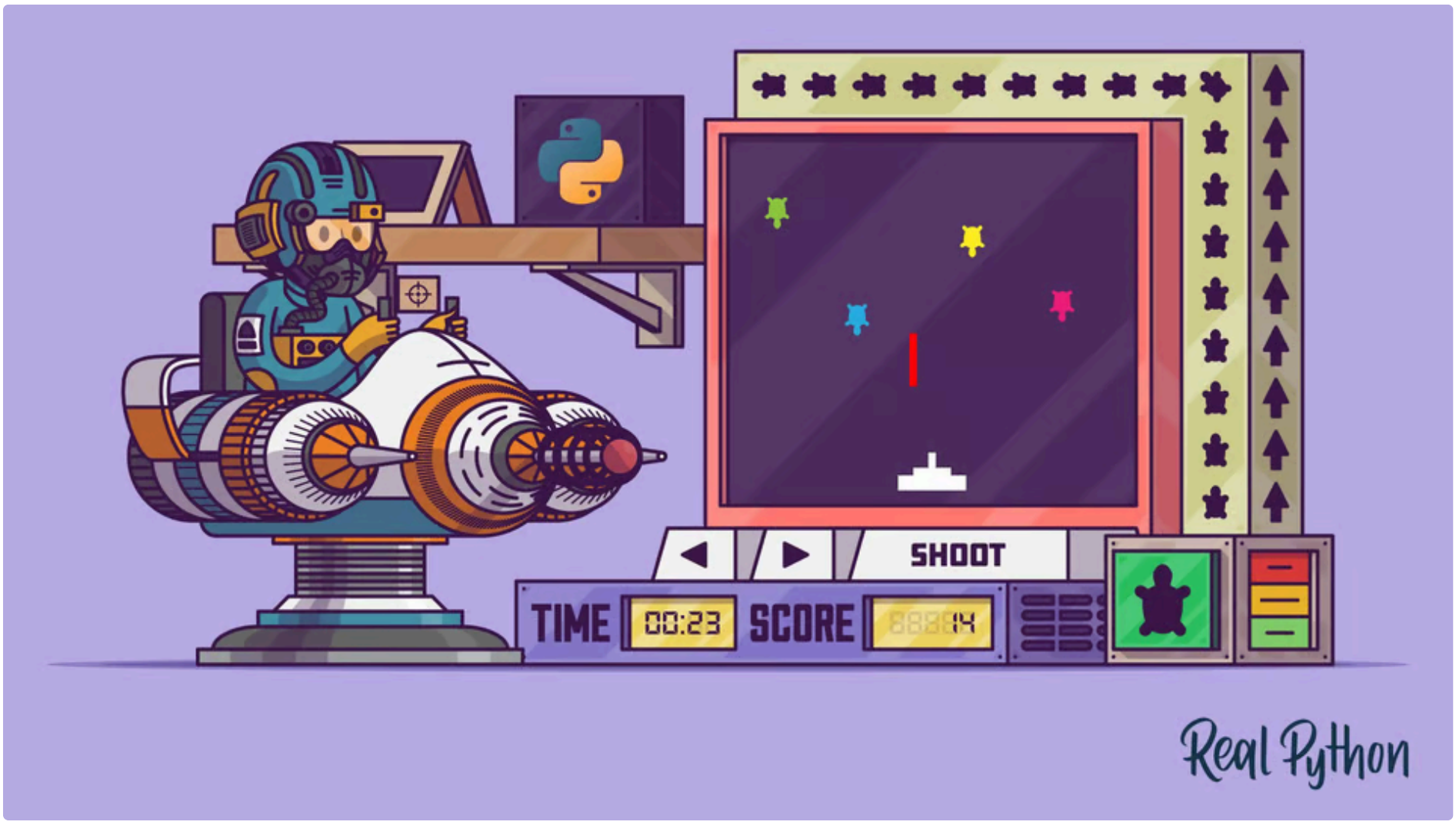


Build a Python Turtle Game: Space Invaders Clone

by Stephen Gruppetta Mar 20, 2024 2 Comments basics python

Mark as Completed

Share Share Email

Table of Contents

Help

In this tutorial, you'll use Python's `turtle` module to build a Space Invaders clone. The game Space Invaders doesn't need any introduction. The original game was released in 1978 and is one of the most recognized video games of all time. It undeniably defined its own video game genre. In this tutorial, you'll create a basic clone of this game.

The `turtle` module you'll use to build the game is part of Python's standard library, and it enables you to draw and move sprites on the screen. The `turtle` module is not a game-development package, but it gives instructions about creating a *`turtle` game*, which will help you understand how video games are built.

In this tutorial, you'll learn how to:

- **Design and build** a classic video game
- Use the `turtle` module to **create animated sprites**
- Add **user interaction** in a graphics-based program
- **Create a game loop** to control each frame of the game
- **Use functions** to represent key actions in the game

This tutorial is ideal for anyone who is familiar with the core Python topics and wants to use them to build a classic video game from scratch. You don't need to be familiar with the `turtle` module to work through this tutorial. You can download the code for each step by clicking on the link below:

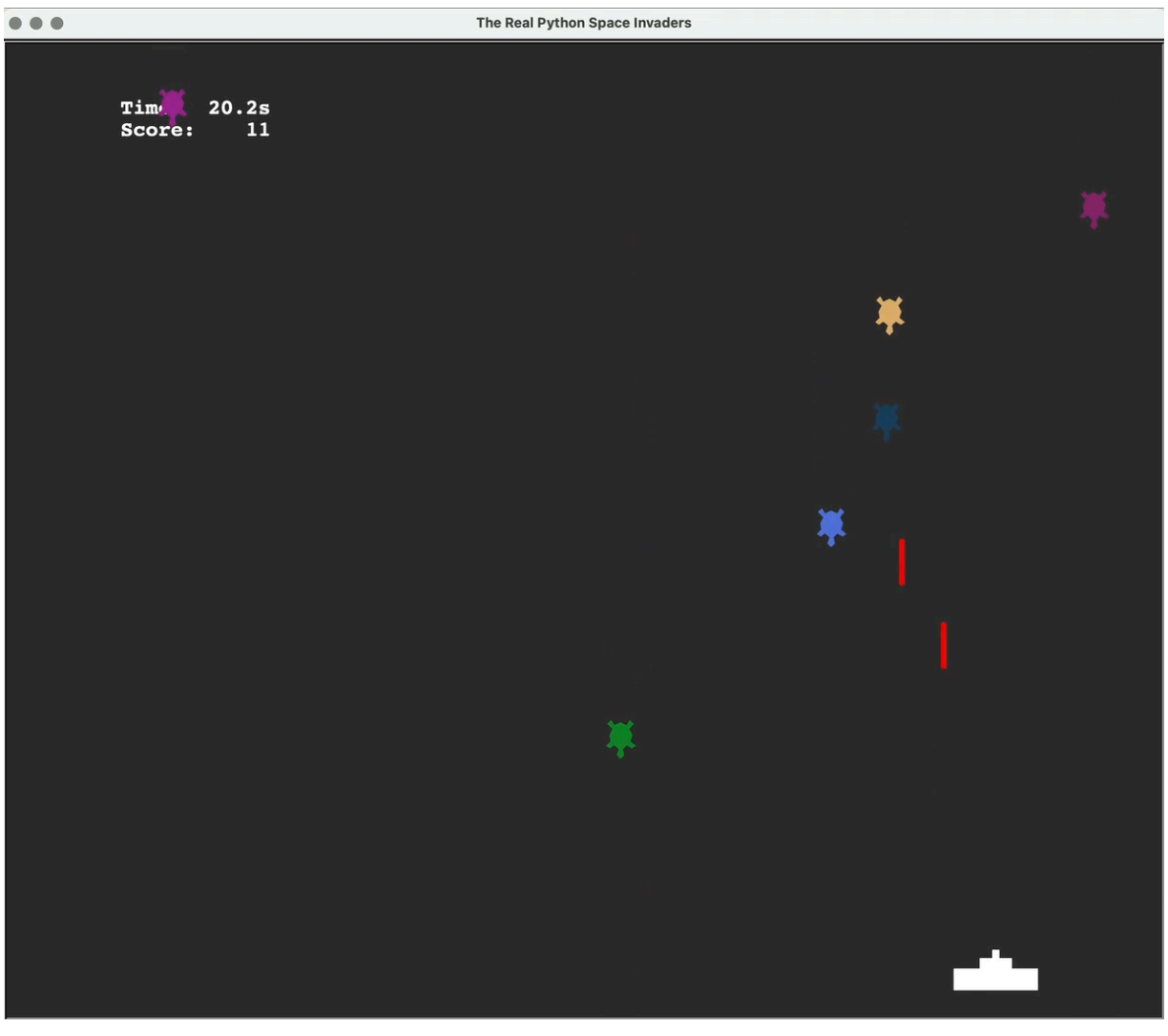
Get Your Code: **Click here to download the free sample code** that shows you how to build a Python turtle game.

In the next section, you can have a look at the version of the game you'll build as you follow the steps outlined in this tutorial.

Demo: A Python Turtle Space Invaders Game

You'll build a simplified version of the classic Space Invaders game and control the laser cannon with the keys on your keyboard. You'll shoot lasers from the cannon by pressing the spacebar, and aliens will appear at regular intervals at the top of the screen and move downwards. Your task is to shoot the aliens before they reach the bottom of the screen. The game ends when one alien reaches the bottom.

This is what your `turtle` game will look like when you complete this tutorial:



Here you can see the main game play for this game, as the laser cannon moves back and forth and shoots the falling aliens. The game also displays the elapsed time and the number of aliens shot down on the screen.

Project Overview

In this project, you'll start by creating the screen that will contain the game. In each step, you'll create game components such as the laser cannon, lasers, and aliens, and you'll add the features required to make a functioning game.

To create this `turtle` game, you'll work through the following steps:

1. Create the game **screen** and the **laser cannon**
2. **Move the cannon** left and right using keys
3. **Shoot lasers** with the spacebar
4. **Create aliens** and move them towards the bottom of the screen
5. Determine when **a laser hits an alien**
6. **End the game** when an alien reaches the bottom
7. Add a **timer** and a **score**
8. **Improve the cannon's movement** to make the game smoother
9. Set the game's **frame rate**

You'll start with a blank screen, and then see the game come to life one feature at a time as you work through each step in this tutorial.

Prerequisites

To complete this tutorial, you should be comfortable with the following concepts:

- Repeating code using `for` loops and `while` loops
- Using `if` statements to control what happens in different conditions
- Defining functions to encapsulate code
- Using lists to store multiple items

You don't need to be familiar with Python's `turtle` to start this tutorial. However, you can read an overview of the `turtle` module to find out more about the basics.

If you don't have all of the prerequisite knowledge before you start, that's okay! In fact, you might learn more by going ahead and getting started! You can always stop and review the resources linked here if you get stuck.

Step 1: Set Up the Turtle Game With a Screen and a Laser Cannon

You can't have a game without a screen where all the action happens. So, the first step is to create a blank screen. Then, you can add sprites to represent the items in the game. In this project, you can run your code at any point to see the game in its current state.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_1/` in the link below:

Get Your Code: Click here to download the free sample code that shows you how to build a Python turtle game.

It's time to take the first step in setting up your game with the `turtle` module and create the screen.

Create the Screen

To start, create a new file and import the `turtle` module:

Python `turtle_invaders.py`

```
import turtle

turtle.done()
```

You call `turtle.done()`, which displays the window and keeps it open. This line should always be the last line in any program that uses `turtle`. This will prevent the program from terminating when it reaches the last line. To stop the program, you close the window by clicking the *close window* icon in the upper corner of the window, just as you would with any other window in your operating system.

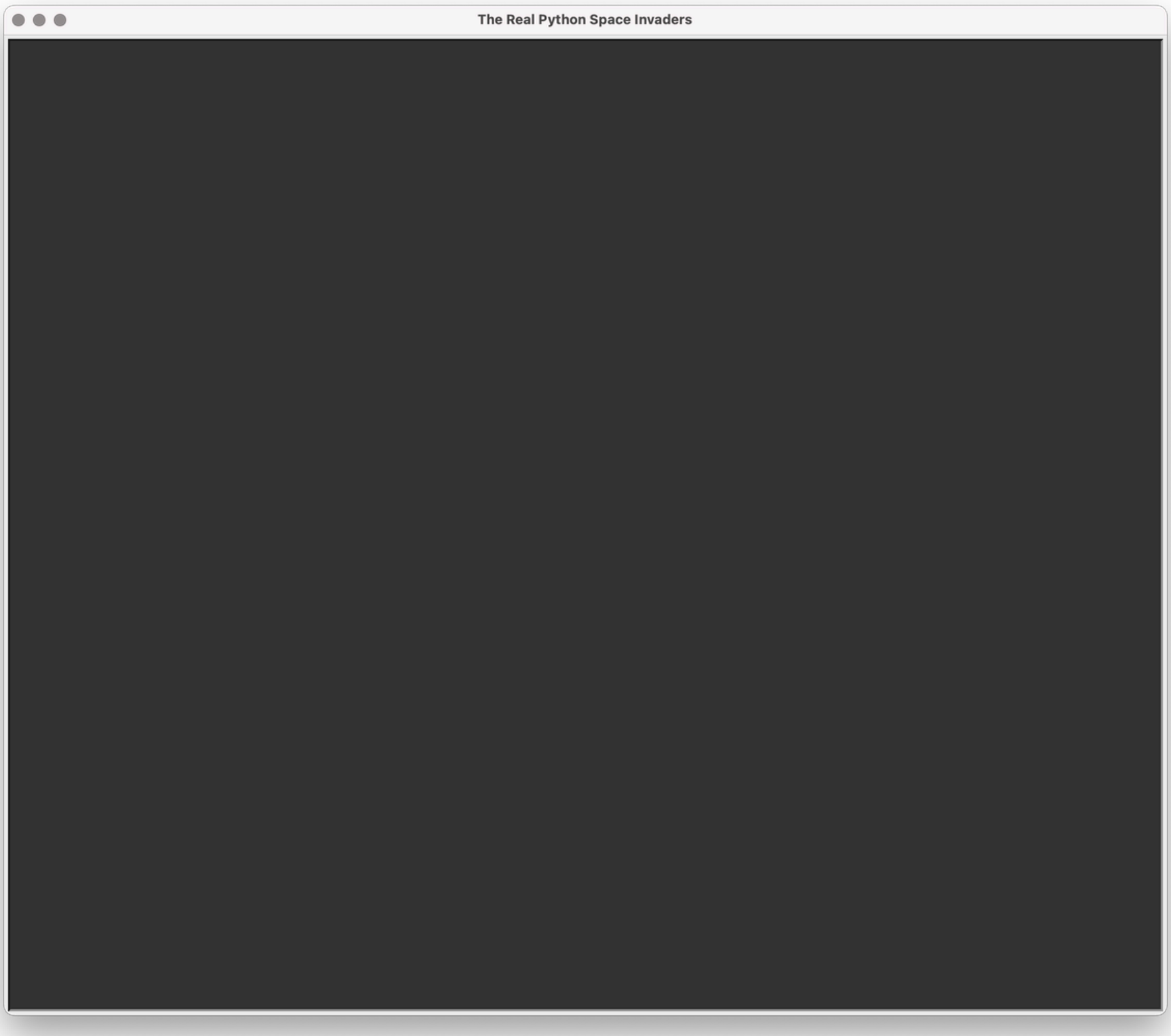
Although this code creates and displays a screen, you can explicitly create the screen in your code and assign it to a variable name. This approach lets you customize the screen's size and color:

Python `turtle_invaders.py`

```
import turtle

window = turtle.Screen()
window.setup(0.5, 0.75)
window.bgcolor(0.2, 0.2, 0.2)
window.title("The Real Python Space Invaders")

turtle.done()
```

You use floats as arguments for `.setup()`, which sets the screen size as a fraction of your computer screen. The window's width is 50 percent of your screen's width, and its height is 75 percent of your screen's height. You can experiment with using pixel values to set the screen dimensions by using integers instead of floats as arguments in `.setup()`.

The default color mode in the `turtle` module requires the red, green, and blue components to be floats in the range [0, 1]. You should also update the text in the title bar. When you run this code, you'll see this screen:



This version has a dark background but you can customize your game with your preferred colors. Now, you're ready to create the laser cannon.

Create the Laser Cannon

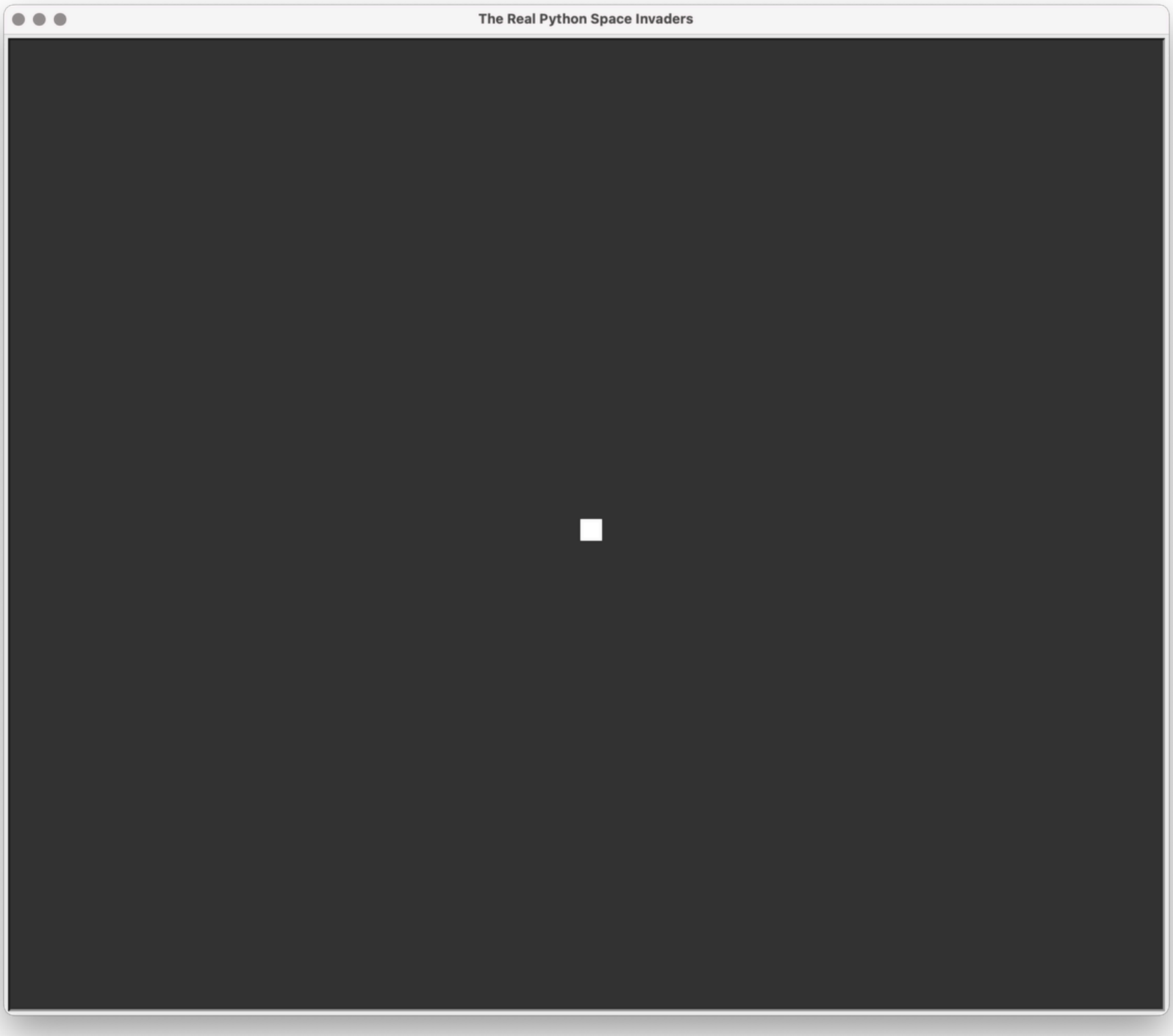
Just like you can't have a game without a screen, you can't have a space invaders clone game without a laser cannon. To set up the cannon, you create a `Turtle` object using `turtle.Turtle()`. Each independent item in the game needs its own `Turtle` object.

When you create a `Turtle` object, it's located at the center of the screen and faces right. The default image is an arrow showing the object's heading. You can change these values using `Turtle` methods:

Python turtle_invaders.py

```
import turtle

window = turtle.Screen()
window.setup(0.5, 0.75)
window.bgcolor(0.2, 0.2, 0.2)
window.title("The Real Python Space Invaders")

# Create laser cannon
cannon = turtle.Turtle()
cannon.penup()
cannon.color(1, 1, 1)
cannon.shape("square")

turtle.done()
```

The cannon is now a white square. You call `cannon.penup()` so that the `Turtle` object doesn't draw a line when you move it. You can see the cannon when you run the code:



However, you want the cannon to be at the bottom of the screen. Here's how you set the cannon's new position:

Python `turtle_invaders.py`

```
import turtle

window = turtle.Screen()
window.setup(0.5, 0.75)
window.bgcolor(0.2, 0.2, 0.2)
window.title("The Real Python Space Invaders")

LEFT = -window.window_width() / 2
RIGHT = window.window_width() / 2
TOP = window.window_height() / 2
BOTTOM = -window.window_height() / 2
FLOOR_LEVEL = 0.9 * BOTTOM

# Create laser cannon
cannon = turtle.Turtle()
cannon.penup()
cannon.color(1, 1, 1)
cannon.shape("square")
cannon.setposition(0, FLOOR_LEVEL)

turtle.done()
```

You define the edges of the screen using `window.window_height()` and `window.window_width()` since these values will be different on different computer setups. The center of the screen has the coordinates `(0, 0)`. So, you divide the width and height by two, to get the *x*- and *y*-coordinates for the four edges.

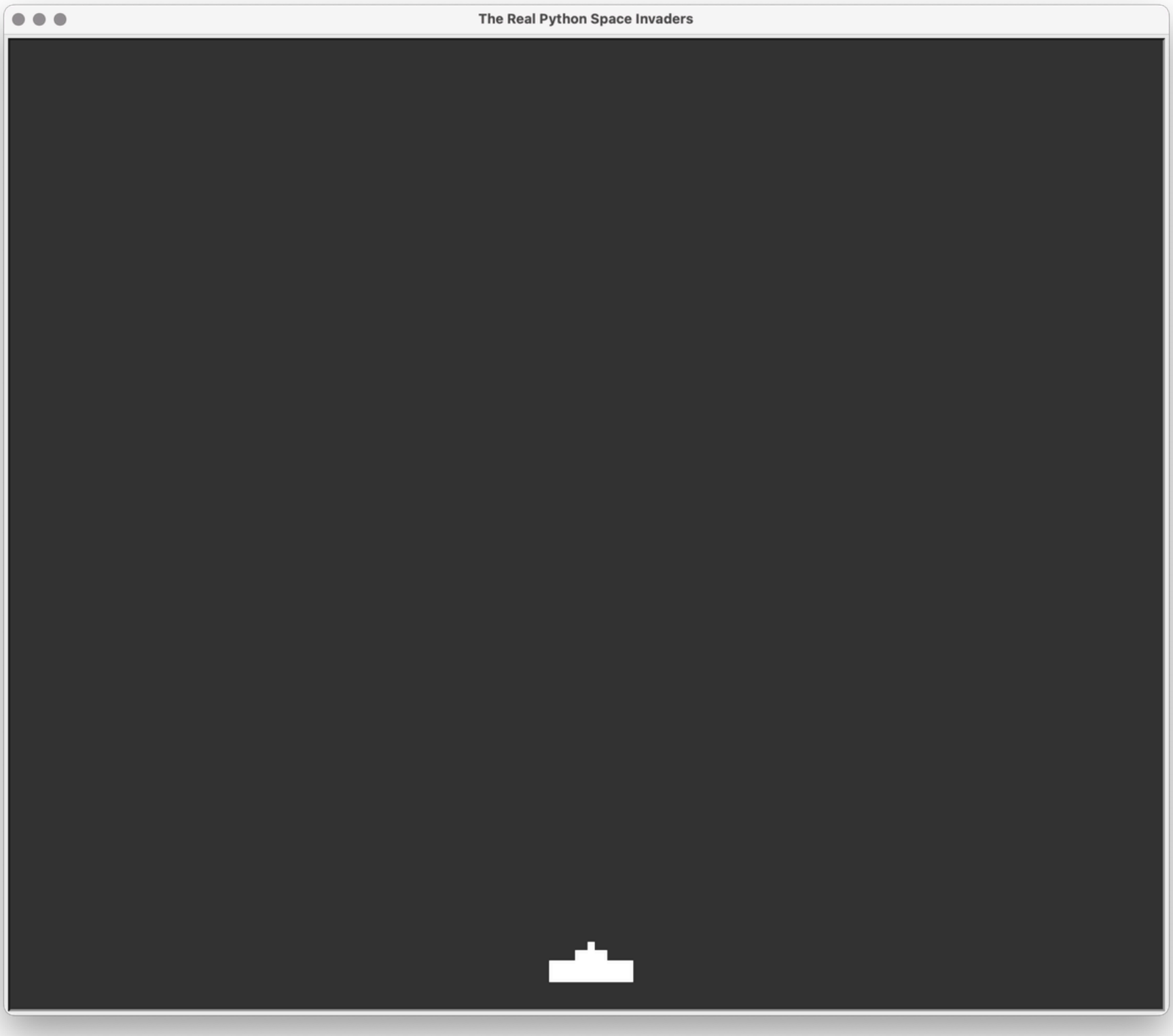
You also define `FLOOR_LEVEL` as 90 percent of the *y*-coordinate of the bottom edge. Then, place the cannon at this level using `.setposition()` so that it's raised slightly from the bottom of the screen.

Remember, this is a clone of a 1970s game, so all you need are simple graphics! You can draw the cannon by stretching the shape of the turtle into rectangles of different sizes using `.turtlesize()` and calling `.stamp()` to leave a copy of this shape on the screen. The code below only shows the section of the code that has changed:

Python turtle_invaders.py

```
# ...

# Draw cannon
cannon.turtlesize(1, 4)  # Base
cannon.stamp()
cannon.sety(FLOOR_LEVEL + 10)
cannon.turtlesize(1, 1.5)  # Next tier
cannon.stamp()
cannon.sety(FLOOR_LEVEL + 20)
cannon.turtlesize(0.8, 0.3)  # Tip of cannon
cannon.stamp()
cannon.sety(FLOOR_LEVEL)

turtle.done()
```

When you run this code, you'll see the cannon at the bottom of the screen:



Your next task is to bring in some interactivity so you can move the cannon left and right.

Step 2: Move the Cannon Using Keys

In the previous step, you wrote code to draw the cannon on the screen. But a game needs a player who can interact with it. So now it's time to learn about keybindings, so you can use the keyboard to move the cannon.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_2/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

To start, you'll familiarize yourself with key techniques used in animation.

Move the `Turtle` Object

You can bind a function to a key using `window.onkeypress()`. The keypress calls the function bound to that key. You can define two functions to move the cannon left and right, and bind them to the `←Left` and `→Right` arrow keys on your keyboard:

Python `turtle_invaders.py`

```
import turtle

CANNON_STEP = 10

# ...

def move_left():
    cannon.setx(cannon.xcor() - CANNON_STEP)

def move_right():
    cannon.setx(cannon.xcor() + CANNON_STEP)

window.onkeypress(move_left, "Left")
window.onkeypress(move_right, "Right")
window.onkeypress(turtle.bye, "q")
window.listen()

turtle.done()
```

Did you notice that you used the function names `move_left` and `move_right` without parentheses as arguments for `.onkeypress()`? If you add parentheses, the program calls the function when it executes the line with `.onkeypress()`, and it binds the function's return value to the key. The functions `move_left()` and `move_right()` return `None`, so if you include parentheses, the arrow keys will remain inactive.

The functions to move the cannon use `CANNON_STEP`, which you define at the top of your code. The name is written using uppercase letters to show that this is a constant value you use to configure the game. Try changing this value to see how this will affect the game.

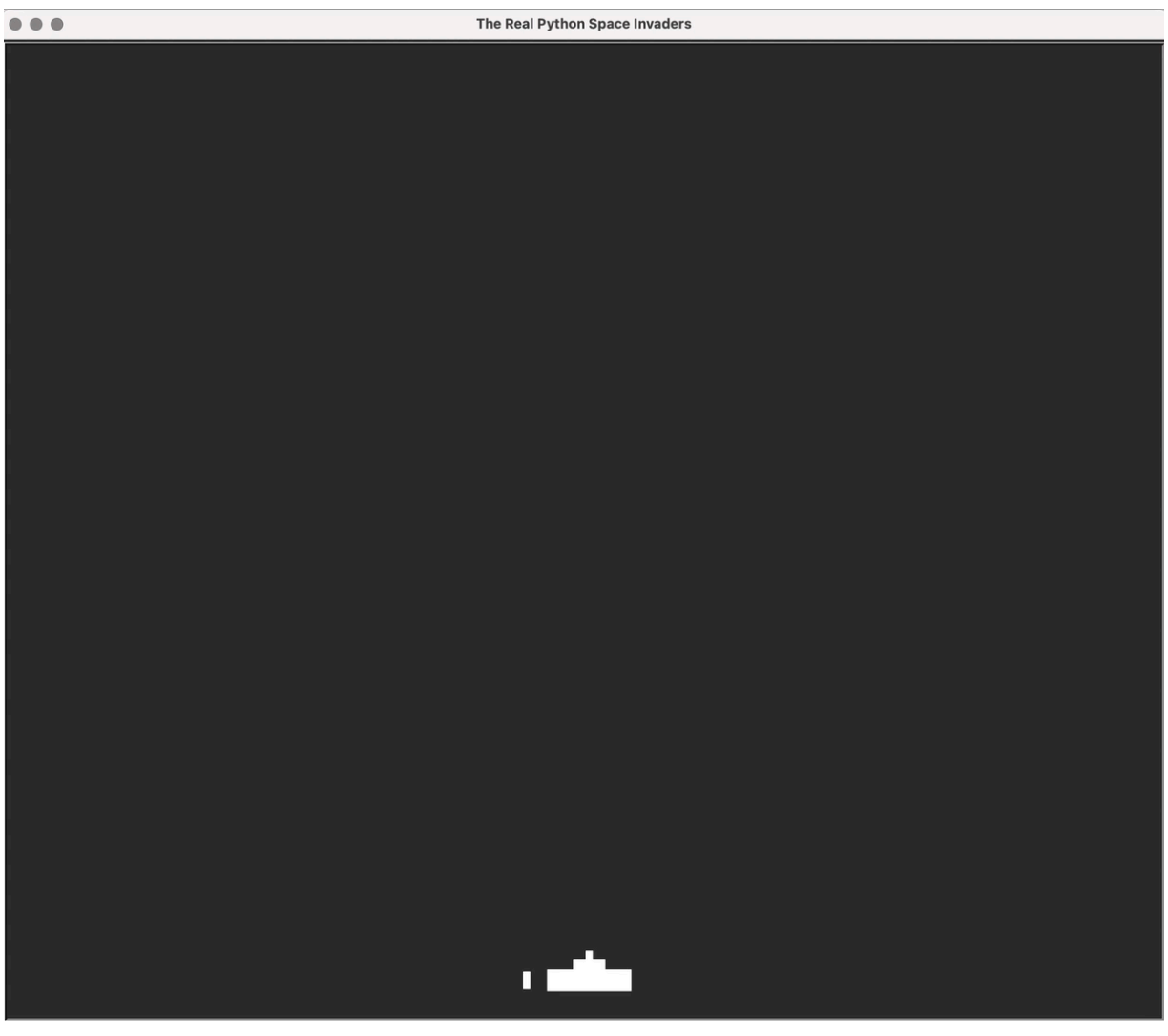
You also bring the focus to the screen using `window.listen()` so that keypress events are collected by the program. By default, the `turtle` module doesn't use computing resources to listen for keypresses while the program is running. Calling `window.listen()` overrides this default. If the keys don't work when you run a `turtle` game, make sure that you included `.listen()` as it's easy to forget about when using `.onkeypress()`.

Now that you're familiar with keybindings, you'll add a feature to quit the program by binding the `Q` key to `turtle.bye`. As you did with the previous functions, you don't add parentheses to `turtle.bye` even though it's a function. Make sure you use a lowercase `q` because if you use an uppercase letter in the code, you'll need to press `⇧ Shift` + `Q` to quit the program.

Now when you run the program, you're able to press the right and left arrow keys to move the `Turtle` object.

> **Note:** Functions used as arguments for `.onkeypress()` can't accept any required arguments. Although the functions can have a `return` statement, you can't access the returned data in your program since these functions are called from within `.onkeypress()`.

Here's what the output looks like at this stage:



This is not quite the behavior you want. Notice how only one part of the drawing moves when you press the arrow keys. This is the sprite representing the `Turtle` object. The rest of the drawing is the result to the `.stamp()` calls in the section of code where you draw the laser cannon.

Move the Entire Laser Cannon

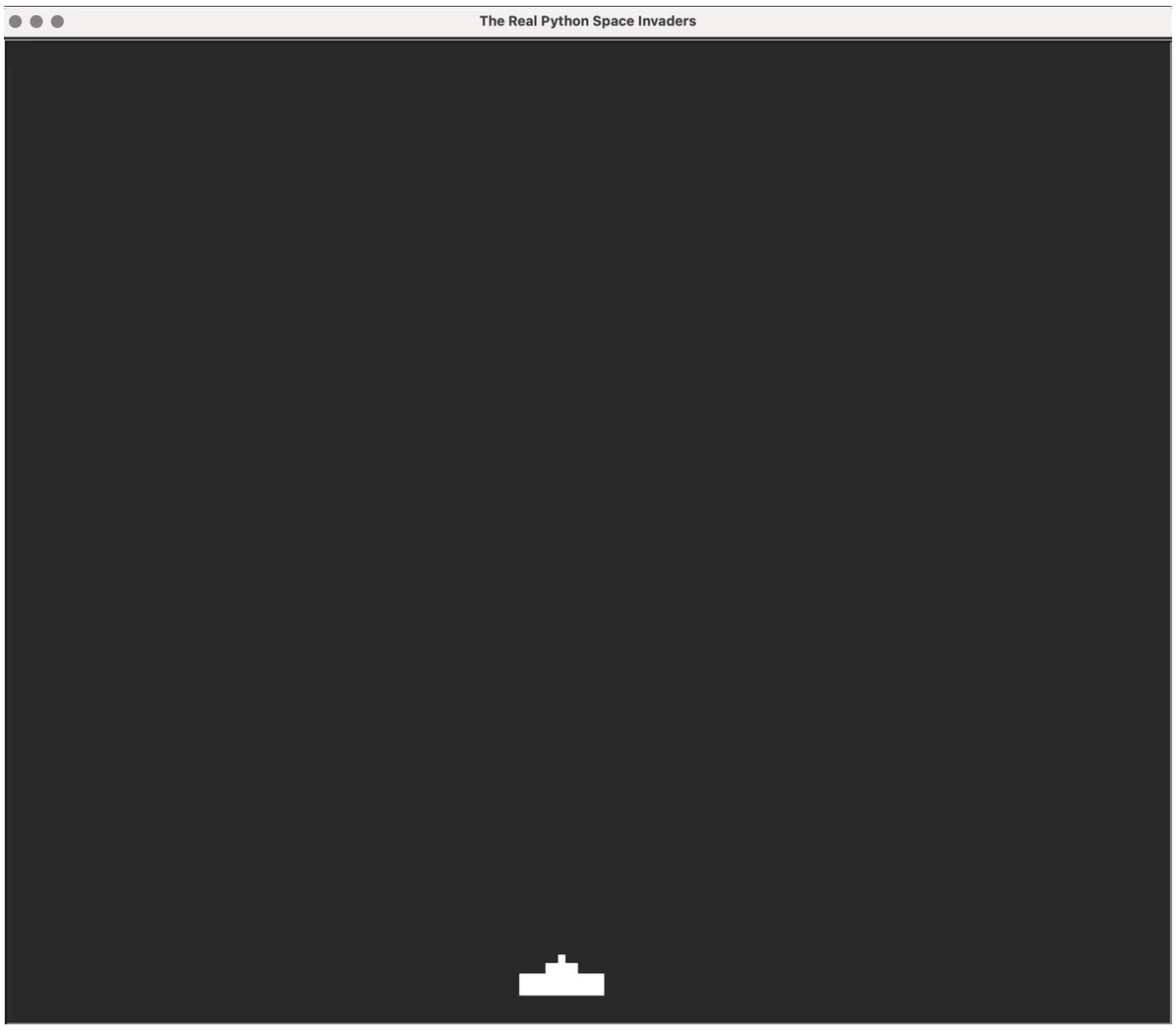
Every time you move the laser cannon, you can clear the previous drawing and redraw the cannon in the new location. To avoid repetition, you can move the code to draw the cannon into a function `draw_cannon()`. In the following code, most of the lines in `draw_cannon()` are the same lines you had in the previous version, but now they're indented:

Python turtle_invaders.py

```
# ...

def draw_cannon():
    cannon.clear()
    cannon.turtlesize(1, 4)  # Base
    cannon.stamp()
    cannon.sety(FLOOR_LEVEL + 10)
    cannon.turtlesize(1, 1.5)  # Next tier
    cannon.stamp()
    cannon.sety(FLOOR_LEVEL + 20)
    cannon.turtlesize(0.8, 0.3)  # Tip of cannon
    cannon.stamp()
    cannon.sety(FLOOR_LEVEL)

def move_left():
    cannon.setx(cannon.xcor() - CANNON_STEP)
    draw_cannon()

def move_right():
    cannon.setx(cannon.xcor() + CANNON_STEP)
    draw_cannon()

window.onkeypress(move_left, "Left")
window.onkeypress(move_right, "Right")
window.onkeypress(turtle.bye, "q")
window.listen()

draw_cannon()

turtle.done()
```

When you call `cannon.clear()`, all the drawings made by `cannon` are deleted from the screen. You call `draw_cannon()` in both `move_left()` and `move_right()` to redraw the cannon each time you move it. You also call `draw_cannon()` in the main section of the program to draw the cannon at the start of the game.

Here's what the game looks like now:



The whole laser cannon moves when you press the left or right arrow keys. The function `draw_cannon()` deletes and redraws the three rectangles that make up the cannon.

However, the movement of the cannon is not smooth and can glitch if you try to move it too rapidly, as you can see in the second part of the video. You can fix this issue by controlling when items are drawn on the screen in the `turtle` module.

Control When Items Are Displayed

The `turtle` module displays several small steps when `Turtle` objects move. When you move `cannon` from its initial central position to the bottom of the screen, you can see the square move downwards to its new position. When you redraw the cannon, the `Turtle` object needs to move to several locations, change shape, and create a stamp of that shape.

These actions take time, and if you press an arrow key during this drawing process, the `Turtle` object moves before it has time to complete the drawing.

This problem becomes even more noticeable when there are more events occurring in the game. Now is a good time to fix this issue. You can prevent any changes from being displayed on the screen by setting `window.tracer(0)`.

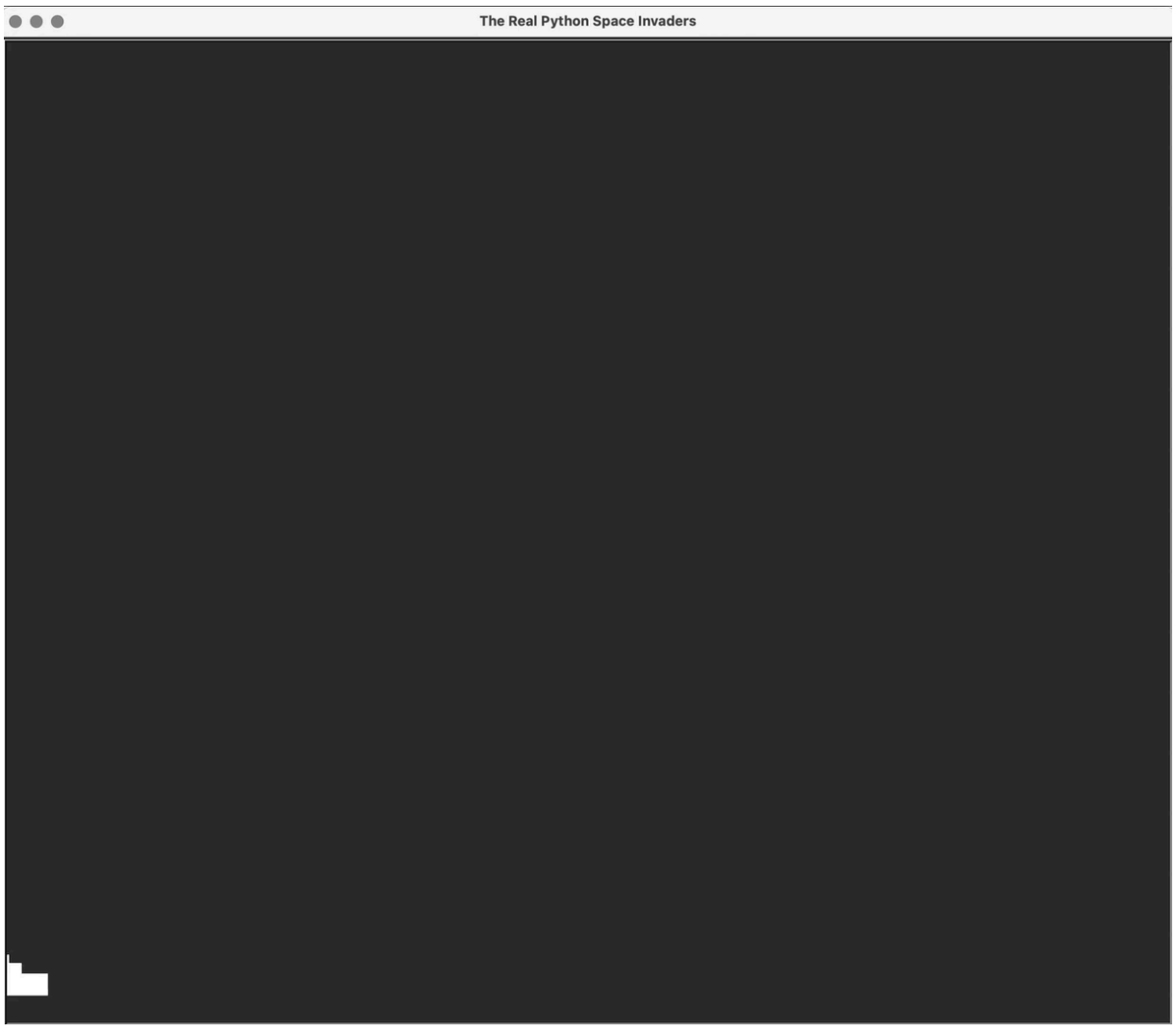
The program will still change the coordinates of a `Turtle` object and its heading, but will not display the changes on the screen. You can control when the screen is updated by calling `window.update()`. This gives you control over when to update the screen.

You can call `window.tracer(0)` right after you create the screen and `window.update()` in `draw_cannon()`:

Python turtle_invaders.py

```
import turtle

CANNON_STEP = 10

window = turtle.Screen()
window.tracer(0)

# ...

def draw_cannon():
    # ...
    window.update()

# ...
```

Now there are no glitches when you move the cannon, and the initial placement of `cannon` from the center to the bottom is no longer visible. Here's this `turtle` game so far:



You'll improve how smoothly the cannon moves later in this tutorial. In the next step, you'll fix another problem. Notice how towards the end of the video, you can see the laser cannon moving out of the screen when the left arrow key is pressed too many times. You need to prevent the cannon from leaving the screen so it can hold center ground and defend the planet from the invading aliens!

Prevent the Laser Cannon From Leaving the Screen

The functions `move_left()` and `move_right()` are called each time the player presses one of the arrow keys to move the laser cannon left or right. To check whether the laser cannon has reached the edge of the screen, you can add an `if` statement. Instead of using `LEFT` and `RIGHT`, which are the edges of the screen, you'll include a gutter to stop the cannon just before it reaches the edges:

Python turtle_invaders.py

```
# ...

LEFT = -window.window_width() / 2
RIGHT = window.window_width() / 2
TOP = window.window_height() / 2
BOTTOM = -window.window_height() / 2
FLOOR_LEVEL = 0.9 * BOTTOM
GUTTER = 0.025 * window.window_width()

# ...

def move_left():
    new_x = cannon.xcor() - CANNON_STEP
    if new_x >= LEFT + GUTTER:
        cannon.setx(new_x)
        draw_cannon()

def move_right():
    new_x = cannon.xcor() + CANNON_STEP
    if new_x <= RIGHT - GUTTER:
        cannon.setx(new_x)
        draw_cannon()

# ...
```

Now, the laser cannon can't leave the screen. You'll use similar techniques in the following steps as you add lasers and aliens to your `turtle` game.

Step 3: Shoot Lasers With the Spacebar

There's no use having a laser cannon if it can't shoot lasers. In this step, you'll create lasers that you can shoot by pressing the spacebar. You'll learn how to store all of the active lasers in a list, and how to move each laser forward in each frame of the game. You'll also start working on the game loop that controls what happens in each frame.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_3/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

To start, you'll use `Turtle` objects to create and store the lasers.

Create Lasers

You want to be able to shoot lasers each time you press `Space`. Since your code needs to store all the lasers present, you can create a `Turtle` for each laser and store it in a list. You'll also need a function to create new lasers:

Python turtle_invaders.py

```
# ...

lasers = []

def draw_cannon():
    # ...

def move_left():
    # ...

def move_right():
    # ...

def create_laser():
    laser = turtle.Turtle()
    laser.penup()
    laser.color(1, 0, 0)
    laser.hideturtle()
    laser.setposition(cannon.xcor(), cannon.ycor())
    laser.setheading(90)
    # Move laser to just above cannon tip
    laser.forward(20)
    # Prepare to draw the laser
    laser.pendown()
    laser.pensize(5)

    lasers.append(laser)

# Key bindings
window.onkeypress(move_left, "Left")
window.onkeypress(move_right, "Right")
window.onkeypress(create_laser, "space")
window.onkeypress(turtle.bye, "q")
window.listen()

draw_cannon()

turtle.done()
```

Spacebar is bound to `create_laser()`, which creates a new `Turtle` object to represent a laser, sets its initial attributes, and appends it to the list of all lasers. To avoid complicating the code, this tutorial makes some simplifications, which you can read about in the following section:

A Note About Simplifications in the Code Show/Hide

There are two simplifications to prevent the code from getting too complex:

1. The height of the cannon is hardcoded to 20 pixels. This value is used in `draw_cannon()` to place the rectangles that make up the cannon, and in `create_laser()`, to place the laser's starting point at the tip of the cannon. In general, you should try to

avoid hardcoding values. This value could be defined as a percentage of the screen height, and the appropriate changes made to `draw_cannon()` to set the widths and heights of the three rectangles used to draw the cannon.

2. The function `create_laser()` modifies a list in the program's global scope. In general, it's best for functions to return values instead of changing the state of existing global variables. However, you'll call `create_laser()` through a keypress, and this prevents you from accessing the returned objects.

The program creates new `Turtle` objects for each laser when you press `Space`. You can add `print(len(lasers))` in `create_laser()` to ensure that the program creates a new laser when you press the spacebar.

In the next section, you'll move the lasers in each frame of the game. Then you'll create a game loop to account for everything that occurs in each frame.

Create the Game Loop to Move the Lasers

The structure of a game can be broken into three parts:

- **Initialization**: This is where you create the objects you need and their initial states to set up the game environment. This is also where you include function definitions for actions that need to occur in the game.
- **Game loop**: This is the central part of a game. It checks and processes user inputs, changes the state of the objects in the game and other values such as the score, and updates the display to reflect the changes. The game loop is a loop because it needs to run continuously to run the game. Each iteration of the game loop represents a frame of the game.
- **Ending**: Once the game loop ends, either because the player wins or loses, the program can end instantly. Typically, games display an end screen, but they can also save any output to file and offer options to the player, such as to play again.

You've already dealt with a signifiant part of the initialization of this `turtle` game. Now, it's time to work on the game loop.

The structure of a game loop depends on the platform you use. In the `turtle` module, you'll need to write your own `while` loop to update the state of the objects in the game and check for events, such as collisions between objects. However, other events are managed through `turtle` methods like `.onkeypress()`. A `turtle` program runs an event loop, which starts when you call `turtle.done()` and handles events such as keypresses.

You can compare the game loop you'll need when working with `turtle`, to the game loop structures in other Python games packages such as `pygame` and `arcade`.

In this `turtle` game so far, the only item that's moving is the cannon, which you move with the arrow keys. The cannon doesn't move at other times.

However, in most games, there are items that need to move or perform other actions continuously. Here, the lasers you create when you press the spacebar need to move up by a fixed amount in each frame of the game. So now you'll create a `while` loop to make these changes that occur in every frame:

Python `turtle_invaders.py`

```
import turtle

CANNON_STEP = 10
LASER_LENGTH = 20
LASER_SPEED = 10

# ...

def create_laser():
    laser = turtle.Turtle()
    laser.penup()
    laser.color(1, 0, 0)
    laser.hideturtle()
    laser.setposition(cannon.xcor(), cannon.ycor())
    laser.setheading(90)
    # Move laser to just above cannon tip
    laser.forward(20)
    # Prepare to draw the laser
    laser.pendown()
    laser.pensize(5)

    lasers.append(laser)

def move_laser(laser):
    laser.clear()
    laser.forward(LASER_SPEED)
    # Draw the laser
    laser.forward(LASER_LENGTH)
    laser.forward(-LASER_LENGTH)

# ...

# Game loop
while True:
    # Move all lasers
    for laser in lasers:
        move_laser(laser)
    window.update()

turtle.done()
```

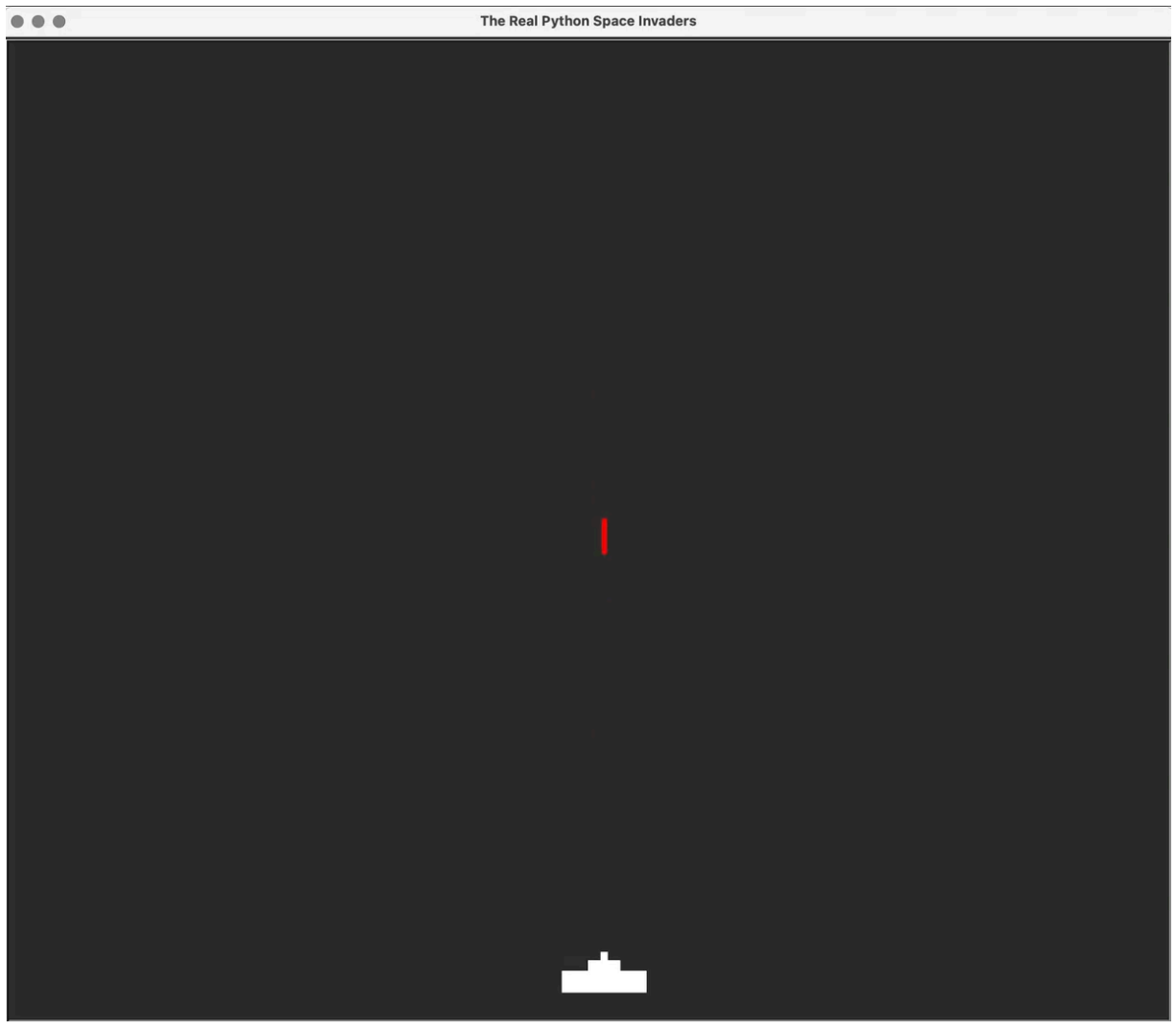
You define the function `move_laser()`, which accepts a `Turtle` object. The function clears the previous drawing for a laser and moves the `Turtle` object forward by an amount that represents the laser's speed. This is the first of the three calls to `laser.forward()`.

To draw the laser on the screen, you use a different technique. When you created the cannon, you used the shape of the `Turtle` object. To draw the laser, you draw a line with the `Turtle` object.

Look back at `create_laser()` and notice you called `laser.hideturtle()` to hide the sprite, and `laser.pendown()` and `laser.pensize(5)` to enable the `Turtle` to draw lines five pixels wide when it moves. Now, you'll need to bring the `Turtle` object back to its starting position in `move_laser()` so it's ready for the next frame.

The `while` loop contains a `for` loop that iterates through all the lasers and moves them forward. You can also add `window.update()` in the game loop to update the display once in each frame. At this point, you can also remove `window.update()` from `draw_cannon()` for you no longer need it.

When you run this code and press the spacebar, you'll see lasers shooting from the cannon:



Next, you'll deal with what to do with the lasers when they leave the top of the screen.

Remove Lasers That Leave the Screen

When a laser leaves the screen, you won't see it any more. But, the `Turtle` object is still in the list of lasers and the program still has to deal with this laser. Even when the laser is out of sight, the program still moves the laser upwards, all the way to infinity and beyond! Over time, the list of lasers will contain a large number of objects that no longer have any role in the game.

When the program moves a laser, the performance bottleneck is the process of drawing the laser on the screen. Updating the `Turtle` object's *y*-coordinates is a quick fix. When a laser leaves the screen, the program doesn't need to draw it any more. Therefore, you'll need to shoot many lasers before you see any noticeable slowing down of the game. However, it's always good practice to remove objects you no longer need.

To remove these objects when they leave the screen, you remove them from the list `lasers`, and this stops the game loop from moving them forward. But because the `turtle` module keeps an internal list of all `Turtle` objects, if you remove the lasers from this internal list by using `turtle.turtles()`, the objects will also be removed from memory:

Python `turtle_invaders.py`

```
# ...

# Game loop
while True:
    # Move all lasers
    for laser in lasers.copy():
        move_laser(laser)
        # Remove laser if it goes off screen
        if laser.ycor() > TOP:
            laser.clear()
            laser.hideturtle()
            lasers.remove(laser)
            turtle.turtles().remove(laser)
    window.update()

turtle.done()
```

Notice how the `for` loop iterates through a copy of `lasers`, since it's best not to loop through a list that's being changed within the same `for` loop.

At this point, you can add `print(len(turtle.turtles()))` and `print(len(lasers))` to the game loop to confirm that lasers are removed from these lists when they leave the screen.

Good job! You're ready to create aliens and move them down the screen.

Step 4: Create and Move the Aliens

It's the quiet before the storm. The laser cannon sits idle at the moment since there are no invading aliens to shoot at. The time has come to add the aliens to your game.

You've already done most of the hard work to create and move the aliens since you can deal with aliens similarly to lasers. But there are some minor differences:

- **New aliens spawn every few seconds** instead of when the player presses a key.
- **Aliens appear at the top of the screen** in a random *x*-position instead of at the tip of the cannon.
- The **shape and color of the aliens** are different from lasers.

Other than these slight differences, the rest of the code will mirror the code you wrote to create and move lasers.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_4/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

To start this part of the project, you'll create aliens to appear at regular time intervals. Once this is in place, you can focus on moving them.

Spawn New Aliens

To start, you define the new function `create_alien()` and a list to store the aliens:

Python turtle_invaders.py

```
import random
import turtle

# ...

lasers = []
aliens = []

# ...

def create_alien():
    alien = turtle.Turtle()
    alien.penup()
    alien.turtlesize(1.5)
    alien.setposition(
        random.randint(
            int(LEFT + GUTTER),
            int(RIGHT - GUTTER),
        ),
        TOP,
    )
    alien.shape("turtle")
    alien.setheading(-90)
    alien.color(random.random(), random.random(), random.random())
    aliens.append(alien)

#  ...
```

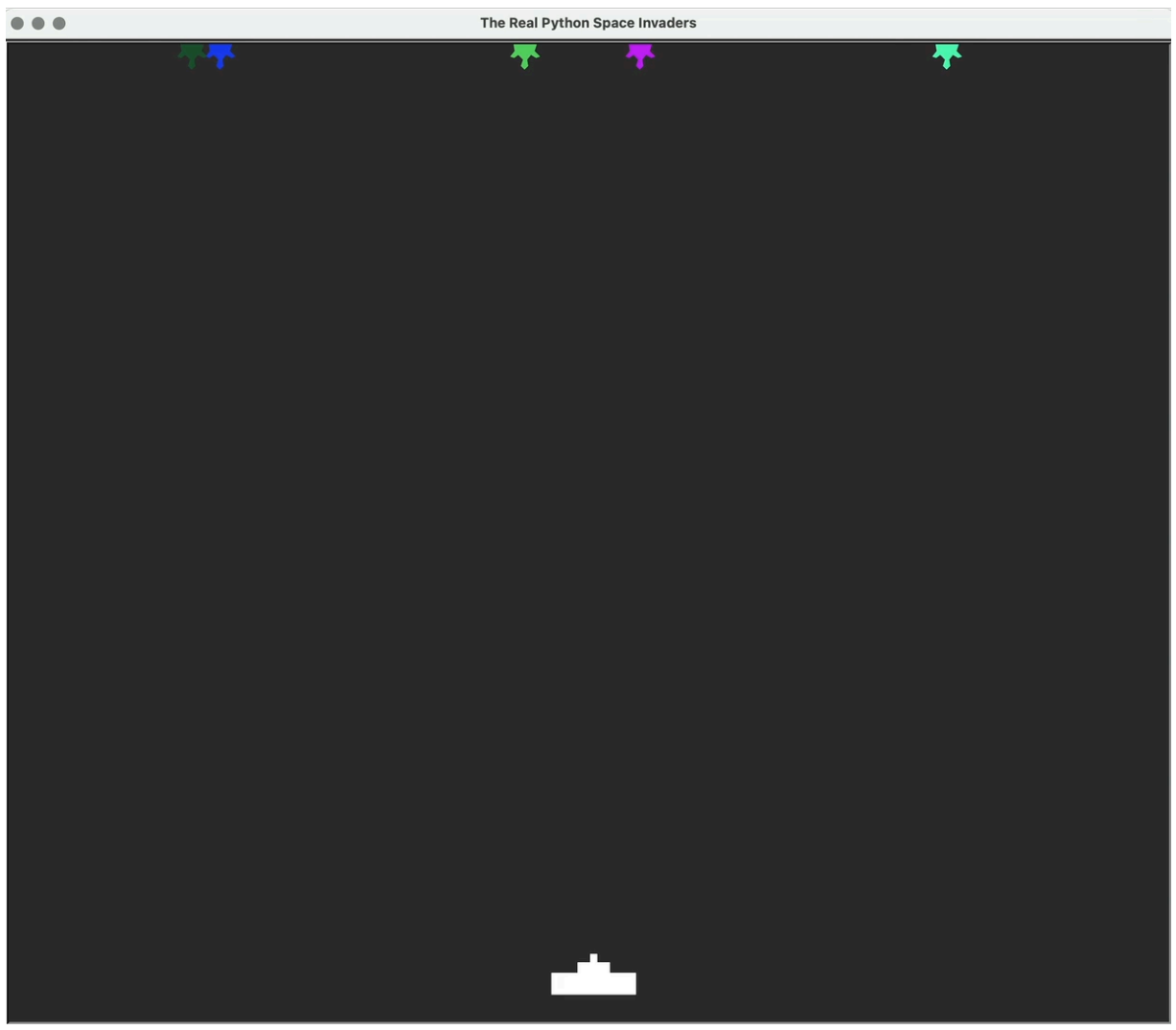
You place the aliens at the top of the screen in a random *x*-position. The aliens face downwards. You also assign random values for the red, green, and blue color components so that every alien has a random color.

Aliens appear at regular time intervals in this `turtle` game. You can import the `time` module and set a timer to spawn a new alien:

Python turtle_invaders.py

```python
import random
import time
import turtle

CANNON_STEP = 10
LASER_LENGTH = 20
LASER_SPEED = 10
ALIEN_SPAWN_INTERVAL = 1.2  # Seconds

# ...

# Game loop
alien_timer = 0
while True:
    # ...

    # Spawn new aliens when time interval elapsed
    if time.time() - alien_timer > ALIEN_SPAWN_INTERVAL:
        create_alien()
        alien_timer = time.time()
    window.update()

turtle.done()
```

The code now spawns a new alien every 1.2 seconds. You can adjust this value to make the game harder or easier. Here's the game so far:



Aliens are spawning at regular intervals, but they're stuck at the top of the screen. Next, you'll learn how to move the aliens around the screen.

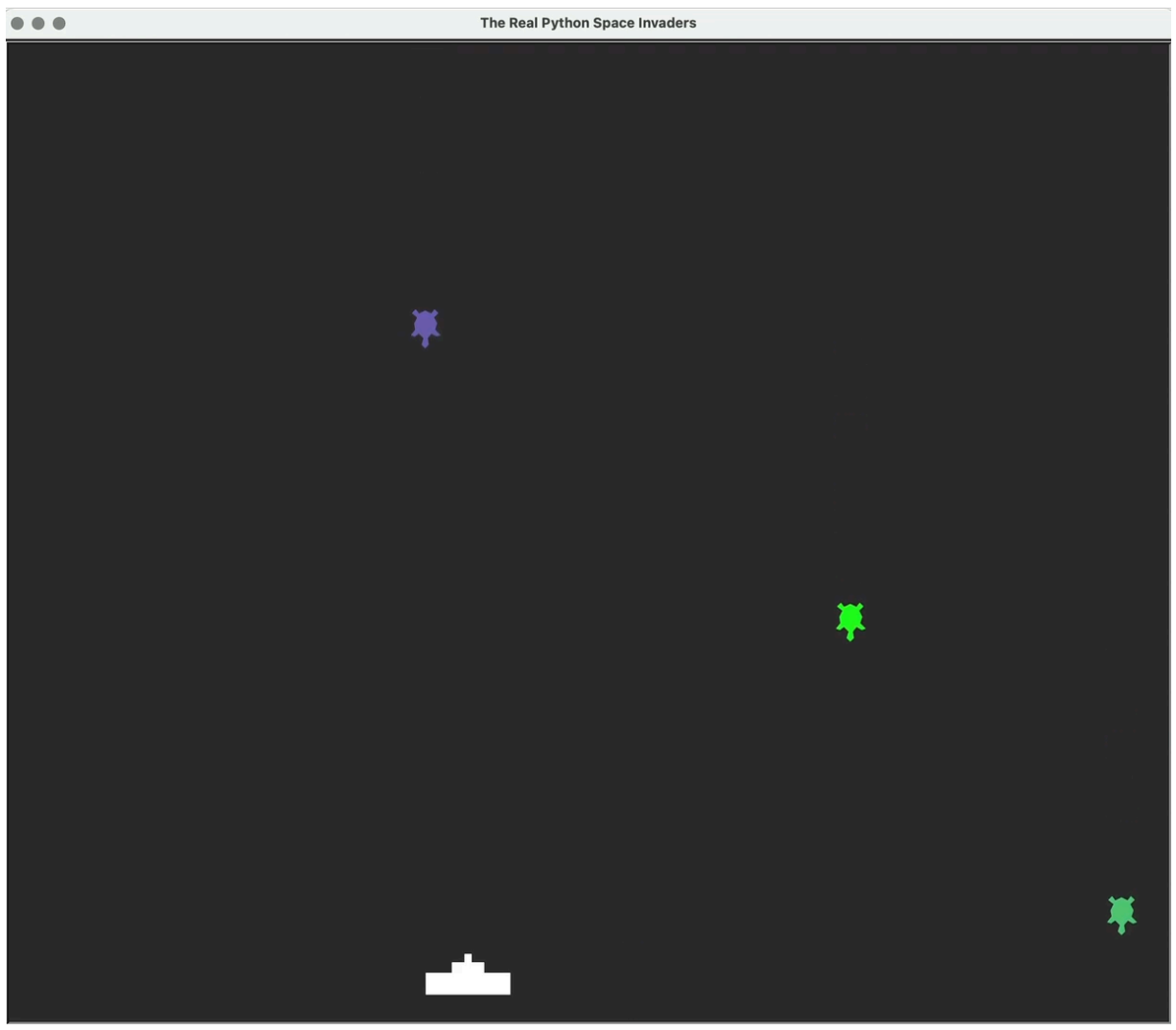
Move the Aliens

This is where you'll move all of the aliens in the game loop's list of aliens:

Python turtle_invaders.py

```
import random
import time
import turtle

CANNON_STEP = 10
LASER_LENGTH = 20
LASER_SPEED = 10
ALIEN_SPAWN_INTERVAL = 1.2  # Seconds
ALIEN_SPEED = 2

# ...

# Game loop
alien_timer = 0
while True:
    # ...

    # Move all aliens
    for alien in aliens:
        alien.forward(ALIEN_SPEED)
    window.update()

turtle.done()
```

The aliens are now continuously moving down:



If the aliens aren't moving at the right speed, you can adjust their speed by using a different value for `ALIEN_SPEED`. Keep in mind that the speed of the game will vary depending on the system you're working on. You'll set the frame rate of the game in the last step of this tutorial, so you don't need to worry about the speed for now.

This `turtle` game is looking closer to completion, but there are a still a few key steps you need to take before this is a game you can play. The first of these steps is to detect when a laser hits an alien.

Step 5: Determine When Lasers Hit Aliens

You probably noticed in the previous video that the lasers were going right through the aliens without affecting them. This behavior won't stop the alien invasion! Currently, the program is not checking whether a laser hits an alien. So you'll need to update the code to deal with this important game feature.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_5/` in the link below:

Get Your Code: Click here to download the free sample code that shows you how to build a Python turtle game.

In each frame of the game, you're likely to have several lasers flying upwards and several aliens falling downwards. So you need to check whether any of the lasers hit any of the aliens. In this part of tutorial, you'll apply a simple technique to deal with this issue—you'll check the distance of each laser from each alien to detect any hits.

Granted this is not the most efficient option, but it's good enough for this simplified version of space invaders.

In the game loop, you already have a loop iterating through the list of lasers. You can add another loop to iterate through each alien for every laser:

Python turtle_invaders.py

```
# ...

while True:
    # Move all lasers
    for laser in lasers.copy():
        move_laser(laser)
        # Remove laser if it goes off screen
        if laser.ycor() > TOP:
            laser.clear()
            laser.hideturtle()
            lasers.remove(laser)
            turtle.turtles().remove(laser)
        # Check for collision with aliens
        for alien in aliens.copy():
            if laser.distance(alien) < 20:
                # TODO Remove alien and laser
                ...

# ...
```

The new `for` loop iterates through a copy of `aliens`, and it's nested within the `for` loop that iterates through `lasers`. You use `aliens.copy()` since you'll remove aliens from this list in some frames. Make sure to avoid changing the list you're using for iteration.

The code above shows a `# TODO` comment instead of the block of code required. The code you need to write here should perform the following actions:

1. **Remove the laser** from the screen and from memory.
2. **Remove the alien** from the screen and from memory.

You already wrote code to remove the laser. These are the lines which clear the drawing and remove the `Turtle` from the two lists when a laser leaves the screen:

Python turtle_invaders.py

```
# ...
laser.clear()
laser.hideturtle()
lasers.remove(laser)
turtle.turtles().remove(laser)
```

You should avoid repeating the same lines of code. So you can put these lines in a function and call it twice. However, these are the same steps required to remove the alien from the game. Therefore, you can write a function that can remove any sprite and use it for the lasers and the aliens:

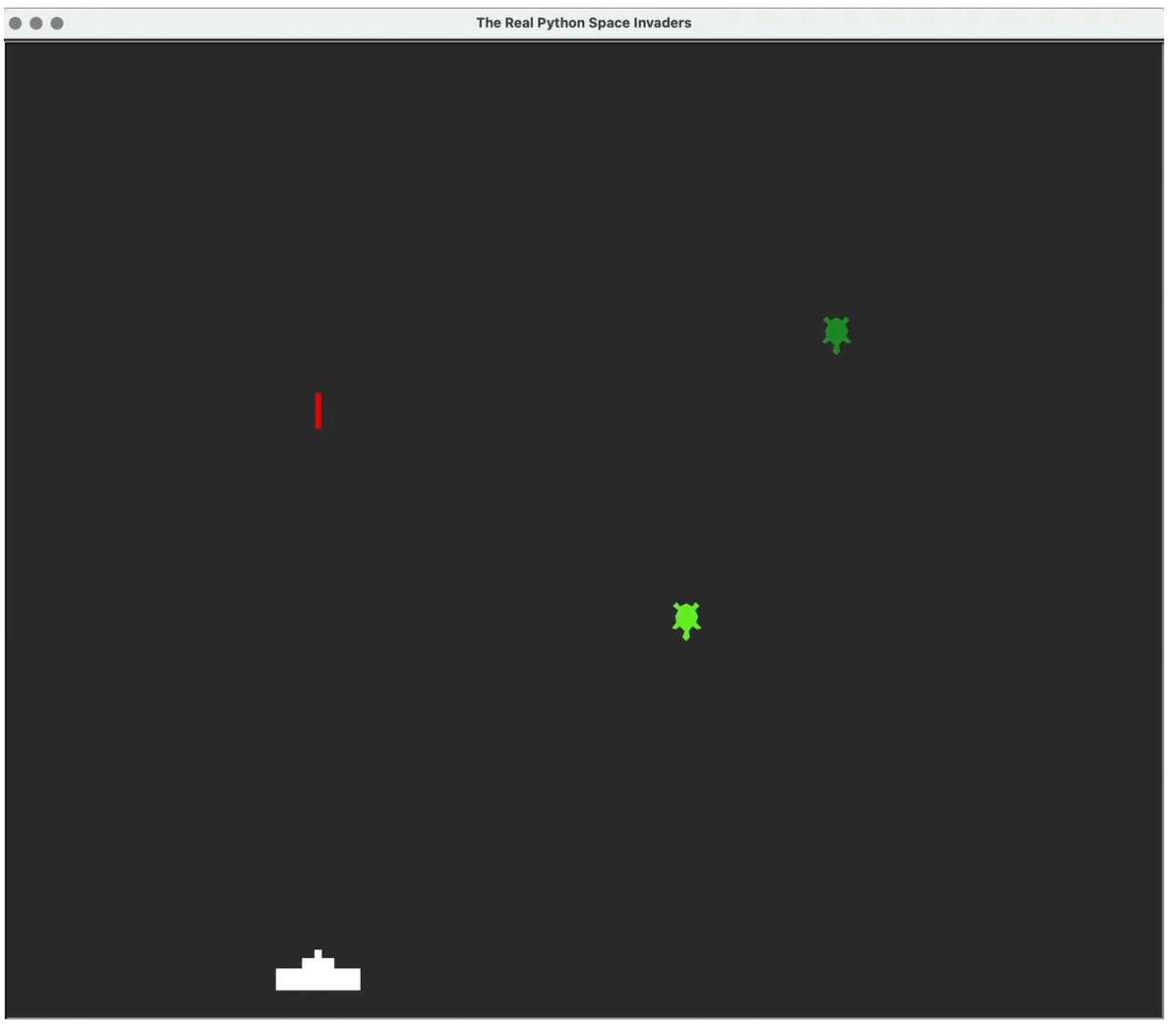
Python turtle_invaders.py

```
# ...

def remove_sprite(sprite, sprite_list):
    sprite.clear()
    sprite.hideturtle()
    window.update()
    sprite_list.remove(sprite)
    turtle.turtles().remove(sprite)

# ...

# Game loop
alien_timer = 0
while True:
    # Move all lasers
    for laser in lasers.copy():
        move_laser(laser)
        # Remove laser if it goes off screen
        if laser.ycor() > TOP:
            remove_sprite(laser, lasers)
            break
        # Check for collision with aliens
        for alien in aliens.copy():
            if laser.distance(alien) < 20:
                remove_sprite(laser, lasers)
                remove_sprite(alien, aliens)
                break

    # ...
```

The function `remove_sprite()` accepts a `Turtle` object and a list, and it clears the `Turtle` and any drawings linked to it. You call `remove_sprite()` twice for the lasers: once when a laser leaves the screen and another time when it hits an alien. You also call `remove_sprite()` once for the aliens in the game loop.

There are also two `break` statements. The first is when the laser leaves the screen, since you don't need to check anymore whether that laser has collided with an alien. The second is in the inner `for` loop when a laser hits an alien, since that laser cannot hit another alien.

Your game should look like this now:



As you can see, both the laser and the alien disappear when there's a collision. There's just one more required step before you can play this game. Then it'll be *Game Over* if an alien reaches the floor level.

Step 6: End the Game

The game ends when one of the aliens reaches the floor level. The alien invasion succeeded! So now you'll add a splash screen to show *Game Over* on the screen.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_6/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

The game loop already iterates through the list of aliens, and you can check whether each alien has reached the ground. There are several ways to stop the game loop when this happens. In this next part, you'll create a Boolean flag to control when the `while` loop should stop:

Python turtle_invaders.py

```
# ...

# Game loop
alien_timer = 0
game_running = True
while game_running:
    # ...

    # Move all aliens
    for alien in aliens:
        alien.forward(ALIEN_SPEED)
        # Check for game over
        if alien.ycor() < FLOOR_LEVEL:
            game_running = False
            break
    window.update()

turtle.done()
```

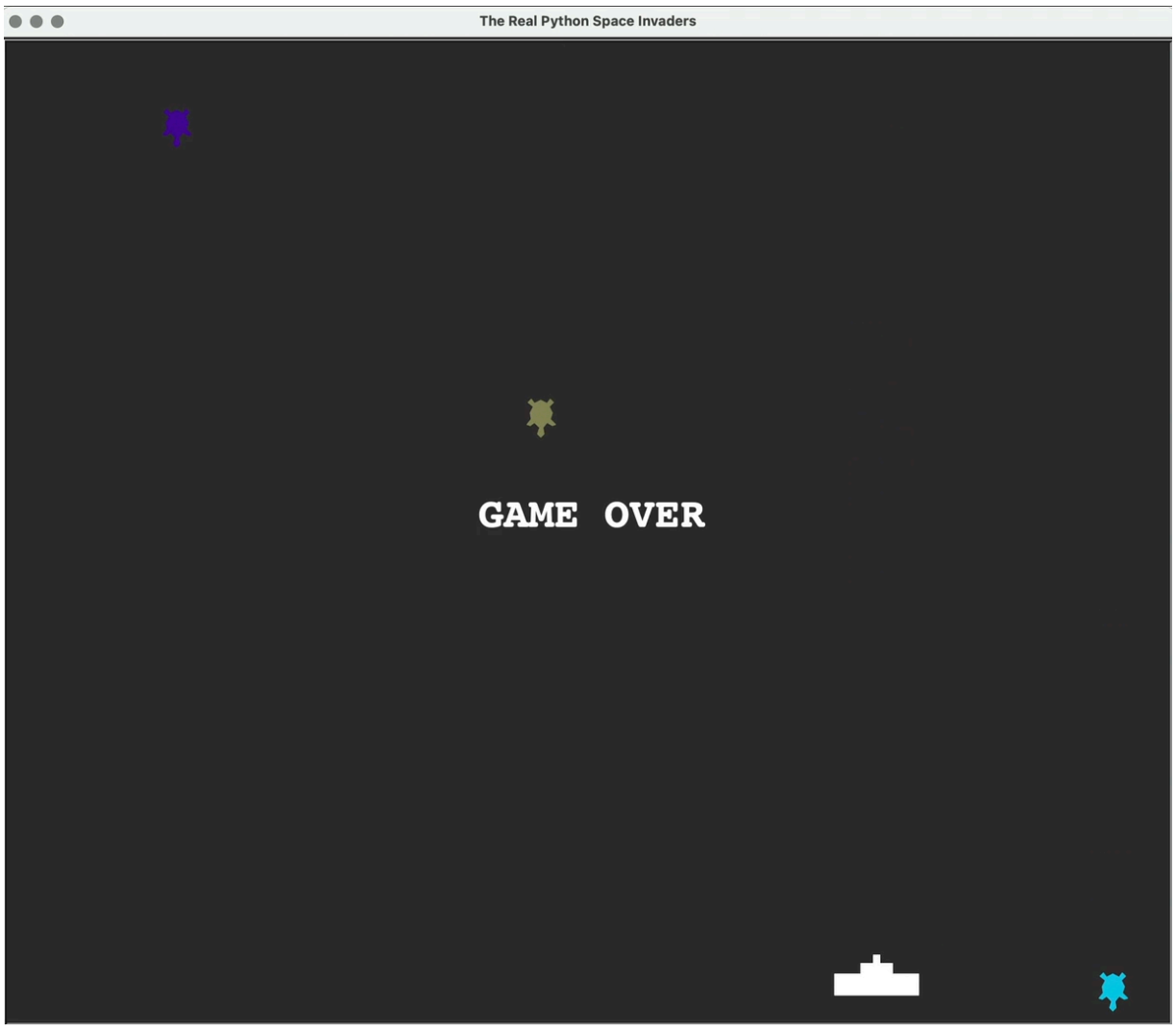
When the alien's *y*-coordinate is below the floor level, you set the Boolean flag `game_running` to `False` and break the `for` loop. The flag signals the `while` loop to stop repeating.

Before you made this last change, you had an infinite `while` loop. Now, the loop is no longer infinite, and you can add code after the loop that will execute once the game ends:

Python turtle_invaders.py

```
# ...

# Game loop
alien_timer = 0
game_running = True
while game_running:
    # ...

    window.update()

splash_text = turtle.Turtle()
splash_text.hideturtle()
splash_text.color(1, 1, 1)
splash_text.write("GAME OVER", font=("Courier", 40, "bold"), align="center")

turtle.done()
```

You create a new `Turtle` to write text on the screen. The game is nearly complete now:



Congratulations, you have a game that works! You can stop here if you wish, but there are three additional steps you can take if you'd like to polish this `turtle` game a bit more.

Step 7: Add a Timer and a Score

In this step, you'll add a timer and display the number of aliens the player has hit.

You can download the code as it'll look at the end of this step from the folder named `source_code_step_7/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

To start, you'll create and display a timer.

Create a Timer

You already imported the `time` module when spawning aliens. You can use this module to store the time at the start of the game loop and to work out how much time has elapsed in each frame. You can also create a new `Turtle` to display text on the screen:

Python `turtle_invaders.py`

```
# ...

# Create laser cannon
cannon = turtle.Turtle()
cannon.penup()
cannon.color(1, 1, 1)
cannon.shape("square")
cannon.setposition(0, FLOOR_LEVEL)

# Create turtle for writing text
text = turtle.Turtle()
text.penup()
text.hideturtle()
text.setposition(LEFT * 0.8, TOP * 0.8)
text.color(1, 1, 1)

# ...

# Game loop
alien_timer = 0
game_timer = time.time()
game_running = True
while game_running:
    time_elapsed = time.time() - game_timer
    text.clear()
    text.write(
        f"Time: {time_elapsed:5.1f}s",
        font=("Courier", 20, "bold"),
    )

    # ...
```

You write the time elapsed in the top-right corner of the screen at the start of each frame. In the first argument in `.write()`, you use the format specifier `:5.1f` to display the time elapsed with a total of five characters and one digit after the decimal point.

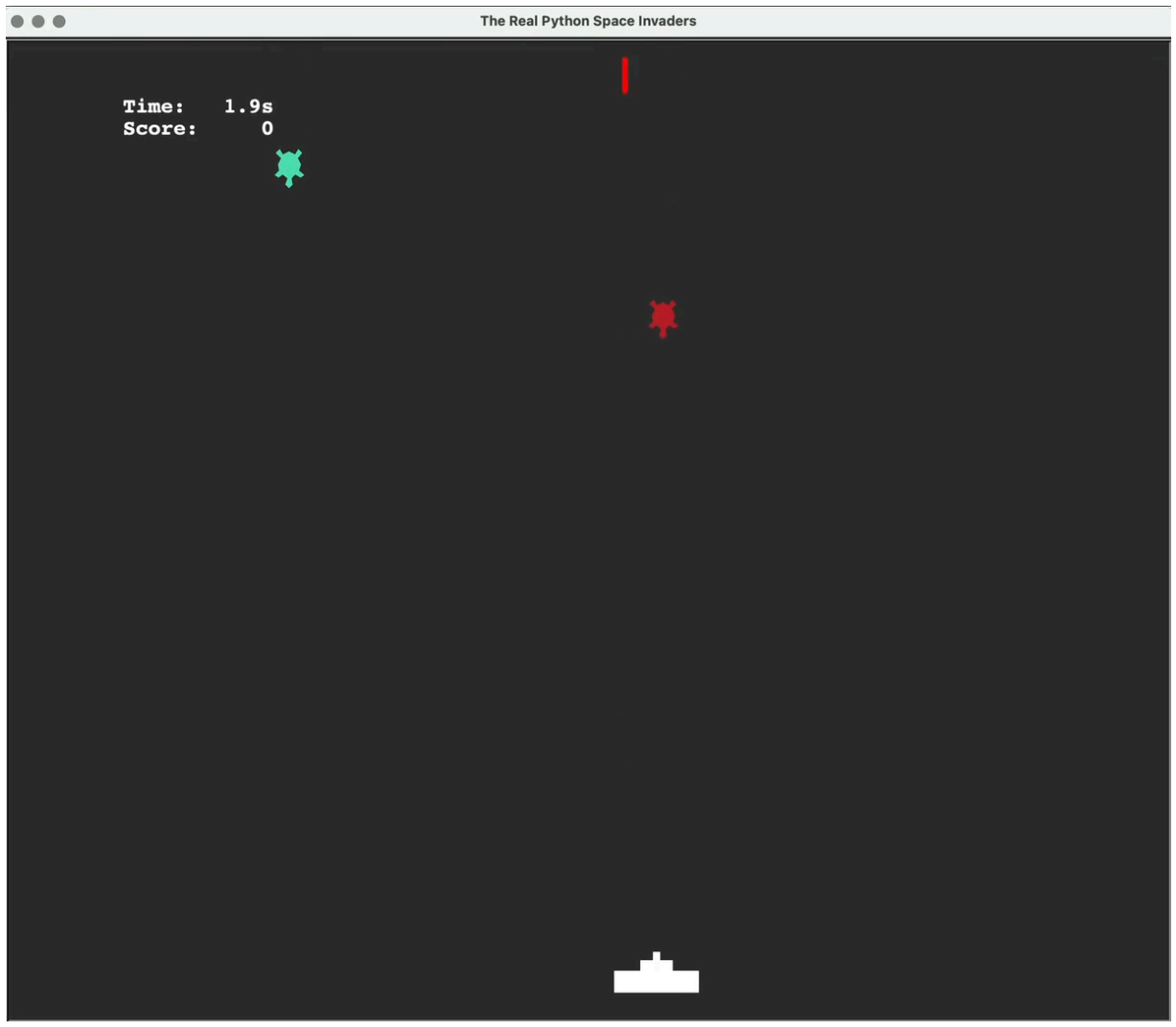
Add the Score

Create another variable to keep the score. Increment the score each time the player hits an alien and display the score below the time elapsed:

Python turtle_invaders.py

```
# ...

# Game loop
alien_timer = 0
game_timer = time.time()
score = 0
game_running = True
while game_running:
    time_elapsed = time.time() - game_timer
    text.clear()
    text.write(
        f"Time: {time_elapsed:5.1f}s\nScore: {score:5}",
        font=("Courier", 20, "bold"),
    )

    # ...
        # Check for collision with aliens
        for alien in aliens.copy():
            if laser.distance(alien) < 20:
                remove_sprite(laser, lasers)
                remove_sprite(alien, aliens)
                score += 1
                break

    # ...
```

This `turtle` game now shows the number of aliens hit and the time elapsed in each frame of the game:



The game is fully functional, but did you notice that the laser cannon jumps from one position to another? And, if you wanted to move the cannon more quickly, the jumps would be even more noticeable. In the next step, you'll change how the laser cannon moves to make it smoother.

Step 8: Improve the Cannon's Movement

In the current version of the game, the laser cannon moves by a fixed number of pixels each time the player presses the left or right arrow keys. If you want to move the cannon faster across the screen, you need to make these jumps bigger. In this step of the tutorial, you'll change the way the cannon moves to make its movement smoother. It may even increase your chances of saving your planet from the alien invasion!

You can download the code as it'll look at the end of this step from the folder named `source_code_step_8/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

The laser cannon moves differently from how lasers and aliens move. The cannon moves in one large step in the functions `move_left()` and `move_right()`. These functions are called when you press the arrow keys. If you want the cannon to have a smoother motion, you need to move it in smaller increments continuously. To achieve this, you need to move the laser cannon in the game loop like the lasers and aliens.

However, you need to control whether the laser cannon is moving to the left, to the right, or whether it's stationary. To do this, you can create a new variable to store which direction the cannon is moving at any point during the game. Here are the three values this new variable can have:

- 1 represents movement to the right
- -1 represents movement to the left
- 0 indicates the cannon is not moving

You can then change this value in `move_left()` and `move_right()`, and use it to control the cannon's movement in the game loop:

Python `turtle_invaders.py`

```
import random
import time
import turtle

CANNON_STEP = 3

# ...

# Create laser cannon
cannon = turtle.Turtle()
cannon.penup()
cannon.color(1, 1, 1)
cannon.shape("square")
cannon.setposition(0, FLOOR_LEVEL)
cannon.cannon_movement = 0  # -1, 0 or 1 for left, stationary, right

# ...

def move_left():
    cannon.cannon_movement = -1

def move_right():
    cannon.cannon_movement = 1

# ...

# Game loop
alien_timer = 0
game_timer = time.time()
score = 0
game_running = True
while game_running:
    time_elapsed = time.time() - game_timer
    text.clear()
    text.write(
        f"Time: {time_elapsed:5.1f}s\nScore: {score:5}",
        font=("Courier", 20, "bold"),
    )

    # Move cannon
    new_x = cannon.xcor() + CANNON_STEP * cannon.cannon_movement
    if LEFT + GUTTER <= new_x <= RIGHT - GUTTER:
        cannon.setx(new_x)
        draw_cannon()

    #  ...
```

The left and right arrow keys are still bound to the same functions, `move_left()` and `move_right()`. However, these functions change the value of the data attribute `cannon.cannon_movement`.

You set this as a data attribute rather than a standard variable. A data attribute is similar to a variable, but it's attached to an object. The `Turtle` object already has other data attributes defined in the `Turtle` class, but `.cannon_movement` is a custom data attribute you add in your code.

The reason you need to use a data attribute instead of a standard variable is so you can modify its value within functions. If you define a standard variable for this value, the functions `move_left()` and `move_right()` would need to return this value. But these functions are bound to keys, and they're called from within `.onkeypress()`, so you can't access any data they return.

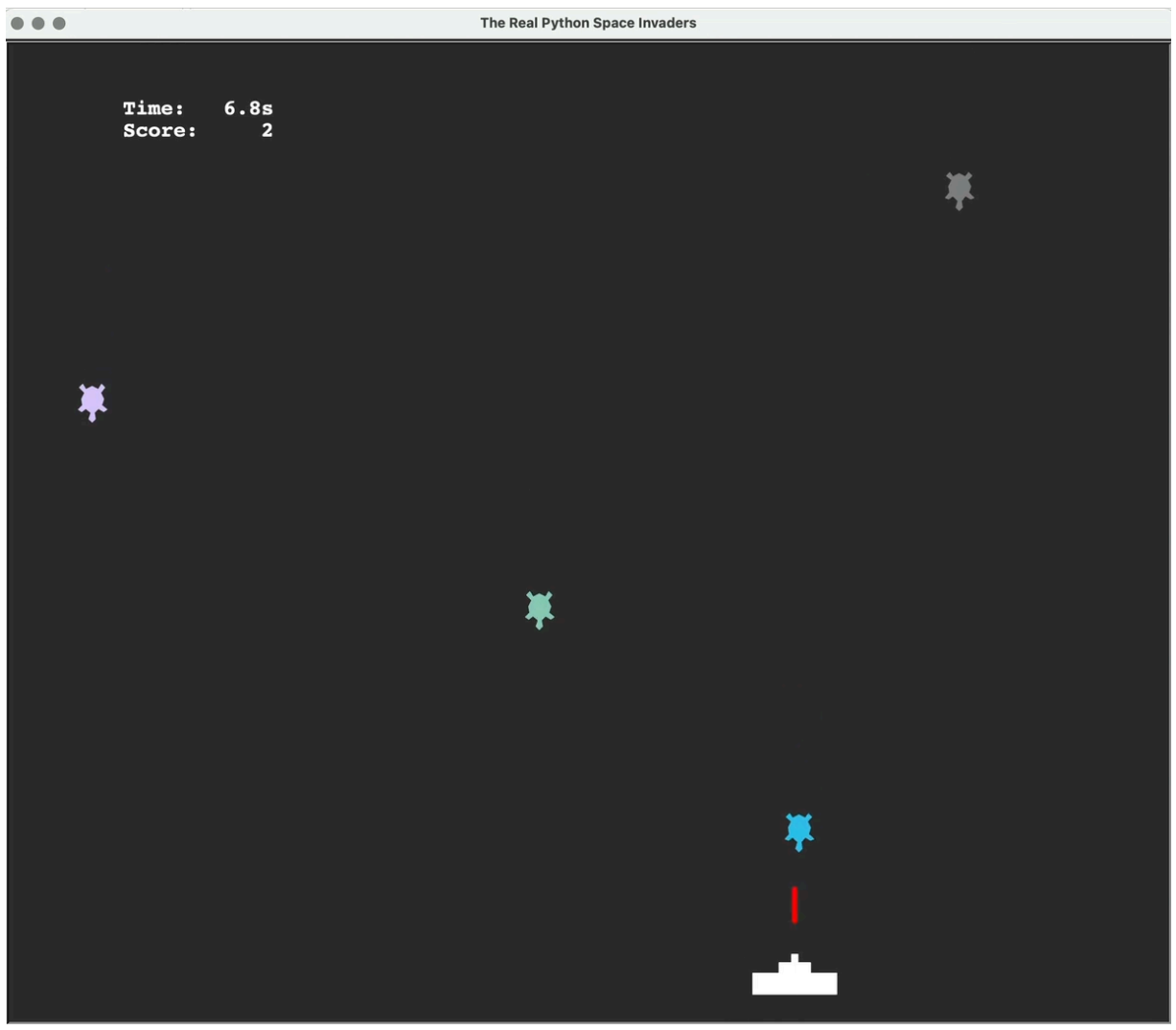
You also decrease the value of `CANNON_STEP` in the last modification to slow down the cannon when it moves. You can pick a value that's suitable for your system.

When you run this code, you'll notice that you can set the cannon to move either left or right, and you can change its direction. But once the cannon starts moving, it never stops. You never set `cannon.cannon_movement` back to 0 in the code. Instead, you want this value to get back to 0 when you release the arrow keys. Use `.onkeyrelease()` to achieve this:

Python turtle_invaders.py

```
# ...

def move_left():
    cannon.cannon_movement = -1

def move_right():
    cannon.cannon_movement = 1

def stop_cannon_movement():
    cannon.cannon_movement = 0

# ...

# Key bindings
window.onkeypress(move_left, "Left")
window.onkeypress(move_right, "Right")
window.onkeyrelease(stop_cannon_movement, "Left")
window.onkeyrelease(stop_cannon_movement, "Right")
window.onkeypress(create_laser, "space")
window.onkeypress(turtle.bye, "q")
window.listen()

# ...
```

Now you can press and hold the arrow keys to move the cannon. When you release the arrow keys, the cannon stops moving:



The cannon's movement is much smoother. The game works perfectly. However, if you try to run this game on different computers, you'll notice that the speed of the game varies. To account for this, you may need to adjust the speed values for the cannon, laser, and aliens depending on the kind of computer running the game.

In the final step of this tutorial, you'll set the game's frame rate so that you can control how fast the game runs on any computer.

Step 9: Set the Game's Frame Rate

Each iteration of the `while` loop represents a frame of the game. There are several lines of code that need to be executed in each iteration of the loop. The time it takes for the program to go through all the operations in the loop depends on how fast your computer is, and whether your operating system is busy with other tasks.

In this step, you'll fix the duration of a frame by choosing a frame rate.

You can download the code as it'll look at the end of this step from the folder named `source_code_final/` in the link below:

> **Get Your Code: Click here to download the free sample code** that shows you how to build a Python turtle game.

Here you'll set the frame rate in frames per second, and work out the time for one frame at that frame rate:

Python turtle_invaders.py

```
import random
import time
import turtle

FRAME_RATE = 30  # Frames per second
TIME_FOR_1_FRAME = 1 / FRAME_RATE  # Seconds

# ...
```

Next, measure the time it takes to run one iteration of the `while` loop. If this time is shorter than the time required for one frame, pause the game until you reach the required time for one frame. The loop can then proceed with the next iteration. As long as the time it takes to run the `while` loop is shorter than the time required for one frame, every frame will take the same amount of time once the pause is completed:

Python turtle_invaders.py

```
import random
import time
import turtle

FRAME_RATE = 30  # Frames per second
TIME_FOR_1_FRAME = 1 / FRAME_RATE  # Seconds

CANNON_STEP = 10
LASER_LENGTH = 20
LASER_SPEED = 20
ALIEN_SPAWN_INTERVAL = 1.2  # Seconds
ALIEN_SPEED = 3.5

# ...

# Game loop
alien_timer = 0
game_timer = time.time()
score = 0
game_running = True
while game_running:
    timer_this_frame = time.time()

    # ...

    time_for_this_frame = time.time() - timer_this_frame
    if time_for_this_frame < TIME_FOR_1_FRAME:
        time.sleep(TIME_FOR_1_FRAME - time_for_this_frame)
    window.update()

# ...
```

This is the right time to adjust the game parameters to suit your preferences, such as the values that determine the speed of the cannon, lasers, and aliens.

And here's the final version of the game:

You'll find the complete game code in the collapsible section below:

Complete Game Code Show/Hide

Python turtle_invaders.py

```
import random
import time
import turtle

FRAME_RATE = 30  # Frames per second
TIME_FOR_1_FRAME = 1 / FRAME_RATE  # Seconds

CANNON_STEP = 10
LASER_LENGTH = 20
LASER_SPEED = 20
ALIEN_SPAWN_INTERVAL = 1.2  # Seconds
ALIEN_SPEED = 3.5

window = turtle.Screen()
window.tracer(0)
window.setup(0.5, 0.75)
window.bgcolor(0.2, 0.2, 0.2)
window.title("The Real Python Space Invaders")

LEFT = -window.window_width() / 2
RIGHT = window.window_width() / 2
TOP = window.window_height() / 2
BOTTOM = -window.window_height() / 2
FLOOR_LEVEL = 0.9 * BOTTOM
GUTTER = 0.025 * window.window_width()

# Create laser cannon
cannon = turtle.Turtle()
cannon.penup()
cannon.color(1, 1, 1)
cannon.shape("square")
cannon.setposition(0, FLOOR_LEVEL)
cannon.cannon_movement = 0  # -1, 0 or 1 for left, stationary, right

# Create turtle for writing text
text = turtle.Turtle()
text.penup()
text.hideturtle()
text.setposition(LEFT * 0.8, TOP * 0.8)
text.color(1, 1, 1)

lasers = []
aliens = []

def draw_cannon():
    cannon.clear()
    cannon.turtlesize(1, 4)  # Base
    cannon.stamp()
    cannon.sety(FLOOR_LEVEL + 10)
    cannon.turtlesize(1, 1.5)  # Next tier
    cannon.stamp()
    cannon.sety(FLOOR_LEVEL + 20)
    cannon.turtlesize(0.8, 0.3)  # Tip of cannon
    cannon.stamp()
    cannon.sety(FLOOR_LEVEL)

def move_left():
    cannon.cannon_movement = -1

def move_right():
    cannon.cannon_movement = 1

def stop_cannon_movement():
    cannon.cannon_movement = 0

def create_laser():
    laser = turtle.Turtle()
    laser.penup()
    laser.color(1, 0, 0)
    laser.hideturtle()
    laser.setposition(cannon.xcor(), cannon.ycor())
    laser.setheading(90)
    # Move laser to just above cannon tip
    laser.forward(20)
    # Prepare to draw the laser
```

```
    laser.pendown()
    laser.pensize(5)

    lasers.append(laser)

def move_laser(laser):
    laser.clear()
    laser.forward(LASER_SPEED)
    # Draw the laser
    laser.forward(LASER_LENGTH)
    laser.forward(-LASER_LENGTH)

def create_alien():
    alien = turtle.Turtle()
    alien.penup()
    alien.turtlesize(1.5)
    alien.setposition(
        random.randint(
            int(LEFT + GUTTER),
            int(RIGHT - GUTTER),
        ),
        TOP,
    )
    alien.shape("turtle")
    alien.setheading(-90)
    alien.color(random.random(), random.random(), random.random())
    aliens.append(alien)

def remove_sprite(sprite, sprite_list):
    sprite.clear()
    sprite.hideturtle()
    window.update()
    sprite_list.remove(sprite)
    turtle.turtles().remove(sprite)

# Key bindings
window.onkeypress(move_left, "Left")
window.onkeypress(move_right, "Right")
window.onkeyrelease(stop_cannon_movement, "Left")
window.onkeyrelease(stop_cannon_movement, "Right")
window.onkeypress(create_laser, "space")
window.onkeypress(turtle.bye, "q")
window.listen()

draw_cannon()

# Game loop
alien_timer = 0
game_timer = time.time()
score = 0
game_running = True
while game_running:
    timer_this_frame = time.time()

    time_elapsed = time.time() - game_timer
    text.clear()
    text.write(
        f"Time: {time_elapsed:5.1f}s\nScore: {score:5}",
        font=("Courier", 20, "bold"),
    )
    # Move cannon
    new_x = cannon.xcor() + CANNON_STEP * cannon.cannon_movement
    if LEFT + GUTTER <= new_x <= RIGHT - GUTTER:
        cannon.setx(new_x)
        draw_cannon()
    # Move all lasers
    for laser in lasers.copy():
        move_laser(laser)
        # Remove laser if it goes off screen
        if laser.ycor() > TOP:
            remove_sprite(laser, lasers)
            break
        # Check for collision with aliens
        for alien in aliens.copy():
            if laser.distance(alien) < 20:
                remove_sprite(laser, lasers)
```

```
                remove_sprite(alien, aliens)
                score += 1
                break
    # Spawn new aliens when time interval elapsed
    if time.time() - alien_timer > ALIEN_SPAWN_INTERVAL:
        create_alien()
        alien_timer = time.time()

    # Move all aliens
    for alien in aliens:
        alien.forward(ALIEN_SPEED)
        # Check for game over
        if alien.ycor() < FLOOR_LEVEL:
            game_running = False
            break

    time_for_this_frame = time.time() - timer_this_frame
    if time_for_this_frame < TIME_FOR_1_FRAME:
        time.sleep(TIME_FOR_1_FRAME - time_for_this_frame)
    window.update()

splash_text = turtle.Turtle()
splash_text.hideturtle()
splash_text.color(1, 1, 1)
splash_text.write("GAME OVER", font=("Courier", 40, "bold"), align="center")

turtle.done()
```

Conclusion

You made it to the end of this tutorial and wrote a fully-functioning Space Invaders clone. Along the way, you explored several features of the `turtle` module, many of which are similar to tools you'll find in other graphics packages. You also worked with techniques used for animations and games.

In this tutorial, you learned how to:

- **Design and build** a classic video game
- Use the `turtle` module to **create animated sprites**
- Add **user interaction** in a graphics-based program
- **Create a game loop** to control each frame of the game
- **Use functions** to represent key actions in the game

Now you can play as long as you want and try to beat your high score. Make sure to take breaks!

Next Steps

Once you play the game long enough, you'll get quite good at it. That's the right time to make the game harder. Here are some ideas on how you can extend this `turtle` game further:

- **Add difficulty levels**: You can increase the speed of the aliens and the spawn rate after a certain time interval has passed, or once you destroy a certain number of aliens. This will make the game progressively harder.
- **Limit the number of lasers**: Set a limit on how many lasers are available so you only get a top-up after a certain amount of time has passed, or when you destroy a certain number of aliens. That's the end of keeping the spacebar pressed down!
- **Create a fixed-cannon version**: Modify the game so that the cannon doesn't move but can rotate when you press the arrow keys to shoot in different directions. This makes for a fun variation of the game.

Enjoy being creative and as you write your own version of this game, and feel free to share your versions in the comments below.

Stephen worked as a research physicist in the past, developing imaging systems to detect eye disease. He now teaches coding in Python to kids and adults. And he's almost finished writing his first Python coding book for beginners

» More about Stephen

Each tutorial at Real Python is created by a team of developers so that it meets our high quality standards. The team members who worked on this tutorial are:

Aldren

Brenda

Geir Arne

What Do You Think?

Rate this article:

LinkedIn Twitter Facebook Email

What’s your #1 takeaway or favorite thing you learned? How are you going to put your newfound skills to use? Leave a comment below and let us know.

Commenting Tips: The most useful comments are those written with the goal of learning from or helping out other students. Get tips for asking good questions and get answers to common questions in our support portal.

Looking for a real-time conversation? Visit the Real Python Community Chat or join the next “Office Hours” Live Q&A Session. Happy Pythoning!

Keep Learning

Related Tutorial Categories: basics python