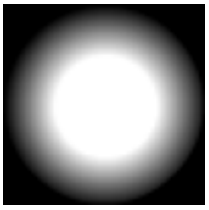
Part 13 – Shooting and Spawners

Time for our hero to defend himself in the quest to reach the prize. He needs to shoot his gun.

What will we use for a bullet? There a few choices, but we'll go for `particle.png` in the `orx/tutorial/data/object` in the Orx project. And copy this to our `data/texture` folder.



Once copied, define it in the config:

```
[BulletGraphic]
Texture = particle.png

[BulletObject]
Graphic  = BulletGraphic
Speed    = (500, 0, 0)
LifeTime = 1.0
Scale    = 0.25
```

Any BulletObject that is created will be scaled down a little, sent flying off to to the right, and can only exist for 1 second before being auto-destroyed.

What is the best way to get our hero to shoot lots of these `BulletObjects`? A spawner is the thing to use.

A spawner can be attached to an object (like our hero). And it can be told to spawn a specific type of object (like a bullet). Let's take this approach to start with. It won't be perfect, but we can always improve it later.

Create a spawner for the hero:

```
[BulletSpawner]
Object          = BulletObject
WaveSize        = 1
WaveDelay       = 0.1
Position        = (10, -7, 0)
```

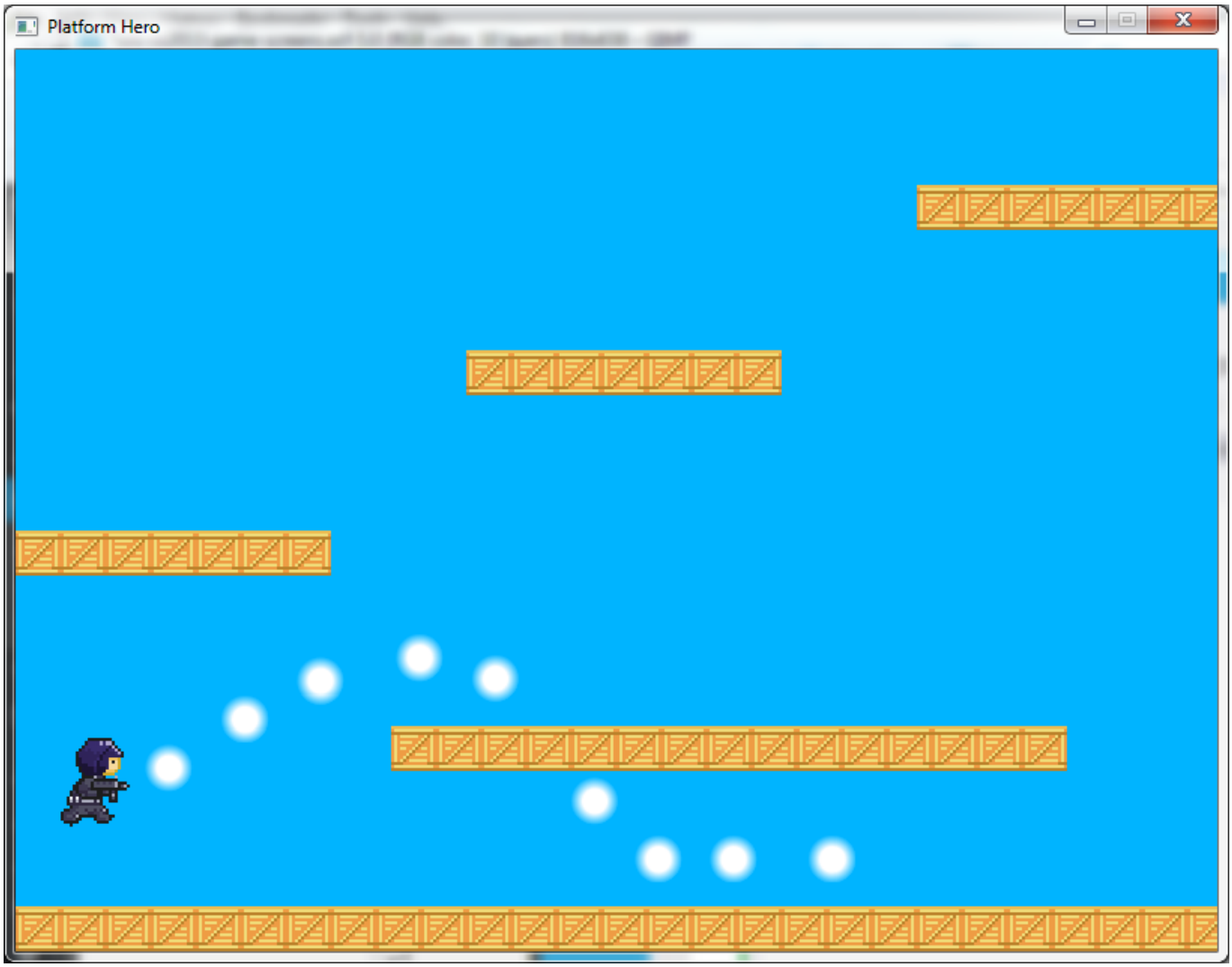
This spawner can shoot `BulletObjects`. It can shoot one at a time with 0.1 seconds between shots.

Time to attach it to the `HeroObject`:

```
[HeroObject]
Graphic      = HeroGraphic
Position     = (-350, 100, 0)
```

```
Scale        = 2
AnimationSet = HeroAnimationSet
Body         = HeroBody
Spawner      = BulletSpawner
```

Run it and your hero will be shooting:



Great, so that's working for us. Now, how to stop it from shooting. The function that is available to us for this purpose is: orxObject_Enable()

If we disabled our hero object, yes it would stop him from shooting, but it would also make him disappear. Not quite what is expected.

There are a number of ways to solve this situation. Let's go with a simple one.

What if the HeroObject had a single empty child object? And what if that child object had the spawner? Because it would be invisible, it wouldn't matter if it was turned on or off.

Define an empty object as the hero's gun:

```
[HerosGun]
```

```
Spawner          = BulletSpawner
Position         = (0, 0, 0)
```

Remove the spawner from the hero object, and make `HerosGun` a child of `HeroObject`:

```
[HeroObject]
Graphic      = HeroGraphic
Position     = (-350, 100, 0)
Scale        = 2
AnimationSet = HeroAnimationSet
Body         = HeroBody
ChildList    = HerosGun
```

When you run this, you'll notice little difference. Our hero is still firing. But now we can turn his gun off at the bottom of the `Init()` function.

But we need a reference to the gun in code, which we don't have yet because [HerosGun] was created automatically with HeroObject.

First declare the gun as a variable just like the hero object:

```
orxOBJECT *hero;
orxOBJECT *herosGun;
```

Then, in the `Init()` function, get the child reference of the gun using the hero object:

```
  hero = orxObject_CreateFromConfig("HeroObject");
  herosGun = (orxOBJECT*)orxObject_GetChild(hero);
  orxObject_CreateFromConfig("Scene");
```

Now that you have the reference, turn off the `herosGun` immediately after creating it:

```
  hero = orxObject_CreateFromConfig("HeroObject");
  herosGun = (orxOBJECT*)orxObject_GetChild(hero);
  orxObject_Enable(herosGun, orxFALSE);
  orxObject_CreateFromConfig("Scene");
```

Compile and run.

Cool, he stopped shooting. The last thing, turn on, and off the shooting based on pressing the left control key in the `Update()`:

```
if (orxInput_IsActive("Shoot"))
{
    orxObject_Enable(herosGun, orxTRUE);
}  else {
    orxObject_Enable(herosGun, orxFALSE);
}
```

Compile and run.

Hmmm... His bullets don't go left when the hero faces left. Change the spawner by adding `ObjectSpeed` and `UseRelativeSpeed` properties:

```
[BulletSpawner]
Object           = BulletObject
WaveSize         = 1
WaveDelay        = 0.1
Position         = (0, 0, 0)
ObjectSpeed      = (500, 0, 0)
UseRelativeSpeed = true
```

The `ObjectSpeed` property overrides the `Speed` property in the `BulletObject` (no it's not really needed anymore) and the `UseRelativeSpeed` means the `ObjectSpeed` with be applied relative to the direction of the spawner.

Compile and Run.

Fantastic! Our hero can now run, jump, and shoot.

Next: Part 14 - FX.

- Part 1 – Downloading Orx
- Part 2 – How Orx works
- Part 3 – Setting up a new game project
- Part 4 – A tour of an Orx project
- Part 5 – Viewport and the camera
- Part 6 – Objects
- Part 7 – Spritesheets and Animation
- Part 8 – Platforms and Texture Repeating
- Part 9 – Physics
- Part 10 – Input Controls
- Part 11 – Running and Standing
- Part 12 – Changing Direction
- Part 13 – Getting our hero to shoot
- Part 14 – FX
- Part 15 – Collision Events.
- Part 16 – Jelly Monsters
- Part 17 – Timeline Tracks
- Part 18 – Exploding Monsters
- Part 19 – The Hero's survival.
- Part 20 – Text and Game Over

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/en/guides/beginners/shooting_and_spawners?rev=1731935164

Last update: **2025/09/30 17:26 (14 months ago)**