Exploders

Classes

There are a few different exploders in *Scripts/exploders*:

1. Exploder (base class of all exploders. Is responsible for 3d ray-based physics and launching components)
2. Exploder2D (is responsivbe for 2d ray-based physics)
3. SphereExploder (simple spherical exploder. Working example of how to implement your own physics based on Exploder)

Parameters

- **Explosion Time**
  Start time of the explosion
- **RandomizeExplosionTime**
  Delays explosion for random value between 0 and **RandomizeExplosionTime** from **Explosion Time**
- **Radius**
  Radius of the explosion. Doesn't change force, only changes radius of influence.
- **Power**
  Power of the explosion.
- **Probe Count**
  Count of probes emitted in one physics iteration (check out **Emulation** chapter for more info)
- **Explode Duration**
  Changes the duration of physics effects

Components

This classes are provided as Components in *Scripts/effects/components*:

1. **ExploderComponent**
   (base abstract class of all exploder components)
2. **LightComponent**
   (applies simple light effects)
3. **ParticleComponent**
   (instantiates a passed in prefab and launches particle effects that are children of the prefab's head object)
4. **PseudoVolumetricComponent**
   (instantiates given count of Game Objects passed in and sets up their PseudoVolumetricExplosion component)
5. **VolumetricComponent**
   (adds a particle system as component of exploded object and creates on base of it a heatwave)
6. **VolumetricComponent2D**
   (does the same as previous, but in 2D)

Pseudo Volumetric Component

The component itself is pretty simple. Just attach it to desired gameObject and pass in *Volumetric* prefab from *Prefabs* folder.
But *Volumetric* prefab is complicated. The mechanism of how it works is explained in this article.

Emulation

Physics

Physics is emulated pretty straightforward:

1. every FixedUpdate an iteration of physics emulation is run.
2. from position of the exploded object **Probe Count** probes are emited.
3. every probe is a ray with random initial direction and initial distance of **Radius.** The ray is casted. If something was hit:
    a. if a rigidBody is hit, the force with scalar value of
    power * Time.deltaTime / probeCount
    is applied at hit.Position. (power is divided by probeCount to make real explosion power independend)
    b. if a non-rigidBody is hit, the ray is refracted in random direction, which though cannot be directed against hit.normal

Heatwave

Heatwave is a bit more tricky than physics. It has some cheats that are implemented to optimize it's behaviour and speed.
First let's assume that the emulation is already running for a few frames (we'll skip the init part). And let's talk about the behaviour only, not about the particle color/size.

Each particle has it's position and it's direction and it's hitCount.
In an update iteration following happens:

1. **emission** * **Time.deltaTime** particles are emitted by particle system. For each particle:
    a. randomly choose one of already created particles.
    b. copy hitCount, position and direction values.
    c. change the direction a little bit
2. particles are moved in their directions. If they hit on their way some obstacle, they simply stop and change their directions to in random, which though cannot be directed against hit.normal. And hitCount increases
3. **Teleportation Iteretions** times: For each particle:
    a. randomly choose one of the particles.
    b. if their hitCounts differ in **Teleportation Threshold** times, current particle is teleportated to other particle. This is needed to avoid particles "indoors" accumulation

summarizing the optimizations:

- Created particles don't really emulate their whole way, they are just "mixed in" to other particles.

- If a particle is stuck in one place, it is teleported to a free particle.

Now. This works only if there already are some particles in the scene. But how do we emit them?

- **Emission** particles are generated in exploded object's position at start of the emulation. With random directions assigned.
- for first **Center Emission Duration** seconds, every second **Center Emission** particles are emitted. And they calculate their positions with no cheating

That's it!
Thanks for your attention.
If you have some questions, feel free to write at mischapanin@gmail.com