Space Invaders

◎ Work tips

- the point is to do personal work, even if you work together and discuss
- all useful notions will be given in class or in the document
- read the questions, most the answers are in them
- ask questions 😄

Context

The **Space Invaders** game is considered to be one of the most influential games of all time. It started the *golden age* of arcade-style video games.

The point of the game is to control a small spaceship that has to fight waves of aliens. The aliens approach at small speed, and the player can shoot bullets to destroy them. If the aliens get to the bottom of the screen, the the game is lost.

The game can mostly be played as an arcade game, meaning unlimited play as long as the aliens do not get to the bottom of the screen.

Idea

Do not forget that the code template gives you a lot of information on where to write the elements for the different steps. Use that to your advantage.

Our first version will look something like this:

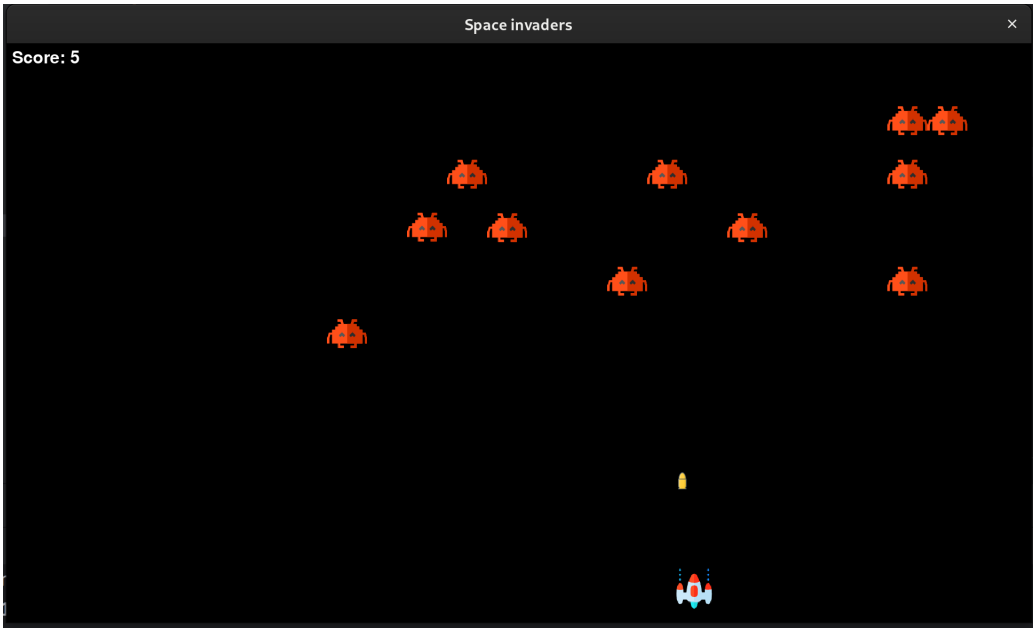

Figure 1: First version of our **Space Invaders** game

Step 1. Base variables and setting up the window

As usual, we will start by creating a few variables to hold the base values for our game construction. You will need:

- A **unit**, which we will set to `1`
- A **window_width** and a **window_height**. They will be set to `1920` units and `1080` units respectively[1].
- A **background_colour**, set it to black for the moment.
- A **player_base_speed** equal to `10` units.
- A **player_base_scale** equal to `75`.
- An **enemy_base_speed** equal to `1` units.
- An **enemy_base_scale** equal to `75`.
- A **bullet_base_speed** equal to `30` units.

We now have to set up the window our game will play in. To do this, you have to update the background colour in the main loop, and correctly set the size information at the screen creation.

Step 2. The player

In **Space Invader**-type games, the player is represented by a spaceship. The player is able to move left and right as well as shoot a cannon.

In this step, we create the necessary information to hold the player, and we draw it on the screen.

1. Create a dictionnary called **player**. It will hold the following information:
    - **score**, set at `0`
    - **x**, set at `half the window width`
    - **y**, set at `window height minus 100 units`
    - **speed**, set at `player_base_speed`
    - **change**, set at `0`
    - **size**, set as a tuple `(player_base_scale*unit, player_base_scale*unit)`
2. Create a dictionnary called **game**. It will hold all the necessary information for the game. Add a key `"player"` that holds the `player` dict.

[1] Check you screen. If the window does not show up, try a smaller resolution such as `1366x768` or `1600x900`

Idea

Instead of drawing the player's spaceship with pygame methods (circles and rectangles), we are going to load a **sprite**.

Take a couple of instants to understand how the **draw_player** function works. How it loads the image, and what it does to it.

3. Call the **draw_player** function with the correct parameter to draw the player on the screen.

Information

You should be able to run the script and see the window with your spaceship in it.

Step 3. Moving the player

Having a spaceship is nice, but it's better if we can move it. The movement in this game is **1-D**, meaning we will only be moving on the x-axis.

We will move the spaceship by using the **left** and **right** arrows on the keyboard. We want to be able to move as long as the key is pressed, and stop when the key is not pressed anymore.

Idea

If `pygame.KEYDOWN` is the event that is triggered when a key is pressed, what do you think is the event triggered when the key is released?

- Handle the press on a key. If an arrow key is pressed, update the `"change"` field in the player dictionnary by the correct value (± the speed).
- Handle the release of an arrow key. The `"change"` field should be reset to `0`.
- Uncomment the line that makes the player move by the value of the `"change"` field.
- Handle the edge cases: the player should not be able to be outside the window. If the player's **x**-coordinate is less than `0`, set it to `0`. Find the second condition by trying[2].

Information

The player should be able to move from side to side of the window now. Keeping the key pressed should continue the movement, while releasing it stops the player.

The player sprite should not be able to go outside of the game area.

[2] don't forget that coordinates are computed from the top-left corner. This also applies to sprites.

Step 4. Creating the enemies

In order to make the game interesting, we need to have enemies. We will start by setting up everything we need to draw the enemies, then we will add their movement.

Enemies will be represented by dictionnaries that contain all the necessary information, just like the player. The fields you will have in the dictionnary are: **x**, **y**, **size** and **speed**.

1. Write a function called **draw_enemy** that takes a dict as parameter and draws it at the correct position on the screen. Use the **draw_player** function as an example. Find the correct sprite ☺
2. Write a **draw_enemy_list** function that takes a list of enemies and draws them all.

This allows us to draw the aliens, we now have to build them. To do this, we will generate a random number of enemies at random positions on a given line.

3. Write a function called **get_random_position** that returns a tuple of two values. The first value should be a random x-coordinate, the second value should be equal to the base scale of the enemies.
4. Write a **get_number_of_enemies** function that returns a random number between `0` and `5`.
5. Write a **create_enemies** function that takes a number **nb** as a parameter and returns a list of nb enemies. When creating an enemy, the position should be generated by a function, the size is computed in a similar way as for the player and the speed is the enemy base speed.
6. Create a list of enemies and add it to our **game** dictionnary.
7. Draw the enemies inside the **game_turn** function.

Information

You should be able to see a row of little aliens at the top of your screen at the moment.

Step 5. Moving and waves

We will now make the enemies move towards the bottom of the screen. We will also add multiple waves to keep the enemies coming.

To make the enemies move, you have to loop over the list at each game turn and update their y-coordinate by their speed.

We will add new enemies once the current line is far enough. To do this:

- Create a `"next_wave"` field in the game dictionnary. Set its value to `2 ticks`.
- At each game loop, decrease this value by `1`.

- If the value is `0`, set it back to `2 ticks` and append to the enemy list new random enemies. Use **get_number_of_enemies** and **create_enemies**.

Step 6. Building a bullet

The spaceship has to be able to fire bullets to destroy the incoming aliens. We will start by setting up everything we need. The bullet will be represented by a dictionnary, which will contain the following fields: **x**, **y**, **speed** and **fired**.

1. Create a bullet. The speed is `30 units`, x and y can be anything and the fired status is `False`. Add it to the game dictionnary.
2. Write a **draw_bullet** function that takes a bullet as parameter and draws it. Use the other drawing functions as examples.
3. Draw the bullet **if it is fired** at each game turn.

Information

You should be able to see a fired bullet if you set the correct fields correctly.

Step 7. Firing a bullet

We limit the rate of fire to one bullet at a time. This means you cannot fire until the bullet has exited the window or hit an enemy.

1. Write a **fire_bullet** function. The function takes a bullet and the player as parameters. If the bullet is not currently fired, then set the state to `True`, as well as the coordinates to be the player's coordinates.
2. Look at the **hits** function. Update the return line to return `True` if the distance is smaller than the enemy's size.
3. Write a **out_of_bounds** function that takes a bullet as parameter and returns `True` if the bullet is outside the game area (hint: think of the direction of travel, do you have to check all the possibilities?)

Now, inside the **game_turn** function:

4. Handle the pressing of the `Space` key. When pressed, fire the bullet.
5. If the bullet is fired, move it by updating the y coordinate with its speed. The bullet should go up.
6. If fired, check whether it is out of bounds. If so, reset the state of the `"fired"` field to `False`.
7. If the bullet is fired, check whether it hits an enemy. If so, reset the bullet, add `1` to the player score, and remove the enemy from the list.

Step 8. Game over

The game is considered to be lost if an enemy reaches an imaginary line that we set in the window.

We will consider this line to be positioned at `3 player sizes` over the bottom of the window.

Add the relevant code to handle this.

More & Image sources

Once you finished the game, please feel free to make it better by changing the gameplay, the images, the sound, etc.

Information

Do not forget to use images you are allowed to use and credit the authors. The current images are sourced from:

- `https://www.flaticon.com/free-icon/ufo_214358`
- `https://www.flaticon.com/free-icon/space-invaders_744737`
- `https://www.freepng.fr/png-drz5lj`
- `https://www.flaticon.com/free-icon/bullet_224681`

You are also welcome to create your own images / animations!