Checking qualifications

Demonstrate your knowledge of:

handling keystrokes using Pygame tools!

Checking qualifications

Which command allows you to get all the events happening in the game at the current moment?

Which part of the program is it appropriate to use this command in?

Checking qualifications

Handling keyboard events

Current events should be analyzed in the game loop.

Command	*Purpose*
`pygame.event.get()`	A set of events that occur during a given frame of the loop.

```
while game_over != True:
    for event in pygame.event.get():
        Action
```

"For each event from the set of current events, perform an action."

Checking qualifications

What is an event type ?

What is an name of the event ?

Give examples of event types and names.

Checking qualifications

Handling keyboard events

The type and name of the event are properties of an "Event" object.

Command	*Purpose*
`pygame.event.get()`	A set of events that occur during a given frame of the loop
`event.type / event.key`	Event type/Event name (already described in the Pygame library)

```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_w:
            Action
```

"If there is an event with a key pressed down among the current events and this key is W, then perform the action."

Checking qualifications

Let the platform sprite be set as an instance of the Picture class.

How can we move the platform 3 pixels to the right when pressing the "Right Arrow" key?

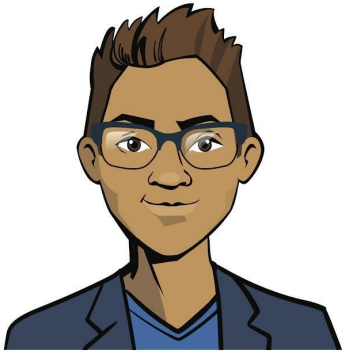
Checking qualifications

Movement of a sprite

```
platform = Picture('platform.png', platform_x, platform_y, 100, 30)
```

```
while not game_over:
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
                platform.rect.x +=3
```

"Until the game is over, if among the current events there is an event with a key pressed down and this key is the "Right Arrow" key, then move the platform 3 pixels to the right."



Checking qualifications

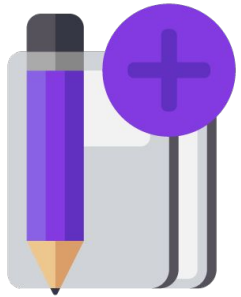
Qualifications confirmed!

Great, you are ready to brainstorm and complete your work task!

Checking qualifications

Brainstorm:

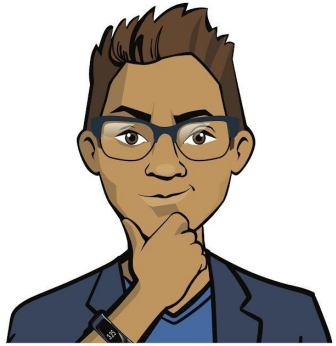
Controlling the platform with the keyboard



Handling keyboard events

An instance of the Picture class, the platform is controlled using the keyboard. The buttons are not defined in the terms of reference; we will choose them ourselves:

1. **When the** Left Arrow key is pressed **once**, the platform moves 3 pixels to the left.
2. **When the** Right Arrow key is pressed **once**, the platform moves 3 pixels to the right.



Brain storm

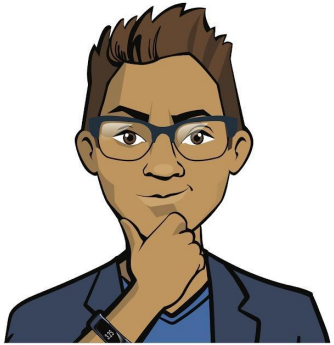
Handling keyboard events

An instance of the Picture class, the platform is controlled using the keyboard. The buttons are not defined in the terms of reference; we will choose them ourselves:

1. **When the** Left Arrow key is pressed **once**, the platform moves 3 pixels to the left.
2. **When the** Right Arrow key is pressed **once**, the platform moves 3 pixels to the right.

*How should the platform behave when the control key is **pressed**?*

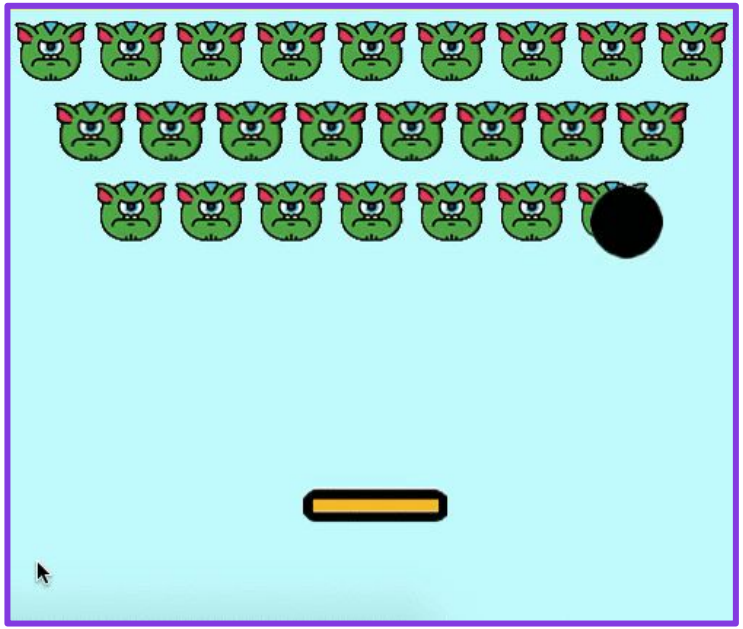
Should it keep moving?



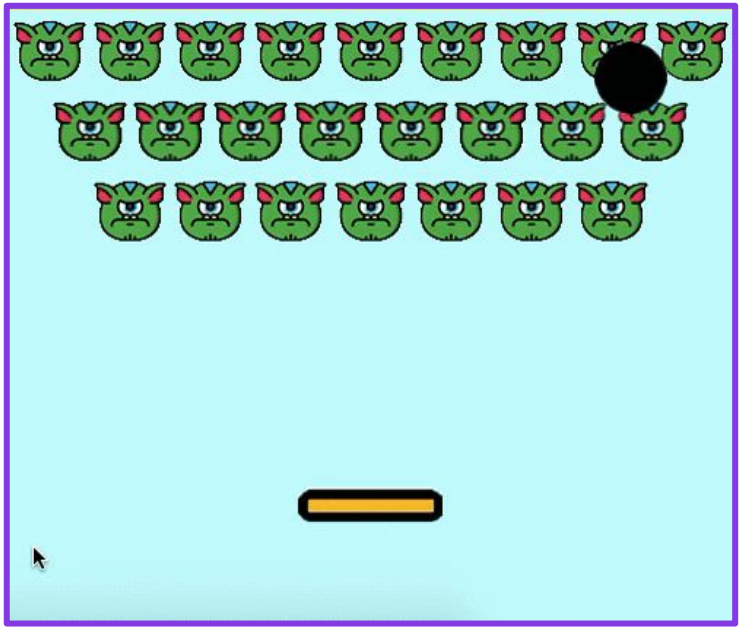
Brain storm

Compare the work of the programs:

Where is the work of the platform implemented more naturally for the user?



Pressing the key is not provided for; multiple keystrokes are required.



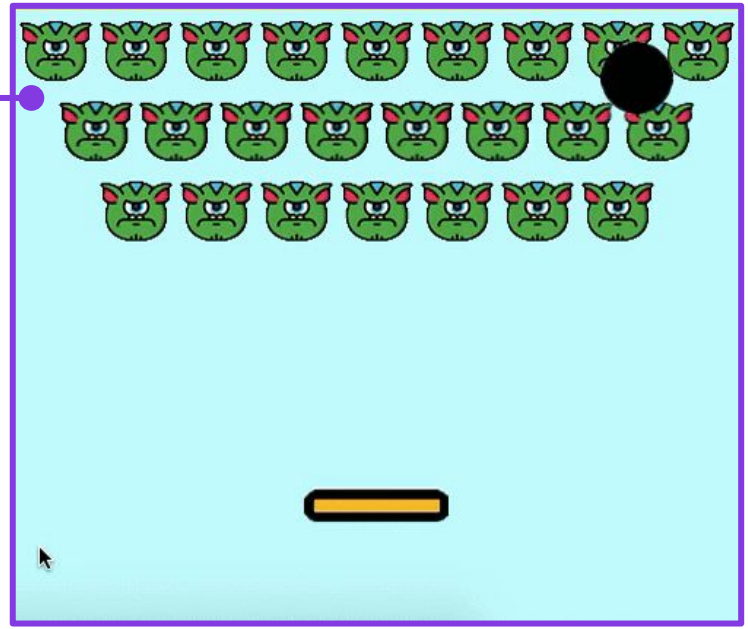
When the key is pressed, the platform continues to move.

Brain storm

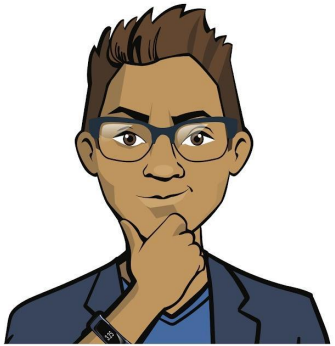
Compare the work of the programs:

Where is the work of the platform implemented more naturally for the user?

This option is more understandable to the players, but requires additional development.



When the key is pressed, the platform continues to move.



Brain storm

Handle keystrokes

Let the key that's pressed also determine the ***direction of movement***.

Introduce the move_right and move_left flag variables.

Flag	*Under what conditions is it True?*	*Under what conditions is it False?*
`move_right` — if True, then move to the right	The "Right Arrow" key is pressed	The "Right Arrow" key is raised (not pressed)
move_left — if True, then move to the left	The "Left Arrow" key is pressed	The "Left Arrow" key is raised (not pressed)

Brain storm

Handle keystrokes

Let the key that's pressed also determine the ***direction of movement***.

Introduce the move_right and move_left flag variables.

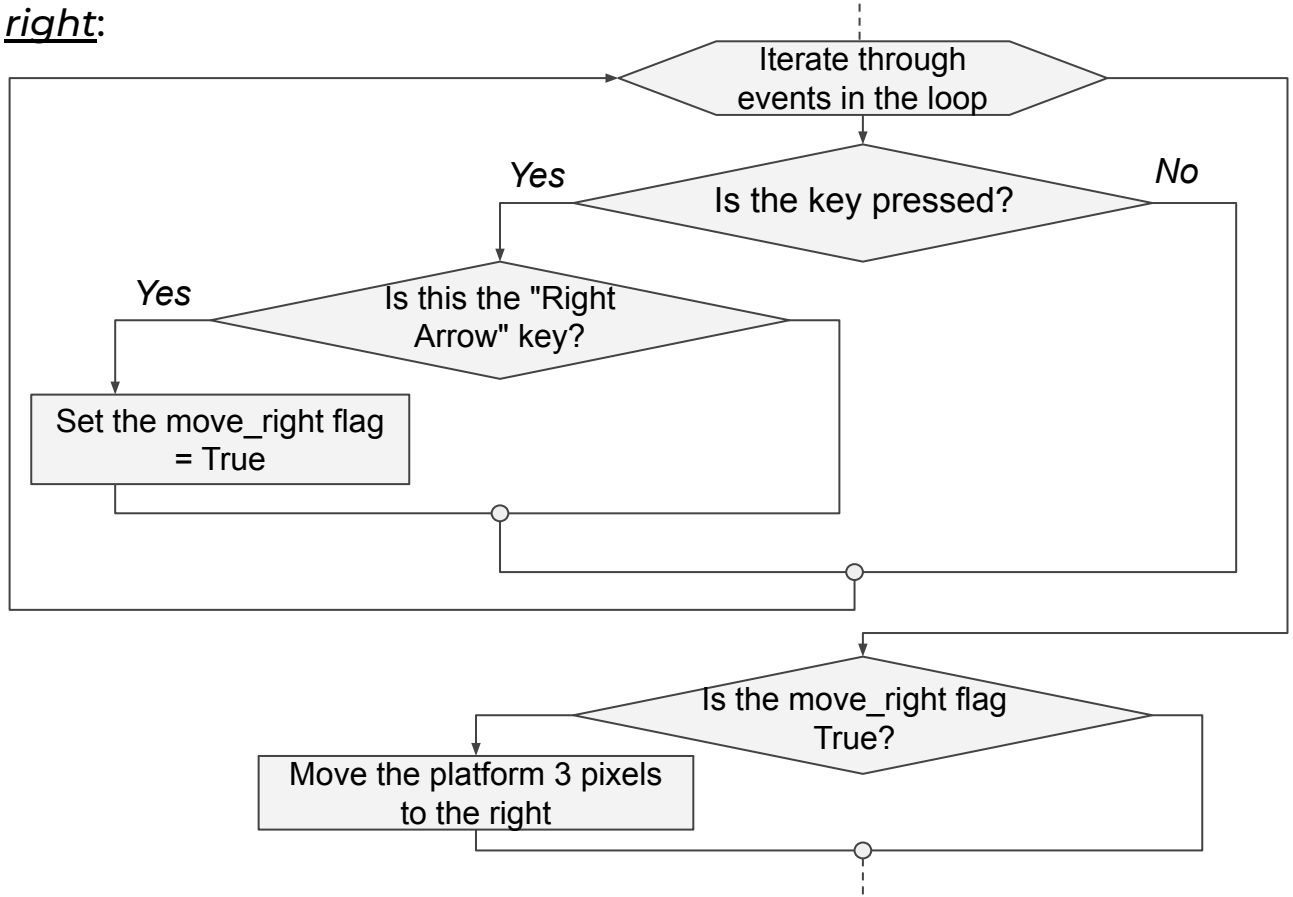
Flag	*Under what conditions is it True?*	*Under what conditions is it False?*
`move_right` — if True, then move to the right	The "Right Arrow" key is pressed	The "Right Arrow" key is raised (not pressed)
move_left — if True, then move to the left	The "Left Arrow" key is pressed	The "Left Arrow" key is raised (not pressed)

When the flag is True, the platform shifts by 3 pixels at each step of the game loop.

Brain storm

Handle keystrokes

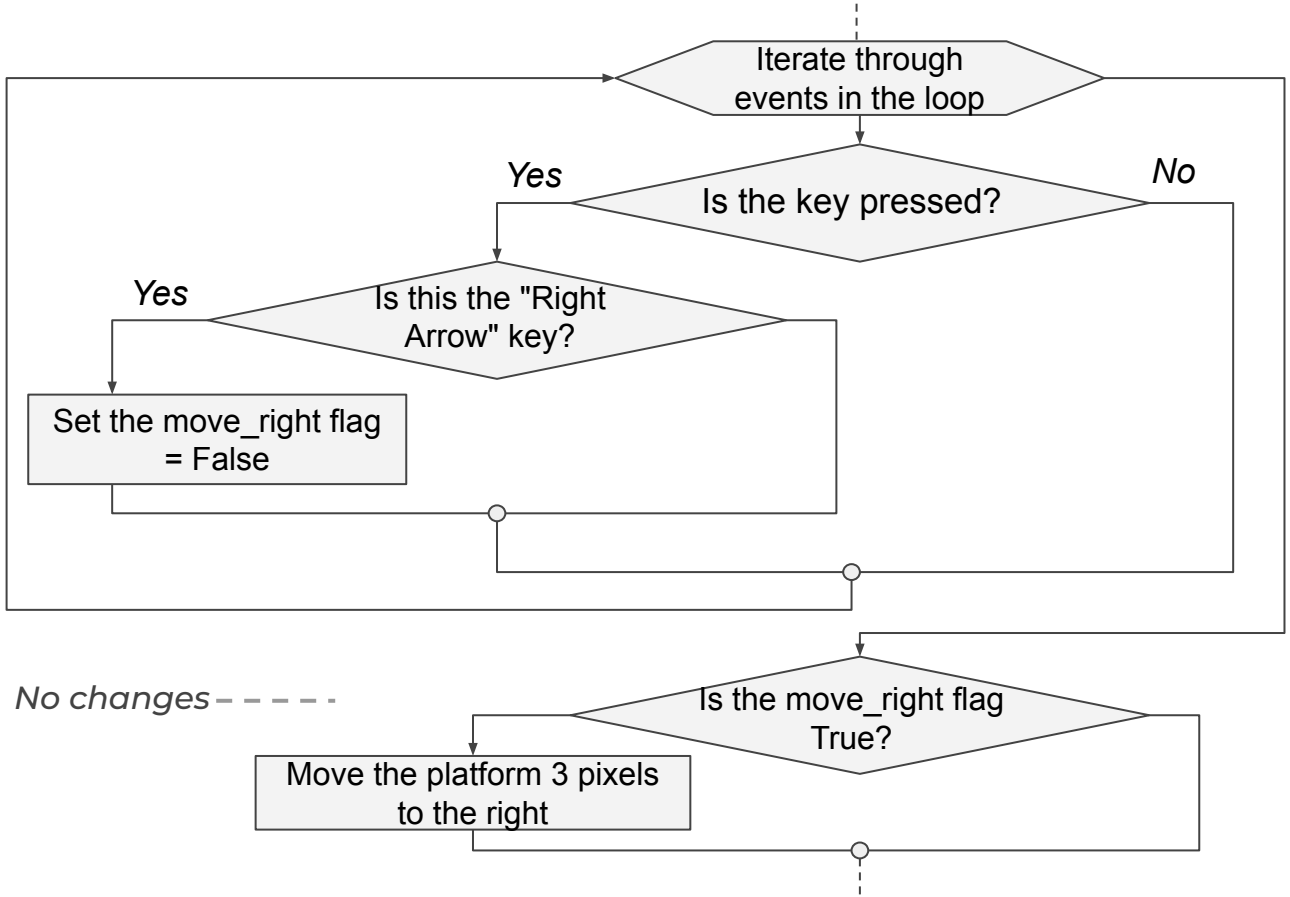
Examine the flowchart with a *fragment* of the algorithm for *movement to the right*:



Brain storm

Handle keystrokes

If the "Right Arrow" key is pressed, you need to set move_right = False.



Brain storm

Handle keystrokes

A fragment of the program code showing the handling of a pressed "Right Arrow" key.

```
while not game_over:
    #...
    for event in pygame.event.get():
        #...
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
                move_right = True
        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                move_right = False


    if move_right:
        platform.rect.x +=3
```

Movement to the left is implemented in the same way.

Brain storm

Program flowchart:

The result is a platform sprite controlled using the keyboard.

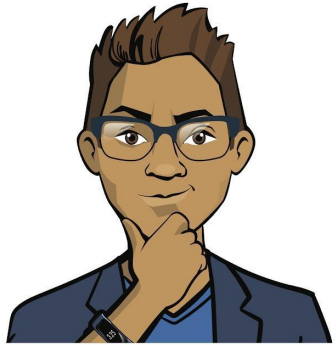
- Create a game scene and sprites
- Create motion flags (= False)
- Until the game is over...
  - Iterate through game events
    - Set flag values when keys are pressed
  - Move the platform according to the flags
  - Redraw sprites, update the scene

Brain storm

Your tasks:

1. Program the movement of the platform sprite when keys are pressed.
2. Make sure that the platform continues to move when the key is held down for a long time.

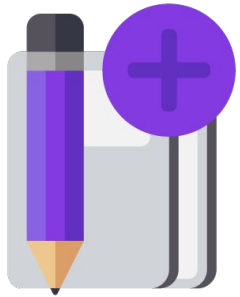
If there is time left over, think about how you could set the initial speed of the ball sprite.



Brain storm

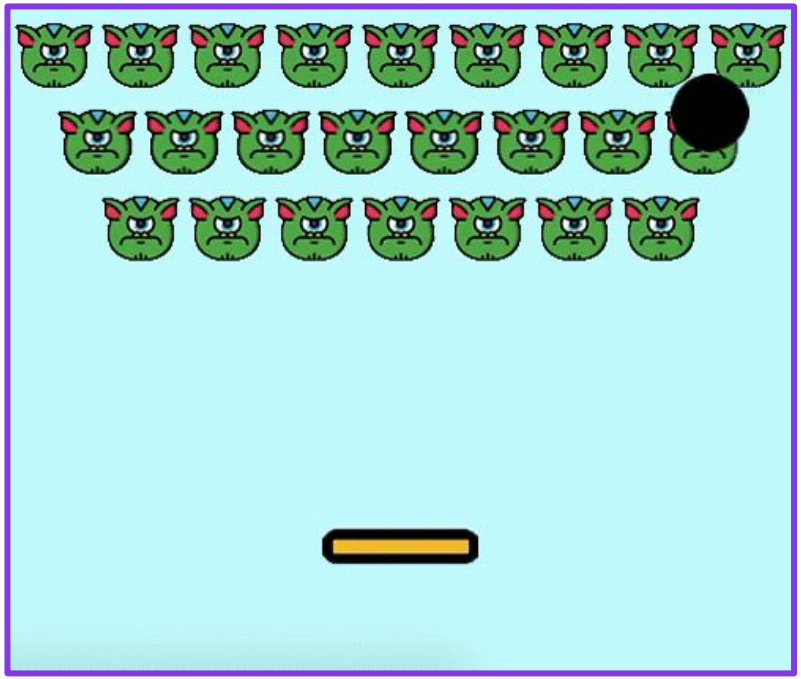
Brainstorm:

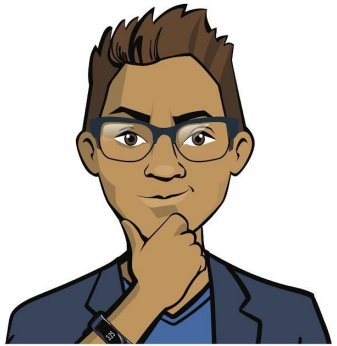
Moving the ball



How can we program the automatic movement of the ball?

Up to now, we've only controlled sprites.
Without user action, a sprite remains in one place.





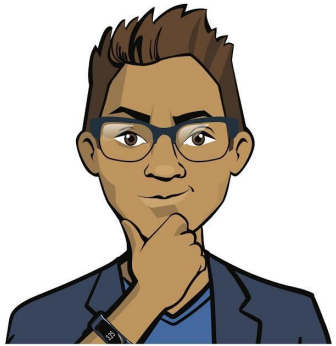
Brain storm

Automatic movement of a sprite

Divide the task into subtasks:

➔ giving the ball an initial constant speed.

What happens when there's a collision with the top and side borders of the scene?



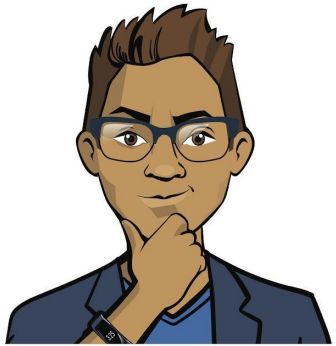
Brain storm

Automatic movement of a sprite

Divide the task into subtasks:

- ➔ giving the ball an initial constant speed.
- ➔ if the ball **reaches the top boundary** of the scene, it bounces off it — and we change its direction to the opposite one;
- ➔ if the ball **reaches the side boundary** of the scene, it bounces off it — and we change its direction to the opposite one;
- ➔ if the ball **touches the platform from above**, then it bounces off it — and we change its direction to the opposite one.

What happens when there's a collision with the lower border of the screen?



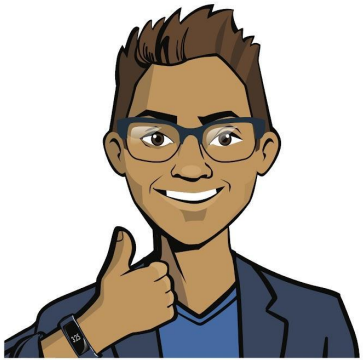
Brain storm

Automatic movement of a sprite

Divide the task into subtasks:

- ➔ giving the ball an initial constant speed.
- ➔ if the ball **reaches the top boundary** of the scene, it bounces off it — and we change its direction to the opposite one;
- ➔ if the ball **reaches the side boundary** of the scene, it bounces off it — and we change its direction to the opposite one;
- ➔ if the ball **touches the platform from above**, then it bounces off it — and we change its direction to the opposite one.

The lower border of the screen is not an obstacle.
When it is crossed, the player loses.

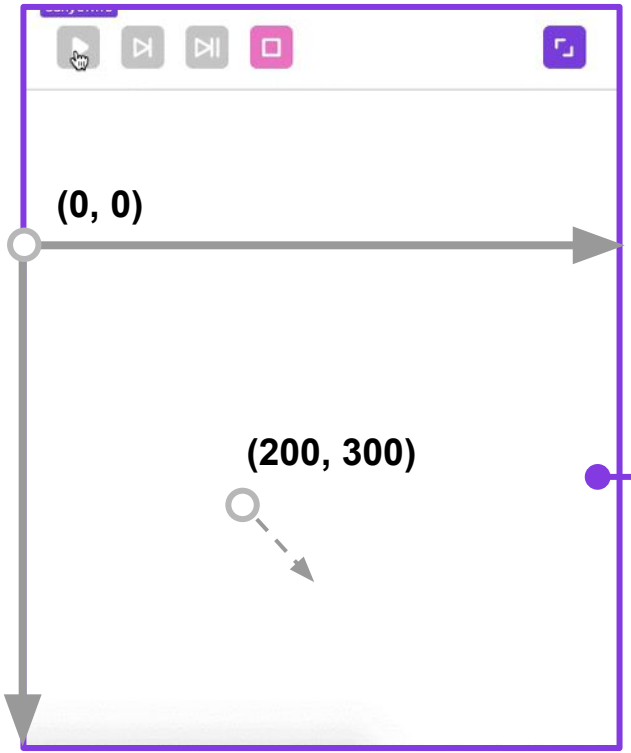


Brain storm

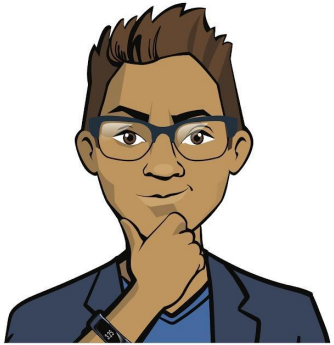
How do we apply a constant speed?

Let the ball sprite appear on the scene at the point (200, 300).

How do we give it a constant speed when the game launches?



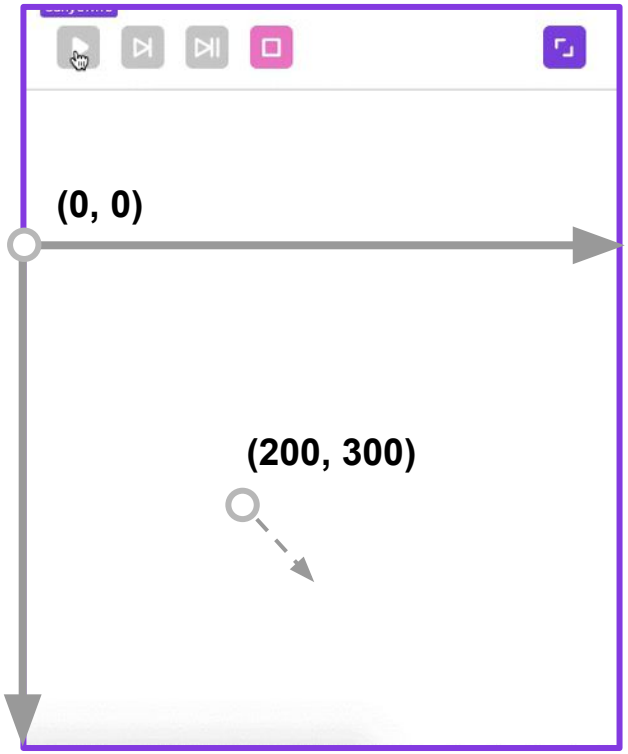
When the game launches, the ball starts moving at a constant speed.



Brain storm

How do we apply a constant speed?

Set the speed as a constant (in the game loop) shift of the sprite by 3 pixels along the horizontal and vertical axes.



At the beginning of the game, enter the variables speed_x and speed_y and assign the value 3:

```
speed_x = 3
speed_y = 3
```

In the game loop, set the movement along the coordinates of both axes:

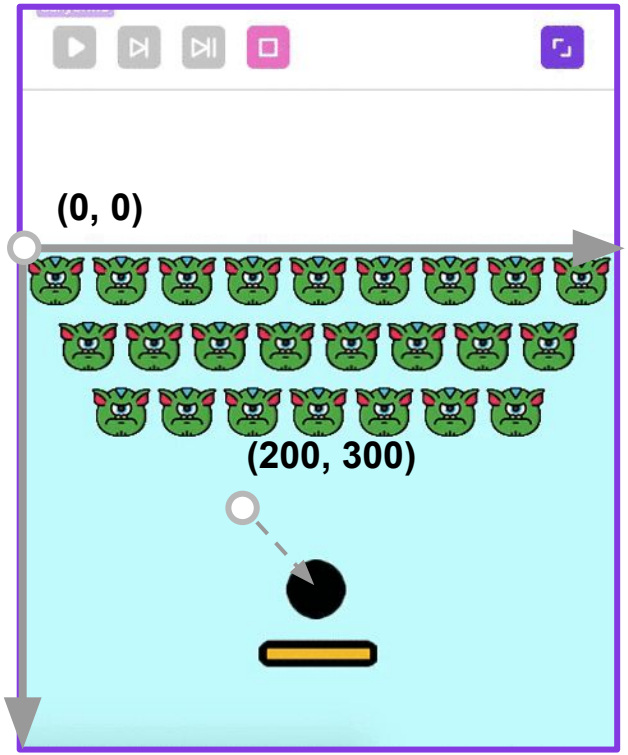
```
while game_over != True:
  #...
  ball.rect.x += speed_x
  ball.rect.y += speed_y
  #...
```

How will the ball behave if you launch the program right now?

Brain storm

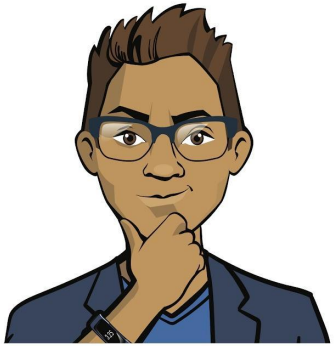
How do we apply a constant speed?

Set the speed as a constant (in the game loop) shift of the sprite by 3 pixels along the horizontal and vertical axes.



At startup, the initial speed will be set, but the sprite will not "see" the obstacles.

How do we program the ball to bounce off the top and side walls and the platform?



Brain storm

Bounce the ball off the walls and platform

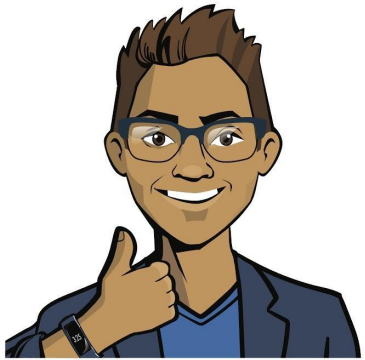
Program the bounce off the platform. To do this, we need to have it recognize the **collision of two Picture-type sprites**.

Add a new **colliderect()** method to the Area class. We will consider contact between the sprites' rectangular frames to be collision.

Command	*Purpose*
`res = rect.colliderect(rect)`	The command returns True if a collision occurred, and False if it did not

Addition of the Area class:

```
class Area():
    #...
    def colliderect(self, rect):
        return self.rect.colliderect(rect)
```



Brain storm

Bounce the ball off the walls and platform

Program the bounce off the platform. To do this, we need to have it recognize the **collision of two Picture-type sprites**.

Add a new **colliderect()** method to the Area class. We will consider contact between the sprites' rectangular frames to be collision.

Command	*Purpose*
`res = rect.colliderect(rect)`	The command returns True if a collision occurred, and False if it did not

Bounce the ball off the platform:

```
while game_over != True:
    #...
    if ball.colliderect(platform.rect):
       speed_y *= -1
```

Change the direction of vertical movement to the opposite. It doesn't change horizontally.

Brain storm

Bounce the ball off the walls and platform

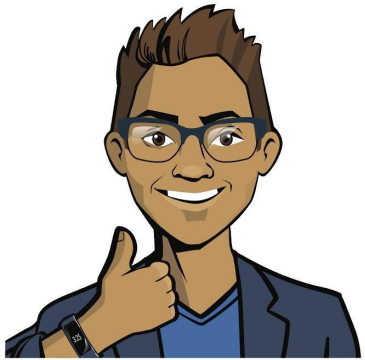
To implement the rebound from the walls, we can analyze the current coordinates of the ball:

```
if  ball.rect.y < 0:
      speed_y *= -1
```

```
if ball.rect.x > 450 or ball.rect.x < 0:
      speed_x *= -1
```

If the ball's y coordinate goes above the upper border of the screen, then we change the direction of vertical movement.

If the ball's x coordinate goes beyond the side border of the screen, then we change the direction of horizontal movement.



Brain storm

Overall scheme for the program

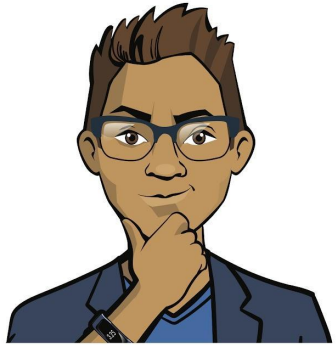
The main part of the program is the movement of the ball:

- Create a game scene and sprites
- Create movement flags and initial vertical and horizontal speeds.
- Until the game is over...
  - Iterate through game events
    - Set flag values when keys are pressed
  - Move the platform according to the flags
  - Automatic movement of the ball
  - Redraw sprites, update the scene

Brain storm

Your tasks:

1. Give the ball sprite the initial movement speed when starting the game.
2. Supplement the Area class with a new method and program the rebound of the ball from the platform and the walls of the scene.



Brain storm