Micro:bit Snake Workshop

Things to do while you're waiting!

Sign up for our mailing list!

tinyurl.com/earsmail

Like our Facebook Page!

fb.me/earsedi

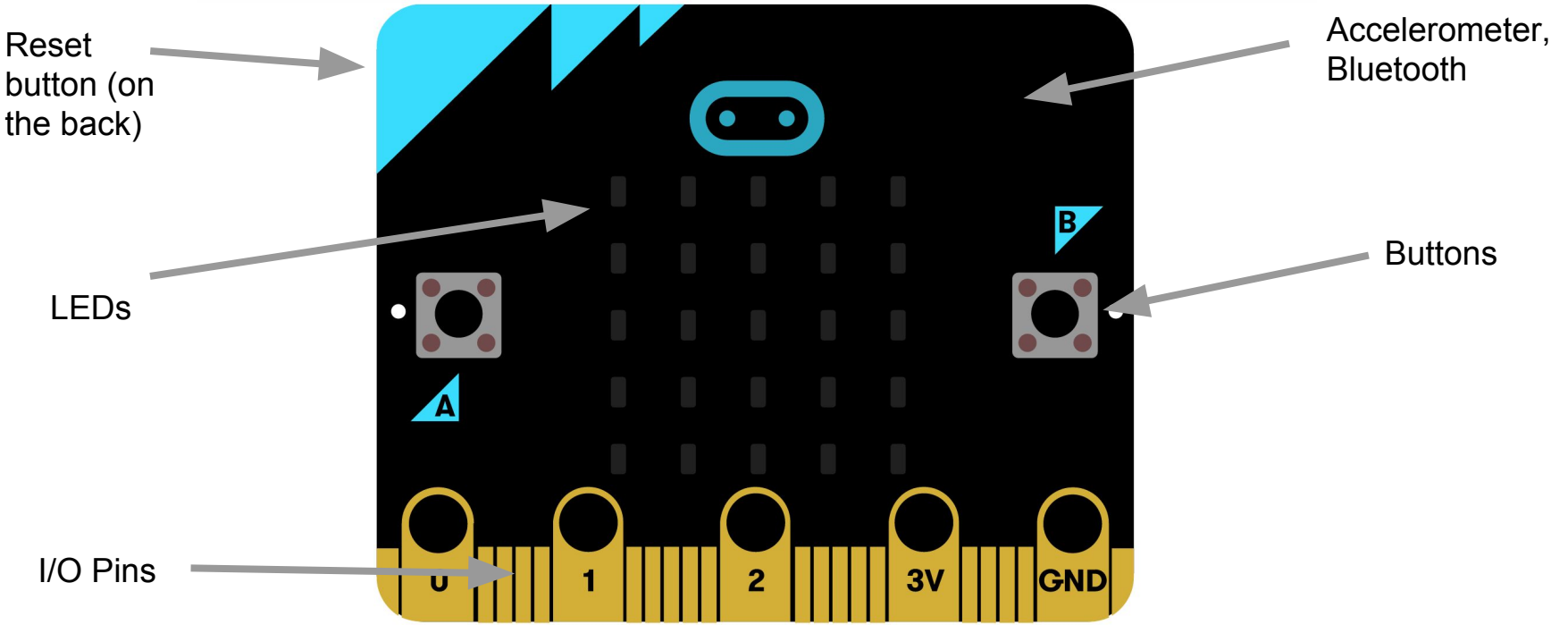
Buy a membership!

tinyurl.com/earsmember

The plan for today

1. Introductions to concepts
2. Environment setup
3. Coding

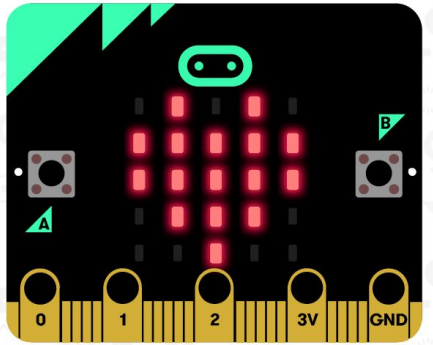
The micro:bit



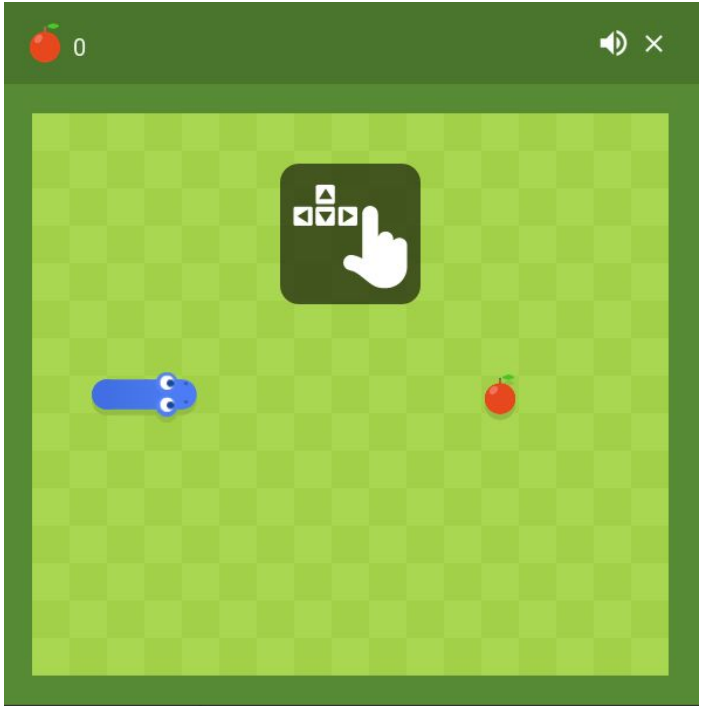
Coding on the micro:bit

```
from microbit import *

display.show(Image.HEART)
```



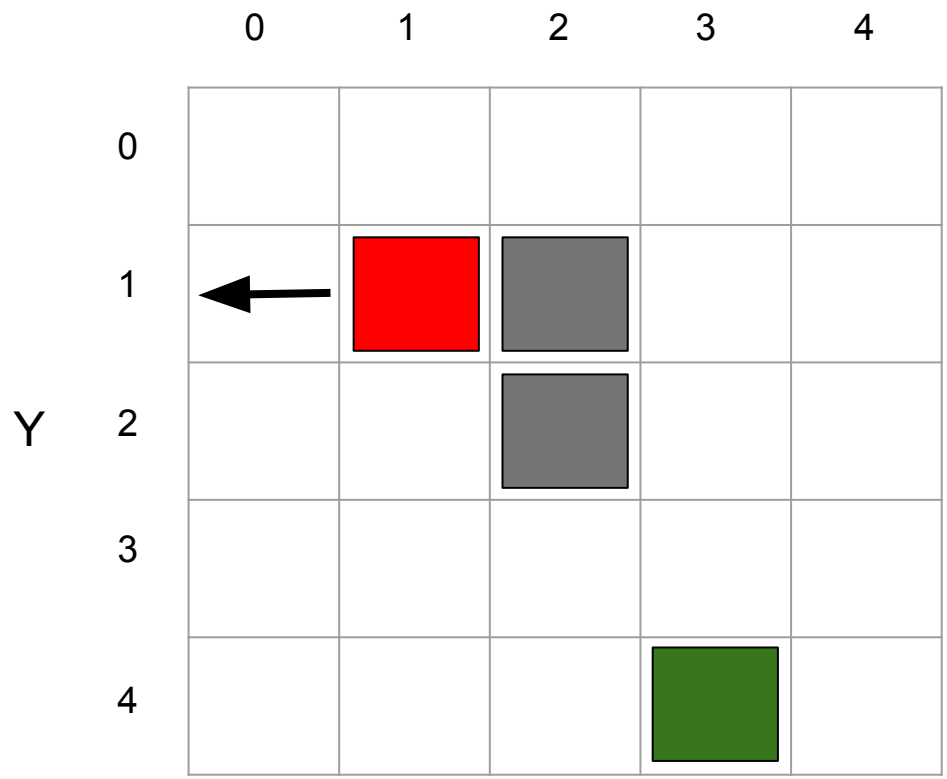
Snake



Components of a snake game?

What are the components of a snake game?

- An N by N grid.
- A snake - list of pairs of coordinates. (X and Y pairs)
- A food item - a coordinate pair. (X and Y)
- A direction of movement - “up”, “down” “right” or “left”
- An end condition - is the game finished?



- Grid - 5 by 5
- Snake - [[1, 1], [2, 1], [2, 2]]
- Food item - [3, 4]
- Direction of movement - “left”
- End condition - False

The game loop

```
while True:
    game.handle_input()
    game.update()
    game.draw()
    sleep(500)
```

Environment set up

python.microbit.org

tinyurl.com/earssnake

Environment set up

Programming the micro:bit

1. Plug micro:bit into USB
2. Click “Download” on Python editor
3. Drag .hex file to micro:bit (appears as USB)

Instances

Class (cookie cutter)

Instances (cookies)

The code in front of you

```
from microbit import *

class Snake:
    def __init__(self):
        ## UNCOMMENT AND FILL IN THE # LINES BELOW ...
        ...
    ...

game = Snake()

while True:
    ...
```

The code in front of you

```
from microbit import *

class Snake:
    def __init__(self):
        ## UNCOMMENT AND FILL IN THE # LINES BELOW ...
        ...
    ...

game = Snake()

while True:
    ...
```

The code in front of you

```
from microbit import *

class Snake:
    def __init__(self):
        ## UNCOMMENT AND FILL IN THE # LINES BELOW ...
        ...
    ...

game = Snake()

while True:
    ...
```

The code in front of you

```
from microbit import *

class Snake:
    def __init__(self):
        ## UNCOMMENT AND FILL IN THE # LINES BELOW ...
        ...
    ...

game = Snake()

while True:
    ...
```

Python lists

```
my_list = [1, 2]

my_list_of_lists = [[1, 2],
                    [2, 2]]
```

Python lists

```
>>> my_list = [[1, 2], [2, 2]]
>>> my_list[0]
[1, 2]
>>> my_list[1]
[2, 2]
>>> my_list.append([3, 2])
>>> my_list
[[1, 2], [2, 2], [3, 2]]
>>> my_list[-1]
[3, 2]
```

Python lists

```
>>> my_list = [[1, 2], [2, 2]]
>>> my_list[0]
[1, 2]
>>> my_list[1]
[2, 2]
>>> my_list.append([3, 2])
>>> my_list
[[1, 2], [2, 2], [3, 2]]
>>> my_list[-1]
[3, 2]
```

Python lists

```
>>> my_list = [[1, 2], [2, 2]]
>>> my_list[0]
[1, 2]
>>> my_list[1]
[2, 2]
>>> my_list.append([3, 2])
>>> my_list
[[1, 2], [2, 2], [3, 2]]
>>> my_list[-1]
[3, 2]
```

Python lists

```
>>> my_list = [[1, 2], [2, 2]]
>>> my_list[0]
[1, 2]
>>> my_list[1]
[2, 2]
>>> my_list.append([3, 2])
>>> my_list
[[1, 2], [2, 2], [3, 2]]
>>> my_list[-1]
[3, 2]
```

Python slices

```
>>> my_list
[[1, 2], [2, 2], [3, 2], [4, 5]]
>>> my_list[1:3]
[[2, 2], [3, 2]]
>>> my_list[1:]
[[2, 2], [3, 2], [4, 5]]
>>> my_list[:3]
[[1, 2], [2, 2], [3, 2]]
```

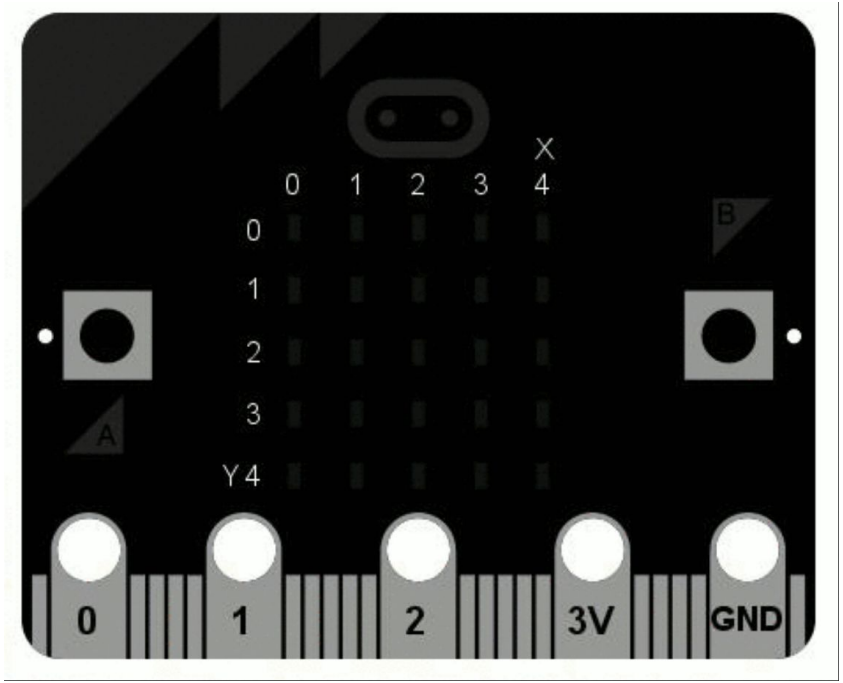
Python slices

```
>>> my_list
[[1, 2], [2, 2], [3, 2], [4, 5]]
>>> my_list[1:3]
[[2, 2], [3, 2]]
>>> my_list[1:]
[[2, 2], [3, 2], [4, 5]]
>>> my_list[:3]
[[1, 2], [2, 2], [3, 2]]
```

Python slices

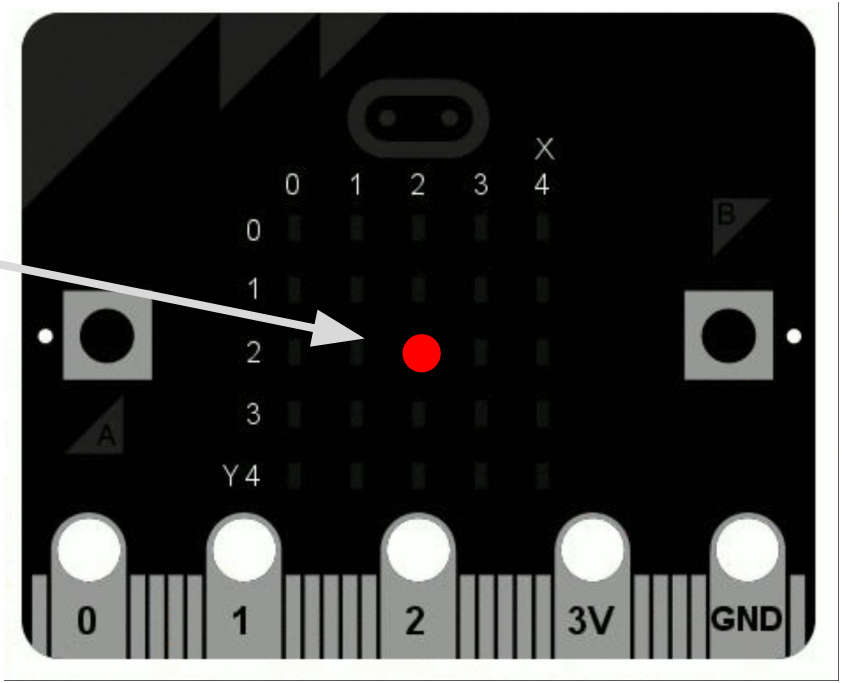
```
>>> my_list
[[1, 2], [2, 2], [3, 2], [4, 5]]
>>> my_list[1:3]
[[2, 2], [3, 2]]
>>> my_list[1:]
[[2, 2], [3, 2], [4, 5]]
>>> my_list[:3]
[[1, 2], [2, 2], [3, 2]]
```

Micro:bit coordinates



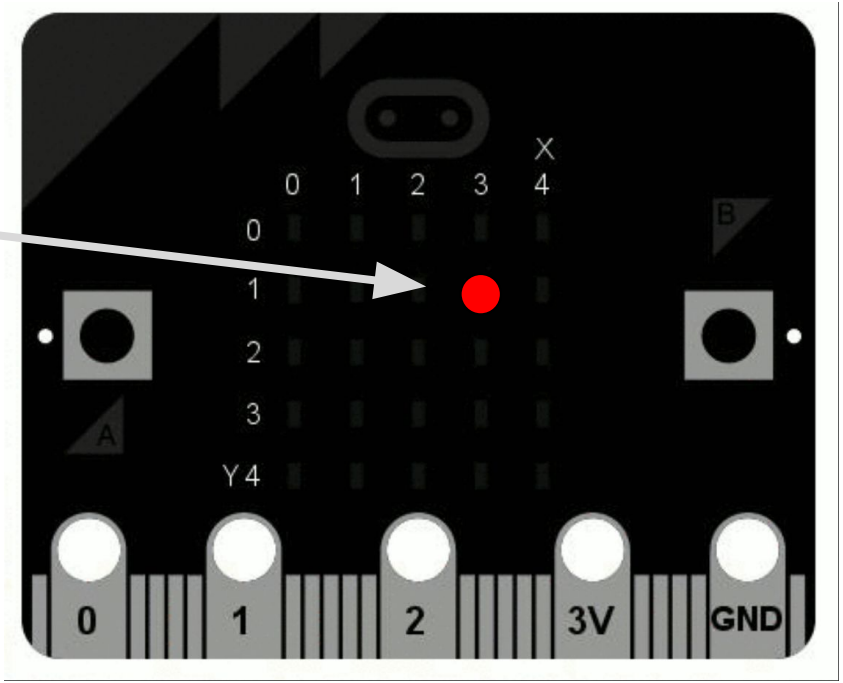
Micro:bit coordinates

Coordinates: (2 , 2)



Micro:bit coordinates

Coordinates: (3 , 1)



Defining game state

TODO: Uncomment and initialise the following variables with some starting values:

- Snake (List of coordinates)
- Direction: up, down, left or right.
- Food location (A coordinate)
- Game end variable (`True` or `False`)

Functions to change:

`__init__()`

Game state

```
def __init__(self):
    self.current_direction = "up"
    self.snake = [[2, 2]]
    self.food = [0, 2]
    self.end = False
```

Python loops

```
>>> my_list = [1, 2, 3]
>>> for element in my_list:
...     print(element)
1
2
3
```

Python loops

```
>>> for element in range(4):
...     print(element)
0
1
2
3
```

Drawing

TODO:

- Clear the display of any pixels
- Draw the snake
- Draw the food
- Make sure the player can distinguish the snake pixels from food pixels.

Functions to change:

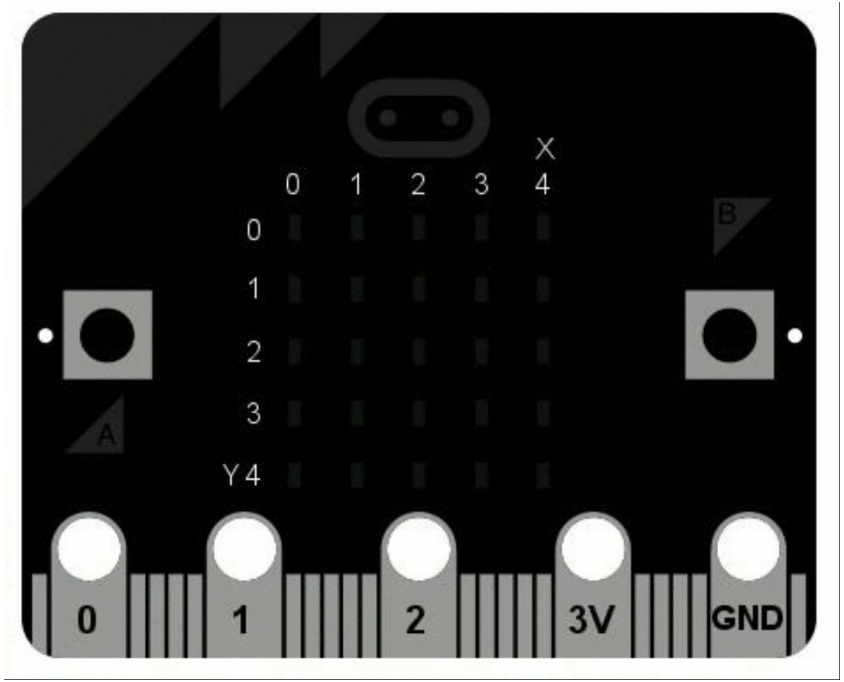
`draw()`

Drawing

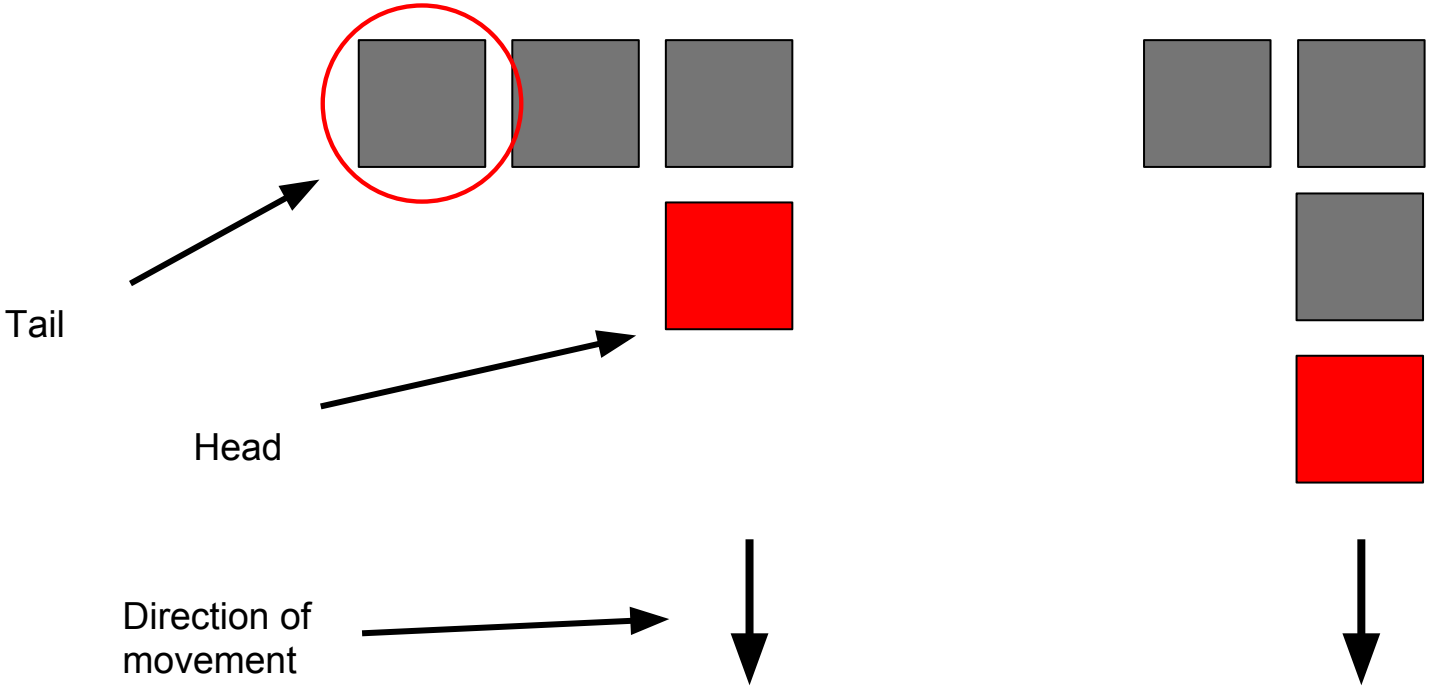
```
def draw(self):
    display.clear()
    display.set_pixel(self.food[0],
                      self.food[1], 5)
    for part in self.snake:
        display.set_pixel(part[0],
                          part[1], 9)
```

Moving the snake

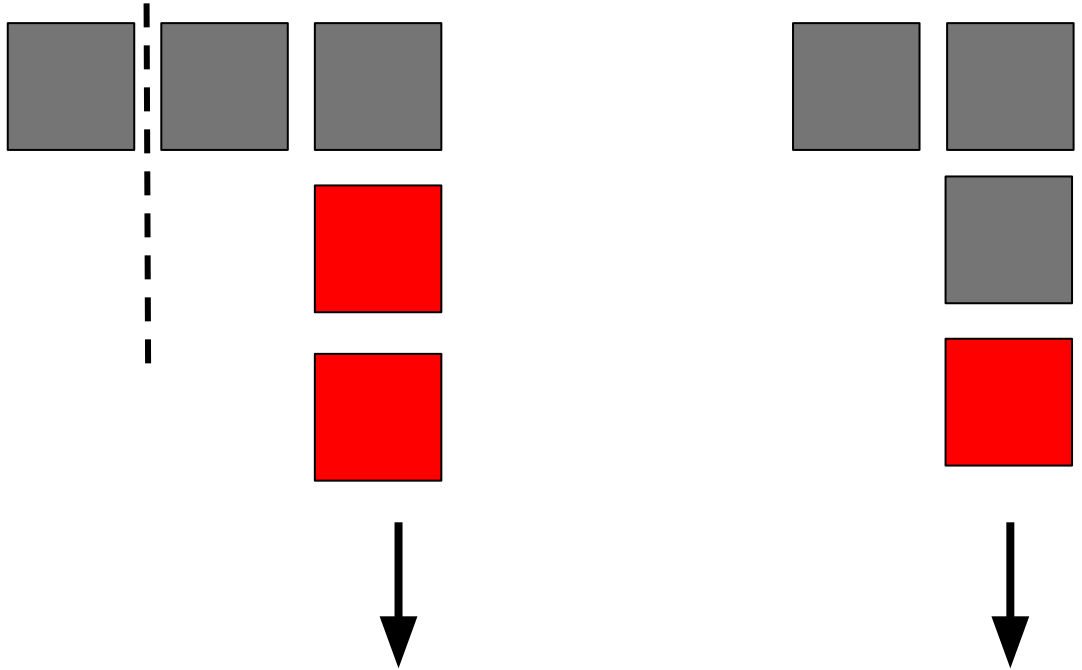
```
update()
```



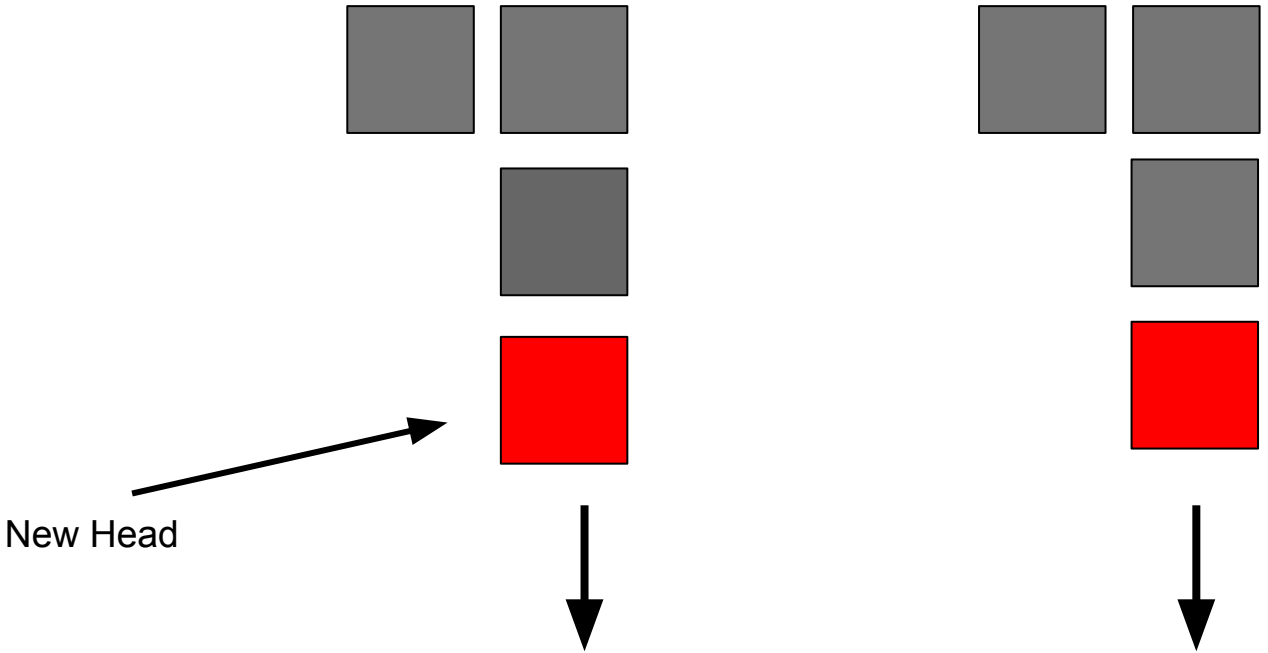
Moving the snake



Moving the snake



Moving the snake



Python conditions (if statements)

```
>>> hungry = True
>>> if hungry:
...     print("Want to eat!")
Want to eat!
```

Python conditions (if statements)

```
>>> hungry = False
>>> if hungry:
...     print("Want to eat!")
... else:
...     print("Don't want to eat!")
Don't want to eat!
```

Python conditions (if statements)

```
>>> color = "blue"
>>> if color == "red":
...     print("I am red!")
... elif color == "blue":
...     print("I am blue!")
... else:
...     print("I am some other color!")
I am blue!
```

Moving the snake

TODO:

- Get the current head of the snake (last element of list `[-1]`)
- Create some new head coordinates for the snake based on the direction of movement (`if` statements)
- Add this new head to the snake coordinates list (`append()` function)
- Remove the last coordinate of the snake (`slices`)

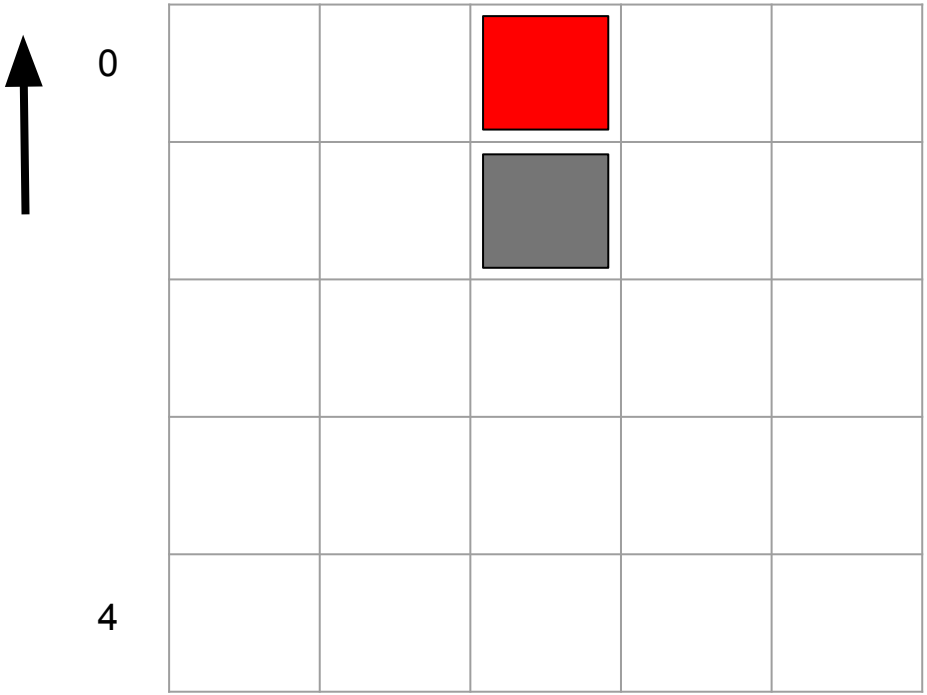
Functions to change:

`update()`

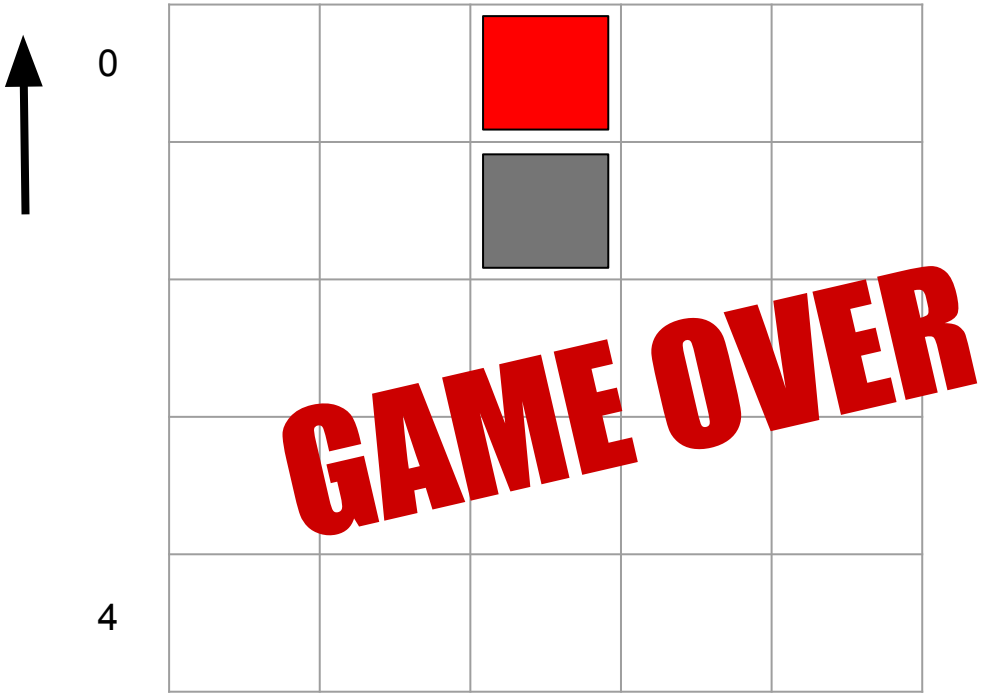
Moving the snake

```
def update(self):
    new_head = self.snake[-1]
    if self.current_direction == "up":
        new_head[1] -= 1
    elif self.current_direction == "down":
        ...
    ...
    self.snake.append(new_head)
    self.snake = self.snake[1:]
```

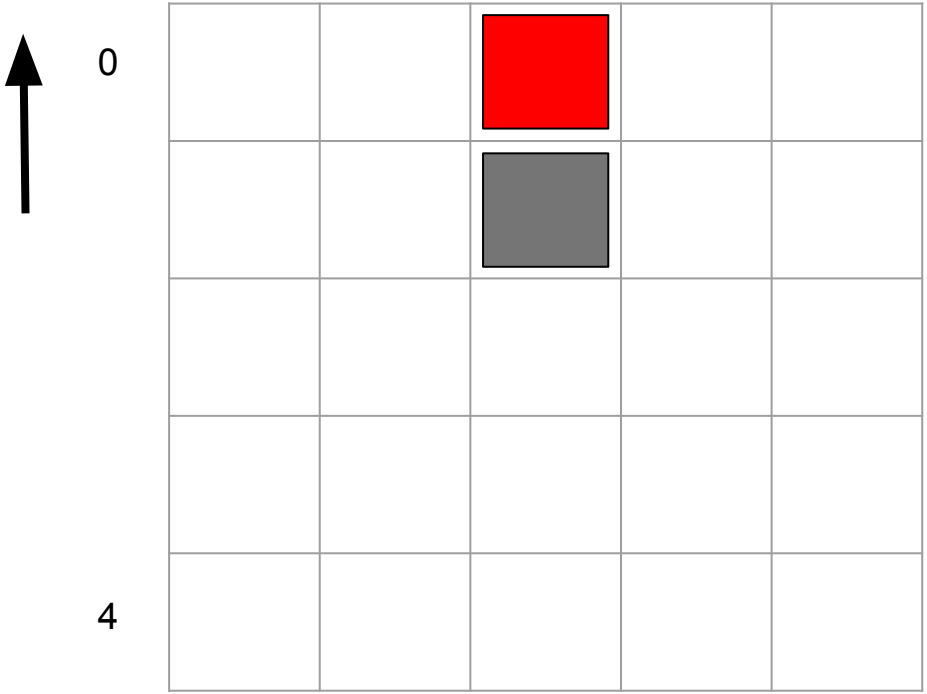
Fixing the wrapping issue



Fixing the wrapping issue



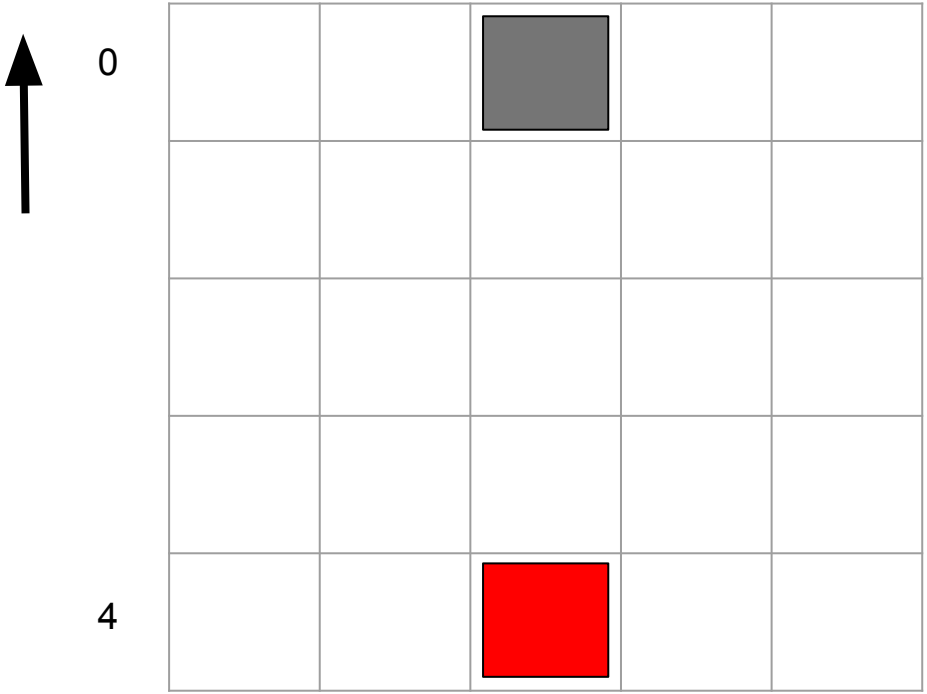
Fixing the wrapping issue



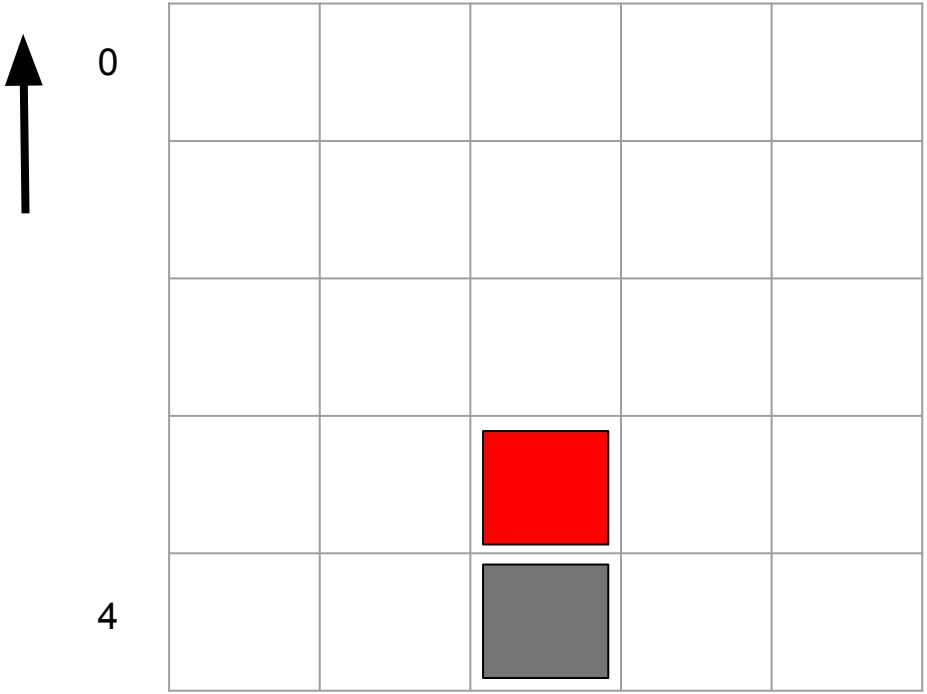
Head coordinates:
(3, 0)

Fixing the wrapping issue

Head coordinates:
(3, 4)



Fixing the wrapping issue



Fixing the wrapping issue

TODO:

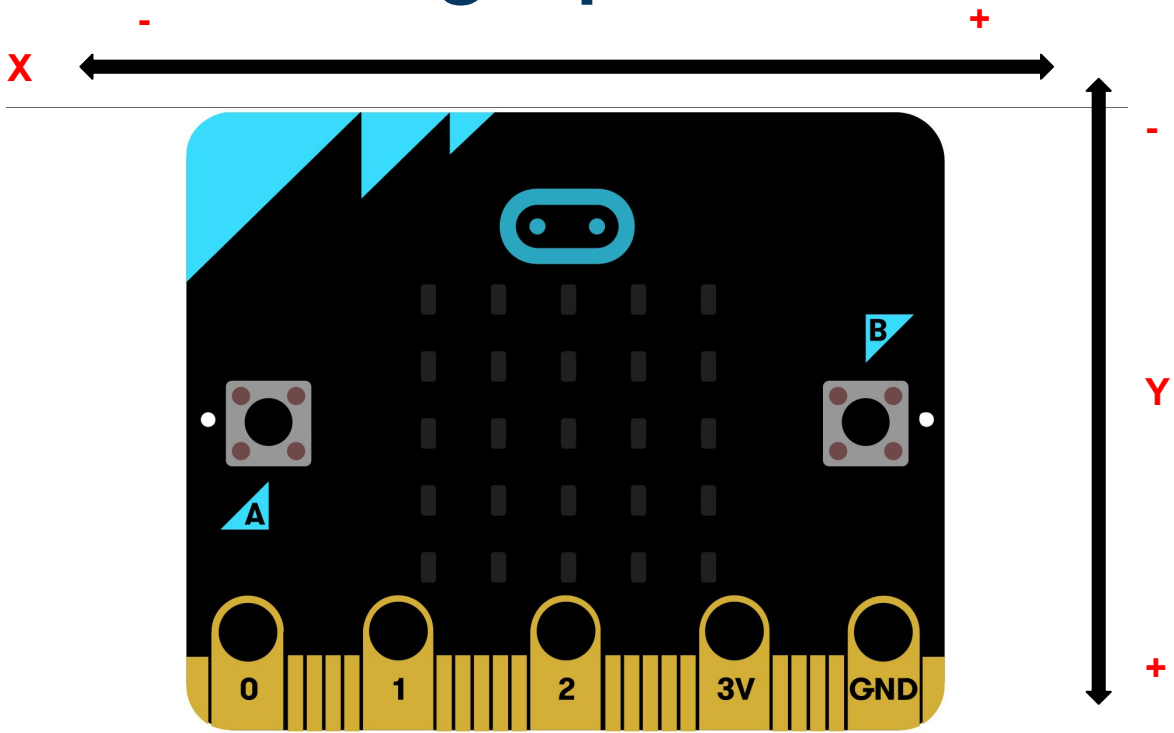
- Check if each coordinate (X and Y) is outside of the range (not between 0 and 4)
- If it is out of range, change the coordinates (- 1 → 4 and 5 → 0)

```
update(self)
```

Fixing the wrapping issue

```
def update(self):
    ...
    if new_head[0] < 0:
        new_head[0] = 4
    elif new_head[0] > 4:
        new_head[0] = 0
    ...
    self.snake.append(new_head)
...
```

Getting input



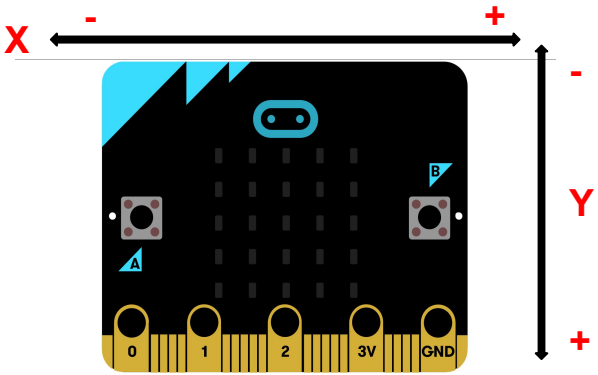
Getting input

TODO:

- Get the tilt values for X and Y (`accelerometer`)
- Update the `current_direction` value (with “left”, “right”, “up” or “down”)

Functions to change:

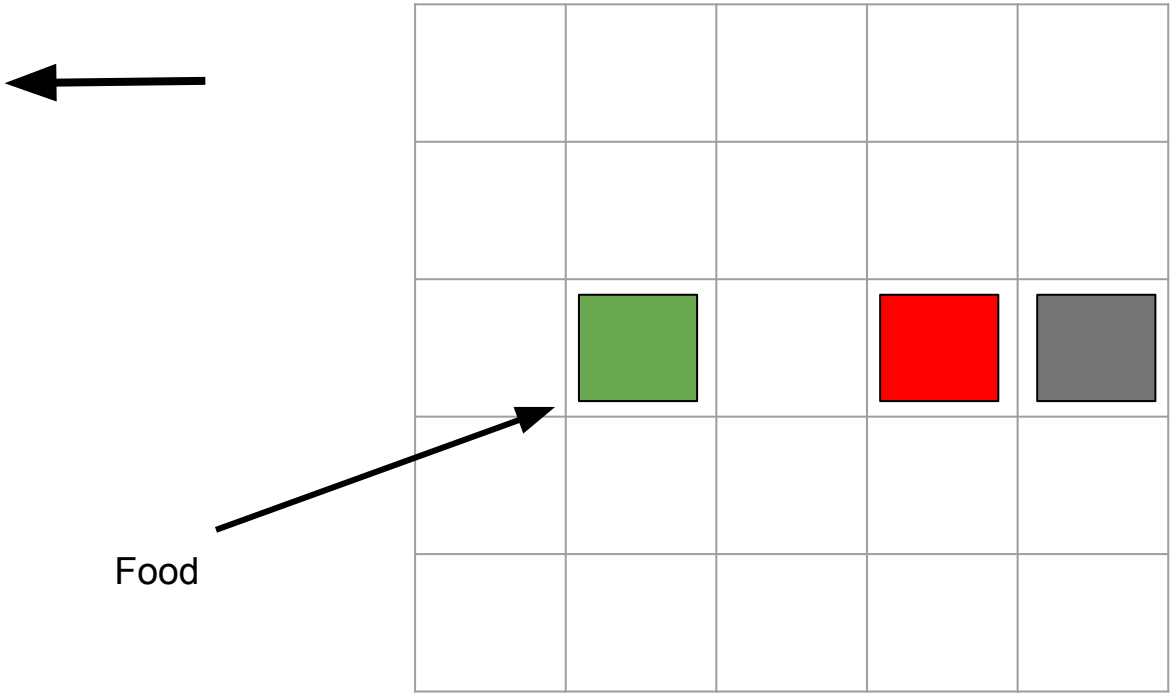
`handle_input(self)`



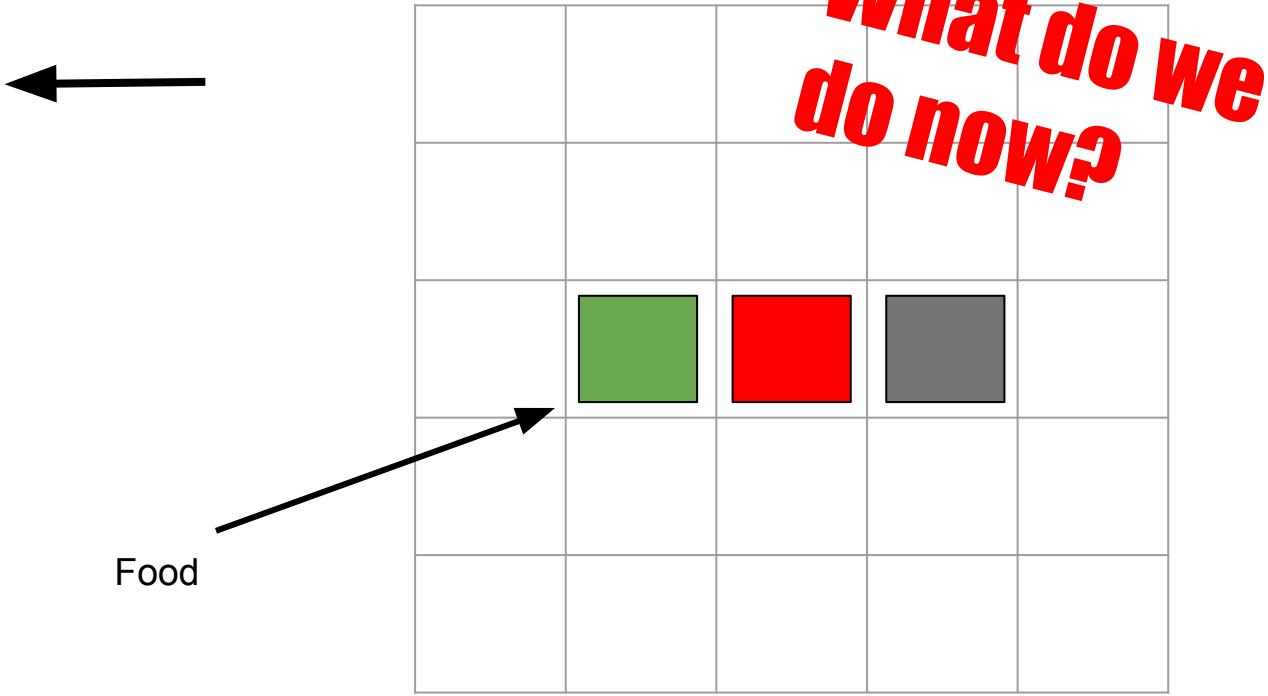
Getting input

```
def handle_input(self):
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    if abs(x) > abs(y):
        if x < 0:
            self.current_direction = "left"
        else:
            self.current_direction = "right"
    else:
        if y < 0:
            self.current_direction = "up"
        else:
            self.current_direction = "down"
```

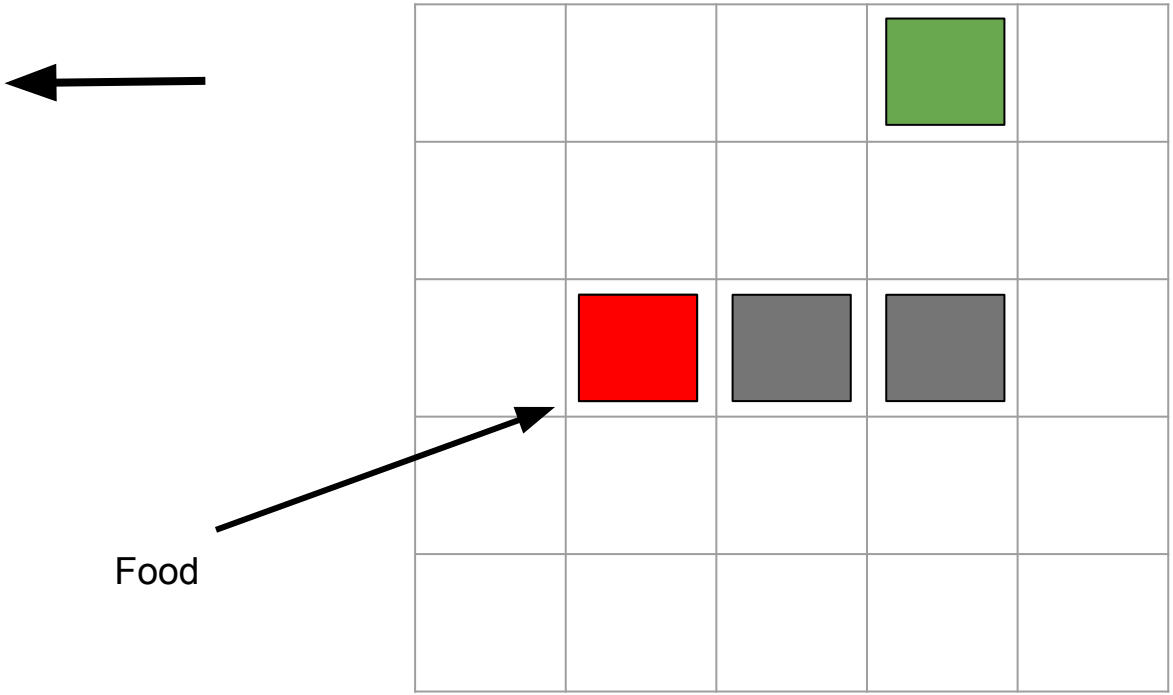
Food



Food



Food



Food eating

TODO:

- Check whether the new head of the snake is in the same tile as the food
- If yes then generate new food (DO NOT code this yet!)
- Otherwise remove tail of snake

Functions to change:

`update(self)`

Food eating

```
def update(self):
    ...
    self.snake.append(new_head)
    if new_head == self.food:
        # generate new food
        pass
    else:
        self.snake = self.snake[1:]
...
```

Random in Python

```
>>> from random import randint
>>> randint(0, 4)
1
>>> randint(0, 4)
4
>>> randint(0, 4)
0
...
```

Food generating

TODO:

- Randomly generate new X and Y coordinates for the food (`randint`)
- Check whether the new food is inside of the snake (`while food in snake`)
- If the new food is inside of snake, generate X and Y again.

Do this if new head is same co-ords as current food

```
update(self)
```

Food generating

```
def update(self):
    ...
    self.snake.append(new_head)
    if new_head == self.food:
        self.food = [randint(0, 4), randint(0, 4)]
        while self.food in self.snake:
            self.food = [randint(0, 4), randint(0, 4)]
    else:
        self.snake = self.snake[1:]
...
```

Python loop control

```
>>> num = 0
>>> while True:
...     print(num)
...     num += 1
...     if num == 4:
...         break
0
1
2
3
```

Ending the game

TODO:

- Check whether the new head is in snake (do this before appending)
- If it is set `end` to `True`
- **In the game loop** display a sad face if the game is over

Functions to change:

`update(self)`, `game loop`

Ending the game

```
def update(self):
    ...
    # we've modified our new_head
    if new_head in self.snake:
        self.end = True
    self.snake.append(new_head)
    ...
```

Ending the game

```
while True:
    ...
    game.update()
    if game.end:
        display.show(Image.SAD)
        break
    game.draw()
    ...
```

Micro:bit simulator

https://create.withcode.uk/

Import to access a Micro:bit simulator:

```
from microbit import *
```

Challenges for the brave :)

1. Make the new_head generation based on the directions code more compact (replace the repeated if statements)
2. Make the keeping the coordinates in bounds more compact (replace the repeated if statements)
3. Currently there is no winning condition. Change the code so that you win when the snake fills the entire screen.

tinyurl.com/microbitfeedback