Checking qualifications

To get started with the work tasks, demonstrate your knowledge level .

Prove that you are ready for brainstorming!

Checking qualifications

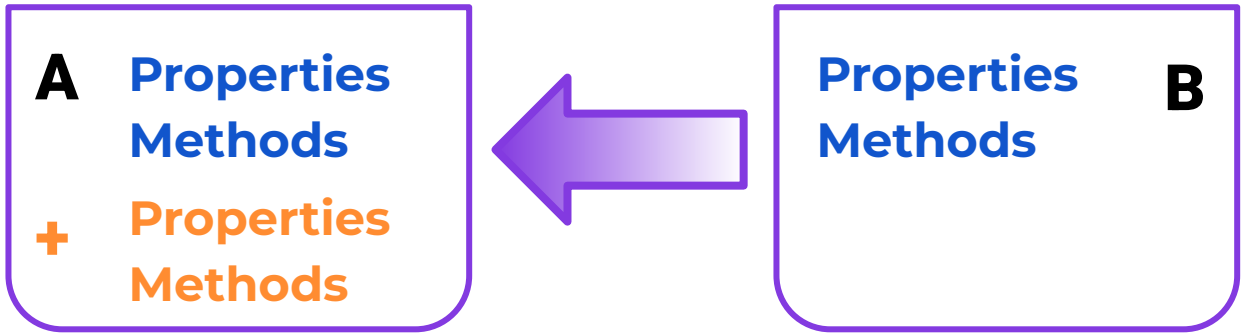
What does inheritance mean?

What is a superclass and a derived class?

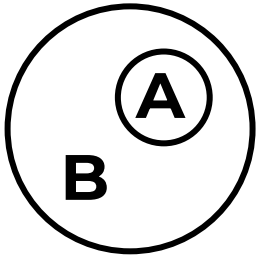
Checking qualifications

Inheritance

Class inheritance helps **transfer all the skills** previously written for a **more general class** into another, more private class, **the derived class** .



Derived class **Superclass**



Class A is nested within class B

Checking qualifications

Think of some real-life examples of classes and subclasses

Checking qualifications

Classes and subclasses

Almost all classes are parents of some classes and derived classes of others.

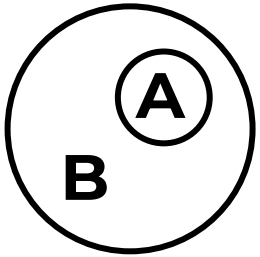
All ***computer games*** **are** ***programs***

All cats are animals

All desks are tables

All comets are celestial bodies

All cars are modes of transport



Checking qualifications

How can you create a derived class by adding new methods?

Checking qualifications

Creating a derived class

Supposing the superclass has already been written, then to create a derived class we need to:

- when creating a derived class, specify the *name of the superclass*;
- add the necessary methods to the derived class.

```
class Derived class name ( Superclass name ):
    def Method name (self, Value ):
        Action with the object and properties

    def Method name (self, Value ):
        Action with the object and properties
```

*Option with the introduction of **only new methods.***

When creating an instance of a derived class, the superclass constructor will be called!

Checking qualifications

How can you create a derived class by adding new properties and methods?

Checking qualifications

Creating a derived class

To create a derived class we need to:

- when creating a derived class, specify the *name of the superclass*;
- create a constructor, introduce the superclass properties, and add new ones;
- add the necessary methods to the derived class.

```
class Derived class name( Superclass name ):
    def __init__(self, Value , Value ):
        super().__init__( Value )
        self. New prop = Value
    def Method name (self, Value ):
        Action with the object and properties
```

Option ***with the introduction of a new property.***

The constructor takes over the properties of the superclass and adds a new one.

Checking qualifications

Qualifications confirmed!

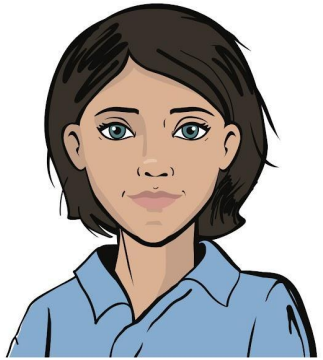
Great, you are ready to brainstorm and complete your work task!

Checking qualifications

Sprite touching

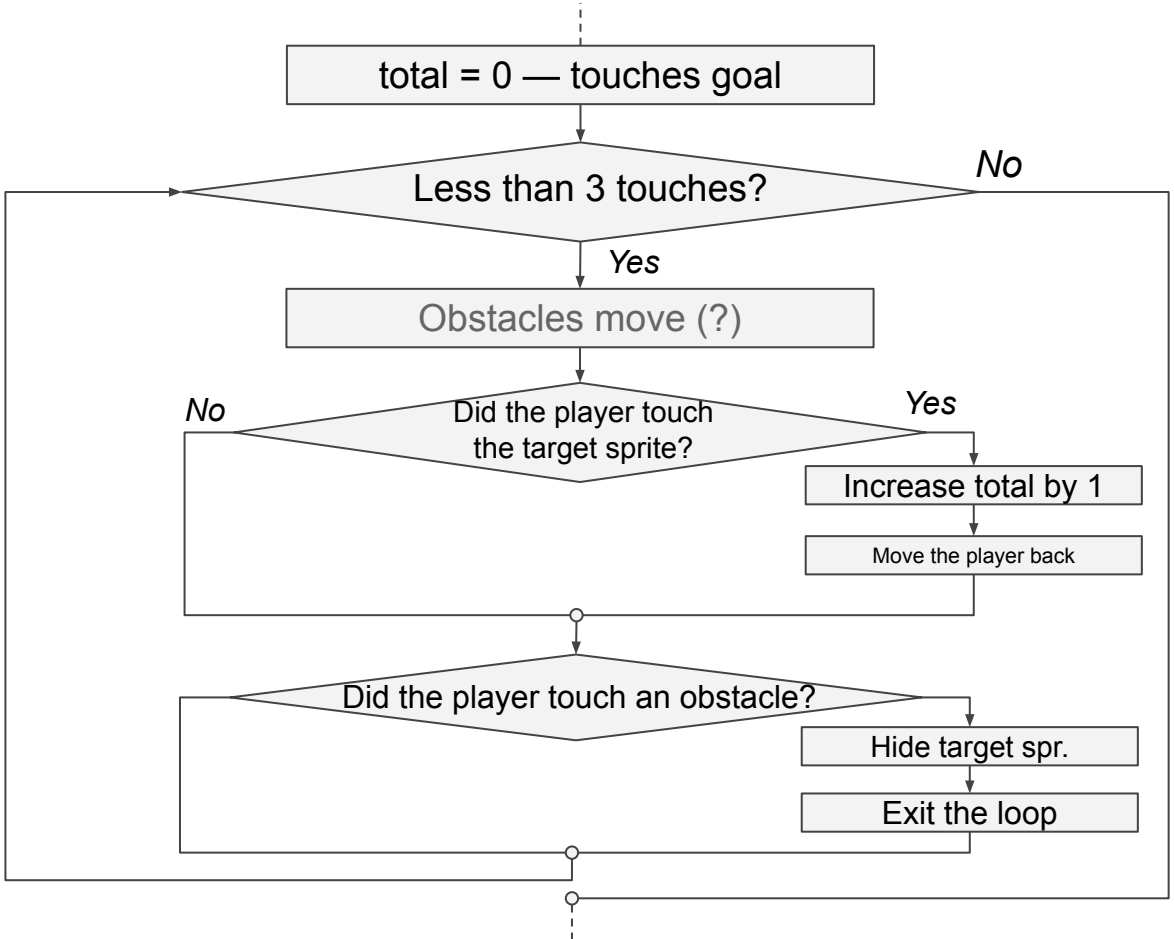
According to the terms of reference, the game should continue until:

- ➔ **the** sprite player **touches the target sprite**;
- ➔ the sprite player **touches any obstacle** at least once.



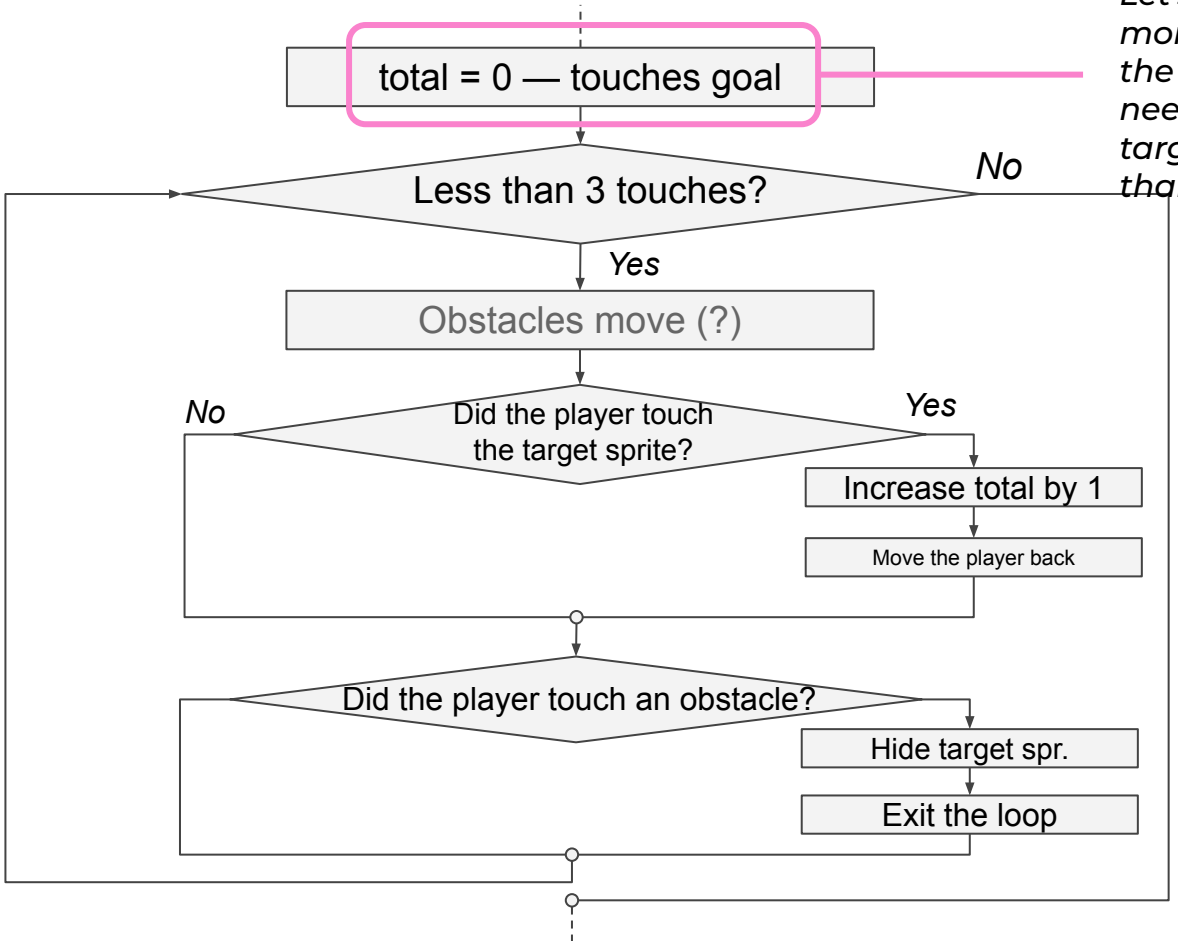
Brain storm

Program flow chart:



Brain storm

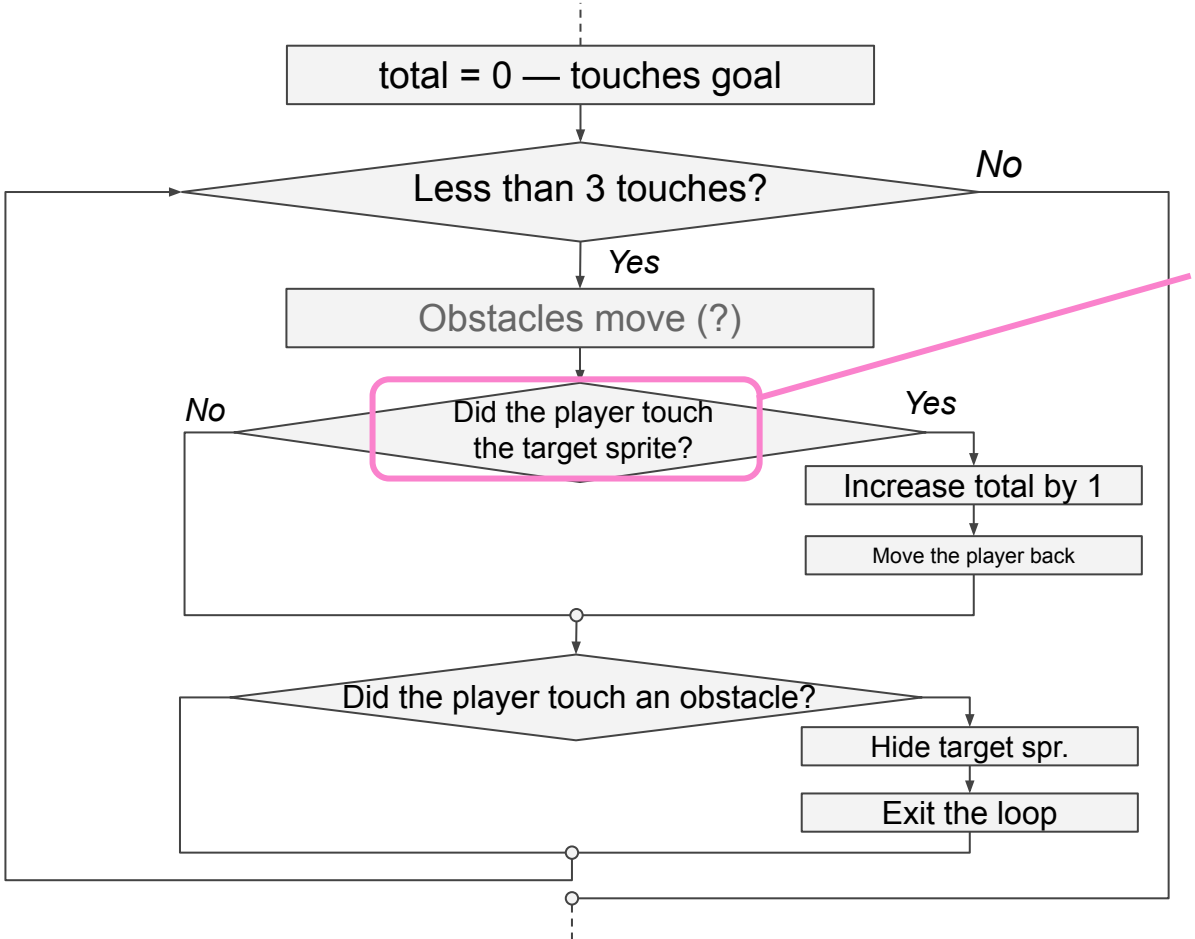
Program flow chart:



Let's make the game more complicated for the user! Let's say they need to 'catch' the target sprite no less than 3 times.

Brain storm

Program flow chart:



We will go over the method determining whether the sprite collisions have occurred.

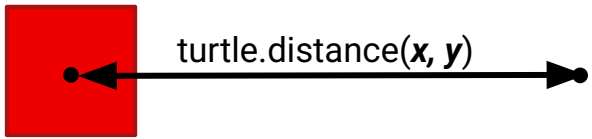
How can touching be detected?

Brain storm

Sprite touching

The method for the Turtle class can help determine the distance between two sprites.

Type	*Comment*
`dist = turtle.distance(x, y)`	The method returning the distance between the Turtle object and the point with the coordinates (x, y)



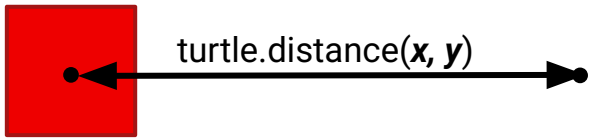
How can we use this method to find the distance between two sprites rather than a sprite and a point?

Brain storm

Sprite touching

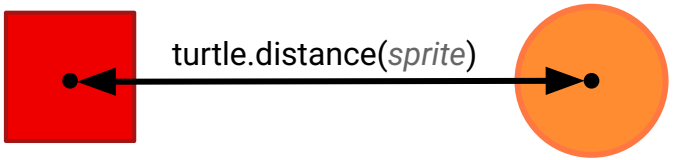
The method for the Turtle class can help determine the distance between two sprites.

Type	*Comment*
`dist = turtle.distance(x, y)`	The method returning the distance between the Turtle object and the point with the coordinates (x, y)




```
dist = t1.distance(t2.xcor(), t2.ycor())
```

We get the position of the second Turtle object by using the xcor() and ycor() methods.



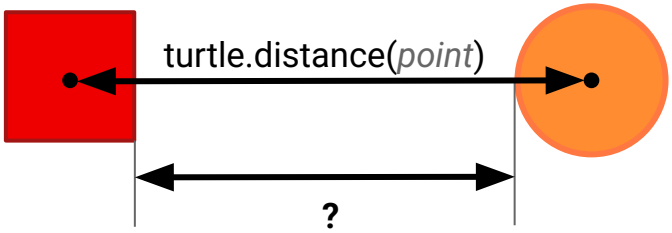
What should the distance be between sprites when they touch?

Brain storm

Sprite touching

The method for the Turtle class can help determine the distance between two sprites.

Type	*Comment*
`dist = turtle.distance(x, y)`	The method returning the distance between the Turtle object and the point with the coordinates (x, y)



We note that the method returns the distance between the centers.
A distance equal to 0 is too small!

A likely condition for the sprites touching: ***distance < 30*** .

Brain storm

Sprite touching

We will add the is_collide() method to the Sprite, which determines whether the sprites touched.

```
class Sprite(Turtle):
    The already implemented part of the class
    def is_collide(self, sprite):
        dist = self.distance(sprite.xcor(), sprite.ycor())
        if dist < 30:          <-- The plausible condition for touching
            return True
        else:
            return False

player.is_collide(goal)
```

Brain storm

Sprite touching

We will add the is_collide() method to the Sprite, which determines whether the sprites touched.

```
class Sprite(Turtle):
    The already implemented part of the class
    def is_collide(self, sprite):
        dist = self.distance(sprite.xcor(), sprite.ycor())
        if dist < 30:
            return True
        else:
            return False
```

```
player.is_collide(goal)
```

"Has the player touched the target?"

How do we use the method in the implementation of the game loop?

Brain storm

Main game loop

Part of the game loop can be implemented already.

```
total_score = 0

while total_score < 3:
    if player.is_collide(goal):
        player.goto(0, -100)
        total_score += 1
    if player.is_collide(enemy1) or player.is_collide(enemy2):
        goal.hideturtle()
        break

enemy1.hideturtle()
enemy2.hideturtle()
```

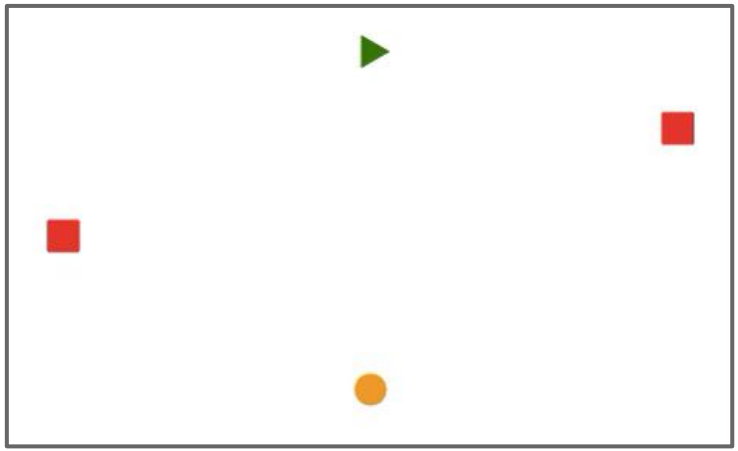
"If the player touched the target..."

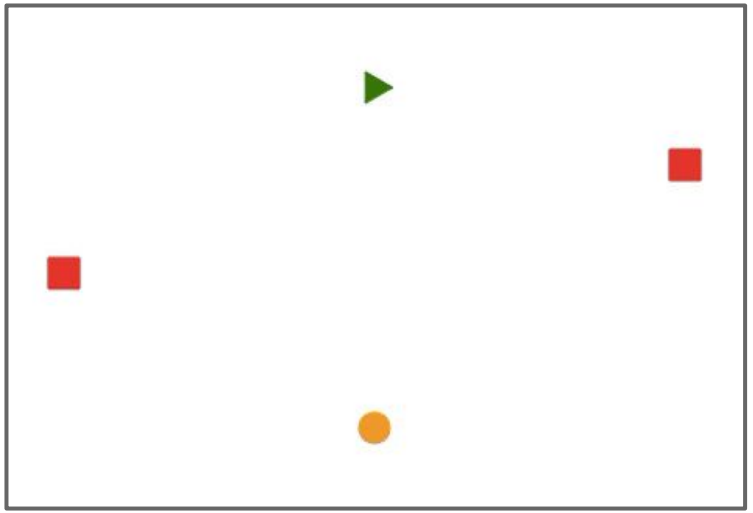
"If the player touched an obstacle..."

Brain storm

Expected look of the game

Games after addition of the game loop:

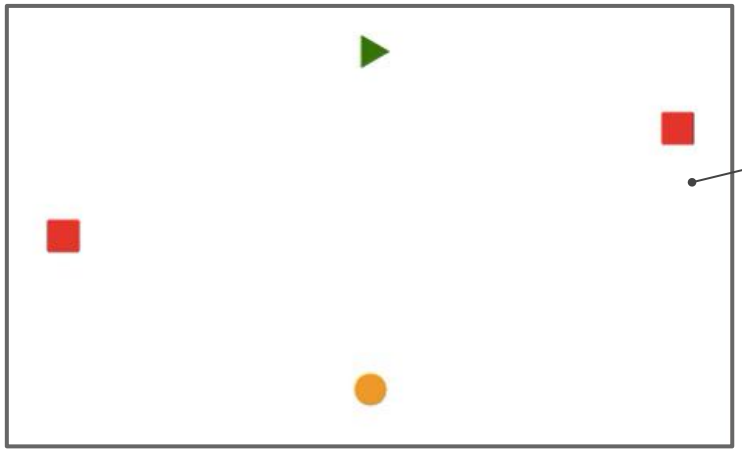




Brain storm

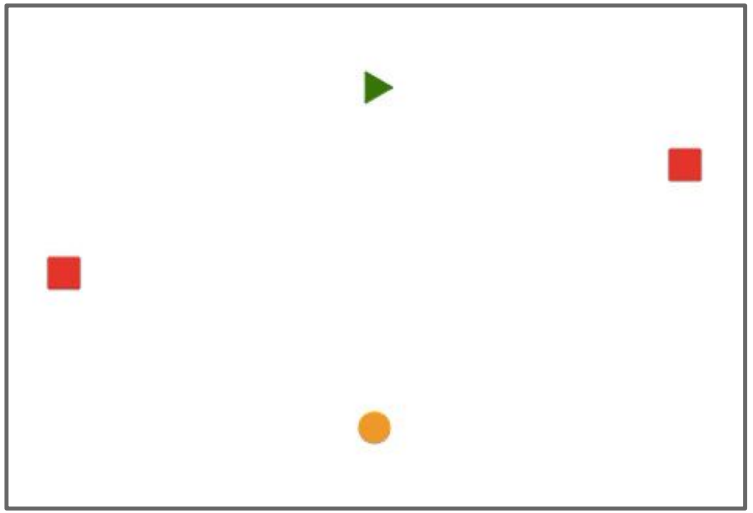
Expected look of the game

Games after addition of the game loop:



Processing events is done by using the Screen object.

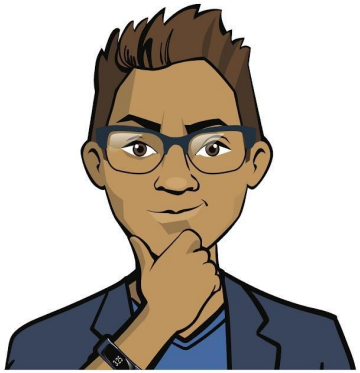
To "activate" the screen when launching the game, you need to click on it with the mouse.



Brain storm

Tasks:

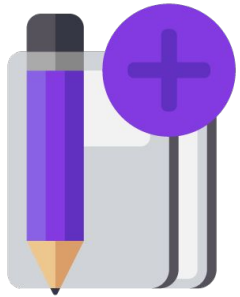
- ➔ Add the Sprite class code for the "Hit It!" game using the is_collide() method for processing sprite touching.
- ➔ Program the game loop and the reaction to touching the target sprite or an obstacle.



Brain storm

Brainstorm:

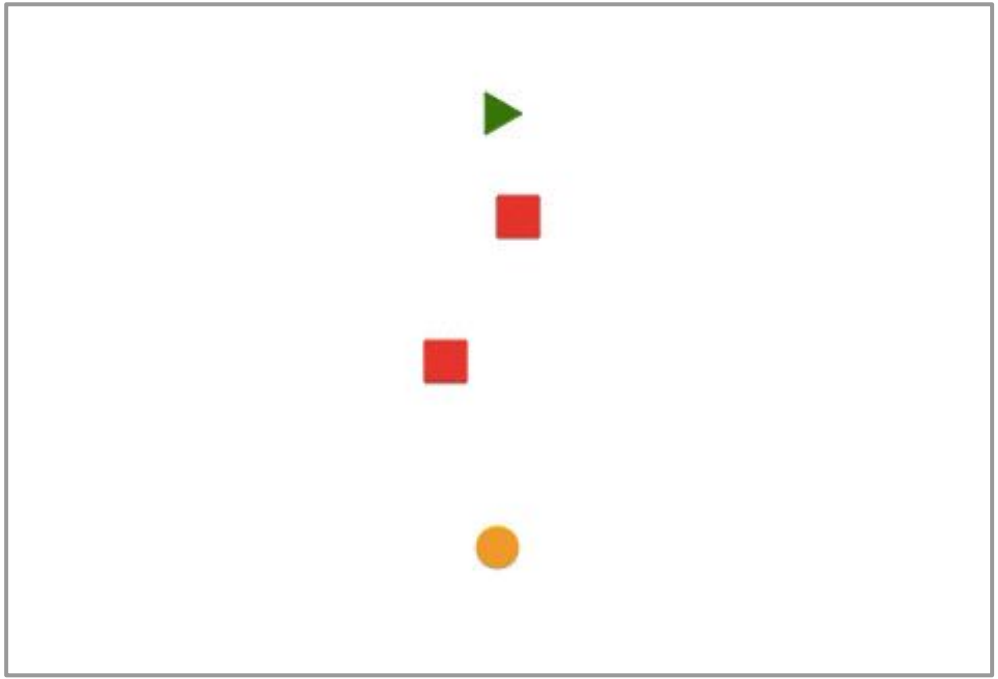
The "Hit It!" game



Movement of sprites

We now have to program the automatic movement of the obstacle sprites.

How can we do this?

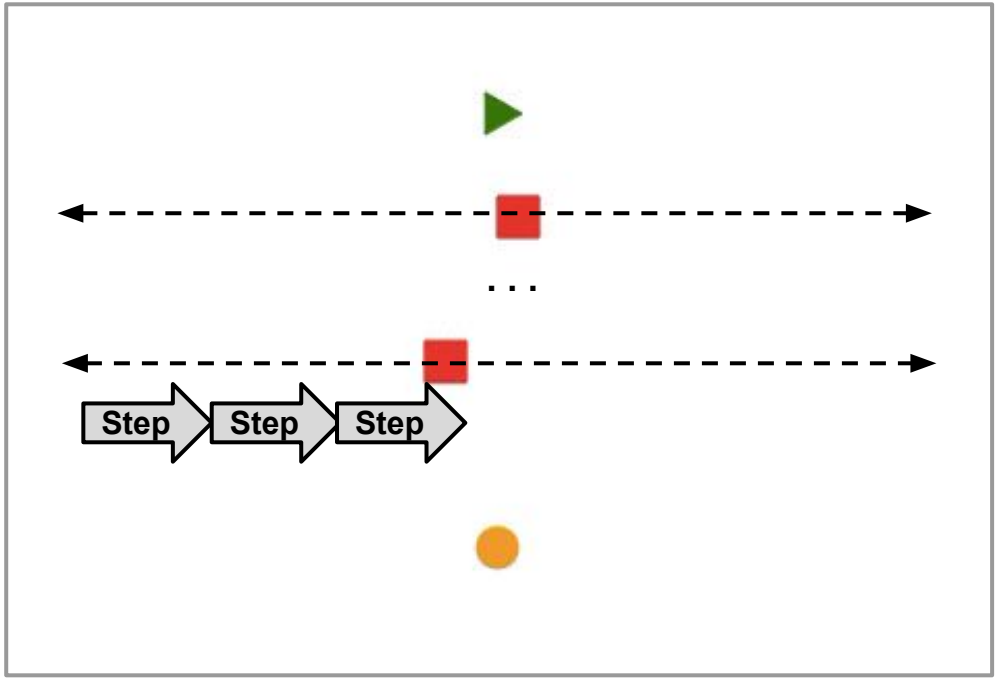


Brain storm

Movement of sprites

We now have to program the automatic movement of the obstacle sprites.

How can we do this?

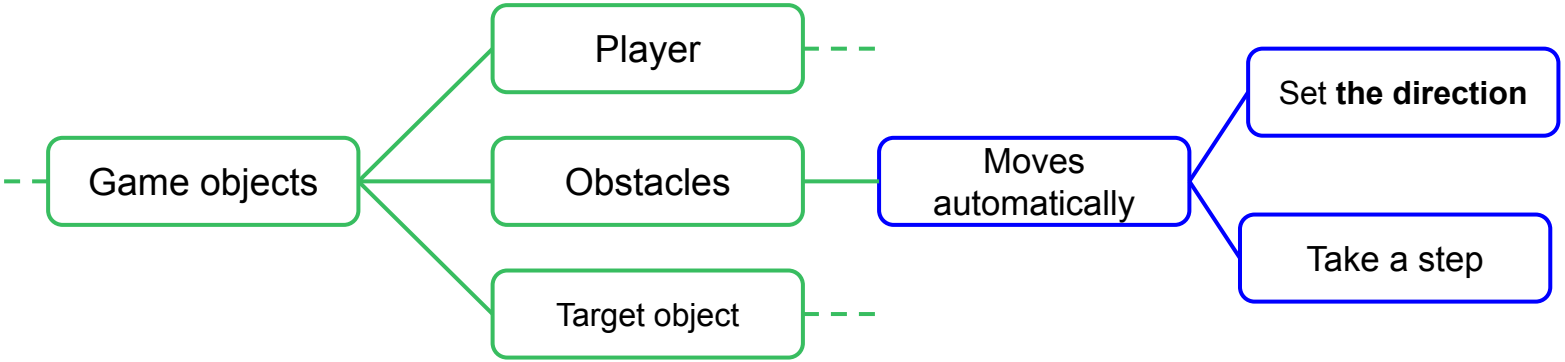


Brain storm

Movement of sprites

We now have to program the automatic movement of the obstacle sprites.

How can we do this?

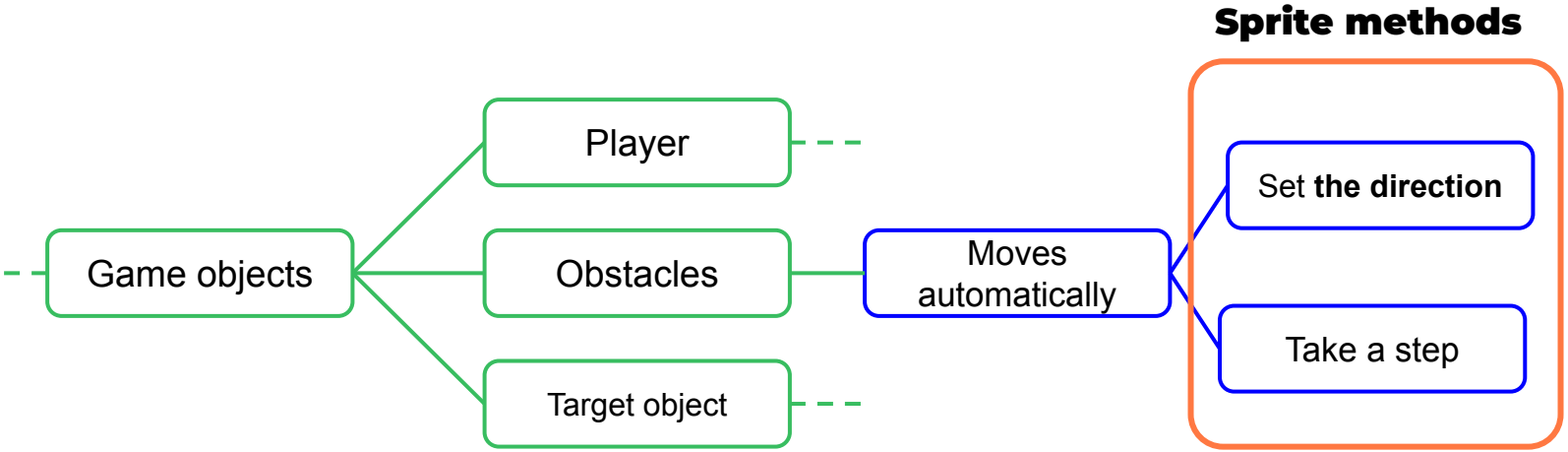


Brain storm

Movement of sprites

We now have to program the automatic movement of the obstacle sprites.

How can we do this?



Brain storm

1. The 'Set direction' method

Using the methods from the Turtle class:

- ➔ place the sprite in its movement starting position;
- ➔ set the direction of movement towards the end point.

Type	*Comment*
`angle = turtle.towards(x, y)`	The method returning the angle between the current direction of the turtle and the direction towards the specified point
`turtle.setheading(angle)`	The method which turns the turtle at the specified angle

The end point (the turning point)

Brain storm

1. The 'Set direction' method

Implement the method using the commands of the Turtle class.

The Sprite class:

```
def set_move(self, x_start, y_start, x_end, y_end):
```

We send the coordinates of the start and end points of the movement as parameters.

Brain storm

1. The 'Set direction' method

Implement the method using the commands of the Turtle class.

The Sprite class:

```
def set_move(self, x_start, y_start, x_end, y_end):
    self.x_start = x_start
    self.y_start = y_start
    self.x_end = x_end
    self.y_end = y_end
```

'Save' the coordinates as new properties of the Sprite class.

Brain storm

1. The 'Set direction' method

Implement the method using the commands of the Turtle class.

The Sprite class:

```
def set_move(self, x_start, y_start, x_end, y_end):
        self.x_start = x_start
        self.y_start = y_start
        self.x_end = x_end
        self.y_end = y_end
        self.goto(x_start, y_start)
        self.setheading(self.towards(x_end, y_end))
```

Move the sprite to the movement start point and set the direction towards the end point.

Brain storm

2. The "Take a step"' method

The length of the step was sent to the sprite when we created the Sprite instance.

The step is saved in the step field.

The Sprite class:

```
def make_step(self):

    self.forward(self.step)
```

Movement is implemented using the Turtle method.

Can a sprite always take a step in the required direction?

Brain storm

2. The "Take a step"' method

The length of the step was sent to the sprite when we created the Sprite instance.

The step is saved in the step field.

The Sprite class:

```
def make_step(self):

    self.forward(self.step)

    if self.distance(self.x_end, self.y_end) < self.step:

        self.set_move(self.x_end, self.y_end, self.x_start, self.y_start)
```

Steps can be taken until the edge of the screen is reached.

Brain storm

2. The "Take a step"' method

The length of the step was sent to the sprite when we created the Sprite instance.

The step is saved in the step field.

The Sprite class:

```
def make_step(self):

    self.forward(self.step)

    if self.distance(self.x_end, self.y_end) < self.step:

        self.set_move(self.x_end, self.y_end, self.x_start, self.y_start)
```

If less than one step remains until the end point of movement...

...we begin to move in the opposite direction.

Brain storm

Introducing the methods into the game code

The main program code:

```
enemy1 = Sprite(-200, 0, 15, 'square', 'red')
enemy1.set_move(-200, 0, 200, 0)
enemy2 = Sprite(200, 70, 15, 'square', 'red')
enemy2.set_move(200, 70, -200, 70)

#...

while total_score < 3:
    enemy1.make_step()
    enemy2.make_step()
    if player.is_collide(goal):
        #...
```

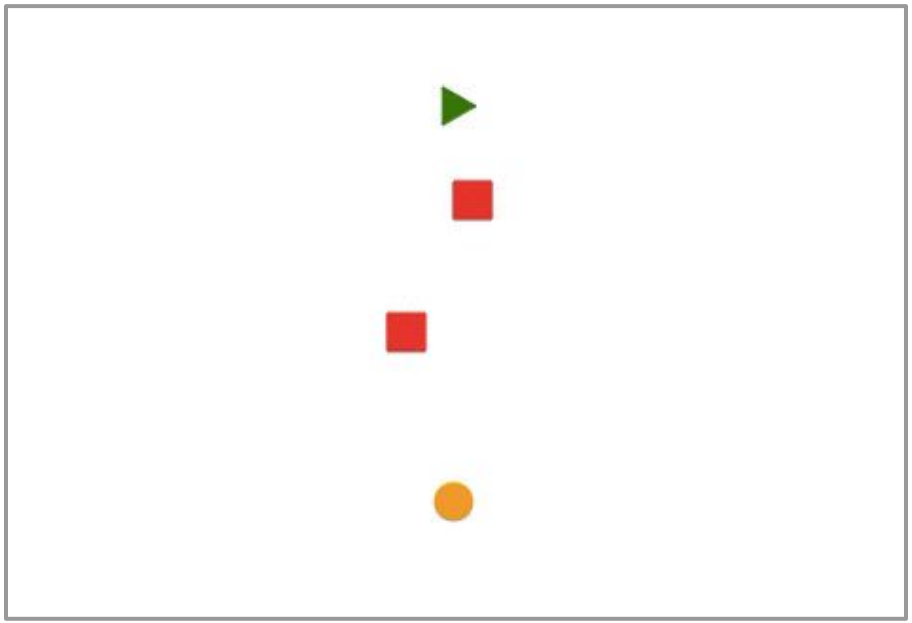
Along with creating obstacles, we set the direction of movement.

We move the sprites in the required direction at every stage of the loop.

Brain storm

Tasks:

➔ Complete the "Hit It!" game by adding the code for making the obstacles move right and left.



Brain storm