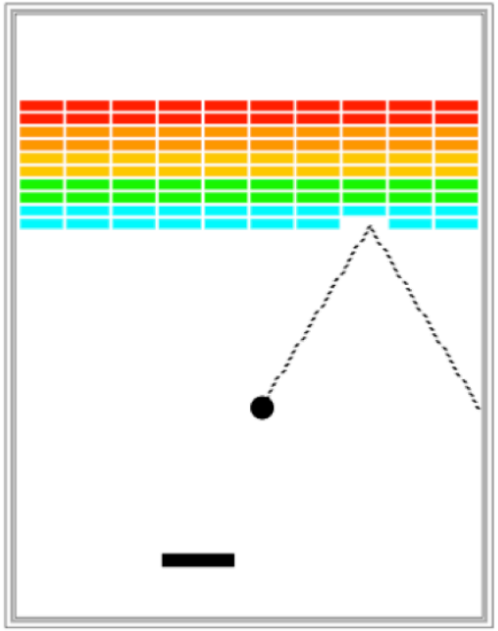
Requirements

- 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



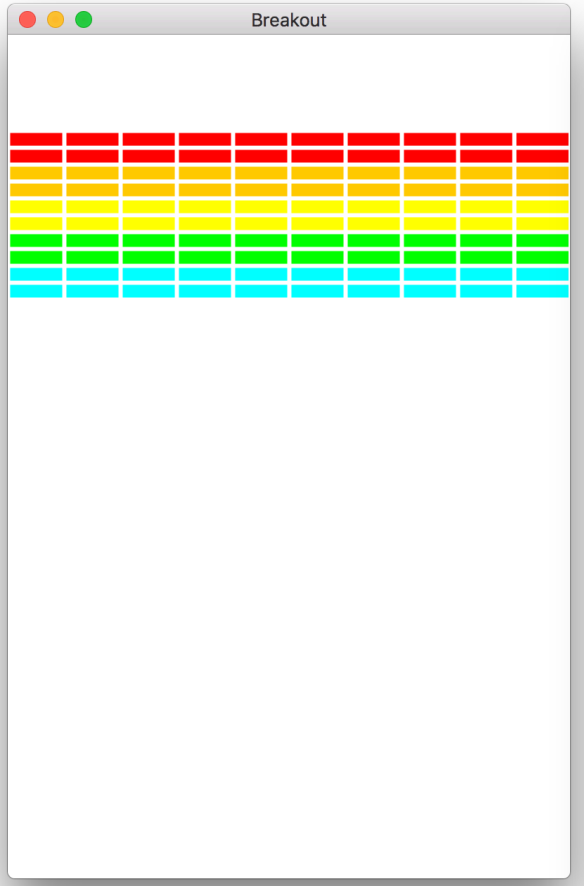
Requirements

- 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

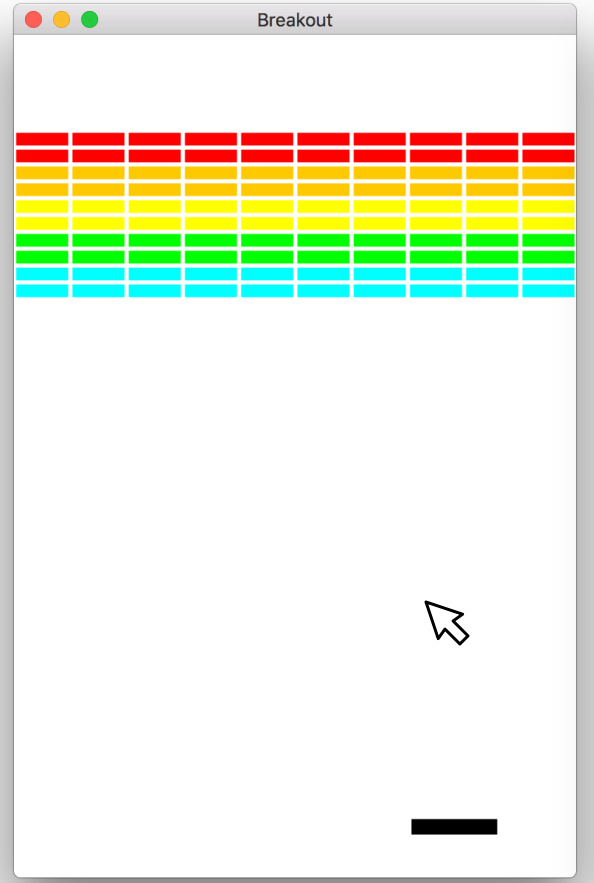


Big program. Do it in parts

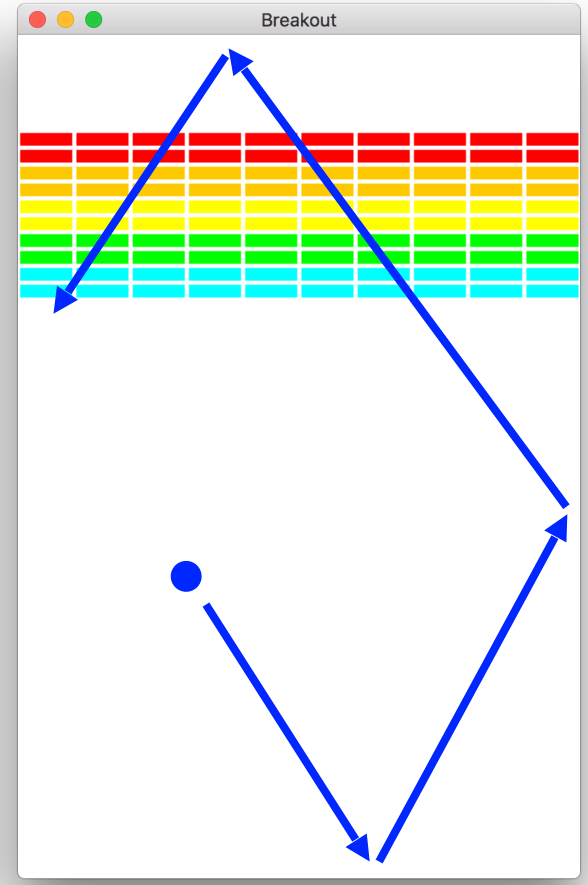
1



2

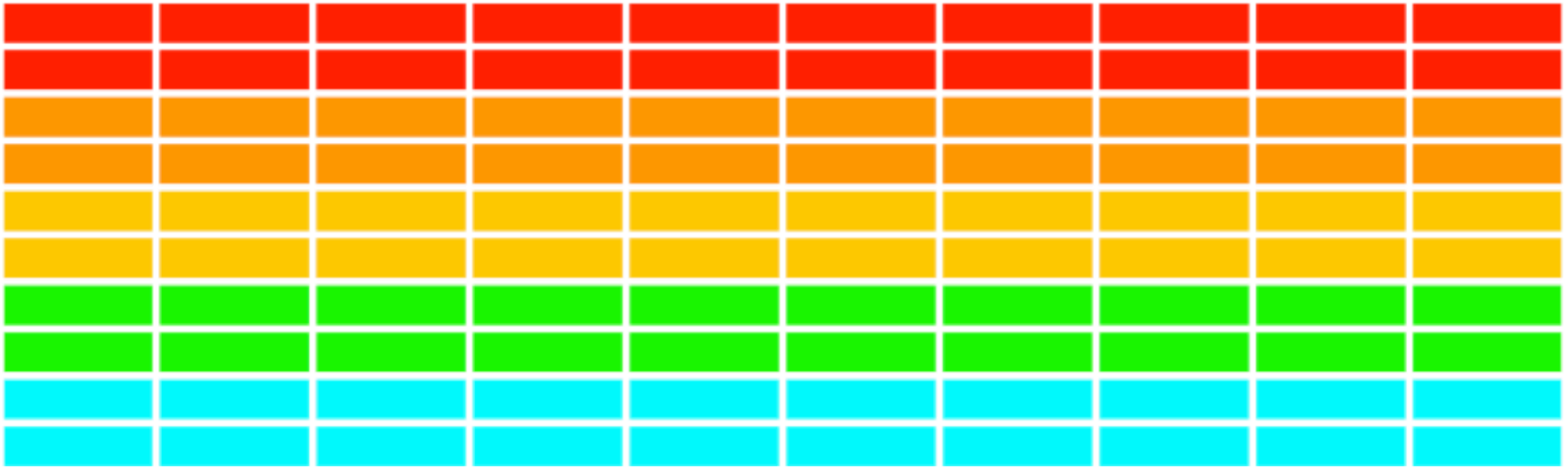


3

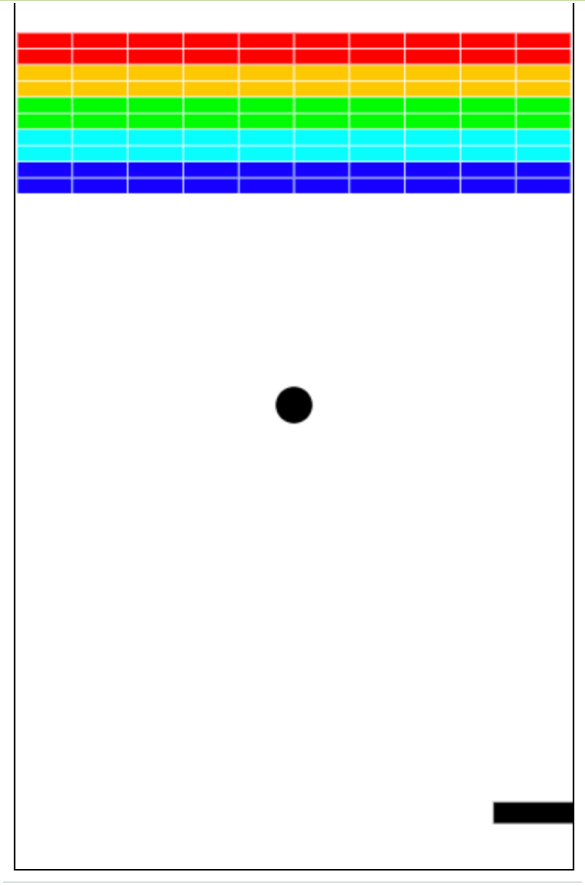


Part 1 - Creating Bricks

- Number, dimensions and spacing of bricks as constants
- Calculate x coordinate of the first column – so that bricks are centered
- Colors of bricks – red, orange, yellow, green and cyan



Part 2 - Add and move ball

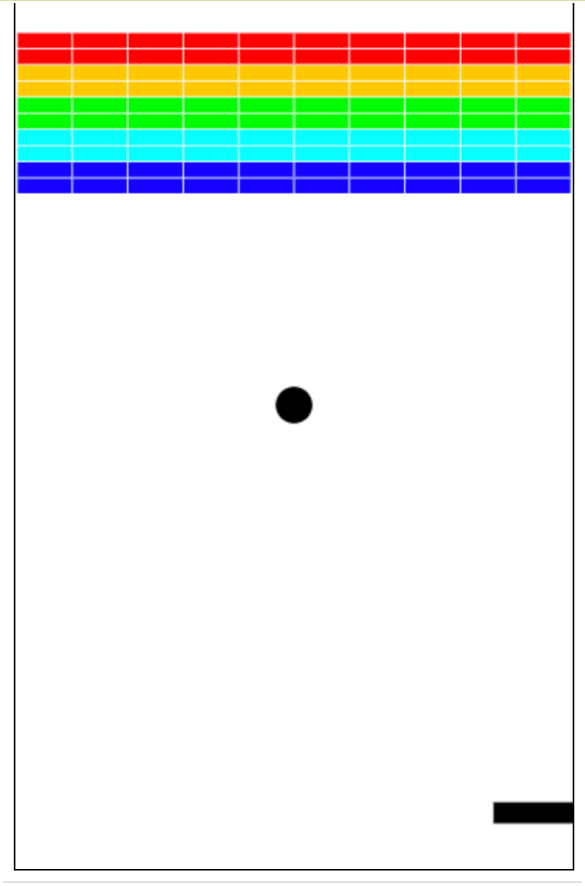


- Put filled ball at the center of the window
- Coordinates of the ball is the upper left corner
- Velocity of the ball – declared as variables

```
change_x = 10
change_y = 10
```

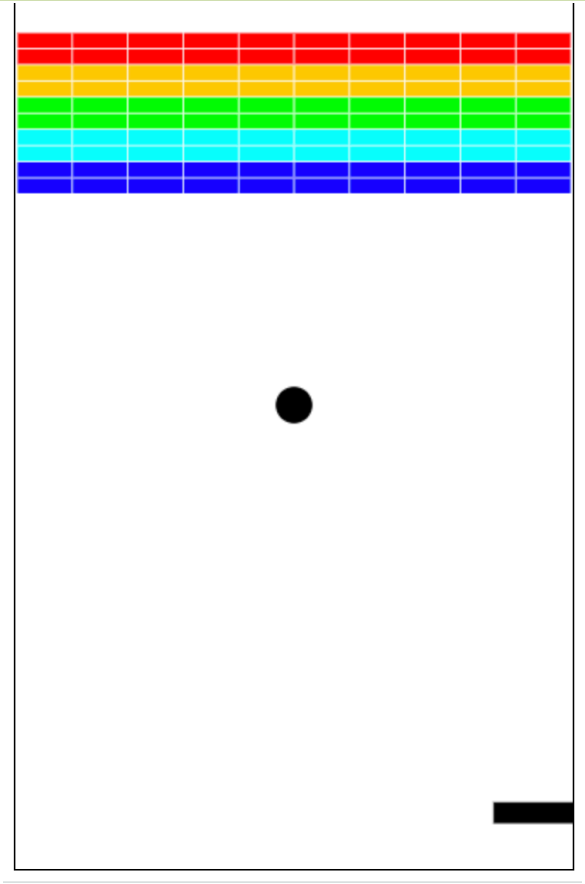
- Pick random values for change in x and y
- Move the ball using
  - **canvas.move(object, change_x, charge_y)**
  - **canvas.moveto(objects, new_x, new_y)**
- Move the ball – Initially – ball heading downwards

Part 2 - Bouncing ball



- Animation loop – where ball is moving
  - Bounce from right, left and top walls
  - Bottom wall – start in the middle again
- Update the canvas – where bricks are removed
- When bounced from top or bottom wall
  - Inverse **change_y**
- When bounced from left or right wall
  - Inverse **change_x**

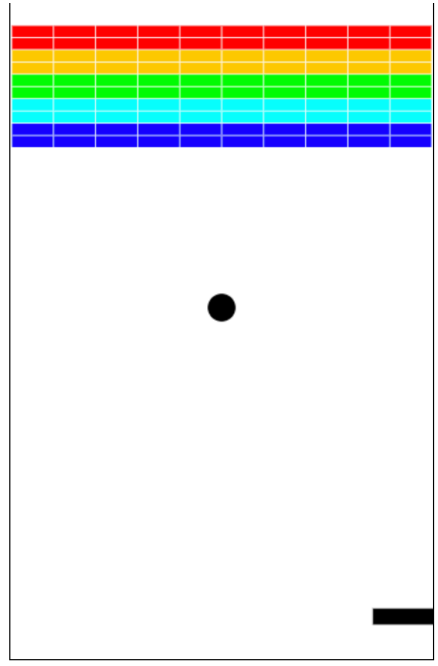
Part 3 - Add Paddle

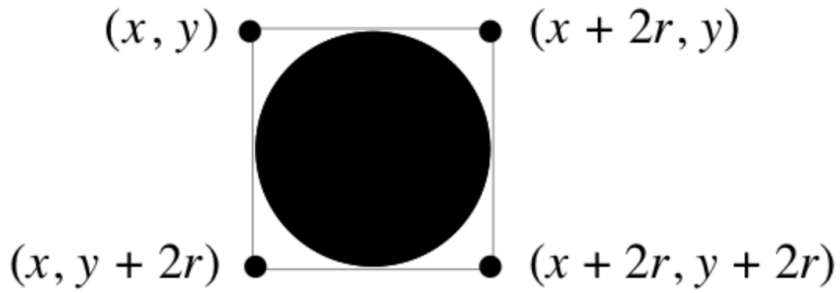


- Create Paddle – filled rectangle at a specific location
- Link paddle move with mouse move
  - Move paddle – track x coordinate of the mouse onl
  - Use **mouse_x = canvas.get_mouse_x()**

Part 4 - Check for collision

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





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

```
Flip change_y direction
```

  - 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!