pytch_

https://pytch.org/
info@pytch.org

Blue Invaders game

Open starting project

This workshop started with a Pytch team member demonstrating the first few stages of making this game. These instructions pick up from the end of that segment. You can start with the following link, which will open the project as it is after the demonstration, ready for the tasks on this worksheet:

- https://www.pytch.org/app/suggested-demo/workshops/blue-invaders-1

Click the "Demo" button once this link has opened.

Finish making the game

There are a few missing pieces before this is a playable game. See how you get on with these tasks:

Task 1 part 1 (Stage)

Problem: The player can see their "score" on the screen, but not their "lives".

Look in the Stage's code where we created the *score* variable with value 0 and we wrote the line to show it on the Stage — this is where you'll need to make changes.

You need to create a *lives* variable as we did with the *score* and give it an initial value, for example the value 3. Then you want to show it on the Stage.

To also show the *lives* variable with the *score* one, you can use code very similar to the

```
pytch.show_variable(Stage, "score", right=236)
```

line we wrote earlier. The code *right=236* puts the score display at the top-right corner of the screen. You can leave out the highlighted part altogether to put the display at the top-left of the screen.

Add your new code for *lives* underneath the existing line for *score*.

Task 1 part 2 (Sprite)

Problem: the lives at the moment are not used to check whether the game still needs to keep going. The game carries on forever, but we want it to keep going only when lives are bigger than zero.

In your current program, the *lives* variable is always 3, but you will soon develop some code to make the player lose lives too. So, you need your existing code to be ready for this. In the sprite's "play-game" script, the code currently runs forever, using a *while True* loop. You need to change this loop into something where your code runs only when lives are bigger than 0. The help to the left of the Code pane will help you find the Python code you need for this with an example you can adapt to your case — look in the *Control* section for the *Repeat until* help.

Task 2 (Sprite)

Problem: The sprites are all enemy invaders (blue) we want to destroy, and never friendly aliens (green) we want to leave alone. (We want that to be random.)

Look in the Sprite's code where we added the line to always switch the costume to the first one (0) — this is where you'll need to make changes.

Look in the *Costumes* tab to see what "costumes" the sprite has. Each sprite can "wear" any one of these costumes.

To start with, change the 0 to a 1 in the

```
self.switch_costume(0)
```

code, then click the green play button, to see what that number does. Looking in the *Costumes* tab, Python counts list positions from zero, so "costume 0" is the first one, as we mentioned before, which is the enemy alien (blue).

Now create a new variable *costume_number* just before the *switch_costume()* line:

```
costume_number = random.choice([0, 1])
```

This new line asks Python to choose randomly from a list of two numbers (0 and 1), and create a variable *costume_number* to refer to the result. Use this variable instead of the constant 0 in *self.switch_costume(0)*, to get a random mixture of enemy (blue) and friendly (green) aliens.

Task 3 (Sprite)

Problem: All the sprites move at the same speed.

Look in the sprite's code for this line:

```
self.glide_to_xy(self.x_position, -200, 4.0)
```

The "4.0" means how long (4 seconds) to take to glide down the screen. Instead of always taking 4 seconds, we want to take a random amount of time between, say, 3.0 seconds and 5.0 seconds, including numbers with a fraction part. *(continued on next page)*

The Python function

```
random.uniform(lowest, highest)
```

will choose a random number like this for you. Instead of *lowest* and *highest*, you'll type actual numbers. Create a new variable *glide_time* for this random value, and change the *self.glide_to_xy()* line to use this *glide_time* variable.

Task 4 part 1 (Sprite)

Problem: Nothing happens when the player destroys a friendly alien.

Look for the Sprite's "when clicked" script. Under where the code says

```
pass
```

you can write some code which runs when the player clicks a friendly alien. Your code needs to do two things, as follows. Test the game after you've done each one.

First: Subtract 1 from the variable *Stage.lives*. You can use code very similar to the line

```
Stage.score += 10
```

— recall that this line adds 10 to the variable *Stage.score*.

Second: Check if the value of *Stage.lives* is now zero. You'll use an *if* statement — there's an example at the top of the script. If *Stage.lives* is zero, that's game over, and we want to "broadcast a message" to all the clones, so they hide themselves. Look in the Stage's code for an example of how broadcasts work, and add code to broadcast a message like "game-over".

Task 4 part 2 (Sprite)

Problem: The other sprites don't immediately vanish when the game is over.

We want each sprite to hide itself when it receives the message broadcast by the code you wrote in the previous task. So we need a new script that hides the sprites when it receives the message "game-over" (or the message string you used, if different).

Task 5 (Sprite)

Problem: The game would look better if the aliens exploded.

For this you'll need to change the script saying what happens when the sprite is clicked. Just before the *self.hide()* line, we can add a short animation to make the alien explode.

The sprite has some costumes which can be worn one after another to make an explosion effect. You can switch to the first explosion image, wait a short while, switch to the next explosion image, and so on. Note: you can do this by copying and pasting your code, but if you have time, think about how to avoid the repetitive code.

The help to the left of the Code pane will help you find the Python code you need for waiting — look in the *Control* section. You can also find information on *self.switch_costume()* — this is in the *Looks* section. Or you might find *self.next_costume()* useful. *(continued on next page)*

The "when-clicked" code needs to be updated, because of the new costumes the sprite might be wearing.

Customise the game

Try some of these suggestions, or work on your own ideas!

- Add a special red alien which is worth 50 points. The sprite already has a costume for this. Or you can use an image from the web instead. Think about how to make the red alien appear, and how to make the game behave correctly when it's clicked.
- Lose points if you miss the sprite (i.e., if you click on the Stage instead).
- Make the red alien rarer than the other two sprites.
- Add some left-to-right randomness to where the sprites start.
- Make the aliens step from side to side as they glide down the screen.
- Lose a life if you don't destroy an enemy alien before it reaches the bottom.
- Make more clones.
- Find different images on the web for the sprite's costumes. You can add costumes to the sprite with the **Add from this device +** button at the bottom of the **COSTUMES** tab.

Compare to our version

This link will open the project with all tasks (but not the challenges) completed:

- https://www.pytch.org/app/suggested-demo/workshops/blue-invaders-2

This is our own version, but of course there are many ways of personalising the project or solving the tasks — there's no such thing as "the right answer". If you've made your own version, please feel free to share your project or ideas with us.

Feedback

Any comments or suggestions to info@pytch.org please!

Pytch Team, 2024-11-16