Adding Components and Systems

1. Introduction

This tutorial introduces the steps required to add simple physics to any **`Entity`**, and it will include adding a new **`Component`** and a new **`System`**.

You should extend your code from the previous lab.

2. Designing the Component and System

We are going to add the functionality of simple physics in the form of linear motion (no gravity) to any **`Entity`**. Simple linear motion can be defined as:

```
newPos = oldPos + velocity*dt
```

where **`dt`** is the time in seconds since the last update.

Therefore, we will need a new **`Component`** to hold the velocity information and then a new **`System`** to calculate the motion.

3. Exercises

Please attempt these exercises, but if you get stuck or you are confused then ask for help during the scheduled lab times.

ComponentVelocity

1. Create a new component called **`ComponentVelocity`**. This should take the same form as the **`ComponetPosition`** (in fact I suggest that you copy the **`ComponentPosition`** and rename the class and attributes appropriately for the velocity).

2. You will need to add a new component type to the **`ComponentTypes`** enum in the **`IComponent.cs`** file and return this from the **`ComponentType()`** method of your new **`ComponentVelocity`** class.

SystemPhysics

3. Create a new system called **`SystemPhysics`**. This should take the same form as the **`SystemRender`** (in fact I suggest that you copy the **`SystemRender`** and rename the class to **`SystemPhysics`**).

4. You need to define the MASK for this system. The simple linear motion requires a position, a velocity, and a time interval. The time interval can be obtained from the **`GameScene`** class, so we need to define the MASK so that an **`Entity`** has to have both a **`ComponentPosition`** and a **`ComponentVelocity`**.

5. Rename the **`Draw()`** method (if you copied the **`SystemRender`** class) to **`Motion`** and change the parameters appropriately to take a **`ComponentPosition`** and a **`ComponentVelocity`**. Add the code to calculate the new position based on the provided velocity. For the time step (**`dt`**) use the one in the **`GameScene`** class.

6. Change the code in the **`OnAction()`** method so that you obtain both the **`ComponentPosition`** and the **`ComponentVelocity`** from the **`Entity`** and then pass these to the **`Motion()`** method. **NOTE**: You **MUST** pass the reference of the Components themselves and not the values. If no entity moves when you run your game then you are using the values and not the reference – ask for help!

7. Add the new **`System`** to the **`SystemManager`** in the **`CreateSystems()`** method in the **`GameScene`** class.

8. Finally, add the new **`ComponentVelocity`** to one of the **`Entity`** created in the **`CreateEntities()`** method in the **`GameScene`** class. You now have a moving entity.