# CPSC 427 A3: Animation and Physics

Due: 23:59 PM, Friday March 25, 2022

## 1 Introduction

The goal of this assignment is to introduce you to basic 2D animation. You will extend the chicken game you made for Assignment 2 by including in it a basic particle system implementation.

## 2 Template

You should use your own Assignment 2 code as a starting point. You will find comments throughout the files to help you guide in the right direction. Entry points are marked with `TODO A3`. The following classes will be important.

**Particle Animation** You will make your chicken shoot eggs by implementing a CPU particle system that instantiates spherical objects and simulates their path in air.

**Bouncing Eggs** To make the eggs collide with other eggs and characters you will need to add functionality to `PhysicsSystem::step`.

## 3 Required Work (80%)

1. Getting Started
   (a) We recommend pushing your code to a **private** git repository. Commit all your edits from Assignment 2 to git, such that you can track and potentially revert changes.
   (b) Keep a separate copy of your Assignment 2 executable and dlls, to be able to showcase your A2 solutions to TAs on request.
   (c) Play the `a3_reference.mp4` video to get a sense of what a possible assignment solution should look like.

2. Particle Animation (40%, prereq: Rigid Body Physics lecture)

   Implement a particle system which generates eggs that shoot from the center of the chicken every few seconds (adjust the timing for a compelling visual effect). The eggs should have randomized initial directions bounded by a cone pointing away from the chicken (see video) and initial velocities. Their subsequent motion should be driven by a combination of their initial directions and velocity, as well as gravity. Implement this animation in stages:

   (a) Generate periodic eggs that shoot from the center of the chicken and follow a fixed straight-line path at a fixed speed. The `RenderSystem::restart_game()` function provides example code for creating static eggs.

   (b) Randomize initial egg directions and velocities.

   (c) Introduce a new `Physics` component and add it to eggs at creation time. It should indicate that an entity is affected by physics and may store related properties such as object mass.

   (d) Add gravity into the system to produce physically plausible (non-straight) egg paths. Note that the game template computes in pixel units and miliseconds; not meters and seconds as is common in physics;

3. Bouncing Eggs (40%, prereq: Rigid Body Physics lecture)

   (a) Add interaction between pairs of eggs. Detect when two eggs collide based on their radius, and make both bounce using physically plausible bounce direction and speed computations.

   (b) Make eggs bounce when colliding with eagles but pass behind bugs and the chicken (see below for depth handling). Make eggs and eagles bounce on collision assuming both have spherical geometry with a suitable radius and mass. You will have to replace the 'SUPER APPROXIMATE' `collides()` function that is provided in the template.

   (c) Handle the interaction between eggs and other assets. One could implement the above with complex if conditionals, but this is inflexible. Introduce a new component alongside `Physics` such that you can ensure that eagles bounce against eggs but are unaffected by gravity. You should implement it such that there is one component to indicate that an object is affected by gravity and another that it bounces on collision.

   (d) Properly render eggs passing behind the chicken and other entities when their screen coordinates overlap by changing the drawing order based on the depth (the distance from the camera) of the different entities. You can sort all render components by calling `registry.renderRequests.sort(sort_by_depth)`. The sort function is provided by tinyECS. You will have to equip each entity with depth and implement the comparisons function `bool sort_by_depth(Entity i, Entity j)` to compare the depth of two entities.

# 4 Creative Part(20%)

The required code changes described so far will let you earn up to 80% of the grade. To earn the remaining 20% you will need to make the game more appealing by implementing one advanced feature. You may also gain bonus points when exceeding our expectations. **Marks for the advanced features will be granted only if both they and all basic features are fully implemented and functional.** Advanced feature suggestions:

1. Machine-oblivious time-stepping that produces consistent animation and AI across different platforms and computational loads. Create and document a test case that showcases the difference.

2. Make eggs into ellipses or egg-shapes instead of circles, while still correctly handling collisions. Also include the force of the wind blowing in the egg motion computation.

3. Use a single draw call to render all the eggs at once (requires good OpenGL knowledge), e.g., using `glDrawElementsInstanced`. This should be more efficient since while eggs have different positions, they have the same appearance (and hence use the same shader).

Use your imagination to make other additions than the ones listed above, however, please make sure you focus on tasks involving advanced OpenGL rendering, advanced physics, and advanced animation knowledge.

To support both basic and advanced visualization and control features, **you need to add a toggle option where the user switches between the two modes** by pushing the 'a' and 'b' keys ('a' for advanced mode and 'b' for basic mode; either at startup or during the game).

**Document all the features you add in the README.md file you submit with the assignment. Advice:** implement and test all the required tasks first before starting the free-form part.

To get full credit you should add at least one of the advanced features above and make it fully functional **and** free from bugs. The grading of additional bonuses, features, and the size of bonuses will be at the marker's discretion. A bonus is given for solutions that go beyond the suggestions listed above. **Multiple partially implemented features will not receive full credit.**

# 5 Hand-in Instructions

1. Create a folder called "a3". As before, copy all your source files and the CMakeLists.txt as present in the template to this folder (same folder structure; the TA should be able to run CMake and compile). Double check that you include the `shader` folder. Excluded all generated files, such as /build, .vs, /out and the example videos! These would consume a lot of space on our server.

2. In addition, create a README.md file (Markdown language as used on GitHub) that includes your name, CWL, student number, and any information you would like to pass on to the marker.

3. The assignment should be handed in with the exact command `handin cs-427 a3` on a department machine (use SSH to do this remotely).

   This will handin your entire a3 directory tree by making a copy of your a3 directory, and deleting all subdirectories. If you want to know more about this handin command, use: man handin. You can also use the web interface on your myCS page to upload the assignment.

**Recall, do not publish your solution on github or any other place. Neither during the course nor after; both is considered cheating.**