

CPSC 427, A0: A Tiny Entity Component System in C++

Due: 23:59 PM, Friday January 21, 2022

1 Introduction

The goal of this assignment is to introduce you to the entity component system (ECS) pattern, which is commonly used in game development. The assignment is intended to bring everybody on the same page in terms of programming essentials and sets the foundation for the other assignments and the course project. It will also prepare you for implementing your own games more efficiently and conveniently by using the ECS pattern.

2 Code and Installation (10 %)

Download the *A0_template.zip* from the link provided on the course page. The directory `src` contains the entry point `ecs_demo.cpp`. The tinyECS library is one folder down, `tinyECS/tiny_ecs.hpp` and `tinyECS/tiny_ecs.cpp`.

The project uses CMake. If CMake is not already installed, you can download and install it from the CMake website: <https://cmake.org/download/>; or use the package manager of your choice on Linux/macOS systems.

For Windows it should be sufficient to open the repository folder (the one containing the `CMakeLists.txt`) with Visual Studio (you may have to install the VS CMake extension) and pressing *Build*. For a manual CMake setup on Windows follow the instructions below.

In all other cases create an empty build directory named `build` in the `template` folder (`template/build`). The `.gitignore` file is configured to ignore any files with `build` in the path, to avoid tracking temporary files. You can configure the project using the CMake GUI or the command line. For the GUI, enter the assignment template folder (which contains a `CMakeLists.txt` file) as Source and the `build` folder as Build. Then, press *Configure*, and if the configuration is successful, press *Generate*. For the command line, `cd` inside the `build` directory and run:

```
cmake [path_of_assignment_template] -DCMAKE_BUILD_TYPE=[Debug|Release]
```

Now you can build the generated project using ‘make’ from the command line at the top-level directory, or using your favorite IDE or text editor (Visual Studio or Vim). You

will require a compiler that supports C++14. If you use Visual Studio or Xcode, you should specify it in CMake. It will then generate the respective VS/Xcode project files.

Note: running CMake and building the project will copy files and data to the **build** folder. **Do not edit any files in the build folder**, only edit files in the **src** and **shader** folders (create new assets in the **data** folder).

Note: Going forward we strongly recommend you use Windows or Linux operating systems for your assignments/game. Our A1 template does not support MacOS and you are likely to find writing a game more challenging on a Mac.

2.1 Debugging

To verify that the installation was successful, start the program in your favorite IDE. If on Windows and CMake configured for Visual Studio, you can open the Visual Studio `tinyECS.sln` in your build folder.

The demo program should print a few lines to the terminal and then exit. Set a breakpoint in the `main()` function in `ecs_demo.cpp` and run in debug mode until that point to verify that you can use your IDE's debugging capabilities. Make yourself familiar with the inspection of local and global variables and stepping through instructions in debug mode.

2.2 Entity Component System (ECS) Basics

The ECS pattern stores the entire game state into components that are associated to entities. Each entity uniquely identifies an object, such as a chicken or eagle in the game. Whereas the components equip entities with properties, such as name and movement capabilities.

We created a simple example in the `main()` function in `ecs_demo.cpp` that implements dog, fish, and turtle characters, once using object oriented inheritance and once using the ECS. Inspect the provided toy example to get familiar with creating components in tinyECS by calling `insert()` or `emplace()` and retrieving components with the `get()` function. It is also easy to loop over all entities or components of a type. Examples are given for each operation.

Task 1: (50 %) To get proficient with ECS, add a "Duck" that can walk, swim, and fly. Moreover, remove the dog with `registry.remove_all_components_of(...)`. After that, change the name of the fish to "Old Fish" (it aged a lot during the program execution).

This toy example has no graphical output but prints to the terminal. Make sure that your changes are reflected in the output. In particular, when modifying properties of components stored in the ECS registry, make sure to work on the reference returned by `get` or create a pointer to it. It is a common mistake to work with a copy of the object, which does not change the original properties stored in the ECS registry.

2.3 The tinyECS Library and C++ Templates

The tinyECS implementation uses a template class to collect components of an arbitrary type and to associate each component with an entity without having to duplicate code.

The central part in our tinyECS implementation is the `ComponentContainer` defined in `tiny_ecs.hpp`. It is merely an `std::map` that maps entities to components stored in tightly packed containers. It could be implemented as follows with Entities having a unique integer `id`:

```
template <typename Component>
struct ComponentContainer
{
    // Map from Entities to Components
    std::map<unsigned int, Component> entity_component_map;

    // Inserting a Component c associated to Entity e
    void insert(Entity e, Component&& c) {
        entity_component_map[e.id] = c;
    };

    // Check if entity has a component of type 'Component'
    bool has(Entity e) {
        return entity_component_map.count(e.id) > 0;
    };

    // Return the component of an entity
    Component& get(Entity e) {
        return entity_component_map[e.id];
    };

    // A wrapper to return the component of an entity
    void remove(Entity e) {
        entity_component_map.erase(e.id);
    };
};
```

In practice there are multiple strategies used to increase efficiency; additional steps are taken to make the memory layout linear, hasmaps are used to lookup the component associated to an entity in constant time, unnecessary copy operations are avoided by using move operations, and parameter packs (a C++11 feature) are used for the `emplace` function. It helps to think of it as a simple `std::map`, however, do inspect the tinyECS code (`tinyECS/tiny_ecs.hpp` and `tinyECS/tiny_ecs.cpp`) to get an idea of the inner workings.

As for the standard `std::map` container, the `ComponentContainer` can be explicitly instantiated for any `Component` class, for instance for the `Swim` and `Walk` classes,

```
ComponentContainer<Swim> registry_water_animals;
ComponentContainer<Walk> registry_land_animals;
registry_water_animals.insert(fish_entity, Swim());
registry_land_animals.insert(horse_entity, Walk());
```

We group all the containers in the game in a RegistryECS class. Don't forget to add them there when you add new components to your game.

Task 2: (50 %) The tinyECS can be used to store data in the array of structs (AoS) or struct of arrays (SoA) data layout; just by adding differently structured components. To enable animal motion, we would like to equip animals with variables storing the position x and the velocity y (for simplicity, you can assume a 1D game, where both are scalars).

1. Create and add one or more new components to give the fish and duck velocity and position information with an AoS layout (the 'array' runs over entities; fish & duck).
2. Create new components to give the turtle position and velocity information in the SoA layout (the 'array' still runs over all entities; in this case a single turtle).

Hint1: Inspect the tinyECS implementation to see which part can take the role of the array. **Hint2:** The term "array" can be misleading, any contiguous section of memory can be thought of as an array. Focus on how to use as much of the existing code to pack your data as tightly as possible. In short, don't add more structures than you absolutely need to. **Hint3:** You will have to add new component types to the RegistryECS class but do not edit any of the other tinyECS files.

3 Hand-in Instructions

1. Create a folder called `cs-427` with subfolder `a0`. Copy all your source and CMake files as present in the template to this folder (same folder structure). Double check that you excluded all generated files, such as `/build`, `.vs`, `/out`! These would consume a lot of space on our server. But do include the CMakeLists file as we need it to compile your code, the rule of thumb is, if it's ignored by the `.gitignore` file, you should ignore it as well.
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 a0` on a department machine (use SSH to do this remotely).

This will handin your entire a0 directory tree by making a copy of your a0 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.