Videregående udvikling af 3D applikationer

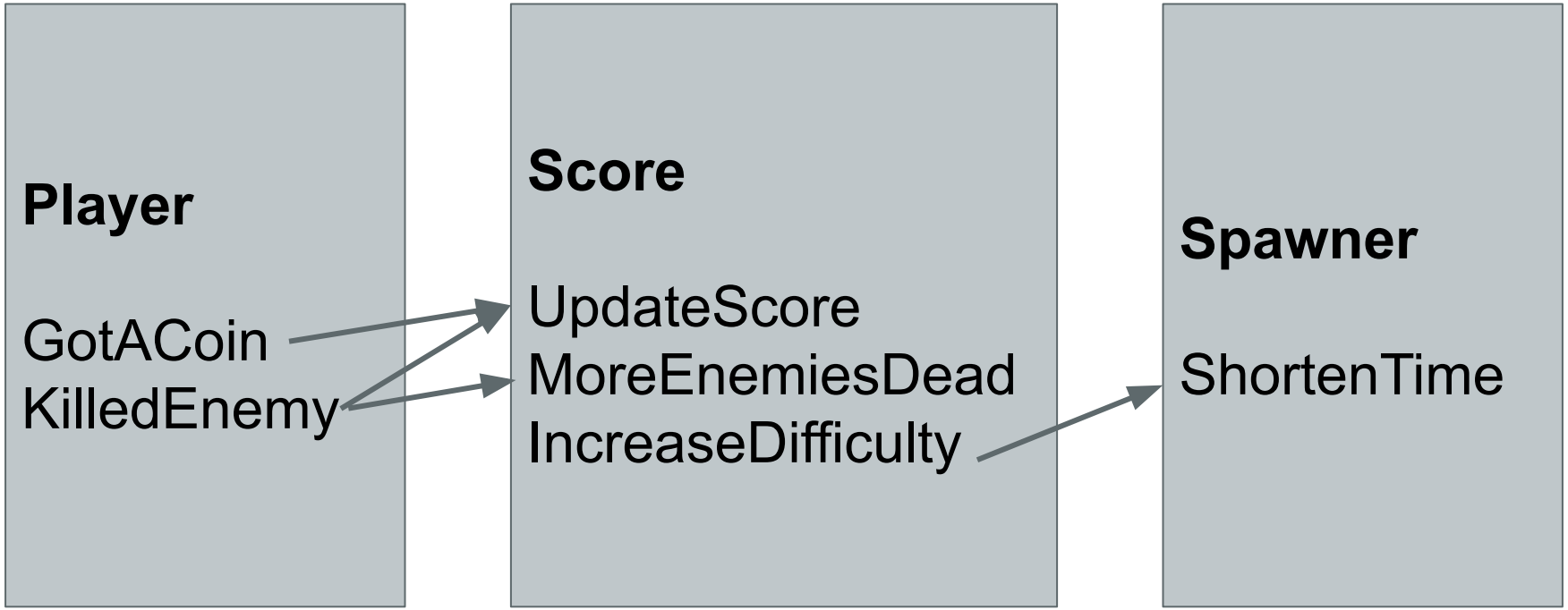
Lecture 11: Events

Events

The problem

- Something happens in the program/game
- Other parts of the code have to react to that, how do we do that?

Example, this is what we want



How would you structure the code?

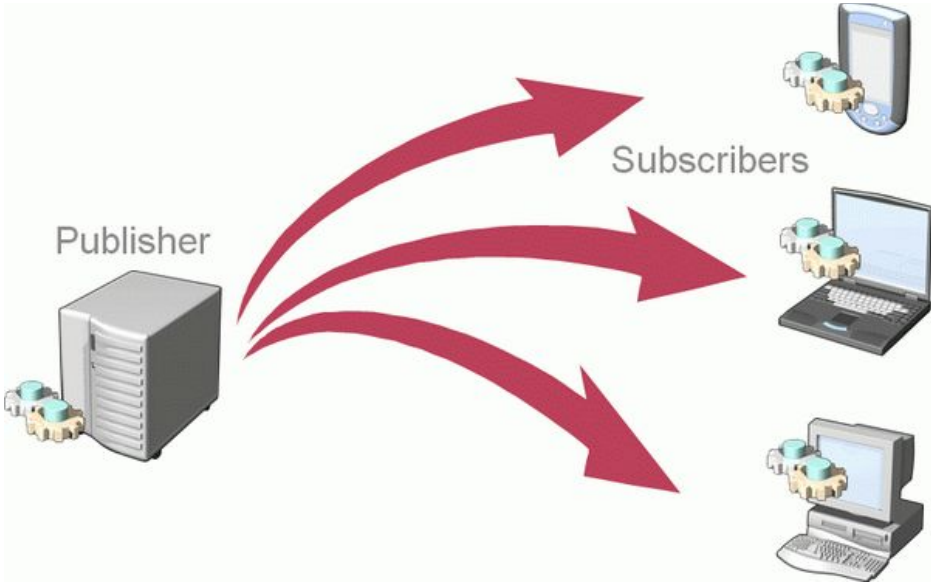
A possible solution

```
public class Player {
    Score s;
    void GotACoin(){
        s.UpdateScore();
    }
    void KilledAnEnemy(){
        s.Update();
        s.MoreDeadEnemies();
    }
    ...
}
```

```
public class Score {
    Spawner spawner;
    int score;
    int enemiesKilled;
    public void UpdateScore(){
        score++;
        if (score % 100 == 0)
            spawner.ShortenTime();
    }
public void MoreDeadEnemies(){
        enemiesKilled++;
        if (enemiesKilled % 10 == 0)
            spawner.ShortenTime();
    }
...
}
```

The Publisher/Subscriber pattern

- Publishers
  - Can define **events**
- Subscriber
  - Can “sign-up” to be notified when a certain event happens
- The process
  - The publisher can “**raise**” events (when they happen)
  - All subscribers are notified of the event
  - Each subscriber handles the event using a specified **event handler**



Events

- Declared by the publisher
- uses the special keyword “event”
- Requires a type that is (or derives from) EventHandler
- Has to be **public** (otherwise nobody can subscribe)
- Cannot be declared in a block of executable code

```
public event EventHandler GotACoin;
```

What can an event do?

- Gives structured access to handlers
  - if you are interested in more, look up delegate methods
- You cannot directly access them!
- You can only do two things: **add** and **remove** handlers
- When the event is raised it will call all its handlers sequentially
  - In the order you added them

Defining a Handler

- Any method with these specific **parameters**:
  - an "**object**" object: object is the base class of everything, so any object is accepted. Used to pass the object that raised the event
  - an **EventArgs** object, these are used to pass extra arguments/info about the event, we'll see how later today :)
- Can have any accessibility (public, private, etc…)
- Can return whatever

```
//make a handler method
private void MyHandler(object caller, EventArgs args){
     ...
}
```

Subscribing and unsubscribing

- Very simple: just add/subtract a method to the event!

```
//define an event
public event EventHandler MyNewEvent;
...
//make a handler method
private void MyHandler(object caller, EventArgs args){
     ...
}
...
//add the handler
MyNewEvent += MyHandler;
//remove the handler
MyNewEvent -= MyHandler;
```

Raising an event

```
//raising the event
if (MyNewEvent != null) //if the event has at
least one subscriber
    MyNewEvent(this, null);
//you need to pass an object (generally the
caller object) and optionally an EventArgs
object
```

- This calls all the methods that have subscribed to MyNewEvent

A possible solution

```
public class Player {
     Score s;
     void GotACoin(){
          s.Update();
     }
     void KilledAnEnemy(){
          s.Update();
          s.MoreDeadEnemies();
     }
     ...
}
```

```
public class Score {
     Spawner spawner;
     int score;
     int enemiesKilled;
     public void UpdateScore(){
          score++;
          if (score % 100 == 0)
               spawner.ShortenTime();
     }
public void MoreDeadEnemies(){
          enemiesKilled++;
          if (enemiesKilled % 10 == 0)
               spawner.ShortenTime();
     }
...
}
```

With events!

```
public class Player {
    public event
EventHandler GotACoin;
    public event
EventHandler KilledAnEnemy;
    ...
}
```

```
public class Score {
    public event EventHandler IncreaseDifficulty;
    int score;
    int enemiesKilled;
    public Score(Player p){
        p.GotACoin += UpdateScore;
        p.KilledAnEnemy += UpdateScore;
        p.KilledAnEnemy += MoreDeadEnemies;
    }
    private void UpdateScore(object caller, EventArgs
args){
        score++;
        if (score % 100 == 0 && IncreaseDifficulty!=null)
            IncreaseDifficulty(this, null);
    }
private void MoreDeadEnemies(object caller, EventArgs args){
        enemiesKilled++;
        if (enemiesKilled % 10 == 0 &&
IncreaseDifficulty!=null)
            IncreaseDifficulty(this, null);
    }
```

With events!

```
public class Player {
     public event
EventHandler GotACoin;
     public event
EventHandler KilledAnEnemy;
     ...
}
```

```
public class Spawner {
     float timeBetweenSpawns = 10f;
     public Spawner(Score s){
          s.IncreaseDifficulty +=
ShortenTime;
     }
     private void ShortenTime(object
caller, EventArgs args){
          timeBetweenSpawns/=2;
     }
}
```

What changed?

- Handler methods can now be private! Before to be accessible they had to be public
- Since they are now handlers, they have to have as parameters (object, EventArgs)
- A class doesn’t have to explicitly call the methods of other classes, it just needs to raise its own event
- Classes have to subscribe to events
- Classes can also un-subscribe to events at runtime!

How would this change in Unity?

```
public class Player {
    public event
EventHandler GotACoin;
    public event
EventHandler KilledAnEnemy;
    ...
}
```

```
public class Score {
    public event EventHandler IncreaseDifficulty;
    int score;
    int enemiesKilled;
    public Score(Player p){
        p.GotACoin += UpdateScore;
        p.KilledAnEnemy += UpdateScore;
        p.KilledAnEnemy += MoreDeadEnemies;
    }
    private void UpdateScore(object caller, EventArgs
args){
        score++;
        if (score % 100 == 0 && IncreaseDifficulty!=null)
            IncreaseDifficulty(this, null);
    }
private void MoreDeadEnemies(object caller, EventArgs args){
        enemiesKilled++;
        if (enemiesKilled % 10 == 0 &&
IncreaseDifficulty!=null)
            IncreaseDifficulty(this, null);
    }
```

In Unity

```
public class Player {
     public event EventHandler
GotACoin;
     public event EventHandler
KilledAnEnemy;
     ...
}
```

```
public class Score {
     public event EventHandler IncreaseDifficulty;
     int score;
     int enemiesKilled;
     public Start(){
          Player p =
GameObject.FindGameObjectWithTag("Player");
          p.GotACoin += UpdateScore;
          p.KilledAnEnemy += UpdateScore;
          p.KilledAnEnemy += MoreDeadEnemies;
     }
     private void UpdateScore(){
          score++;
          if (score % 100 == 0 && IncreaseDifficulty!=null)
               IncreaseDifficulty(this, null);
     }
private void MoreDeadEnemies(){
          enemiesKilled++;
          if (enemiesKilled % 10 == 0 &&
IncreaseDifficulty!=null)
               IncreaseDifficulty(this, null);
```

Passing arguments

- What do you think you get?
- You already get the caller object, so everything that is public in there is accessible :)

```
//raising the event
if (MyNewEvent != null) //if the event has at
least one subscriber
    MyNewEvent(this, null);
//you need to pass an object (generally the
caller object) and optionally an EventArgs
object
```

Passing arguments

- You can pass arguments using the EventArgs class
- Except that, by definition, this class is empty!?
- How can we create a new version of the class EventArgs?
- Why, yes, we can extend it (or derive from it)

Passing arguments

- class MyEventArgs : EventArgs
- note: when defining an event that will take a custom argument class EventHandler becomes EventHandler<MyEventArgs>
- By convention the name should contain “EventArgs”

```
public class MyEventArgs : EventArgs
{
    public string info1 = "test";
    public string info2 = "test2";

    public MyEventArgs(string i1, string i2)
    {
    info1 = i1;
    info2 = i2;
    }
}
```

Passing arguments

- You can also define properties!

```
public class MyArgs : EventArgs
{
     public string Info1 { get; set; }
     public string Info2 { get; set; }
}
...
MyArgs args = new MyArgs();
args.Info1 = "something";
args.Info2 = "something else";
```

Exam Questions

5 questions spanning the course topics

- You will pick a random one
- You can prepare a short presentation (like last semester), *optional but recommended*
- The question is meant as a starting point for a conversation, it is possible that you will be asked about topics that are not covered by the question you picked

Questions

1. OOP
    - Describe the fundamental idea of OOP, how does it differ from procedural programming?
    - Describe what is a class and what is an object
    - How do objects interact with each other?
    - Describe and discuss the Factory pattern.
    - How do events work in C#?

Questions

2. Data types and structures
    - What is the difference between a primitive type and an object type? Is there any difference in how variable assignment is made for the two types?
    - What is a data structure? Choose a couple of examples and describe how they differ.
    - What structure would you use to represent a sequence of attacks (see Dragonage as an example), or player information (assuming you want to store login information of all players in one place)? Discuss why.
    - How would you make a custom data structure? If you wanted to make a data structure to save cartesian coordinates (x,y) would you use a struct or a class? Why?
    - Describe and discuss Interfaces: how do they work, when should we use them, and how do they differ from abstract classes?

Questions

3. Trees and Graphs
    - What is a tree? What are their main characteristics and elements? Can you give a couple of examples of different trees?
    - How can you traverse a tree?
    - Define what is a graph. What are the differences between directed/undirected and weighted/unweighted graphs? Give an example of how you would implement a graph
    - Describe and discuss the Iterator pattern.
4. Debugging and testing
    - What is the difference between a syntactical, a semantical, and a logical error?
    - What is the difference between debugging and testing?
    - What do we mean with white box and black box testing?
    - What is an exception? How can we raise them and catch them? What does it mean to "propagate" an exception?
    - What are assertions, and what can we use them for?

Questions

5. Recursion and sorting
    - What is a recursive function? Make an example and describe how it works
    - How can you sort in C#? How do we define custom sorting orders?
    - Choose either MergeSort of Quicksort and explain how it works
    - Can you make an example of a design pattern?

Event exercise

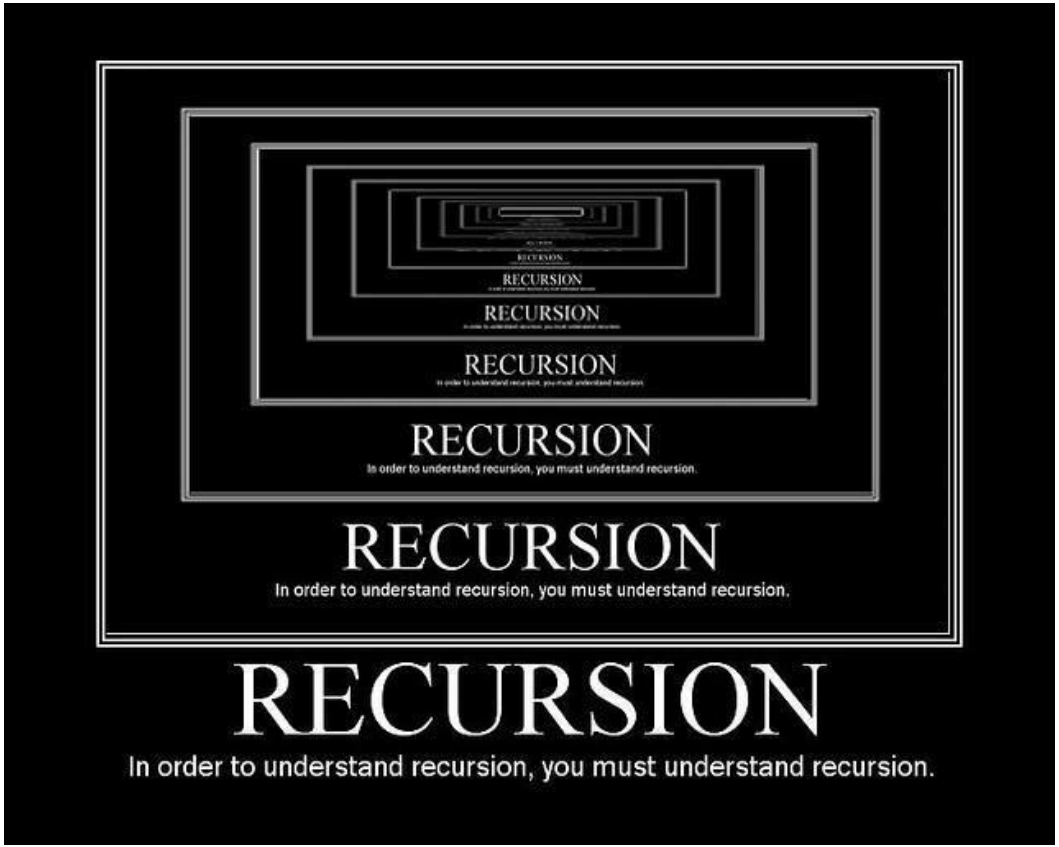
Let’s add an event to Zork

- Add a PlayerFaints event to Program
- Add a SendBackToJail handler to World
- Subscribe the handler to the event (I think in the Main would be the easiest)
- Create some condition to raise the event (maybe a 1/10 chance each time the player moves

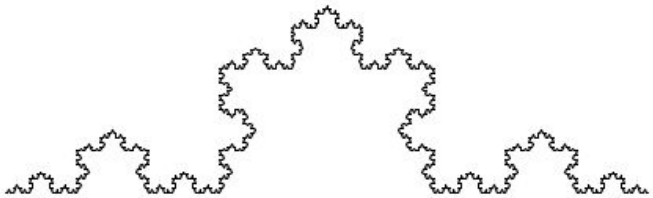
Fractal fun with Unity

Inspired by http://catlikecoding.com/unity/tutorials/constructing-a-fractal/

Recursion is back!

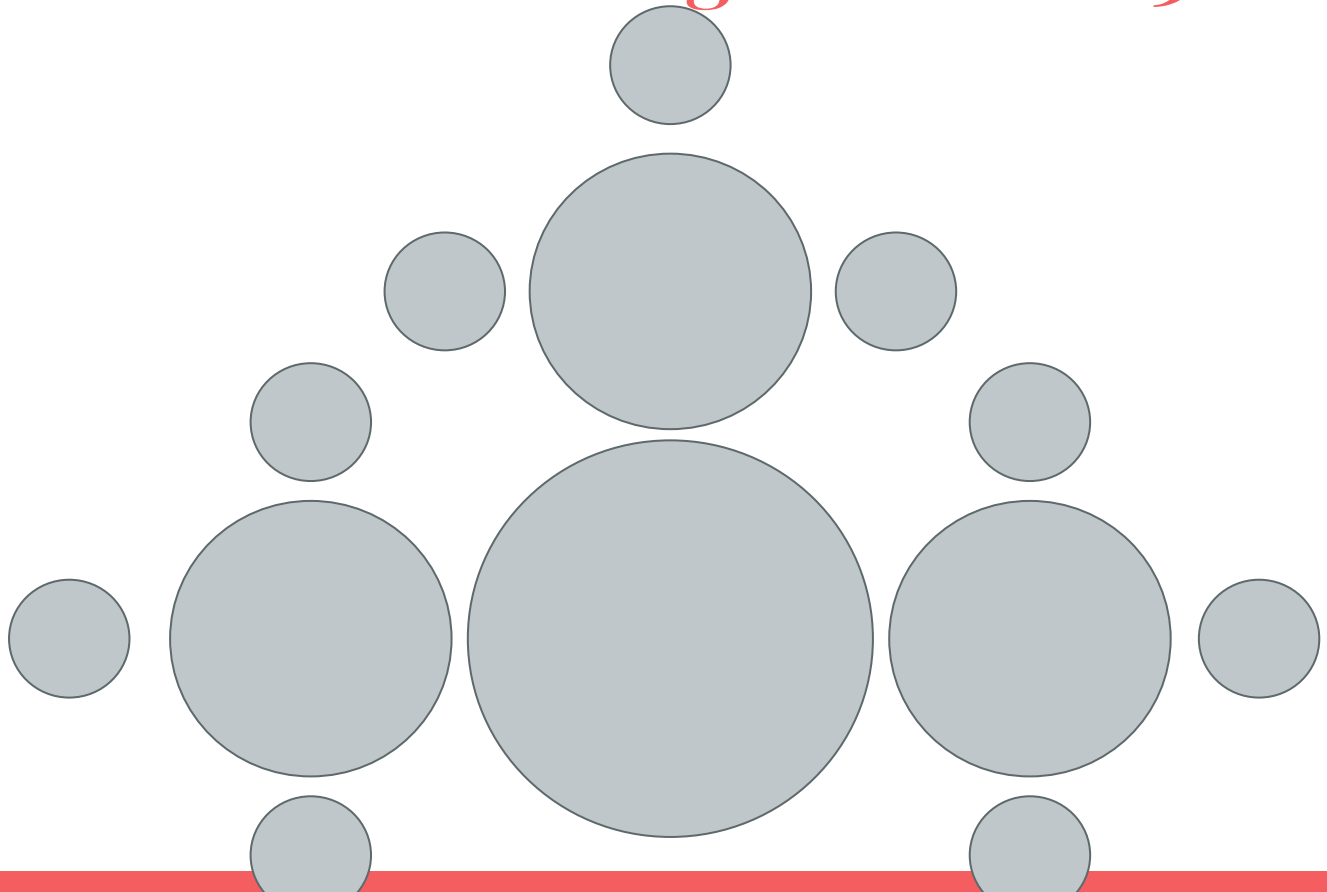


Graphical example: Koch fractals

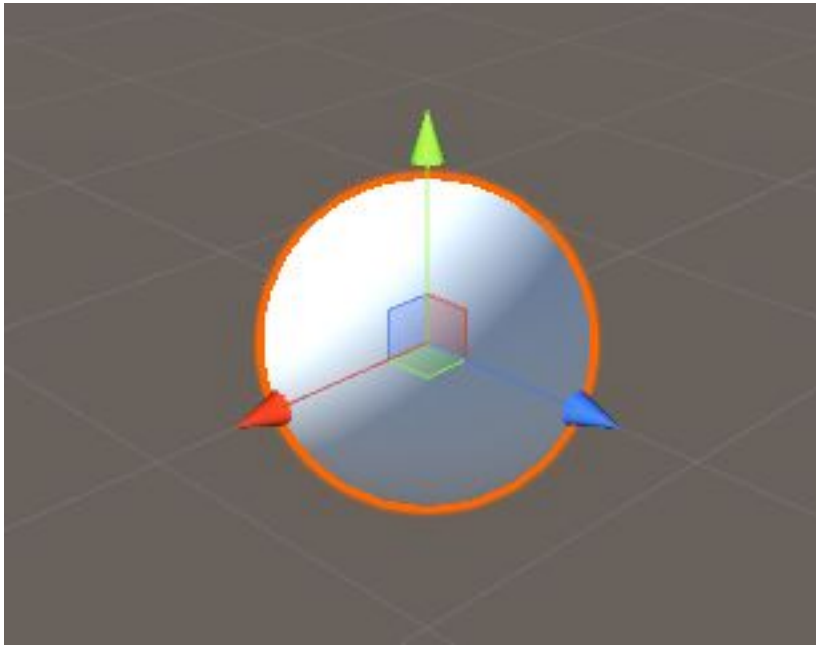


koch(5)

Can we make something similar in 3D?



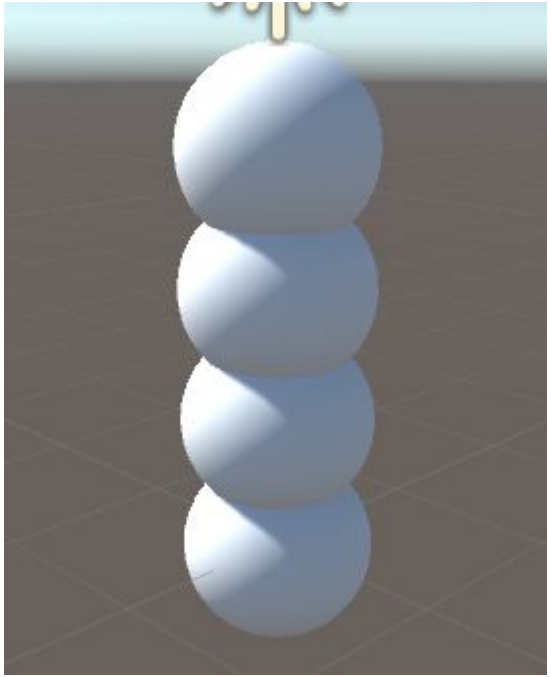
Let’s create a “procedural” sphere



Meaning, let’s make an empty object that creates a sphere

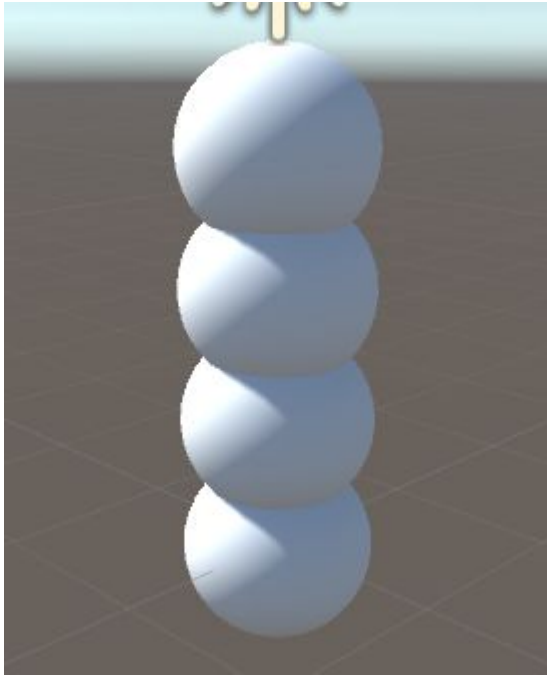
How do we create a new “child” sphere?

- new GameObject
- Add the Fractal component
- Initialize the fractal component with some values (possibly the ones from the parent



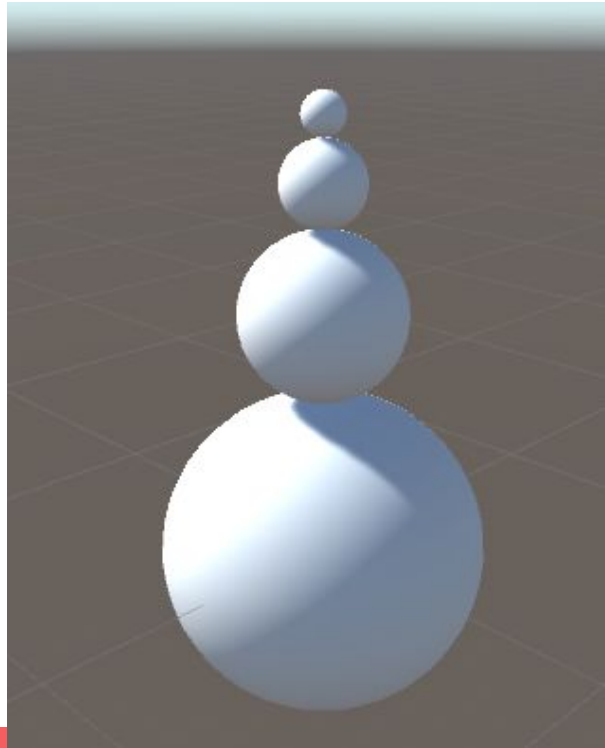
In this case we don’t have a “base case” so we should specify how deep we want our fractal to be

- In this example 3
- How do we add this constraint to the script?
- Remember that this value should decrease in the children, or we will continue creating spheres until the PC melts

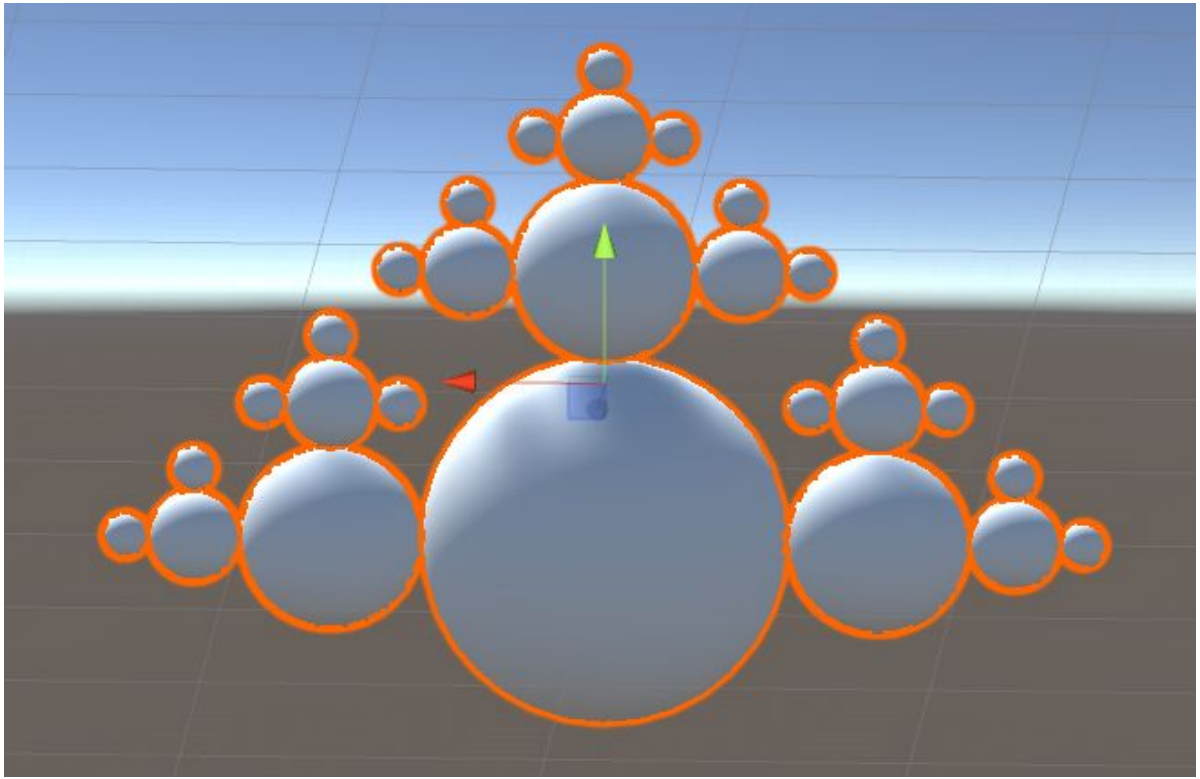


Now all the children are the same as the parent (and in the same place), how do we move them?

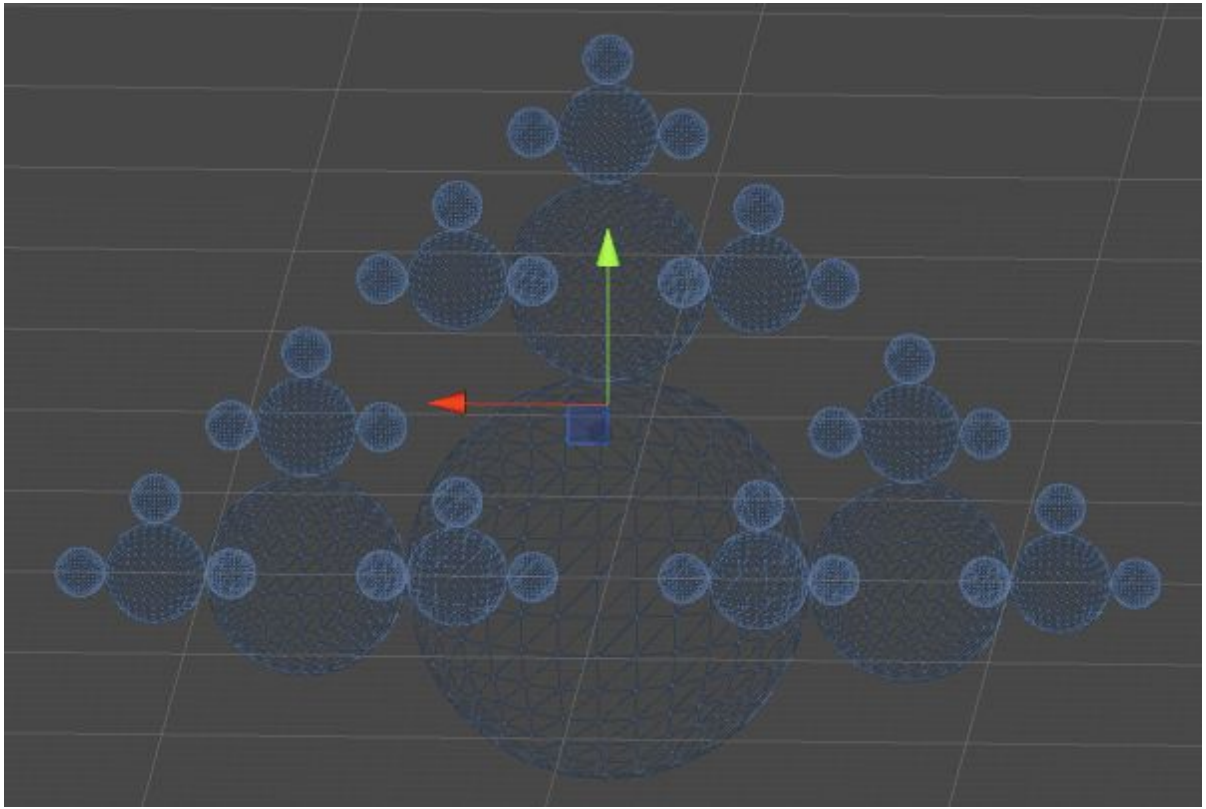
- In the Initialize function we can change their position and scale!
- We should probably have a scale value as a configurable parameter



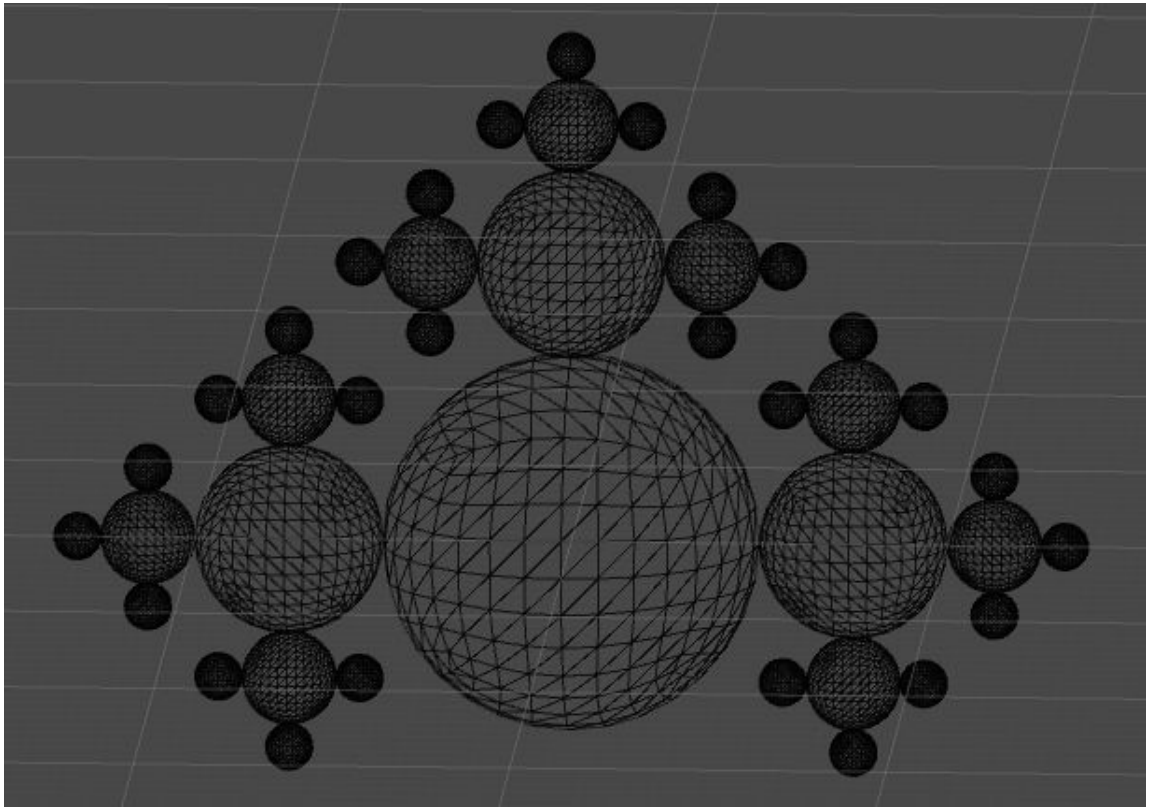
More children! How do we do that?



Ops! Something is not right!



Adding some rotation makes it all good



It’s starting to look nice, but we get the fractal instantaneously, can we make it appear a little at a time? (just for fun)

Coroutines!
(where we create the children)

Let’s cleanup and make the code a bit more general

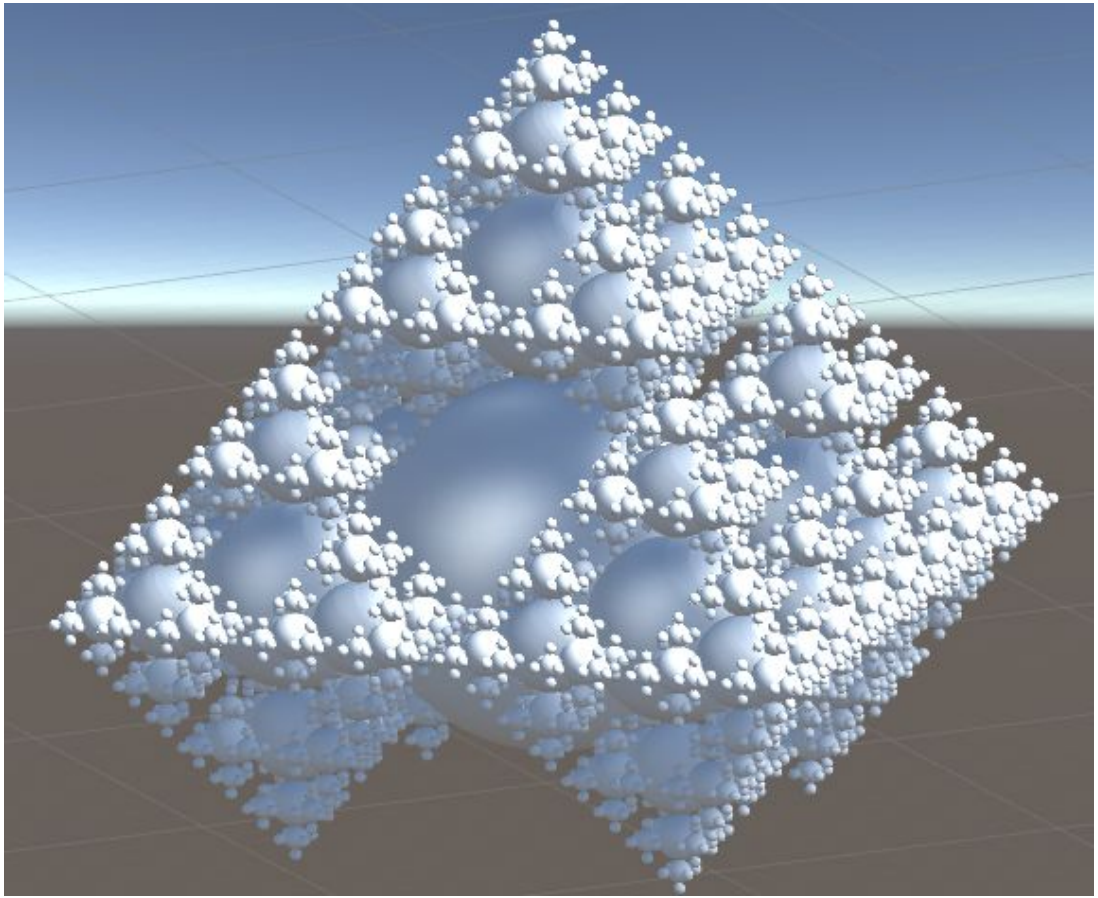
```
Vector3[] childDirections =
{
    Vector3.up,
    Vector3.right,
    Vector3.left
};

Quaternion[] childOrientation =
{
    Quaternion.identity,
    Quaternion.Euler(0f, 0f, -90f),
    Quaternion.Euler(0f, 0f, 90f)
};
```

How should the coroutine method change?

The final fractal!

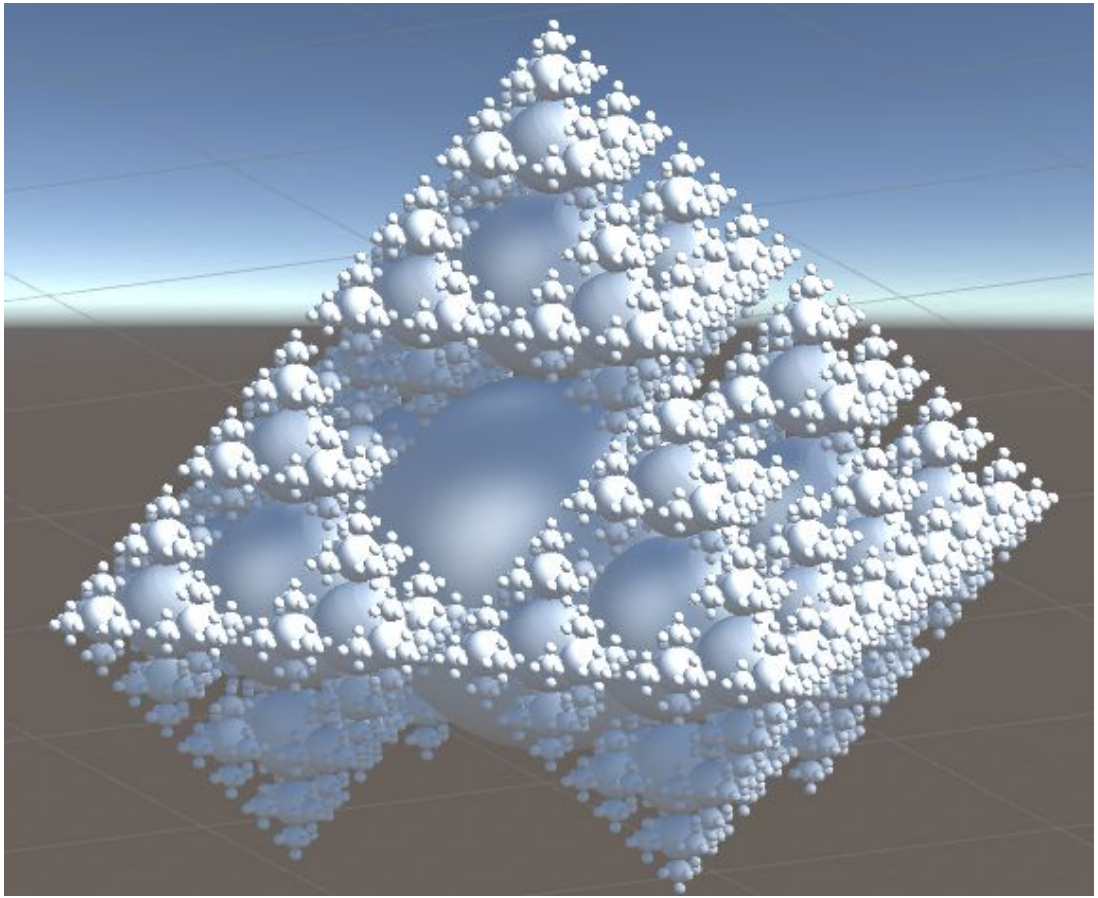
When adding also the forward and back directions

Warning!
the number of objects increases dramatically as you increase the depth, go above depth 5 at your own risk!



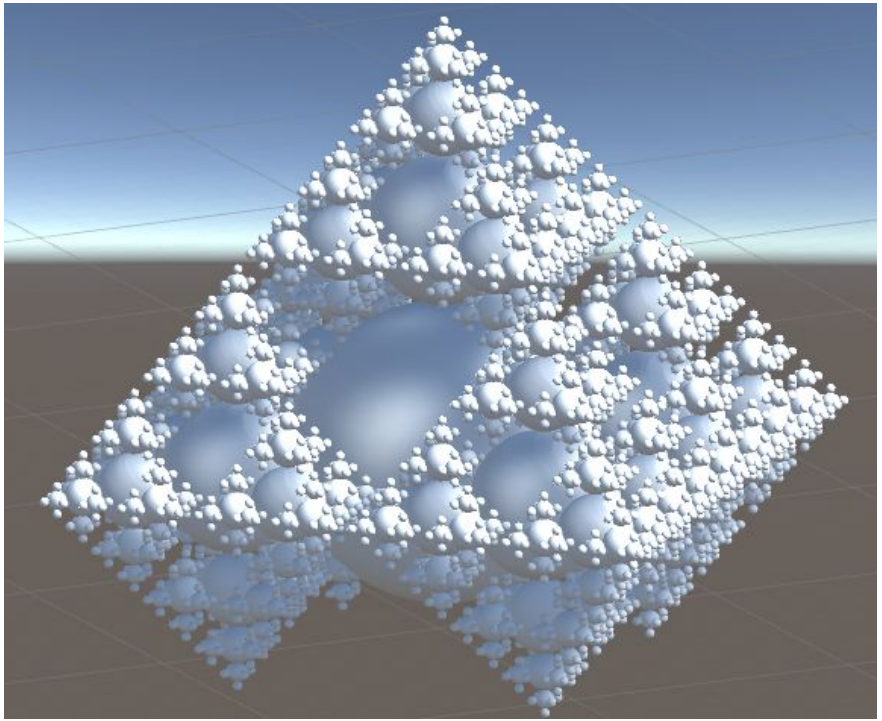
Creativity time!

- How would you make it more interesting?
- Ideas:
  - Colors
  - Different shapes
  - Asymmetrical
  - Offset the directions a little
  - ...



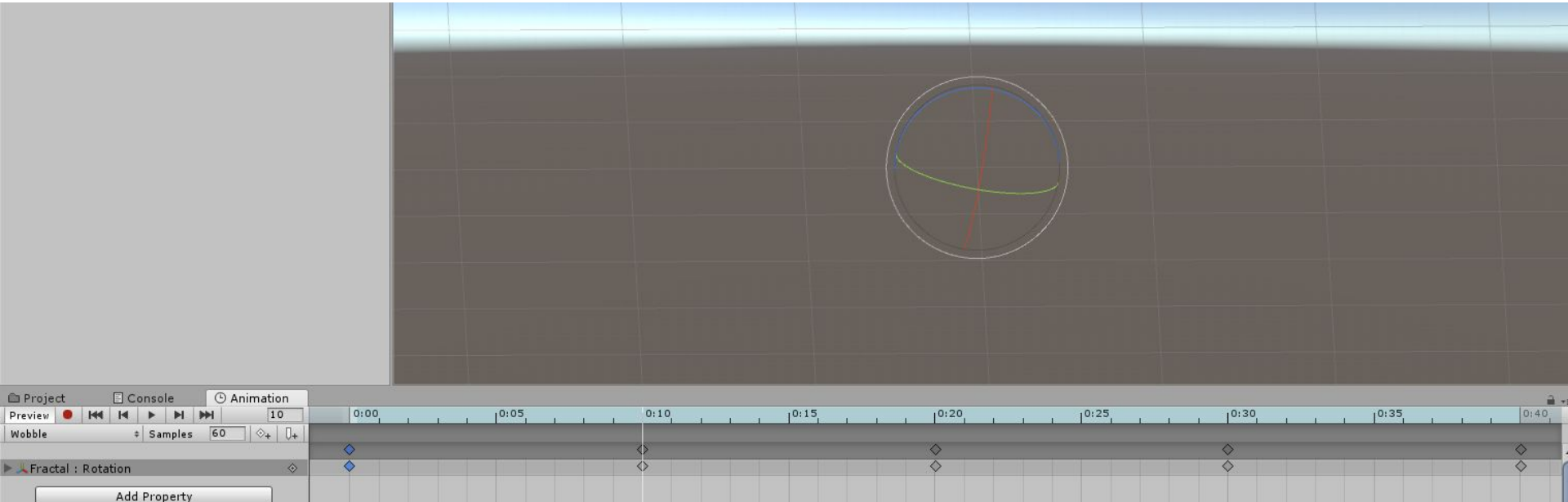
What about adding some animation?

- If we attach an animation to the root object and to the new ones what do we get? Recursive animation!
- Let’s create a simple wobble animation



What about adding some animation?

- Let's set it up so it goes from starting rotation-> rotate 10 degrees on the Z axis -> starting rotation ->rotate -10 degrees on the Z axis -> starting rotation



How should we change the Fractal script to have the animation added to each child?

- We need to add an Animator, which needs to get the correct RuntimeAnimatorController
- We need to also pass the controller to the children, like we did before with the other parameters

Look at it go!

You might notice that the orientation of the children is broken, but I didn’t have time to find a workaround

Next time

- Back to Unity
- Final assignment presentation & kickstart