# Programming Fundamentals I
# Lab 7 - If Statements Part II

In this lab we are going to continue practicing *if* statements while improving on our first-person shooter game. We will also introduce new concepts. Here are the new additions that we are going to program in this lab:

- The ability to shoot using the *spacebar* button
- Reload ammo using the *r* button
- Determine whether a shot hits a target
- Update the score when a target is hit
- The ability of the game to have random choices

EXERCISE:

1. Which of the above additions require the use of an *if* statement?
2. What are the steps needed to introduce the shooting capability to our game?

## SHOOTING & RELOADING:

Let us start with the gun shooting when the *spacebar* key is pressed. We first introduce a new property to the *gun* Actor by adding the following line of code after line 15:

```
fps.fired = False
```

Then, we add the following to the end of the `on_key_down(key)` function:

```
elif key == keys.SPACE:
        fps.fired = True
```

Finally, we add the following instructions to the end of the `update()` function:

```
if fps.fired:
        fps.fired = False
        fps.ammo -= 1
        fps.image = 'explosion.png'
        clock.schedule(resetImage, 0.2)
```

We first make sure to set the *fired* property to *False*. Then, we reduce the ammo by one. We also change the image of our first-person shooter from crosshair to explosion. The last instruction is our first example of a timed instruction. This instruction will call the `resetImage()` function after 0.2 seconds have passed. The `resetImage()` function changes the image of *fps* back to crosshair.

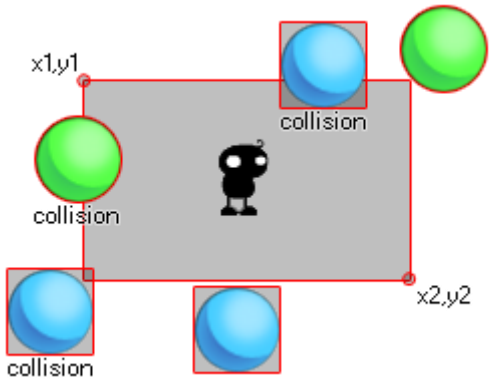
## EXERCISES:

1. Did we introduce any new flag variable in this lab?
2. Currently our game allows us to fire more than 5 times. This can be seen when the Ammo value on the screen becomes negative. Modify your program so that the gun cannot fire more when the ammo becomes zero.
3. Modify the program so the game reloads the ammo to 5 when the player presses the *r* button.

## COLLISION OF ACTORS:

Figuring out whether one actor collides with another actor is an important part of game development. The use of collision detection allows the programmer to determine several game actions as: the player hitting an enemy, two cars colliding, or in our game determining whether a shot hits a target. There are several ways to determine whether two actors collide with (hit) each other. The simplest method is based on rectangles, where each actor will have a rectangle to enclose it. A collision of two actors is detected if their rectangles intersect. Every actor we add to our game has a behavior (function) that can be used to determine whether it collides with another actor. For example, to determine whether the *fps* actor (the crosshair) collides with a target the following instruction:

```
fps.colliderect(target)
```




will return *True* if fps's rectangle intersects with the target's rectangle, otherwise it will return *False*. With this behavior, we can make a target get destroyed when shot. In order to do that we need to introduce a new flag variable by inserting the following instruction after line 20:

```
target.visible = True
```

This means that initially the target should be displayed on the screen. Next, we need to draw the target only when the *visible* flag is set to *True*. To do that we need to replace in the `draw()` function the following instruction `target.draw()` with:

```
if target.visible:
   target.draw()
```

Finally, insert the following instruction before the `clock.schedule(resetImage, 0.2)` instruction in the `update()` function:

```
if fps.colliderect(target):
            target.visible = False
```

## EXERCISES:

1. Update the score when the target is hit. The target has a property called *value*. Use this property to update the score.
2. Currently, and after a target is destroyed, the score increases if the area that the target occupied is shot again. Fix the code to prevent this situation.

## INTRODUCING RANDOMNESS TO THE GAME:

The ability to choose between two or more options is a human quality that we can program in a game using two types of instructions:

- Nested *if* statements
- Random number generator

We should be now familiar with how *if* statements work, and their need of a Boolean expression to implement a decision. Let us see how we implement choice using Boolean expression. A Boolean expression can either be:

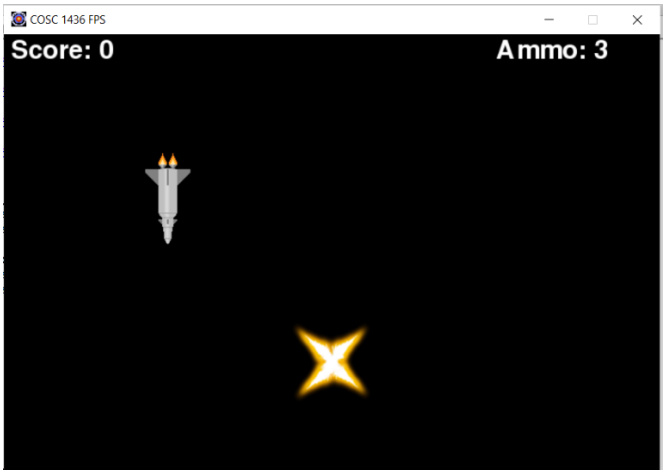
- Boolean value
- Comparison of two values using a relational operator
- One or more Boolean expressions combined using *and*, *or*, *not*

So how do we represent choice using a Boolean expression? First, we need to learn more how we can generate a random number in Python. Then, we can use this generated random number and compare it with a value of our choice. Python provides us with the following instruction:

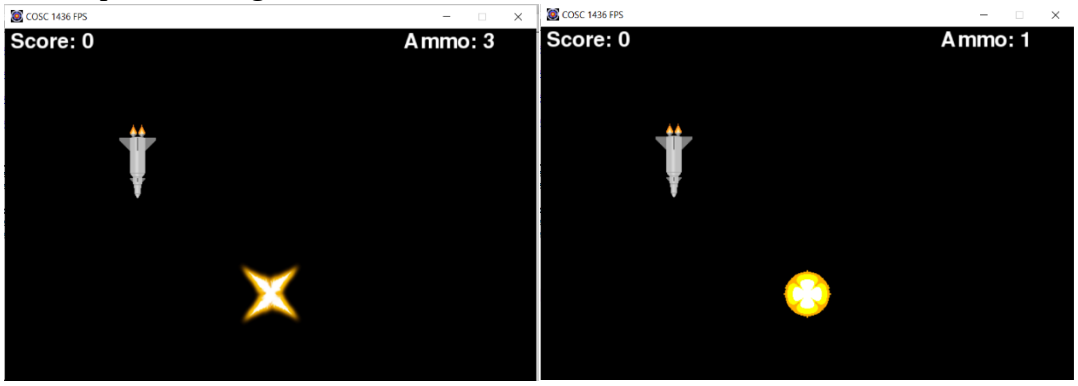
```
randomNumber = randint(1, 52)
```

Here, we are generating a random number between 1 and 52 (inclusive) and saving this number into a variable called randomNumber. This is like asking a person to randomly pick a card from a deck of cards.

Let us introduce the capability of random choice into our game. Until now whenever we fire a shot we get the same explosion, this can be seen in the figure below:




In real life we do not usually get the same explosion. Let us do something similar in our game. Our program will sometimes use one image of an explosion, and in other times the program will use another explosion image:




To do that, we want our program to randomly choose between the two explosion images: *explosion* or *explosion2*. First, add the instruction below after line 1:

```
import random
```

Then, we want our program to randomly pick between two choices by generating a random number between 1 and 2:

```
choice = random.randint(1, 2)
```

EXERCISE:

1. Replace the following instruction from the `update()` function:

   ```
   fps.image = 'explosion.png'
   ```

   With the code snippet below, and then complete it:

```
choice = random.randint(1, 2)
if choice == ----------:
     fps.image = ----------
else:
     ----------
```

(A) (B) (C)

2. How do we modify our code so it favors one type of explosion image (explosion.png) over the other one?