# Participants

Each group should include 3 to 4 participants.  Please enter the following information for each participant:

|  |  |
| --- | --- |
| Name (First, Last) | DigiPen Email ([@digipen.edu](mailto:@digipen.edu)) |
|  |  |
|  |  |
|  |  |
|  |  |

# Topic: *Particles!*

You have been asked to implement a simple particle system consisting of the following classes:

* **ParticleEmitter**
  + A component derived from Component.h.
  + Contains variables for spawning and updating particles.
* **ParticleContainer**
  + A wrapper around a dynamically-allocated array of Particle objects.
  + Contains functions to manage, update, and render multiple particles.
* **Particle**
  + Contains variables for a single particle.
  + Contains functions to update and render a single particle.

# Objective

The objectives for this project are as follows:

* Implement the missing functionality to display a simple “explosion” particle effect.
* (Optional) Modify existing data and code to create alternate particle effects.

# Description

A modified version of the CS230 engine framework has been converted into an external library. You have been given access to only the three classes mentioned above. Some of the functionality, especially in ParticleContainer, has already been provided for you.

As a group, perform the following steps:

* Assign the work to be completed amongst the team members, as you see fit. You might consider working in pairs, or all together, using a concept known as *pair programming*.
* Complete the assigned work.
* Integrate all of the changes into a single project.
* Test and fix any issues discovered.
* Once the project is working correctly, notify the instructor.
* If additional time remains, then experiment with the code and data to see if you can create additional effects.

# Cheat Codes

The following cheats have been implemented to help during development

* <CTRL>+1: Switch to the Particles scene or restart it if it is already running.
* <CTRL>+0: Switch to the Stub scene. Switching to this scene and back to the Particles scene will cause all archetypes to be reloaded. This is useful when only the data files have been changed.

# Files

NOTE: For this exercise, you may freely add/remove/edit the code and header files. However, the primary objective can be achieved without changing the header files.

Please note that the sections of the code that must be modified contain the following comment:

* @@@TEMPORARY:

## Particle.c

This file contains two functions that must be completed:

* **ParticleUpdate**
  + If the particle is active,
    - Decrement the lifetime by dt.
    - If the particle is still active,
      * Update the particle’s position.
    - Else
      * “Kill” the particle.
        + Hint: There is a function for this.
* **ParticleRender**
  + If the particle is active,
    - Call the DGL library function for setting the particle’s transform.
    - Render the mesh.

## ParticleContainer.c/.h

This file contains three functions that must be completed:

* **ParticleContainerIsEmpty**
  + If there are no active particles, then return true, otherwise return false.
* **ParticleContainerIsFull**
  + If there are no free particles, then return true, otherwise return false.
* **ParticleContainerKillParticle**
  + If particles are “recyclable”,
    - Set the particle’s lifetime to 0 (indicates “free”).
    - Increment the number of free particles.
  + Else,
    - Set the particle’s lifetime to a negative value (indicates “dead”).
  + Decrement the number of active particles.

## ParticleEmitter.c/.h

This file contains three functions that must be completed:

* **ParticleEmitterRead**
  + Read the variables defined in the “General Emitter Properties” section of the structure.
    - Hint: For the mesh, read the mesh name and have the MeshLibrary build the mesh.
* **ParticleEmitterUpdate**
  + Update the emit accumulator (accumulator += emitRate \* dt).
  + While there are new particles to spawn,
    - Spawn a particle.
    - Decrement the accumulator.
  + If the container is both “empty” and “full”,
    - Destroy the emitter’s parent entity.
* **ParticleEmitterRender**
  + If the container is NOT empty (active particles exist),
    - Set the shader mode to COLOR.
    - Set the alpha value to 1.0f.
    - Set the tint color.
    - Render all of the particles in the container.
* **ParticleEmitterEmit**
  + If the container is NOT full (free particles exist),
    - Allocate a particle from the particle container.
    - If a particle was successfully allocated, then set EVERY variable in the particle structure:
      * Set the lifetime to a random value in the specified range.
      * Get the transform from the emitter’s parent entity.
        + Hint: Use ComponentGetParent() and EntityHas().
      * Set the position using the transform’s translation.
      * Set the scale to the specified value.
      * Set the rotation to 0.
      * Get a random angle between 0 and 2\*M\_PI.
      * Get a random speed in the specified range.
      * Set the velocity using the angle and speed values.