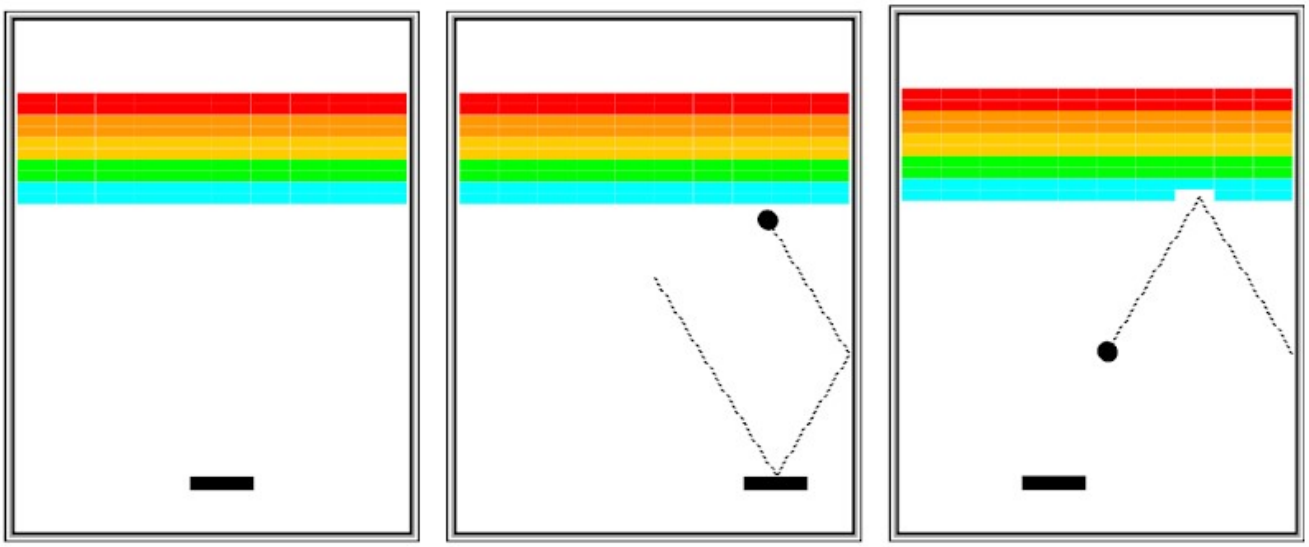
Let's play Breakout!



We have 3 files in this project:

- *game.py* - this is the file that runs the game. You will do all of your work in this file.
- *breakout.py* - this file has a lot of functions that will help you with your game. You do not need to change this file for your game to work.
- *constants.py* - this is where you will find all of the settings for our game. Feel free to change them once your game is working.

Download these files at: http://bit.ly/breakoutcode

Important Tips

- Implement the program in stages, as described in this handout.
- Don't try to extend the program until you get every step working.

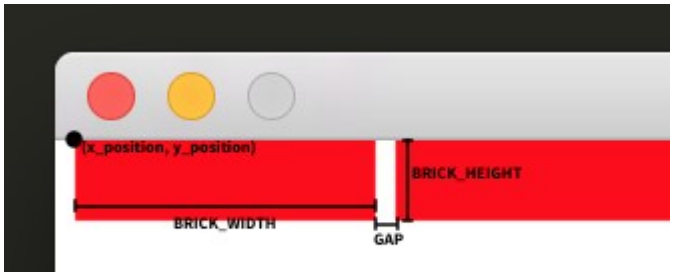
Step #1: Setup the Bricks

Function: build_bricks()

In this step, we will draw `constants.NUM_ROWS` rows of bricks, with `constants.BRICKS_PER_ROW` bricks in each row.

1. First, find where your `build_bricks` function is defined and delete "`pass`". This function should **return** a list named `bricks` that holds every brick.
    a. Create one row of `constants.BRICKS_PER_ROW` bricks using a **for loop**.
        i. Figure out the `x_position` and `y_position` of each brick in the row. Leave `constants.GAP` space between each brick. Note that `(x_position, y_position)` is the **top left** corner of the rectangle. For example, if the **first brick** is at position `(constants.GAP, 0)` then the

second brick will be at x position: 0 + constants.BRICK_WIDTH + 2 * constants.GAP. So if the width is 10 and the gap is 2, then the **second brick** will be at position (14, 0). Note that the y position is still 0 for the second brick since we want every brick in the row to be at the same height.

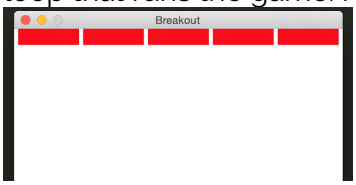


ii. To create one brick, use `breakout.create_new_brick`, and then set its x-position, y-position, and color. At the end, add the brick to the `bricks` list. This allows the brick to get drawn in `draw_objects`.

```
brick = breakout.create_new_brick()
breakout.set_x(brick, x_position)
breakout.set_y(brick, y_position)
breakout.set_color(brick, color)
bricks.append(brick)
```

b. Of course, at the end of the function, you have to **`return bricks`**, the new list you just created.

2. Now, find where the `draw_objects()` function is defined and delete "pass". Eventually, this function will draw everything on the screen. For now, `bricks` is a list of all bricks. Draw every brick in `bricks` using a **for** loop.
   a. Use `breakout.draw_rectangle(x, y, width, height, color)` to draw each brick. Remember, you get `x` by calling `breakout.get_x(brick)` and the same for `y, width, height` and `color`
3. Now that you've changed your `build_bricks` function, you need to call it. Do so before the main `while` loop that runs the game. Remember that build_bricks has a return value - so use it!




```
bricks = build_bricks()
```

   b. When you finish this, there should be a single row of bricks on screen, like the picture on the right.
4. Change your code in `build_bricks` to add `constants.NUM_ROWS` rows of bricks to `bricks`. Hint: You'll need a second **for loop**.
   a. It will help to draw out a diagram on paper to figure out the `x_position` and `y_position` of each brick.

Step #2: Setup the Paddle

Function: paddle_update_position()

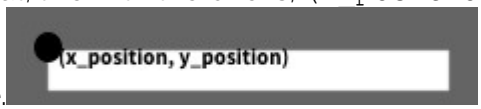
In this step, we will create the paddle and make it follow the mouse.

1. First, create the paddle using the function `breakout.create_new_paddle()`. Do this before the main `while` loop that runs the game.
2. Now, in the `draw_objects` function, draw your paddle.
   a. Use the `breakout.draw_rectangle(x, y, width, height, color)` function. Use x, y, width, height and color of the paddle by calling the `breakout.get_x(paddle)`, `breakout.get_y(paddle)`, and so on.

3. Find where `paddle_update_position(paddle)` is defined and delete "pass". The goal of this function is to make the paddle move with the mouse.
    a. Get the x-position of the mouse using `breakout.get_mouse_location()`.

```
location = breakout.get_mouse_location()
x_position = location[0]
```

    Note that the paddle always stays at the bottom of the screen, so we don't need to change its y-position, just the x-position.
    b. Use `breakout.set_x(paddle, x_position)` to set the x-position of the paddle.
    c. But, notice that when you move your mouse off the screen, towards the right, the paddle moves off the screen too. You should fix this in your code. If the mouse is off the screen, the paddle stays where it is on the edge of the screen instead of moving off. Remember that, like with the bricks, `(x position, y_position)` is the **top left** of the paddle.



4. Now that you've defined the `paddle_update_position` function, you need to figure out where to call it inside the `while` loop.

At this point, you should have the bricks and the paddle on the screen, with the paddle moving with your mouse.

Step #3: Ball & Bounce

Function: ball_update_position

1. First, create the ball using the function `breakout.create_new_ball()`. Do this before the main `while` loop that runs the game.
2. Now, in the `draw_objects` function, draw your ball.
    a. Use the `breakout.draw_circle(x,y,radius, color)` function. You can get the x, y, radius and color of the ball by calling `breakout.get_x(ball)`, `breakout.get_y(ball)` and so on.
3. Find where `ball_update_position(ball)` is defined and delete "pass". The goal of this function is to make the paddle move with the mouse. This function looks a lot like what you did in the last game.
    a. Get the current `x`, `y`, `x_velocity`, and `y_velocity` of the ball using `breakout.get_x`, `breakout.get_x_velocity`...
    b. Get the new x and y for the ball by adding `x_velocity` and `y_velocity` to x and y. (`x_velocity` and `y_velocity` are like the `x_direction` and `y_direction` we had in the last game.)
    c. Save the new x and y for the ball using `breakout.set_x` and `breakout.set_y`.
    d. Like last time, the ball needs to bounce off the walls. For now, ignore the bricks and paddle. When the ball hits a wall, you should either change `x_velocity` or `y_velocity` so that the ball bounces back. Use `breakout.set_x_velocity` and `breakout.set_y_velocity` to save these new numbers.
4. Now that you've defined the `ball_update_position` function, you need to figure out where to call it inside the `while` loop. You only want to move the ball if start is True, which would mean that the game has started (and that the user has clicked the mouse). Find "`if start == True`" in the while loop, delete "`pass`", and call `ball_update_position` there.

Step #4: Check if the paddle and ball collide

Here, we will make the ball bounce off the paddle.

1. Go to the definition of the `ball_bounce_off` function and delete "`pass`". The goal of this function is to switch the y-velocity of the ball if it hits the object. We will use it when the ball hits the paddle or when it hits a brick.
    a. If this function has been called, we know that the ball has hit something. If the ball is moving downwards, we need to make it move upwards. If it's moving upwards, we need to make it move downwards.
    b. Use `breakout.get_y_velocity` and `breakout.set_y_velocity` to make the y-velocity negative, and save the new y-velocity.
        i. Note: You may find that this results in some weird behaviour: the ball can get "trapped" inside the paddle, This is because the paddle keeps telling the ball to change direction, so it starts going down, hits paddle, goes up, hits paddle again, goes down etc. As an extension, think about a way to fix this.
2. In the main while loop, first check whether the ball hit the paddle. Use `breakout.ball_did_collide_with(ball, obj, obj_width, obj_height)` to check whether the ball hit `obj` (where `obj` has width `obj_width` and height `obj_height`). This function will return `True` if the ball hit `obj`, and `False` if the ball did not.
    a. If the ball hit the paddle, make the ball bounce. Use `ball_bounce_off` to do this.

Step #5: Check if the ball and a brick collide

Now we will handle the case where a ball hits a brick. The brick should disappear, and the ball should bounce off.

1. In your main while loop, we need to use a **for** loop to look through all of the bricks, and check one-by-one whether the ball hit each brick.
    a. For each brick, to check whether the ball hit it, call the `ball_did_collide_with` function we used earlier. But this time, instead of checking whether the ball hit the paddle, we're checking whether the ball hit a brick. Make sure you use the brick's width and height (and not the paddle's width and height).
    b. If this is true, make the ball bounce using `ball_bounce_off`.
    c. Finally, if this is true, remove the brick from the `bricks` list using `bricks.remove(brick)`. For example:

```
for i in range(NUM_BRICKS):
        brick = bricks[i]
        if ____________________:
                ____________________
                bricks.remove(brick)
```

Step #6: Winning and losing

The last thing to do is handle winning and losing.

1. We win the game when we've hit all the bricks. Notice that if that's true, the `bricks` list will be empty!
    a. In the **while** loop, check whether there are any bricks left by checking whether the length of `bricks` is 0:

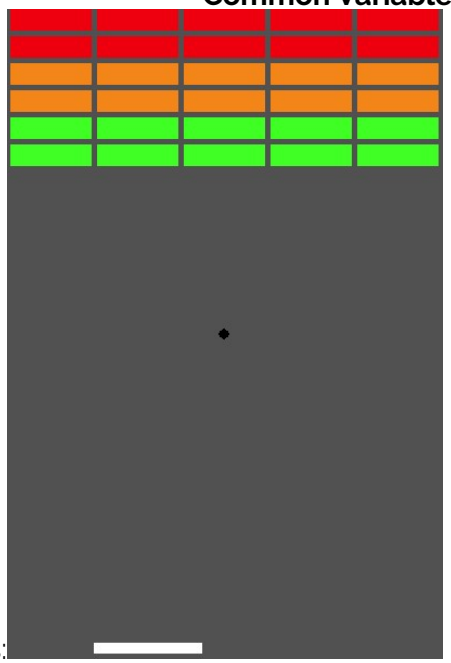
```
if len(bricks) == 0:
```

    Think about where it makes sense to do this - before you check if the ball has hit any other bricks? After?
    b. If so, we should make the **while** loop end. Hint: Notice that the loop continues as long as `running` is `True`. If `running` is not `True`, the loop will end.

2. We lose the game when the ball moves past the paddle and hits the bottom wall.
   a. Right now in your `ball_update_position`, the ball just bounces off the bottom wall. Instead, the ball should disappear (so, not change direction). You should change this.
   b. Next, go to your main while loop. Find the part that says if the ball hit the bottom wall, you lose.

Check if the ball hit the bottom wall. If so, make the **while** loop end by changing `running`, like you did for winning.

i. **Common Variables**



- **Brick** contains:
  - x
  - y
  - width
  - height
  - color
- **Ball** contains:
  - x
  - y
  - radius
  - x_velocity
  - y_velocity
  - color
- **Paddle** contains:
  - x
  - y
  - width
  - height
  - x_velocity
  - color

```
constants.BRICKS_PER_ROW
constants.NUM_ROWS
constants.GAP
constants.BRICK_WIDTH
```

```
constants.BRICK_HEIGHT
constants.BRICK_COLOR
constants.PADDLE_COLOR
constants.PADDLE_WIDTH
constants.PADDLE_HEIGHT
constants.PADDLE_COLOR
constants.BALL_RADIUS
constants.BALL_COLOR
constants.SCREEN_WIDTH
constants.SCREEN_HEIGHT
constants.SCREEN_COLOR
constants.NUM_LIVES
constants.WHITE
constants.YELLOW
constants.GREEN
constants.ORANGE
constants.RED
constants.BLACK
```

Common Methods

```
breakout.create_new_ball(): return Ball
breakout.create_new_paddle(): return Paddle
breakout.create_new_brick(): return Brick
breakout.get_x(obj)
breakout.get_y(obj)
breakout.get_velocity_x(obj)
breakout.get_velocity_y(obj)
breakout.get_radius(obj)
breakout.get_width(obj)
breakout.get_height(obj)
breakout.get_color(obj)
breakout.set_x(obj, x)
breakout.set_y(obj, y)
breakout.set_velocity_x(obj, velocity_x)
breakout.set_velocity_y(obj, velocity_y)
breakout.set_radius(obj, radius)
breakout.set_width(obj, width)
breakout.set_height(obj, height)
breakout.set_color(obj, color)
breakout.clamp(n, min_n, max_n)
breakout.draw_rectangle(obj)
breakout.draw_circle(obj)
breakout.get_mouse_location()
breakout.build_screen(width, height)
breakout.clear_screen()
breakout.ball_did_collide_with(ball, obj, width, height)
```

Possible Extensions

- Make the bricks different colors based on the row. [EASY]
- Add multiple lives to the game, so that you have 3 (or more!) chances to win. [EASY]
- Make the game harder as you go along. For example, make the speed faster after a certain number of bricks have been hit. [EASY]
- Fix the "ball-in-paddle" problem. See the note in Step 4, part 1.b.i [EASY]
- Keep score. You could easily keep score, with points for each brick (and perhaps more points for bricks that are higher up). [EASY]
- Print messages to the screen. [MEDIUM]
    - For example, tell the user to click the mouse at the beginning of the game, or announce whether the user has won or lost.
    - To do this, you need to make a `draw_text` function in palgame.py that draws the text using the `pygame.blit` function. This link may be helpful: http://stackoverflow.com/questions/10077644/python-display-text-w-font-color
- Use your imagination. What else have you always wanted a game like this to do?