YEAH Hours: Breakout

10/16/18

Ryan Eberhardt
(some images borrowed from Nick Troccoli)

Nested/"closure" Functions

```
function DrawDots() {
    let gw = GWindow(
        GWINDOW_WIDTH,
        GWINDOW_HEIGHT);

    gw.addEventListener(
        "click", clickAction);
}
```

```
function clickAction(e) {
    let dot = GOval(
        e.getX() - DOT_SIZE / 2,
        e.getY() - DOT_SIZE / 2,
        DOT_SIZE, DOT_SIZE);
    dot.setFilled(true);
    gw.add(dot);
};
```

This doesn't work, because within `clickAction`, gw is out of scope.

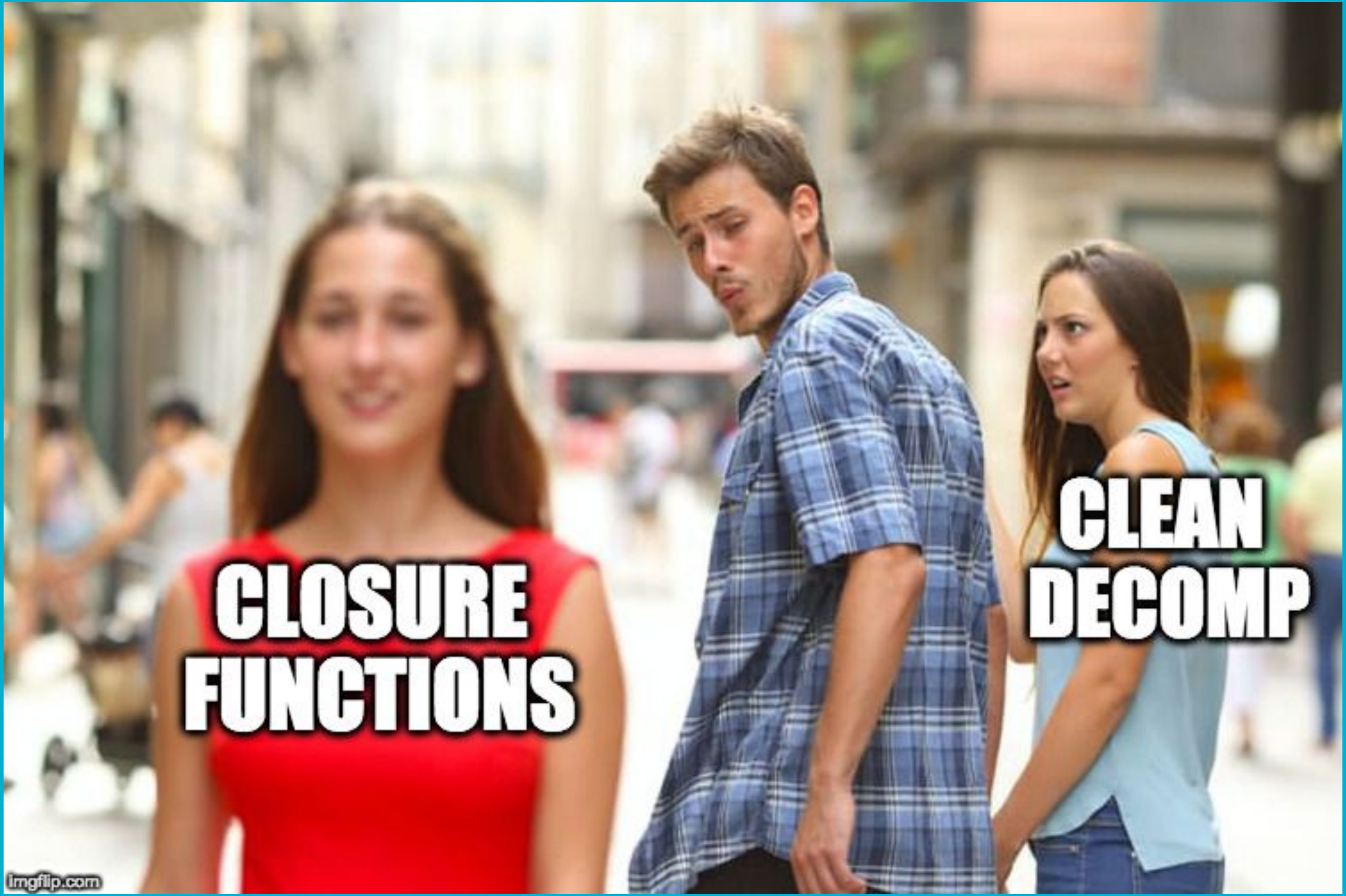
Nested/"closure" Functions

```
function DrawDots() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    function clickAction(e) {
        let dot = GOval(e.getX() - DOT_SIZE / 2,
                        e.getY() - DOT_SIZE / 2,
                        DOT_SIZE, DOT_SIZE);
        dot.setFilled(true);
        gw.add(dot);
    };
    gw.addEventListener("click", clickAction);
}
```

Nested functions inherit their parents' scope!

Inheriting Scope

```
function DrawLines() {
  let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
  let line = null;
  let mousedownAction = function(e) {
    line = GLine(e.getX(), e.getY(), e.getX(), e.getY());
    gw.add(line);
  };
  let dragAction = function(e) {
    line.setEndPoint(e.getX(), e.getY());
  };

  gw.addEventListener("mousedown", mousedownAction);
  gw.addEventListener("drag", dragAction);
}
```


CLOSURE FUNCTIONS
CLEAN DECOMP
imgflip.com

When is sharing variables okay?

- If you need to access the variables from within event listener or animation functions
- You access/change the variable all over the place
- There's just no other way

Handling User Interaction (Event Listeners)

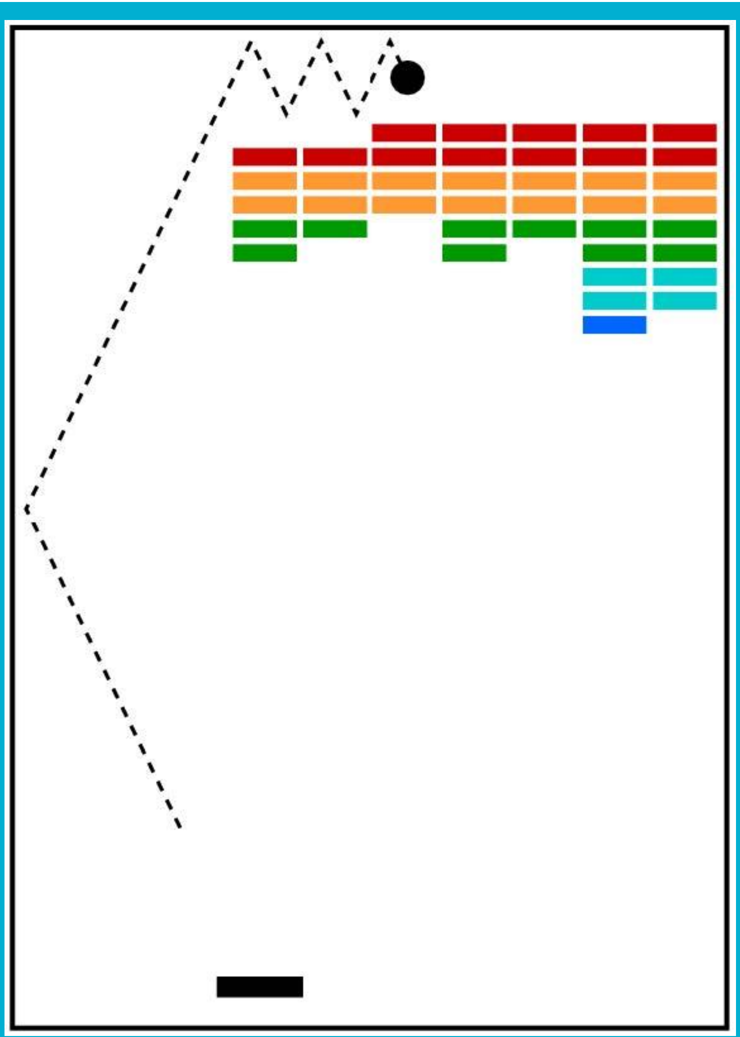
- click
- dblclk
- mousedown
- mouseup
- mousemove
- drag

Handling User Interaction (Event Listeners)

- click
- dblclk
- mousedown
- mouseup
- mousemove
- drag

Animation

```
function AnimatedSquare() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    let square = GRect(0, 0, SQUARE_SIZE, SQUARE_SIZE);
    square.setFilled(true);
    gw.add(square);
    let stepCount = 0;
    let step = function() {
        square.move(dx, dy);
        stepCount++;
        if (stepCount === N_STEPS) clearInterval(timer);
    };
    let timer = setInterval(step, TIME_STEP);
}
```



Breakout

Assignment info

- Due next Friday
- Working in pairs is allowed!
- One huge assignment! Pay attention to the milestones, and set a schedule for yourself

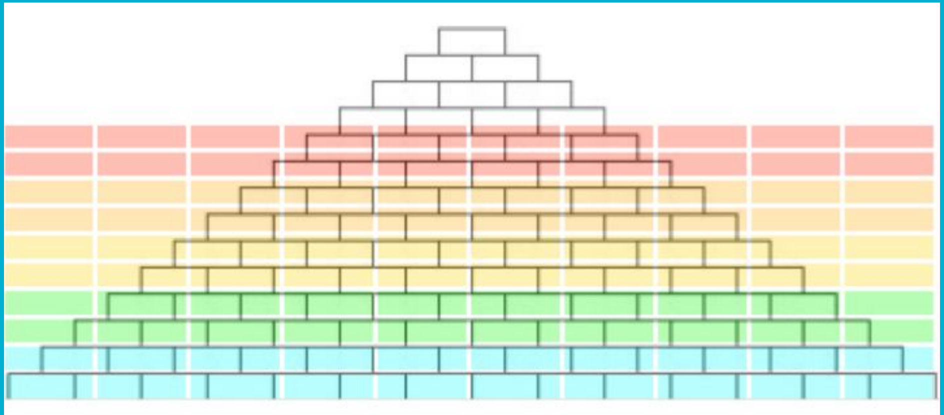
Constants

```
const GWINDOW_WIDTH = 360;        /* Width of the graphics window   */
const GWINDOW_HEIGHT = 600;       /* Height of the graphics window  */
const N_ROWS = 10;                /* Number of brick rows           */
const N_COLS = 10;                /* Number of brick columns        */
const BRICK_ASPECT_RATIO = 4 / 1; /* Width to height ratio of a brick  */
const BRICK_TO_BALL_RATIO = 3 / 2;   /* Ratio of brick width to ball size */
const BRICK_TO_PADDLE_RATIO = 2 / 3; /* Ratio of brick to paddle width   */
const BRICK_SEP = 2;              /* Separation between bricks      */
const TOP_FRACTION = 0.1;         /* Fraction of window above bricks   */
const BOTTOM_FRACTION = 0.05;     /* Fraction of window below paddle   */
const N_BALLS = 3;                /* Number of balls in a game      */
const TIME_STEP = 10;             /* Time step in milliseconds      */
const INITIAL_Y_VELOCITY = 3.0;   /* Starting y velocity downward   */
const MIN_X_VELOCITY = 1.0;       /* Minimum random x velocity      */
const MAX_X_VELOCITY = 3.0;       /* Maximum random x velocity      */
```

Constants

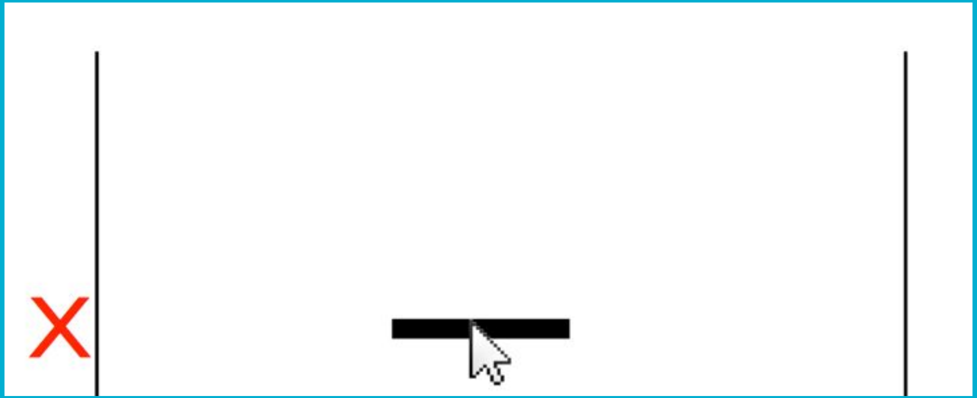
```
/* Derived constants */

const BRICK_WIDTH = (GWINDOW_WIDTH - (N_COLS + 1) * BRICK_SEP) / N_COLS;
const BRICK_HEIGHT = BRICK_WIDTH / BRICK_ASPECT_RATIO;
const PADDLE_WIDTH = BRICK_WIDTH / BRICK_TO_PADDLE_RATIO;
const PADDLE_HEIGHT = BRICK_HEIGHT / BRICK_TO_PADDLE_RATIO;
const PADDLE_Y = (1 - BOTTOM_FRACTION) * GWINDOW_HEIGHT - PADDLE_HEIGHT;
const BALL_SIZE = BRICK_WIDTH / BRICK_TO_BALL_RATIO;
```

Milestone 1: Set Up the Bricks



- Much like the pyramid problem from Assignment 2
- However, instead of placing bricks right next to each other, there should be BRICK_SEP spacing between each brick and each row
- In addition, you should color each pair of rows. (You might do so by writing a function to return a color given a row number, or a function to color a brick given a row number.)

Milestone 2: Create the Paddle



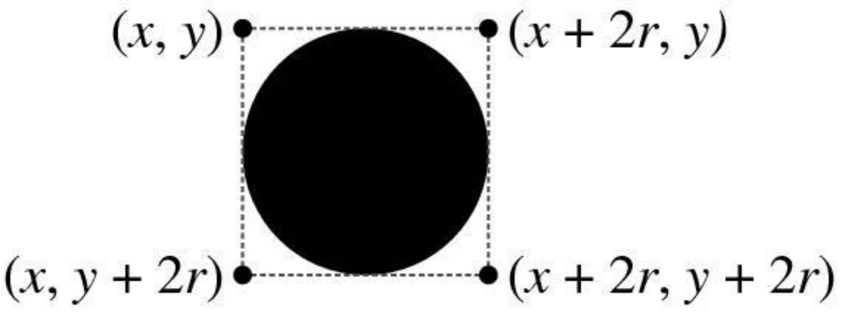
- The paddle is a simple filled GRect
- The middle of the paddle should stay anchored to the mouse: call paddle.setLocation (it's much easier than using paddle.move() here)
- The paddle should not be allowed to move off the screen, even when the mouse moves to the edges of the screen

Milestone 3: Create the Ball

Milestone 3: Create the Ball

- Draw the ball in the center of the screen
- Wait for the user to click the screen (set up a "click" event listener)
- Animate the ball moving
  - Choose vx and vy (see assignment handout)
    - vy = INITIAL_Y_VELOCITY;
    - vx = randomReal(MIN_X_VELOCITY, MAX_X_VELOCITY);
    - if (randomChance()) vx = -vx;
  - Call an animation function every TIME_STEP milliseconds
  - In the animation function, move the ball by vx and vy
- Check for collisions with walls
  - Check if the coordinates of the ball exceed the dimensions of the GWindow, and if so, set vx = -vx or vy = -vy (depending on which wall was hit)

Milestone 4: Checking for Collisions (with bricks)



- `gw.getElementAt(x, y)` will return the object at a particular point (or `null` if there is no object there)
- However, the ball occupies more than a single pixel
- You should write a function `getCollidingObject(gw, ball)` that returns the object that the ball is colliding with, by checking the 4 "corners" of the ball (or `null` if the ball isn't colliding with anything)
  - This function should be pretty simple (somewhere around 8 lines long)

Milestone 4: Checking for Collisions (with bricks)

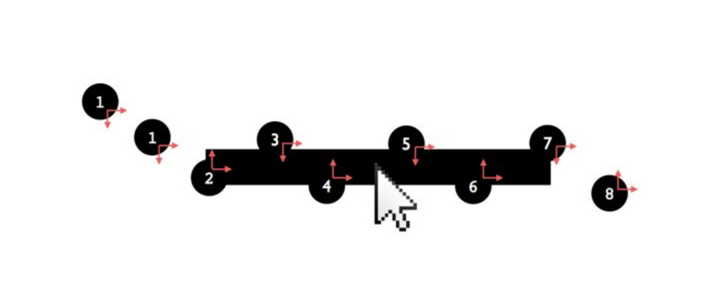
- In your animation function, on each step, call your getCollidingObject function to check whether the ball is colliding with anything
  - If colliding with the paddle or a brick, vy = -vy
  - If colliding with a brick, remove the brick from the screen
- How can you tell if you've collided with the paddle or the brick??

Milestone 4: Checking for Collisions (with bricks)

- In your animation function, on each step, call your getCollidingObject function to check whether the ball is colliding with anything
  - If colliding with the paddle or a brick, vy = -vy
  - If colliding with a brick, remove the brick from the screen
- How can you tell if you've collided with the paddle or the brick??
  - When you create the paddle, keep your `paddle` variable around
  - When checking for collisions, check `if (collidingObject === paddle)` (and if not, then it must be a brick, because there are no other objects drawn in the window)

Milestone 4: Checking for Collisions (with bricks)

- You will likely experience a "sticky paddle" bug:



How might you fix this? (Find a way to make sure that vy is negative after colliding with the paddle, so that the ball is forced to go up!)

Milestone 5

- When the ball hits the bottom of the screen, you need to stop the animation, reset the ball, and wait for the user to click to start the next turn
- The user should have 3 "lives"
- Stop the animation when the user is out of lives, or when all the bricks are gone
- Test, test, test!

Debugging