ImpactLab - Game Dev
Lecture 5: More VI

Summer 2024

School of Computing and Data Science

Wentworth Institute of Technology

Volcano Island

Wentworth
Computing & Data Science

Visual Effects Graph

The Visual Effects Graph is a drag-and-drop, node based tool, built into Unity, that allows us to create complex particle effects very easily.

It's a complex tool! Just like with the regular particle system, you should experiment with different settings to see what you can create.

Notes:

Only fully available on Unity 2019.3 or greater

Uses the GPU to simulate particles, so it's not always applicable for your target platform.

It can also be a laptop battery hog!

The feature must be added to your project:

Window -> Package Manager

Search for Visual Effects Graph and install.

Wentworth
Computing & Data Science

Visual Effects Graph

The Visual Effects graph contains several "Context" areas:

Spawn:
Spawns the Particles

How Many Particles? When are they Spawned?

Initialize:
Starting Values

Longevity? Where are they Spawned? Velocity?

Update:
Behavior over Time

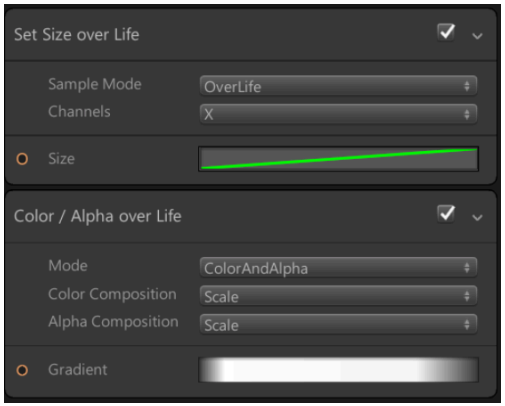
Forces? Color Change?

Quad Output:
Rendering The Particles

What do they look like? Shader? Facing?

Visual Effects Graph

Within each Context, there are a number of blocks, of which you can add more or delete.

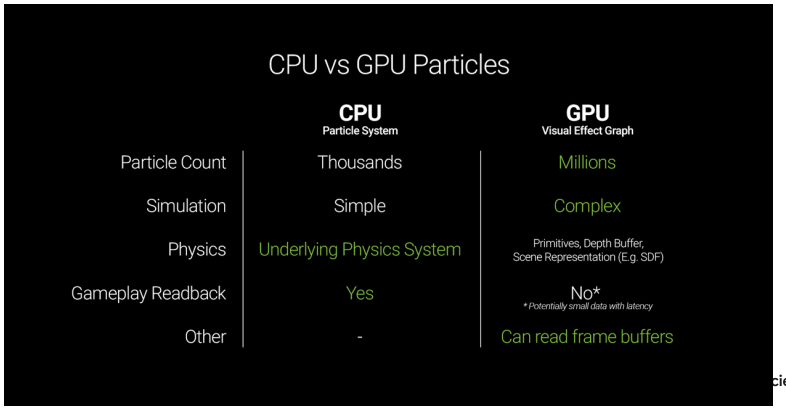


Each block sets a specific property related to the context. Many are the same as what you see in the regular Unity particle system.

Wentworth
Computing & Data Science

Visual Effects Graph

The Visual Effects Graph uses the GPU to render particles whereas the standard particle system uses the CPU. This has several important consequences:



Visual Effects Graph

Now we're going to play around and create a few effects. Keep in mind, this can severely impact your laptop battery!

Additional Resources:

- For more examples and VFX Graph information, take a look at the Unity Blog. e.g. https://blogs.unity3d.com/2019/03/06/visual-effect-graph-samples/
- The Brackys YouTube channel also has several good Visual Effects Graph tutorials (smoke and fireworks)

Volcano Island: What's Left?

- Particle Effects
- Animation
- Sound/Music
- Start/Game Over Screen

I've provided some assets for you to help with creating this items. As we go through this list, feel free to experiment with the settings.

Particle Effects

We'll start by making a new gameobject and adding a particle system component.

Drag the gameobject into your Prefabs folder. In the future, we'll instantiate this object when we need to spawn the particle effect.

The goal is to use three effects:

When the fireball hits the ground.

When the fireball hits the player.

When the player dies.

We'll keep the effect simple for now. Later, you may want to increase the fidelity.

Instantiating the Particle Effects

Make sure you have a couple of effects ready to go, specifically the ones for the enemy hitting the ground and the player.

Enemy Class

```
//death particle effects
public GameObject deathEffect;

private void OnTriggerEnter2D(Collider2D collision) {
  if (collision.tag == "Player") {
    //reduce player health
    p.takeDamage(damage);
    GameObject.Destroy(gameObject);
  }
  if (collision.tag == "Ground") {
    Instantiate(deathEffect, new
Vector3(transform.position.x, transform.position.y-0.5f,
-0.3f), Quaternion.identity);
    GameObject.Destroy(gameObject);
  }
}
```


Instantiating the Particle Effects

Make sure you have a couple of effects ready to go, specifically the ones for the enemy hitting the ground and the player.

Enemy Class

```
//death particle effects
public GameObject deathEffect;

private void OnTriggerEnter2D(Collider2D collision) {
  if (collision.tag == "Player") {
    //reduce player health
    p.takeDamage(damage);
    GameObject.Destroy(gameObject);
  }
  if (collision.tag == "Ground") {
    Instantiate(deathEffect, new
Vector3(transform.position.x, transf
-0.3f), Quaternion.identity);
    GameObject.Destroy(gameObject);
  }
}
```

Ensure that you drag your death effect prefab into this script in the inspector.

Don't forget to add a new tag, "Ground" and set the ground object to it.

Instantiating the Particle Effects

Enemy Class

```
//death particle effects
public GameObject deathEffect;
public GameObject hitEffect;

private void OnTriggerEnter2D(Collider2D collision) {
  if (collision.tag == "Player") {
    //reduce player health
    p.takeDamage(damage);
    Instantiate(hitEffect, new
Vector3(transform.position.x, transform.position.y-0.3f,
-0.3f), Quaternion.identity);
    GameObject.Destroy(gameObject);
  }
  if (collision.tag == "Ground") {
    Instantiate(deathEffect, new
Vector3(transform.position.x, transform.position.y-0.5f,
-0.3f), Quaternion.identity);
    GameObject.Destroy(gameObject);
  }
}
```

Instantiating the Particle Effects

Enemy Class

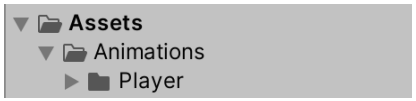
```
//death particle effects
public GameObject deathEffect;
public GameObject hitEffect;

private void OnTriggerEnter2D(Collider2D collision) {
  if (collision.tag == "Player") {
    //reduce player health
    p.takeDamage(damage);
    Instantiate(hitEffect, new
Vector3(transform.position.x, transform.position.y-0.3f,
-0.3f), Quaternion.identity);
    GameObject.Destroy(gameObject);
  }
  if (collision.tag == "Ground") {
    Instantiate(deathEffect, new
Vector3(transform.position.x, tran
-0.3f), Quaternion.identity);
    GameObject.Destroy(gameObject);
  }
}
```

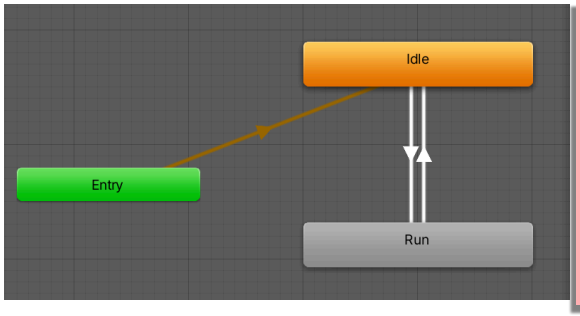
Do the same thing with the hitEffect (the particles that play when the player is hit by the fireball).

Animation

- Let's keep organized:



Animation in Unity uses a *state machine* to control transitions to different animation states



Here, the starting animation is Idle (transition upon Entry).

Under certain conditions, the animation will transition to Run and/or back to Idle.

Animation

- Create a new animation controller (Right Click in the Project window -> Create -> Animation Controller), Rename to PlayerAnimation.
- Create two animations (Right Click in the Project window -> Create -> Animation), Rename to Idle and Run.

The animation controller gives you the state machine for controlling transitions.

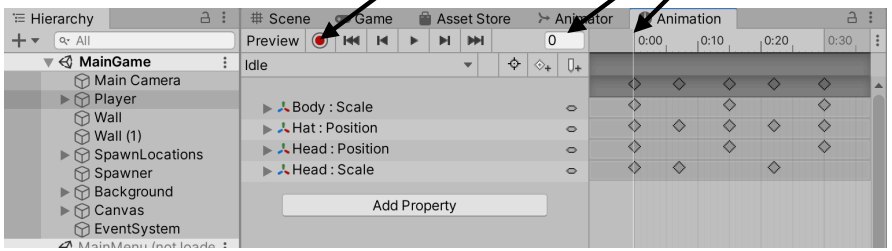
The two animations are where we will record changes to properties of our snowman that will play when the controller is in a specific state.

Animation: Idle

- We're going to start by opening the Animation window, attaching it to one of the frames and selecting our top-level snowman object.

Record Button!

Current Frame

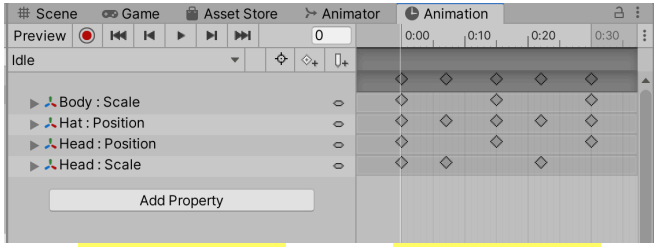


Properties

Keyframes

Animation: Idle

- To animate: Set a property to an initial value (the first keyframe) then add more keyframes and change the value (like the scale of the Head object).
- Unity will interpolate the position of each frame in between and play it like an animation

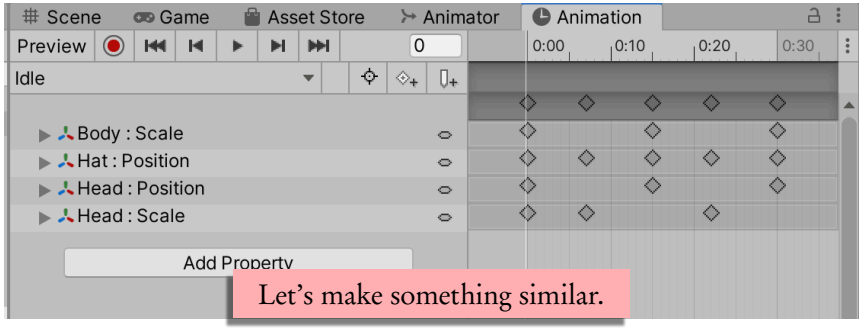


Let's make a head banging snowman

Properties

Keyframes

Animation: Idle

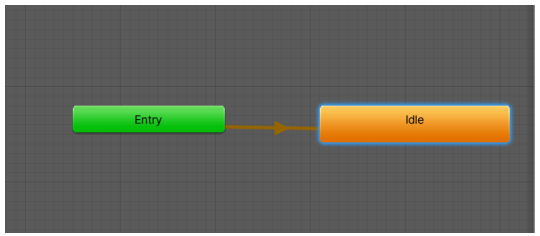


Let's make something similar.

This is my idle animation, notice that I change the scale and position of different pieces of the snowman.

To make a repeating animation, I copy the initial keyframes at 0 and move them down the timeline, then I make changes in between.

Animation: State Machine



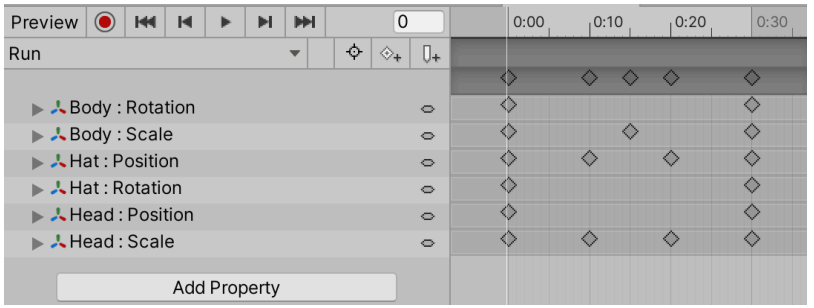
State Machines are used to keeping track of objects that can be in one of any number of "state".

Unity uses this idea of animations. At any given time, the object will be playing a specific animation.

The basic state machine will have an Entry state (for when the object is created), this will connect to one of the animations that you have created (Idle here)

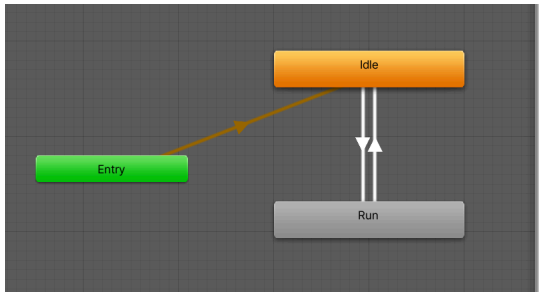
Animation: Run

For the Run animation, I'm going to have the snowman lean forward (in the direction we are running). In addition, it will still do some of the Idle style scaling and transforming.



I change more properties here, so let's try to make something similar.

Animation: State Machine Again



To add more animation states, just drag the animation unto the state machine interface.

We'll add the transitions in just a moment (the white arrows). In general, you'll have transitions between states that are triggered by variables or functions in your code.

Animation: Transitions

- Transitions are how we tell Unity to change from one animation to another.
- Typically this is done through some kind of trigger.
- The triggers are associated with an Animator component that has the Animation Controller as a property.

Add an Animator to the Player gameobject.

Drag the AnimationController that you created to the appropriate place in the Animator.

Now we are ready to go, any animation that we add to our controller will be hooked up to the player object.

Animation: Transitions

- Setting up a transition is as easy as selecting a block, right clicking and selecting Make Transition.
- This attaches an arrow to your mouse cursor which allows you to select another block.
- The transition will go from where you first right clicked to where you clicked.
- Once the transition is created, highlight it.

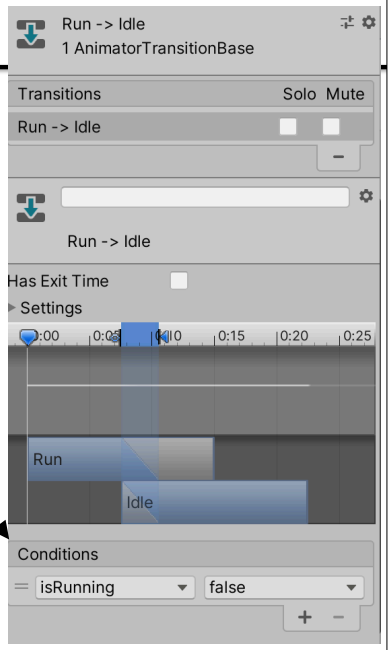
Animation: Transitions

This is my transition from Run to Idle.

There is some info about Exit Time and a timeline graph.

For now the important part is the *Conditions*.

This transition is triggered when the isRunning (boolean) is false, which we will set in our code.

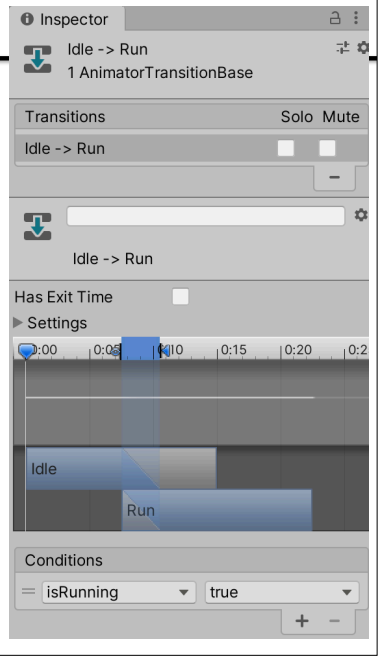


Animation: Transitions

Similarly, the transition from Idle to Run use the same boolean, isRunning.

The timeline shows the amount of time that the transitions between states will take. Play with it to see what happens.

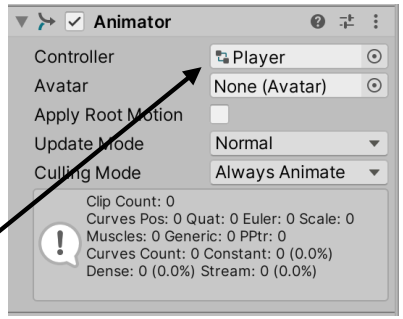
This transition is triggered when the isRunning (boolean) is true, which we will set in our code.



Animation: Attaching to Object

Lastly, we need to attached the animation controller to the object that will be animating.

On your snowman, add the Animator component and drag the animation controller (Player in this case) into the Controller location.



When you play the game, the snowman should play the Idle animation. Next we need to hook up the running animation into the code.

Wentworth
Computing & Data Science

Hooking up the Code

Player Class

```
void Start()
{
  ...
  anim = GetComponent<Animator>();
}

private void Update() {
  if (input > 0 || input<0) {
    anim.SetBool("isRunning", true);
  } else {
    anim.SetBool("isRunning", false);
  }

  if (input > 0) {
    transform.eulerAngles = new Vector3(0f, 0f, 0f);
  }
  else if(input < 0){
    transform.eulerAngles = new Vector3(0f, 180f, 0f);
  }
}
```

We must grab the animator component

Depending on the input (from FixedUpdate) we set the isRunning bool to true or false.

What does this do?

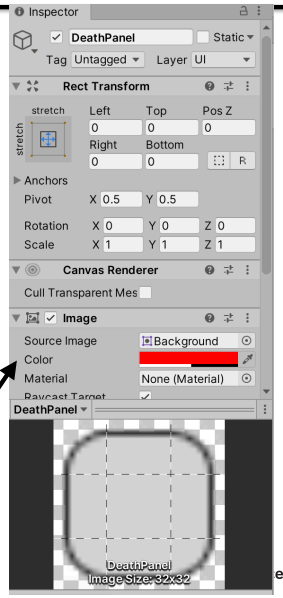
Game Over Screen

We're going to be using the standard Unity UI system to create our Main Menu and Game Over screen

Start by creating a new UI->Image in your Hierarchy.

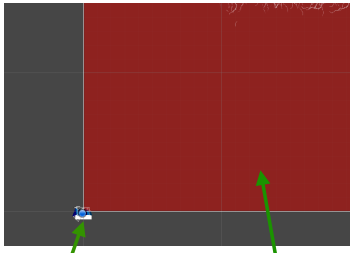
This automatically creates a Canvas for you that contains the Image object as a child.

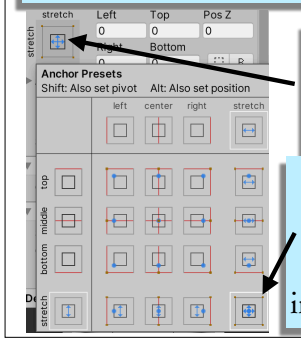
We'll have to set the image that will be displayed. Initially, we'll set the tint to a solid color (with transparency). I'm using the standard button image that is build in.



Game Over Screen

In your scene view, the canvas will look huge. Don't worry, it will display properly when you run the game. However, we'll have to mess with the anchor settings to get it to look how we want.



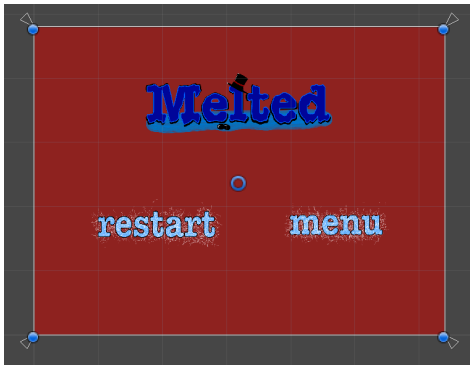


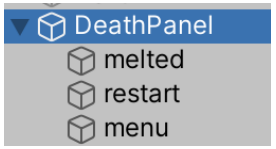
Click here to see the Anchor Presets

Notice that I have the stretch selected (in both dimensions. Using modifier keys (shift and alt) will set different anchor properties. Add more images or text and play around with the settings.

Game Over Screen

Here's an example:



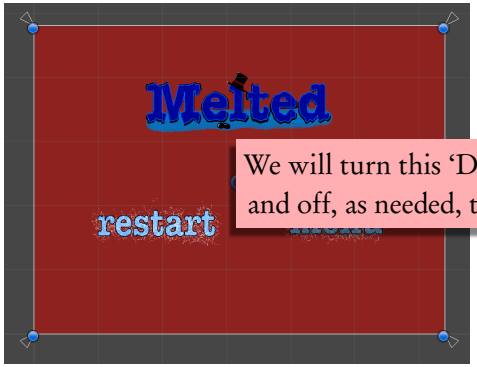


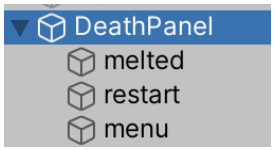
All of the objects are children to the panel that contains the background color (which itself is a child of the canvas).

'Melted' is just an image, but 'Restart' and 'Menu' contain an image and a button component. It's easy to add more components to your UI elements.

Game Over Screen

Here's an example:





We will turn this 'DeathPanel' on and off, as needed, through code.

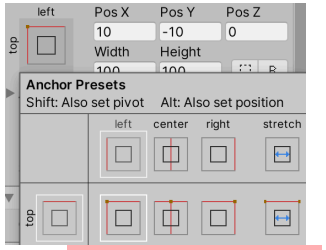
e objects are
the panel that
contains the background color (which itself is a child of the canvas).

'Melted' is just an image, but 'Restart' and 'Menu' contain an image and a button component. It's easy to add more components to your UI elements.

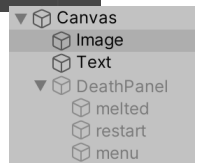
Health UI

Here's the Health UI on the same Canvas:

Note that the anchor is in the top left:



And the Image and Text are children of the Canvas (not the DeathPanel):



We'll update the text via code, for now I've entered "x3" as the default.

Code Hookup

```
public Text healthText;
public GameObject DeathPanel;

public void updateHealthText(int health){
  healthText.text = "x" + health
}

public void deathPanelSwitch(bool state){
  deathPanel.SetActive(state)
}
```

GameManager.cs

Remember: public fields are updated in the inspector.

These are the public methods that allow us to update the GUI information, i.e. show the DeathPanel and change the text for the health.

Code Hookup

Now we can add the previous calls to the appropriate places in our current code:

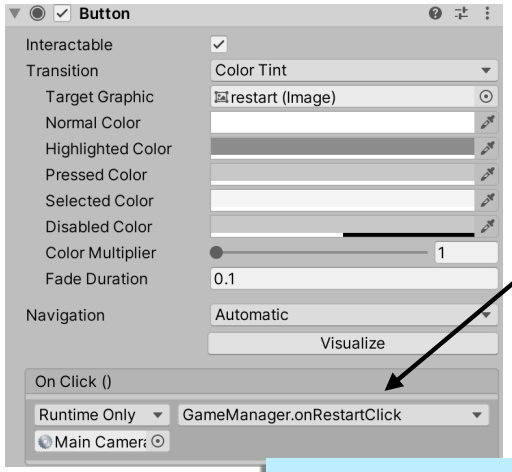
GameManager.cs

```
void Start(){
  deathPanelSwitch(false);
  ...
}

public void onRestartClick(){
  if(deathPanel.activeSelf){
    deathPanelSwitch(false);
    player.reset();
    spawner.reset()
  }
}
```

And create the method to call when the restart button is pressed

Code Hookup



Easy to hook up function calls to button presses.

Restart Button

Back to Player and Spawner

Our onRestartClick() method resets the player and the spawner:

Position the player wherever you want, and ensure that the player gameObject is active.

```
public void reset(){
  health=3;
  this.transform.position=new Vector3(0f,0f,0f);
  GameManager.instance().updateHealthText(health);
  this.gameObject.SetActive(true)
}
```

Player.cs

Spawner.cs

```
public void reset(){
  timeBetweenSpawns=1.25f;
}
```

Reset spawner timing.

Back to Player and Spawner

When the player takes damage and/or "dies" we need to make sure we trigger the GameOver screen

```
public void takeDamage(int value){
  health-=value;
  GameManager.instance().updateHealthText(health);
  if(health<=0){
    //Instantiate death effect here if you want
    this.gameObject.setActive(false);
    GameManager.instance().deathPanelSwitch(true);
  }
}
```

This will make the player gameobject disappear (without destroying it in memory) and show the "DeathPanel"

Coming Up:

- Finishing Volcano Island