# Provenance in Games

**Troy C. Kohwalter,**

*Instituto de Computação – Universidade Federal Fluminense (UFF)*

Introduction
Provenance in Games
Unity3D Provenance scripts
Unity3D Example
Conclusion

# INTRODUCTION

# Context

- Analysis Process
  - Technical Issues
  - Gameplay Mechanics


- Beta Testing
  - Indispensable Source of Data
  - Artisanal
  - Volunteers
  - Superficial Analysis

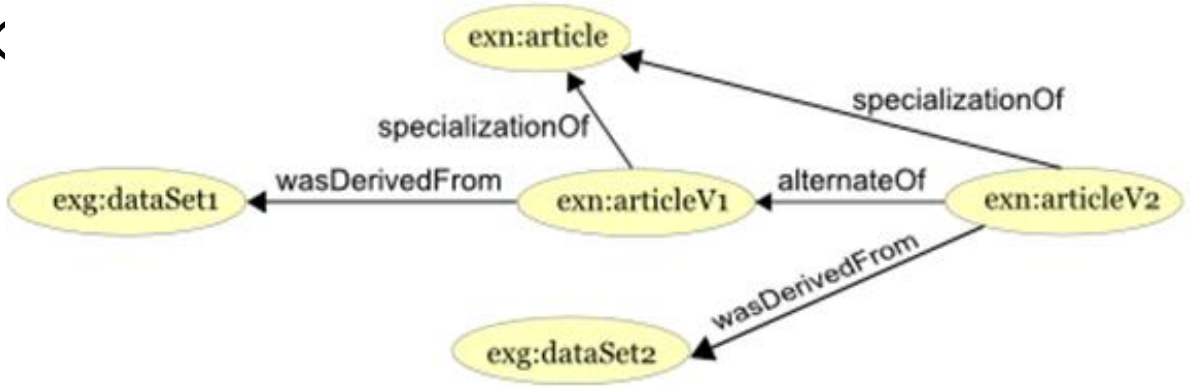
# Motivation

- Cause-and-Effect
  - How to detect?
  - How to display?
- Archeology, Paleontology
  - Provenance

# Provenance

*"Refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle"*

- Provenance Grap
  - Causality Graph




http://www.w3.org/TR/prov-primer/

# Goals

- Cause-and-Effect Relationships
  - Detect
  - Extract
  - Display
- Assist
  - Detect Gameplay Issues
- Visualization
  - Game Session Provenance

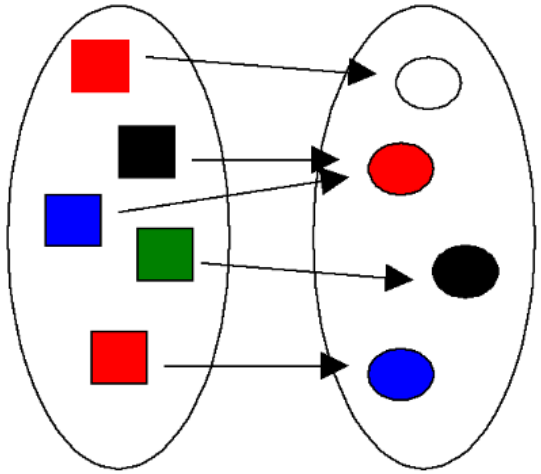
Introduction
Provenance in Games
Unity3D Provenance scripts
Unity3D Example
Conclusion

# PROVENANCE IN GAMES

# Provenance in Games

- Conceptual Framework

- Map Domains
  - Provenance to Games



- Gather
  - Provenance Information
  - Causal Relationships

# Provenance Gathering

- Entity
  - Objects
- Activity
  - Actions
  - Events
- Agent
  - NPCs
  - Player

# Provenance Gathering

- Entity
  - Objects
- Activity
  - Actions
  - Events
- Agent
  - NPCs
  - Player

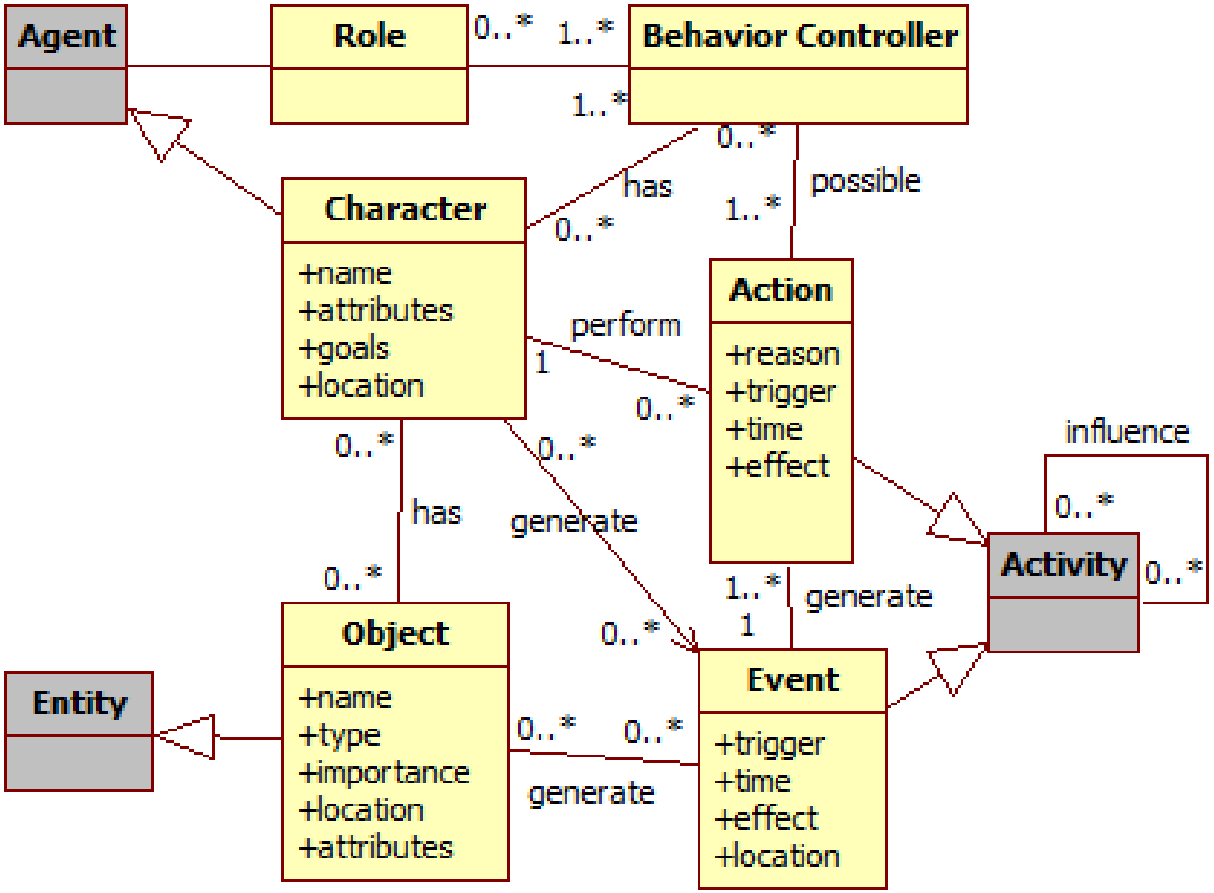
# Provenance Gathering

- Entity
  - Objects
- Activity
  - Actions
  - Events
- Agent
  - NPCs
  - Player



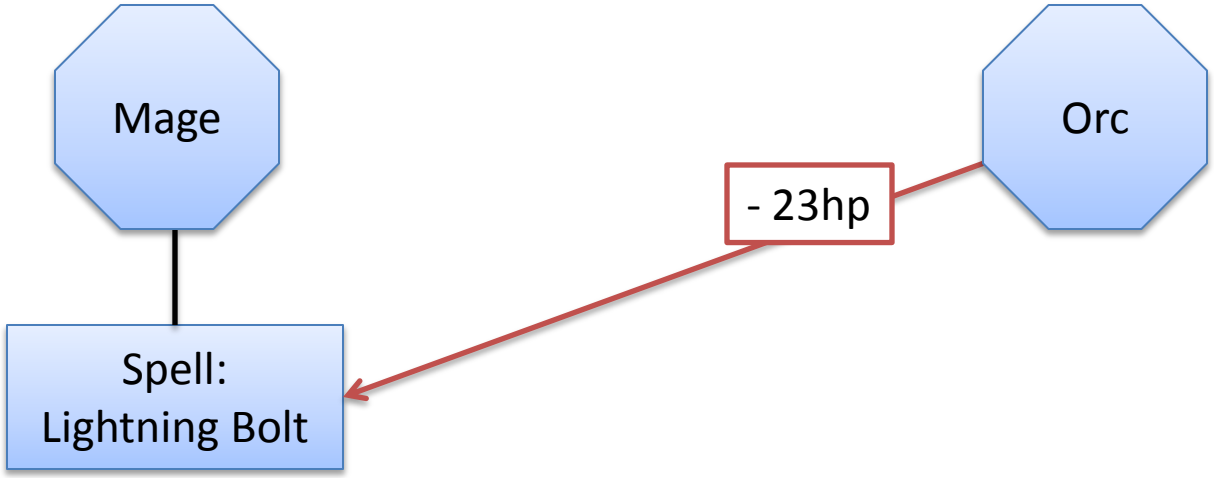



# Provenance to Games

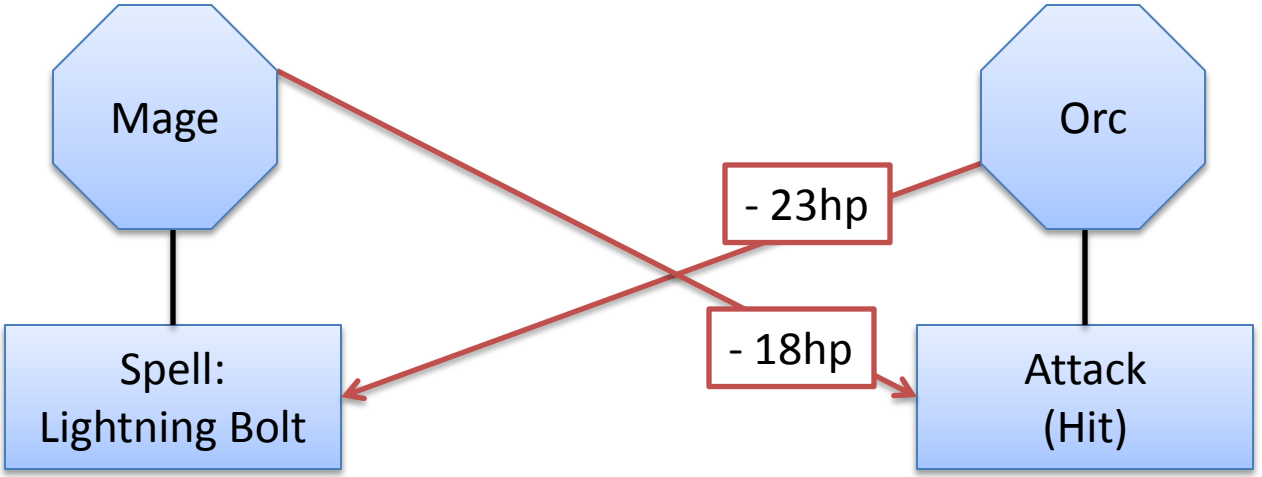


# Graph Construction

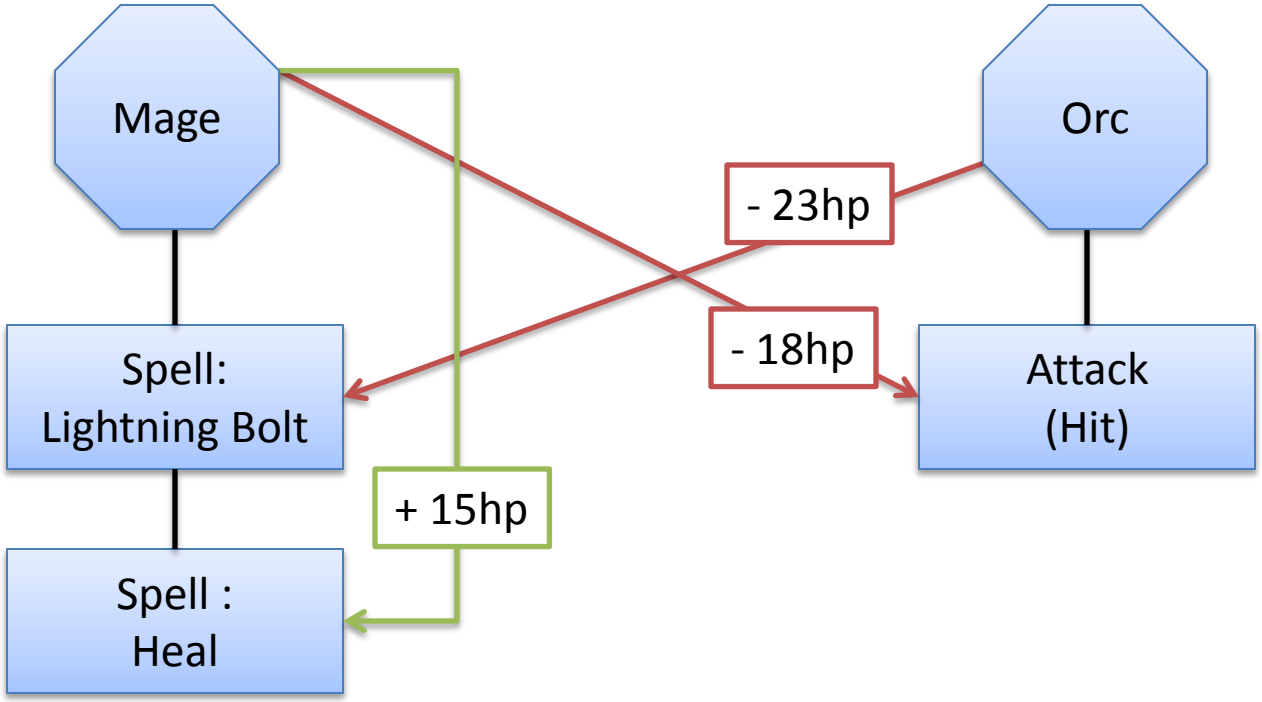
# Graph Construction



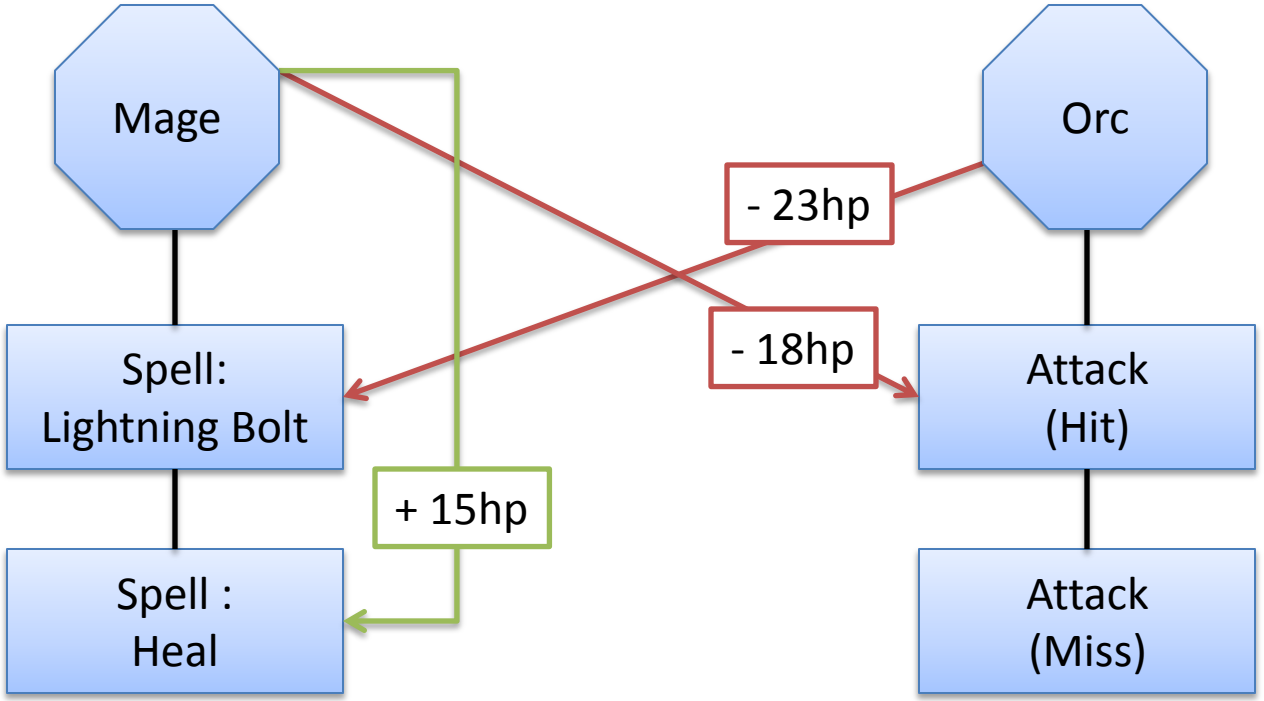
# Graph Construction



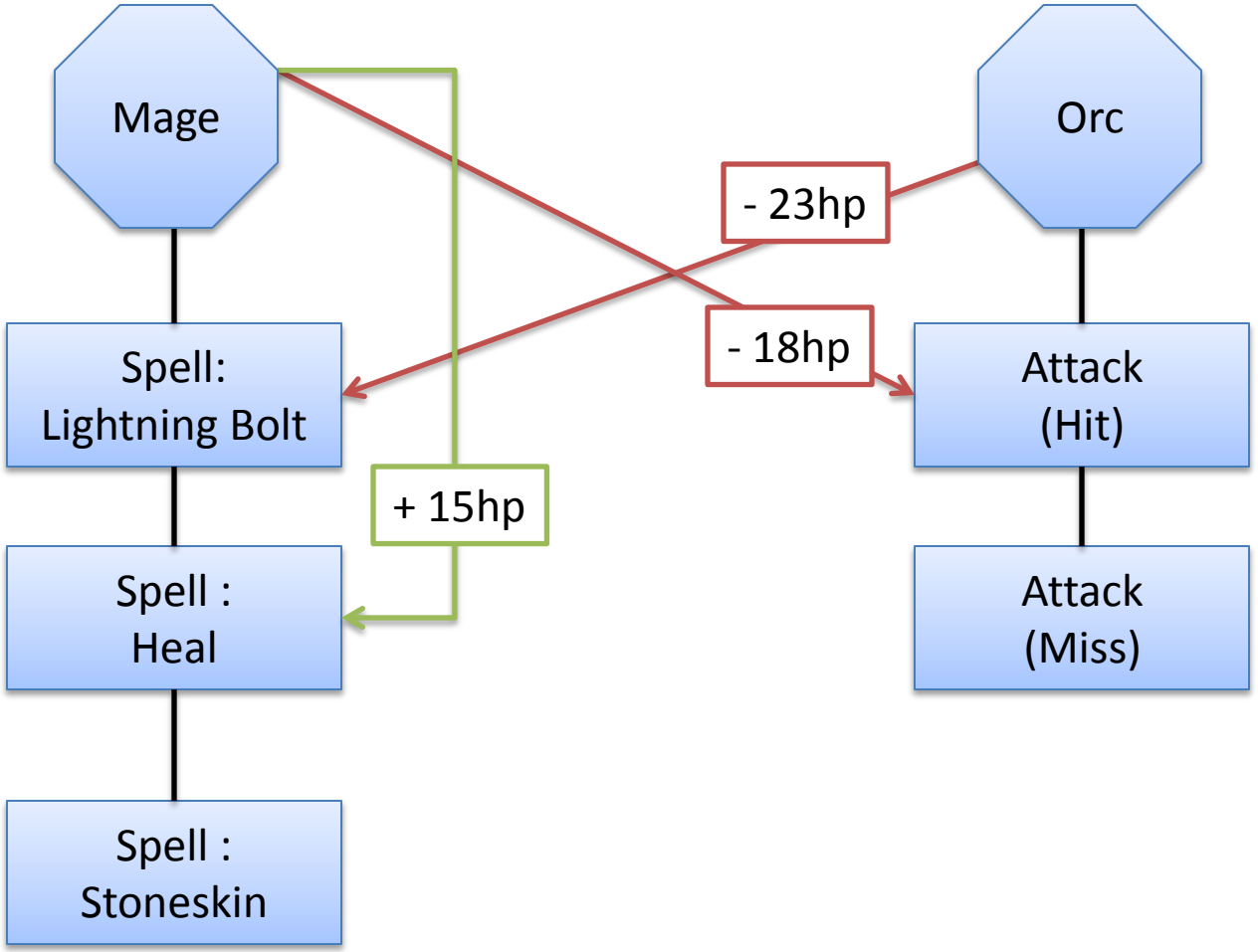
# Graph Construction



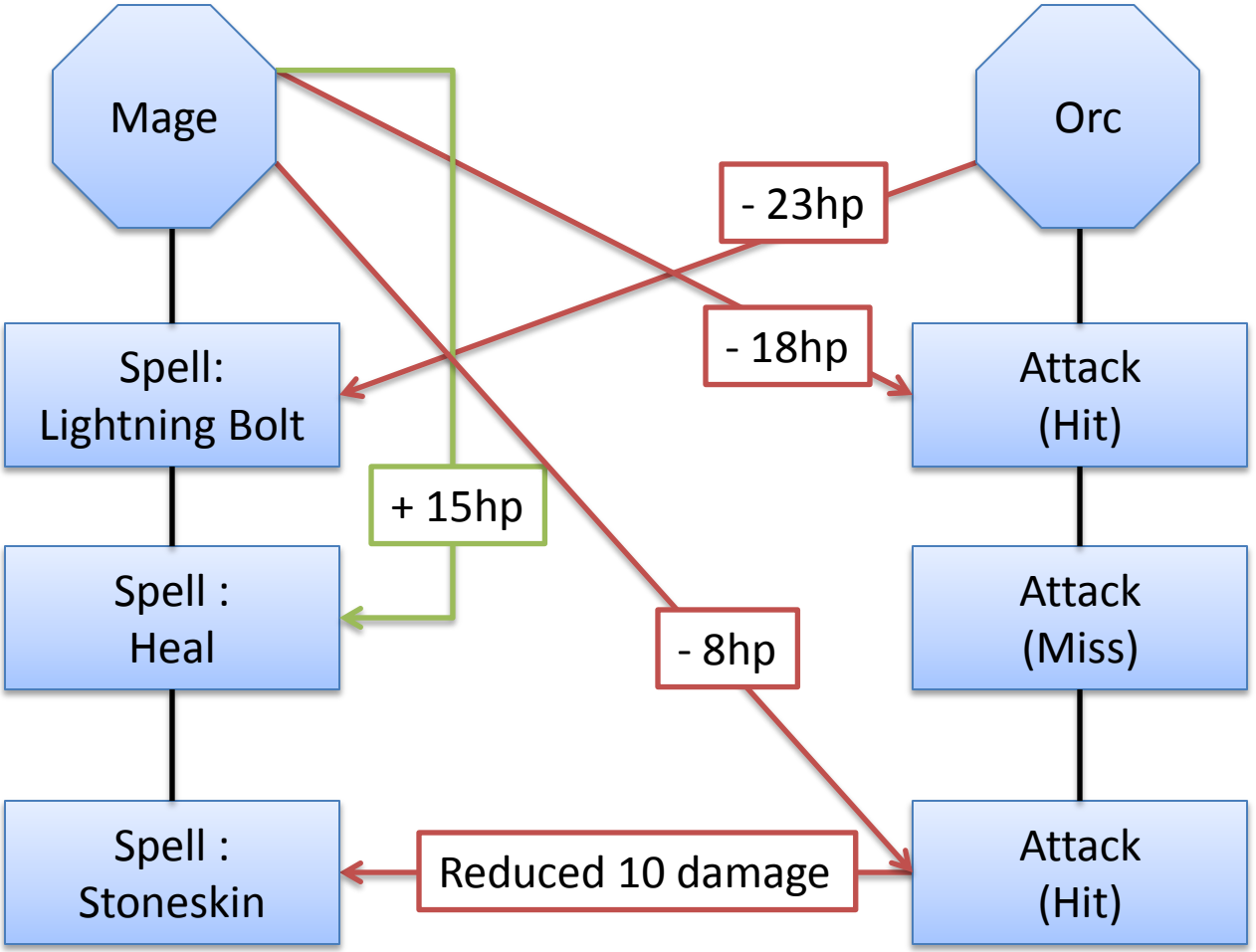
# Graph Construction



# Graph Construction



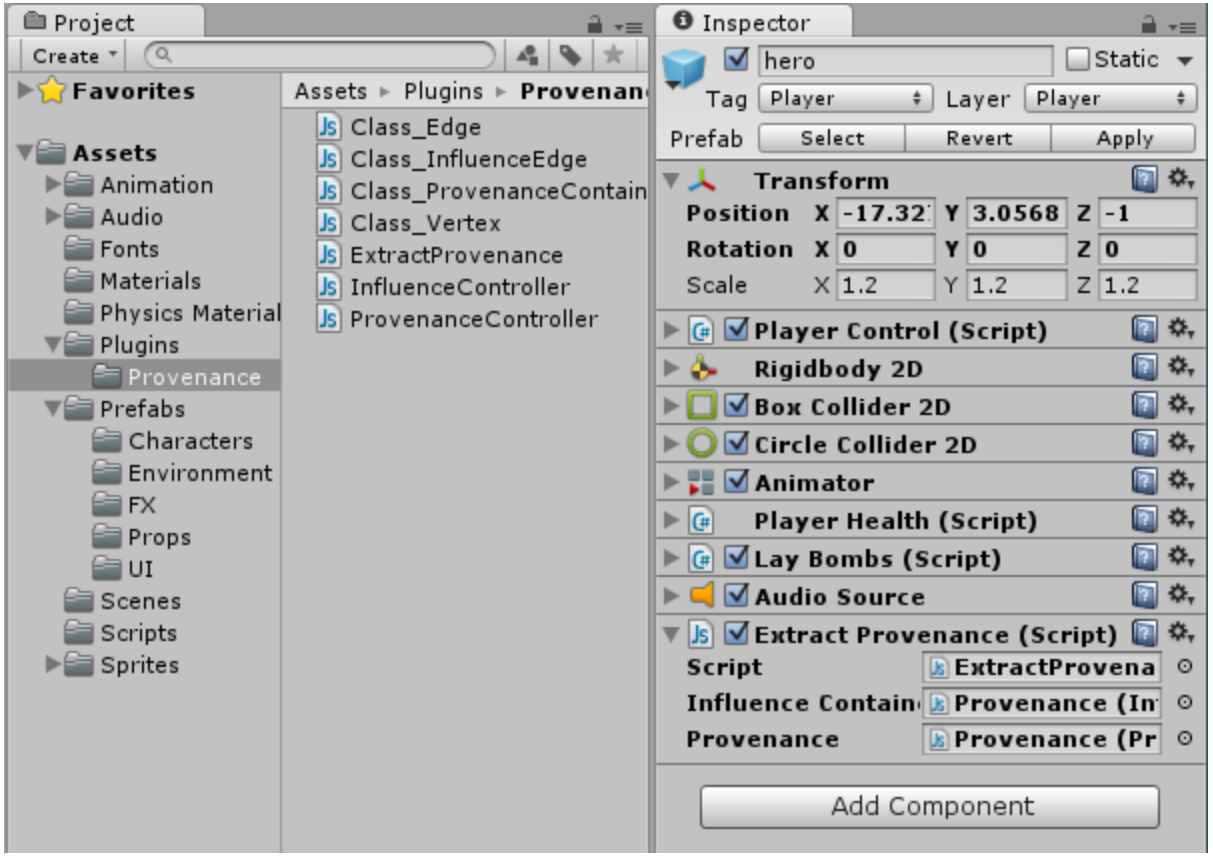
# Graph Construction



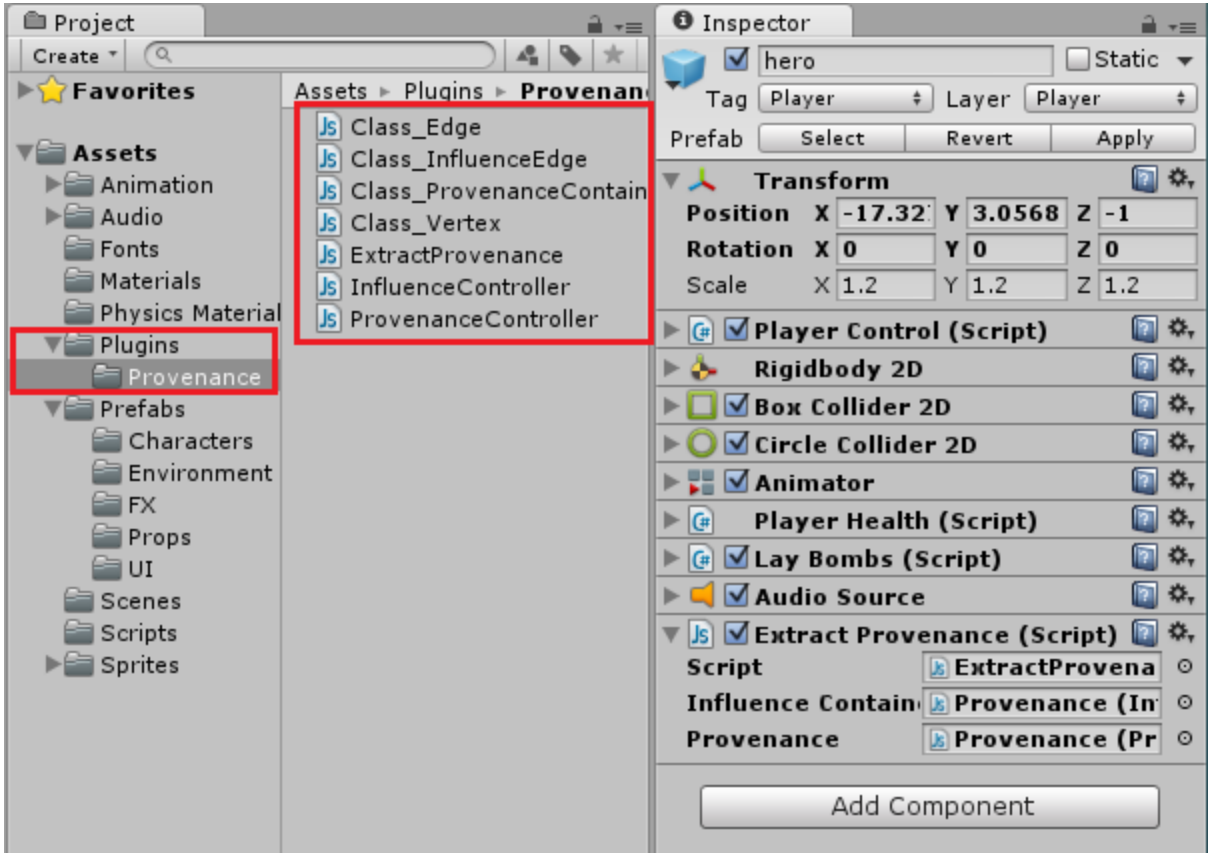
Introduction
Provenance in Games
Unity3D Provenance scripts
Unity3D Example
Conclusion

# UNITY3D PROVENANCE SCRIPTS

# Unity3D


Project
Create
Favorites
Assets
Animation
Audio
Fonts
Materials
Physics Material
Plugins
Provenance
Prefabs
Characters
Environment
FX
Props
UI
Scenes
Scripts
Sprites
Assets ▸ Plugins ▸ Provenan
Class_Edge
Class_InfluenceEdge
Class_ProvenanceContain
Class_Vertex
ExtractProvenance
InfluenceController
ProvenanceController
Inspector
hero
Static
Tag
Player
Layer
Player
Prefab
Select
Revert
Apply
Transform
Position X -17.32 Y 3.0568 Z -1
Rotation X 0 Y 0 Z 0
Scale X 1.2 Y 1.2 Z 1.2
Player Control (Script)
Rigidbody 2D
Box Collider 2D
Circle Collider 2D
Animator
Player Health (Script)
Lay Bombs (Script)
Audio Source
Extract Provenance (Script)
Script
ExtractProvena
Influence Contain
Provenance (In
Provenance
Provenance (Pr
Add Component

# Unity3D


Project
Create
Favorites
Assets
Animation
Audio
Fonts
Materials
Physics Material
Plugins
Provenance
Prefabs
Characters
Environment
FX
Props
UI
Scenes
Scripts
Sprites
Assets ▸ Plugins ▸ Provenan
Class_Edge
Class_InfluenceEdge
Class_ProvenanceContain
Class_Vertex
ExtractProvenance
InfluenceController
ProvenanceController
Inspector
hero
Static
Tag
Player
Layer
Player
Prefab
Select
Revert
Apply
Transform
Position X -17.32 Y 3.0568 Z -1
Rotation X 0 Y 0 Z 0
Scale X 1.2 Y 1.2 Z 1.2
Player Control (Script)
Rigidbody 2D
Box Collider 2D
Circle Collider 2D
Animator
Player Health (Script)
Lay Bombs (Script)
Audio Source
Extract Provenance (Script)
Script
ExtractProvena
Influence Contain
Provenance (In
Provenance
Provenance (Pr
Add Component

# Support Classes

```
#pragma strict

//=============================================================================================================
// 'Edge' Class Definition
// This script is to define the Edge class
// Do not attach this script in any GameObject
// It is only necessary to be on your resources folder
// The 'Edge' class is used for the Provenance-Scripts
//=============================================================================================================
```

```
#pragma strict

//=============================================================================================================
// 'InfluenceEdge' Class Definition
// This script is to define the influenceEdge class
// Do not attach this script in any GameObject
// It is only necessary to be on your resources folder
// The 'InfluenceEdge' class is used for the Provenance-Scripts
//=============================================================================================================
```

```
#pragma strict

//=============================================================================================================
// 'ProvenanceContainer' Class Definition
// This script is to define the ProvenanceContainer class
// Do not attach this script in any GameObject
// It is only necessary to be on your resources folder
// The 'ProvenanceContainer' class is used for the Provenance-Scripts
// It is responsible for exporting provenance information into a XML file
//=============================================================================================================
```

```
#pragma strict

//=============================================================================================================
// 'Vertex' Class Definition
// This script is to define the Edge class
// Do not attach this script in any GameObject
// It is only necessary to be on your resources folder
// The 'Vertex' class is used for the Provenance-Scripts
//=============================================================================================================
```

# ExtractProvenance

```
1  #pragma strict
2
3  //=============================================================================================================
4  // Script for creating vertices for the attached GameObject
5  // Attach this script in the desired game object and invoke the functions described below to gather provenance data
6  //
7  // Link it to InfluenceController
8  // Link it to ProvenanceGatherer
9  //-------------------------------------------------------------------------------------------------------------
10 // Brief explanations of each function used to record provenance information:
11 //
12 //  NewActivityVertex(label, details): Creates an Activity type vertex. Custom game attributes must be inserted by 'AddAttribute' function fi
13 //  NewAgentVertex(label, details): Creates an Agent type vertex. Custom game attributes must be inserted by 'AddAttribute' function first
14 //  NewEntityVertex(label, details): Creates an Entity type vertex. Custom game attributes must be inserted by 'AddAttribute' function first
15 //  NewVertex(): Creates an user-defined <type> vertex.
16 //  AddAttribute(name, value): Adds a new attribute to the attribute list.
17 //                  The attribute's name and value are informed by the user and before invoking NewVertex or any of its variants.
18 //  PopulateAttributes(): Add unity-related attributes to the attribute list. Invoked from NewVertex or any of its variants.
19 //  ClearList(): Clean the attribute list for the next vertex. Invoked from NewVertex or any of its variants.
20 //  GenerateInfluence(tag, ID, name, value): Stores information about the current vertex that is used to influenciate other vertices
21 //  HasInfluence(tag): Checks if there is any influence instance of 'tag' for the current vertex and generates their appropriate edges
22 //  HasInfluence_ID(ID): Checks if there is any influence instance of 'ID' for the current vertex and generates their appropriate edges
23 //  RemoveInfluenceTag(tag): Removes all influences that belongs to the group 'tag' defined by the user
24 //  RemoveInfluenceTag(ID): Removes all influences of 'ID' defined by the user
25 //
26 //-------------------------------------------------------------------------------------------------------------
27 // How to use:
28 //
29 // 1) Invoke 'AddAttribute' to add any custom or game specific attributes that is desired to be stored
30 // 2) Invoke the any of the 'NewVertex' typed functions when an action is executed to store provenance information about the action
31 // 3) Then invoke 'HasInfluence' function for each desired 'tag' or 'ID' to check if there is anything stored that influenced the current act
32 // 4) If the current action can influence another action, then invoke 'GenerateInfluence' by defining its 'tag' and influence 'ID'
33 // 5) If any influence effect expired, then invoke 'RemoveInfluenceTag' or 'RemoveInfluenceID' to remove that influence
34 //=============================================================================================================
```

# Influence &Provenance Controller

```
#pragma strict

//=============================================================================================================================
// Script for storing influence edges for the entire game
// Attach this script in an Empty GameObject that is never destroyed during the game
//(In the same GameObject for ProvenanceGatherer)
// Link it to ProvenanceGatherer
//
// Uses ArrayList for influence edges
// All functions are automatically invoked and controlled by 'ExtractProvenance' script script
//
// If you desire to manually clean/erase the influence list, then invoke 'CleanInfluence' function
//=============================================================================================================================
```

```
#pragma strict

//=============================================================================================================================
// Script for storing vertices and edges
// Attach this script in an Empty GameObject that is never destroyed during the game
//(In the same GameObject for InfluenceController)
//
// Uses one ArrayList for vertices and another for edges
// All functions are automatically invoked and controlled by 'ExtractProvenance' script or 'InfluenceController' script
//=============================================================================================================================

```

Introduction
Provenance in Games
Unity3D Provenance scripts
Unity3D Example
Conclusion

# UNITY3D EXAMPLE

# Unity3D

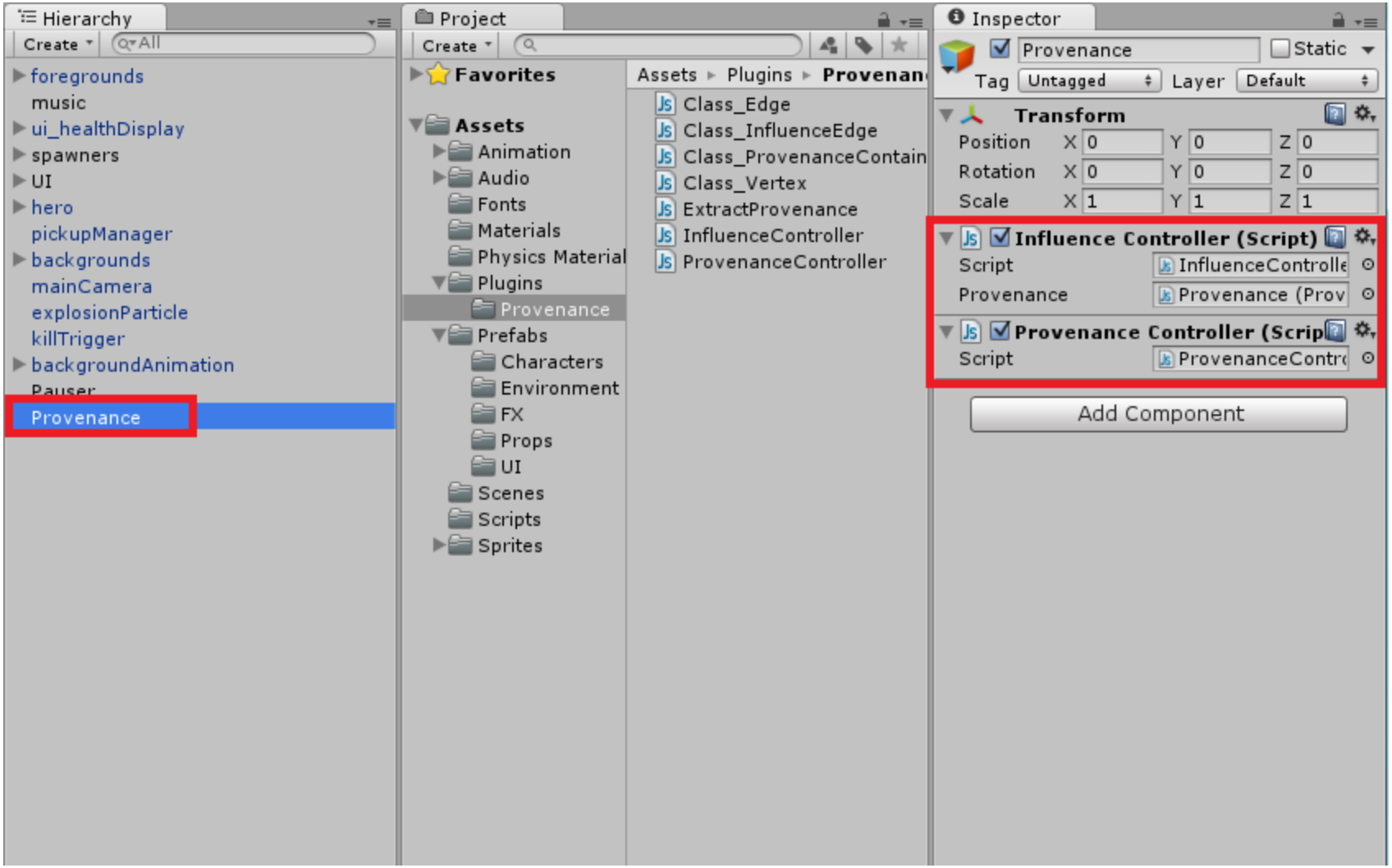

Unity - Level.unity - 2D Tutorial - PC, Mac & Linux Standalone*
File Edit Assets GameObject Component Window Help
Center Local
Layers 2 by 3
Game
Free Aspect
Maximize on Play Stats Gizmos
Score: 0

# Unity3D

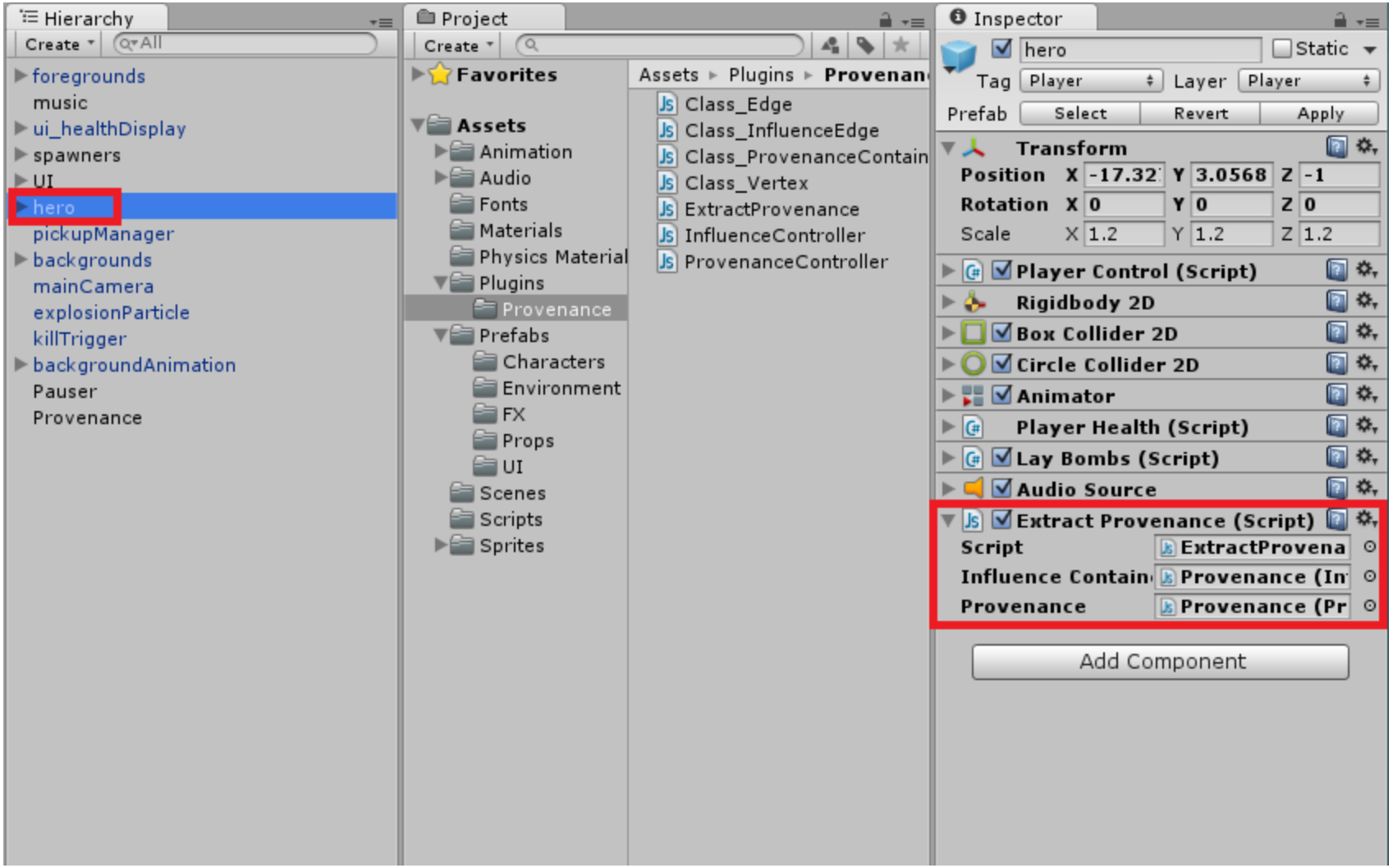
- Create an Empty Game Object
  - Attach ***Provenance Controller*** script
  - Attach ***Influence Controller*** script

- For each Character/Agent
  - Attach ***Extract Provenance*** script

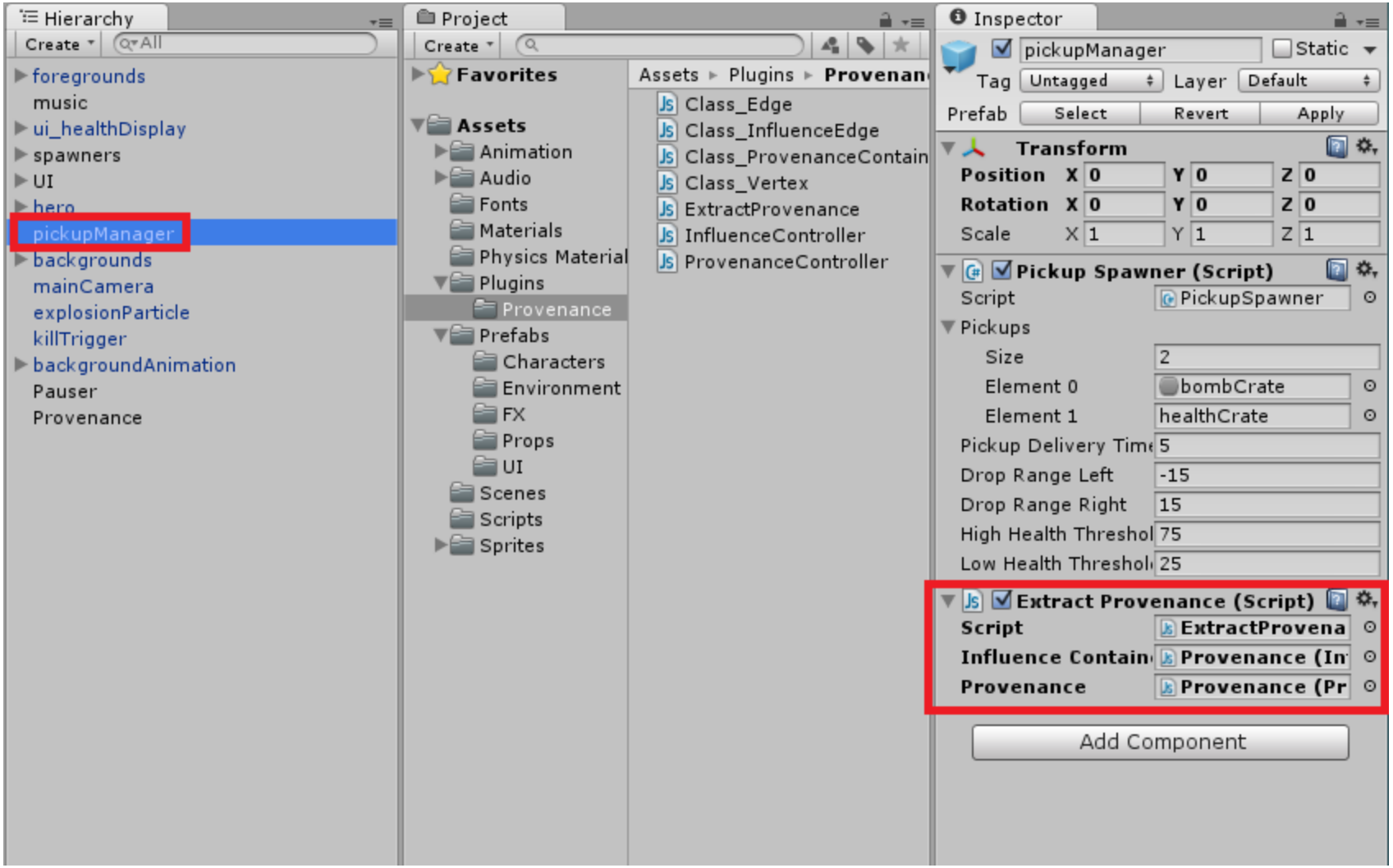
# Unity3D

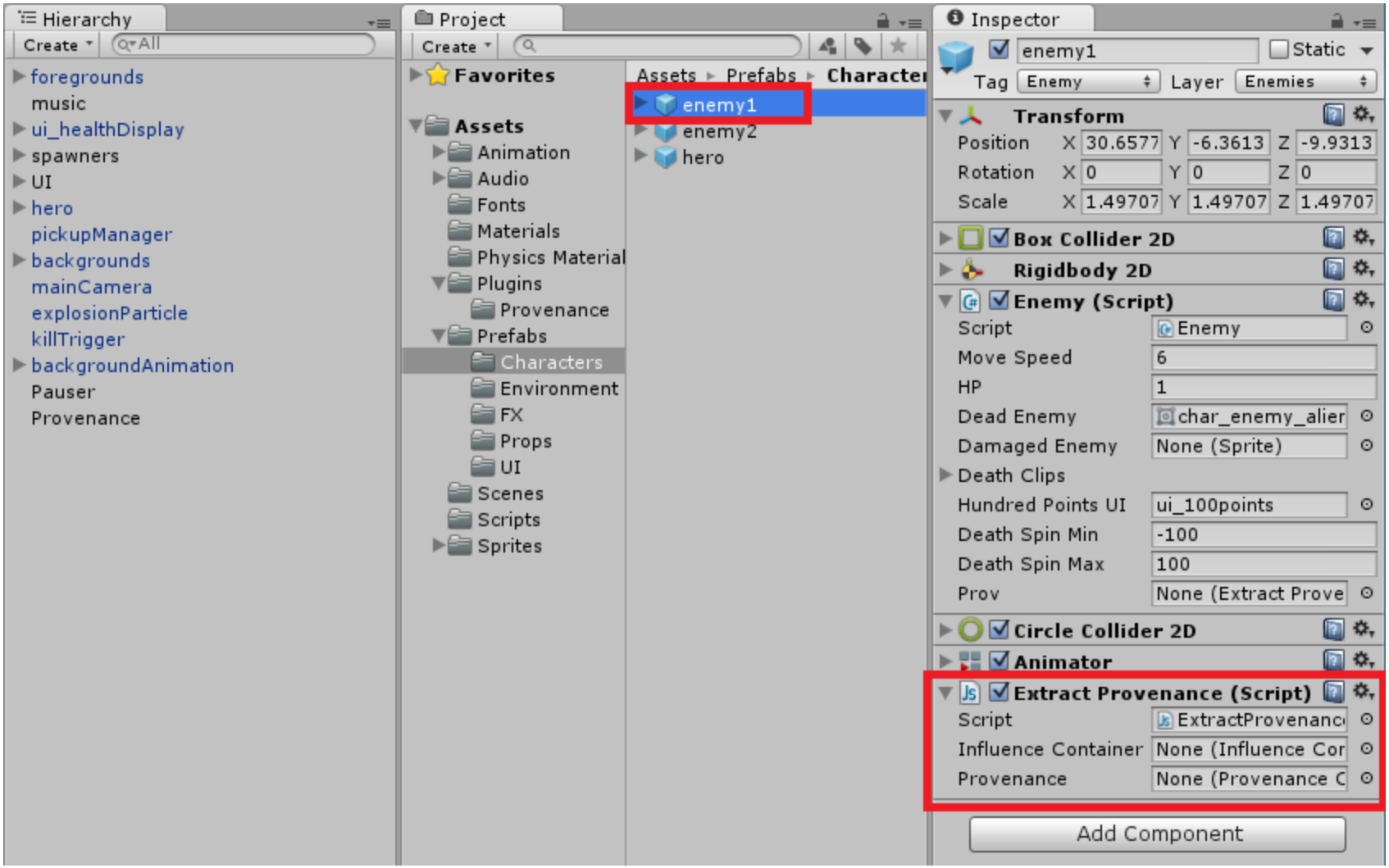

Hierarchy
Create
All
foregrounds
music
ui_healthDisplay
spawners
UI
hero
pickupManager
backgrounds
mainCamera
explosionParticle
killTrigger
backgroundAnimation
Pauser
Provenance
Project
Create
Favorites
Assets
Animation
Audio
Fonts
Materials
Physics Material
Plugins
Provenance
Prefabs
Characters
Environment
FX
Props
UI
Scenes
Scripts
Sprites
Assets ▸ Plugins ▸ Provenan
Class_Edge
Class_InfluenceEdge
Class_ProvenanceContain
Class_Vertex
ExtractProvenance
InfluenceController
ProvenanceController
Inspector
Provenance
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
Influence Controller (Script)
Script
InfluenceControlle
Provenance
Provenance (Prov
Provenance Controller (Scrip
Script
ProvenanceContro
Add Component

# Unity3D

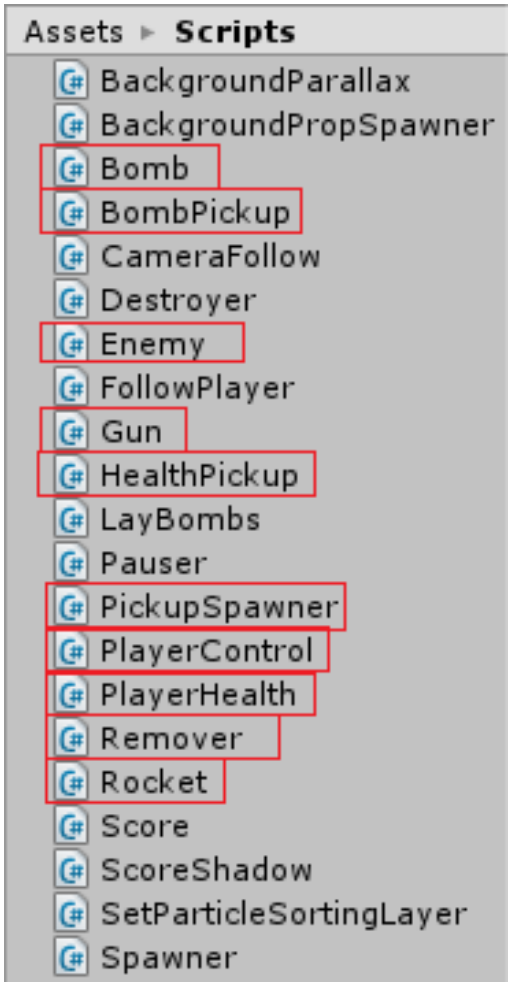

Hierarchy
Create
All
foregrounds
music
ui_healthDisplay
spawners
UI
hero
pickupManager
backgrounds
mainCamera
explosionParticle
killTrigger
backgroundAnimation
Pauser
Provenance
Project
Create
Favorites
Assets
Animation
Audio
Fonts
Materials
Physics Material
Plugins
Provenance
Prefabs
Characters
Environment
FX
Props
UI
Scenes
Scripts
Sprites
Assets ▸ Plugins ▸ Provenan
Class_Edge
Class_InfluenceEdge
Class_ProvenanceContain
Class_Vertex
ExtractProvenance
InfluenceController
ProvenanceController
Inspector
hero
Static
Tag
Player
Layer
Player
Prefab
Select
Revert
Apply
Transform
Position X -17.32 Y 3.0568 Z -1
Rotation X 0 Y 0 Z 0
Scale X 1.2 Y 1.2 Z 1.2
Player Control (Script)
Rigidbody 2D
Box Collider 2D
Circle Collider 2D
Animator
Player Health (Script)
Lay Bombs (Script)
Audio Source
Extract Provenance (Script)
Script ExtractProvena
Influence Contain Provenance (In
Provenance Provenance (Pr
Add Component

# Unity3D


Hierarchy
Create
All
foregrounds
music
ui_healthDisplay
spawners
UI
hero
pickupManager
backgrounds
mainCamera
explosionParticle
killTrigger
backgroundAnimation
Pauser
Provenance
Project
Create
Favorites
Assets
Animation
Audio
Fonts
Materials
Physics Material
Plugins
Provenance
Prefabs
Characters
Environment
FX
Props
UI
Scenes
Scripts
Sprites
Assets ▸ Plugins ▸ Provenan
Class_Edge
Class_InfluenceEdge
Class_ProvenanceContain
Class_Vertex
ExtractProvenance
InfluenceController
ProvenanceController
Inspector
pickupManager
Static
Tag
Untagged
Layer
Default
Prefab
Select
Revert
Apply
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Pickup Spawner (Script)
Script
PickupSpawner
Pickups
Size
2
Element 0
bombCrate
Element 1
healthCrate
Pickup Delivery Time
5
Drop Range Left
-15
Drop Range Right
15
High Health Threshol
75
Low Health Threshol
25
Extract Provenance (Script)
Script
ExtractProvena
Influence Contain
Provenance (In
Provenance
Provenance (Pr
Add Component


Hierarchy
Create
All
foregrounds
music
ui_healthDisplay
spawners
UI
hero
pickupManager
backgrounds
mainCamera
explosionParticle
killTrigger
backgroundAnimation
Pauser
Provenance
Project
Create
Favorites
Assets
Animation
Audio
Fonts
Materials
Physics Material
Plugins
Provenance
Prefabs
Characters
Environment
FX
Props
UI
Scenes
Scripts
Sprites
Assets ▸ Prefabs ▸ Character
enemy1
enemy2
hero
Inspector
enemy1
Static
Tag
Enemy
Layer
Enemies
Transform
Position X 30.6577 Y -6.3613 Z -9.9313
Rotation X 0 Y 0 Z 0
Scale X 1.49707 Y 1.49707 Z 1.49707
Box Collider 2D
Rigidbody 2D
Enemy (Script)
Script Enemy
Move Speed 6
HP 1
Dead Enemy char_enemy_alier
Damaged Enemy None (Sprite)
Death Clips
Hundred Points UI ui_100points
Death Spin Min -100
Death Spin Max 100
Prov None (Extract Prove
Circle Collider 2D
Animator
Extract Provenance (Script)
Script ExtractProvenanc
Influence Container None (Influence Cor
Provenance None (Provenance C
Add Component

# 2D Platformer Example Scripts Breakdown




- **Enemy**
  - PlayerHealth
    - Enemy Attack (Action)
    - Player take Damage (Influence)
- **PlayerControl**
  - Bomb
    - Player Secondary Attack (Action)
    - Enemy take Damage, Area of Effect (Influence)
  - Gun
    - Player Primary Attack (Action)
    - Spawn **Rocket**
  - Remover
    - Player Death (Action)
  - Rocket
    - Enemy Damage (Influence)
- **PickupSpawner**
  - HealthPickup
    - Health Item (Object)
    - Heal (Influence)
  - BombPickup
    - Bomb Ammunition (Object)
    - More Bombs (Influence)

# Enemy

```
//=========================================================
// Provenance
//=========================================================
public void Prov_Enemy()
{
    Prov_GetEnemyAttributes();
    prov.NewAgentVertex("Enemy" + this.GetInstanceID(),"");
    Prov_Idle();
}

public void Prov_GetEnemyAttributes()
{
    prov.AddAttribute("Health", HP.ToString());
}

public void Prov_Hurt(string infID)
{
    Prov_GetEnemyAttributes();
    prov.NewActivityVertex("Taking Hit","");
    prov.HasInfluence_ID(infID);
}

public void Prov_Idle()
{
    Prov_GetEnemyAttributes();
    prov.NewActivityVertex("Walking","");
}

public string Prov_Attack(float damageAmount)
{
    Prov_GetEnemyAttributes();
    prov.NewActivityVertex("Attacking","");
    prov.GenerateInfluence("Player", this.GetInstanceID().ToString(), "Damage", (-damageAmount).ToString(), 1);
    return this.GetInstanceID().ToString();
}

public void Prov_Death()
{
    Prov_GetEnemyAttributes();
    prov.NewActivityVertex("Dead","");
    prov.GenerateInfluence("Player_Score", this.GetInstanceID().ToString(), "Score", "100", 1);
}
```

# Enemy

```
public ExtractProvenance prov = null;

void Awake()
{
    // Setting up the references.
    ren = transform.Find("body").GetComponent<SpriteRenderer>();
    frontCheck = transform.Find("frontCheck").transform;
    score = GameObject.Find("Score").GetComponent<Score>();

    // Provenance
    GameObject provObj = GameObject.Find("Provenance");
    prov = GetComponent<ExtractProvenance>();
    prov.influenceContainer = provObj.GetComponent<InfluenceController>();
    prov.provenance = provObj.GetComponent<ProvenanceController>();
    Prov_Enemy();
}

public void Flip()
{
    // Multiply the x component of localScale by -1.
    Vector3 enemyScale = transform.localScale;
    enemyScale.x *= -1;
    transform.localScale = enemyScale;

    // Provenance
    Prov_Idle();
}
```

# Enemy

```
void Death()
{
    // Find all of the sprite renderers on this object and it's children.
    SpriteRenderer[] otherRenderers = GetComponentsInChildren<SpriteRenderer>();

    // Disable all of them sprite renderers.
    foreach(SpriteRenderer s in otherRenderers)
    {
        s.enabled = false;
    }

    // Re-enable the main sprite renderer and set it's sprite to the deadEnemy sprite.
    ren.enabled = true;
    ren.sprite = deadEnemy;

    // Increase the score by 100 points
    score.score += 100;

    // Set dead to true.
    dead = true;

    // Allow the enemy to rotate and spin it by adding a torque.
    rigidbody2D.fixedAngle = false;
    rigidbody2D.AddTorque(Random.Range(deathSpinMin,deathSpinMax));

    // Find all of the colliders on the gameobject and set them all to be triggers.
    Collider2D[] cols = GetComponents<Collider2D>();
    foreach(Collider2D c in cols)
    {
        c.isTrigger = true;
    }

    // Play a random audioclip from the deathClips array.
    int i = Random.Range(0, deathClips.Length);
    AudioSource.PlayClipAtPoint(deathClips[i], transform.position);

    // Create a vector that is just above the enemy.
    Vector3 scorePos;
    scorePos = transform.position;
    scorePos.y += 1.5f;

    // Instantiate the 100 points prefab at this point.
    Instantiate(hundredPointsUI, scorePos, Quaternion.identity);

    // Provenance
    Prov_Death();
}
```

# Player Health

```
//============================================================
// Provenance
//============================================================
void Prov_TakeDamage(GameObject enemy)
{
    string infID = enemy.GetComponent<Enemy>().Prov_Attack(damageAmount);
    playerControl.Prov_TakeDamage(infID);
}
```

# Player Health

```
void OnCollisionEnter2D (Collision2D col)
{
    // If the colliding gameobject is an Enemy...
    if(col.gameObject.tag == "Enemy")
    {
        // ... and if the time exceeds the time of the last hit plus the time between hits...
        if (Time.time > lastHitTime + repeatDamagePeriod)
        {
            // ... and if the player still has health...
            if(health > 0f)
            {
                // ... take damage and reset the lastHitTime.
                TakeDamage(col.transform);
                lastHitTime = Time.time;

                // Provenance
                Prov_TakeDamage(col.gameObject);
            }
            // If the player doesn't have health, do some stuff, let him fall into the river to reload the level.
            else
            {
                // Find all of the colliders on the gameobject and set them all to be triggers.
                Collider2D[] cols = GetComponents<Collider2D>();
                foreach(Collider2D c in cols)
                {
                    c.isTrigger = true;
                }

                // Move all sprite parts of the player to the front
                SpriteRenderer[] spr = GetComponentsInChildren<SpriteRenderer>();
                foreach(SpriteRenderer s in spr)
                {
                    s.sortingLayerName = "UI";
                }

                // ... disable user Player Control script
                GetComponent<PlayerControl>().enabled = false;

                // ... disable the Gun script to stop a dead guy shooting a nonexistant bazooka
                GetComponentInChildren<Gun>().enabled = false;

                // ... Trigger the 'Die' animation state
                anim.SetTrigger("Die");
            }
        }
    }
}
```

# Player Control

```
//============================================================
// Provenance
//============================================================
public void Prov_Jump()
{
    Prov_GetPlayerAttributes();
    prov.NewActivityVertex("Jump","");
    prov.HasInfluence("Player_Score");
}

public void Prov_Player()
{
    Prov_GetPlayerAttributes();
    prov.NewAgentVertex("Player","");
}

public void Prov_Walking()
{
    Prov_GetPlayerAttributes();
    prov.NewActivityVertex("Walking","");
    prov.HasInfluence("Player_Score");
}

public void Prov_Shoot()
{
    Prov_GetPlayerAttributes();
    prov.NewActivityVertex("Shooting","");
    prov.HasInfluence("Player_Score");
    //Generated Influence in the ammo instantiation (Rocket)
}

public void Prov_TakeDamage(string infID)
{
    Prov_GetPlayerAttributes();
    prov.NewActivityVertex("Being Hit","");
    // Check Influence
    prov.HasInfluence_ID(infID);
}
```

```
public void Prov_Death()
{
    Score score = GameObject.Find("Score").GetComponent<Score>();

    prov.AddAttribute("Health", "0");
    prov.AddAttribute("Score", score.score.ToString());
    prov.NewActivityVertex("Death","Drowned");
    Prov_Export();
}

public void Prov_GetPlayerAttributes()
{
    PlayerHealth hp = GetComponent<PlayerHealth>();
    Score score = GameObject.Find("Score").GetComponent<Score>();
    LayBombs laybomb = GetComponent<LayBombs>();

    prov.AddAttribute("Health", hp.health.ToString());
    prov.AddAttribute("Score", score.score.ToString());
    prov.AddAttribute("Bombs", laybomb.bombCount.ToString());
}

public void Prov_PickUp(string infID)
{
    Prov_GetPlayerAttributes();
    prov.NewActivityVertex("PickedUp","");
    // Check Influence
    prov.HasInfluence_ID(infID);
}

public void RemoveBombInfluence(string bomb)
{
    prov.RemoveInfluenceID(bomb);
}
public void Prov_LayBomb(string bomb)
{
    Prov_GetPlayerAttributes();
    prov.NewActivityVertex("LayingBomb","");
    prov.HasInfluence("Player_Score");
    prov.GenerateInfluence("Enemy", bomb, "Damage", "-2");
}

void Prov_Export()
{
    Debug.Log ("Exported");
    GameObject ProvObj = GameObject.Find("Provenance");
    ProvenanceController prov = ProvObj.GetComponent<ProvenanceController>();
    prov.Save("2D_Provenance");
}
```

# Player Control

```
// Provenance
public ExtractProvenance prov;

 void Awake()
 {
     // Setting up references.
     groundCheck = transform.Find("groundCheck");
     anim = GetComponent<Animator>();

     // Provenance
     Prov_Player();
 }


void Flip ()
{
    // Switch the way the player is labelled as facing.
    facingRight = !facingRight;

    // Multiply the player's x local scale by -1.
    Vector3 theScale = transform.localScale;
    theScale.x *= -1;
    transform.localScale = theScale;

    // Provenance
    Prov_Walking();
}
```

```
void FixedUpdate ()
{
    // Cache the horizontal input.
    float h = Input.GetAxis("Horizontal");

    // The Speed animator parameter is set to the absolute value of the horizontal input.
    anim.SetFloat("Speed", Mathf.Abs(h));

    // If the player is changing direction (h has a different sign to velocity.x) or hasn't reached maxSpeed yet...
    if(h * rigidbody2D.velocity.x < maxSpeed)
        // ... add a force to the player.
        rigidbody2D.AddForce(Vector2.right * h * moveForce);

    // If the player's horizontal velocity is greater than the maxSpeed...
    if(Mathf.Abs(rigidbody2D.velocity.x) > maxSpeed)
        // ... set the player's velocity to the maxSpeed in the x axis.
        rigidbody2D.velocity = new Vector2(Mathf.Sign(rigidbody2D.velocity.x) * maxSpeed, rigidbody2D.velocity.y);

    // If the input is moving the player right and the player is facing left...
    if(h > 0 && !facingRight)
        // ... flip the player.
        Flip();
    // Otherwise if the input is moving the player left and the player is facing right...
    else if(h < 0 && facingRight)
        // ... flip the player.
        Flip();

    // If the player should jump...
    if(jump)
    {
        // Set the Jump animator trigger parameter.
        anim.SetTrigger("Jump");

        // Play a random jump audio clip.
        int i = Random.Range(0, jumpClips.Length);
        AudioSource.PlayClipAtPoint(jumpClips[i], transform.position);

        // Add a vertical force to the player.
        rigidbody2D.AddForce(new Vector2(0f, jumpForce));

        // Make sure the player can't jump again until the jump conditions from Update are satisfied.
        jump = false;

        //Prov Extraction
        Prov_Jump();
    }
```

# Bomb

```
//============================================================
// Provenance
//============================================================
void Prov_LayBomb()
{
    PlayerControl playerControl = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerControl>();
    playerControl.Prov_LayBomb(this.GetInstanceID().ToString());
}
void PROV_Boom(GameObject enemy)
{
    enemy.GetComponent<Enemy>().Prov_Hurt(this.GetInstanceID().ToString());
}

void Prov_RemoveBomb()
{
    PlayerControl playerControl = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerControl>();
    playerControl.RemoveBombInfluence(this.GetInstanceID().ToString());
}
```

# Bomb

```
IEnumerator BombDetonation()
{
    // Provenance
    Prov_LayBomb();

    // Play the fuse audioclip.
    AudioSource.PlayClipAtPoint(fuse, transform.position);

    // Wait for 2 seconds.
    yield return new WaitForSeconds(fuseTime);

    // Explode the bomb.
    Explode();
}
```

```
public void Explode()
{
    // The player is now free to lay bombs when he has them.
    layBombs.bombLaid = false;

    // Make the pickup spawner start to deliver a new pickup.
    pickupSpawner.StartCoroutine(pickupSpawner.DeliverPickup());

    // Find all the colliders on the Enemies layer within the bombRadius.
    Collider2D[] enemies = Physics2D.OverlapCircleAll(transform.position, bombRadius, 1 << LayerMask.NameToLayer("Enemies"));

    // For each collider...
    foreach(Collider2D en in enemies)
    {
        // Check if it has a rigidbody (since there is only one per enemy, on the parent).
        Rigidbody2D rb = en.rigidbody2D;
        if(rb != null && rb.tag == "Enemy")
        {
            // Find the Enemy script and set the enemy's health to zero.
            rb.gameObject.GetComponent<Enemy>().HP = 0;

            // Provenance
            PROV_Boom(rb.gameObject);

            // Find a vector from the bomb to the enemy.
            Vector3 deltaPos = rb.transform.position - transform.position;

            // Apply a force in this direction with a magnitude of bombForce.
            Vector3 force = deltaPos.normalized * bombForce;
            rb.AddForce(force);
        }
    }

    // Provenance
    Prov_RemoveBomb();

    // Set the explosion effect's position to the bomb's position and play the particle system.
    explosionFX.transform.position = transform.position;
    explosionFX.Play();

    // Instantiate the explosion prefab.
    Instantiate(explosion,transform.position, Quaternion.identity);

    // Play the explosion sound effect.
    AudioSource.PlayClipAtPoint(boom, transform.position);

    // Destroy the bomb.
    Destroy (gameObject);
}
```

# Gun

```
void Update ()
{
    // If the fire button is pressed...
    if(Input.GetButtonDown("Fire1"))
    {
        // ... set the animator Shoot trigger parameter and play the audioclip.
        anim.SetTrigger("Shoot");
        audio.Play();

        // If the player is facing right...
        if(playerCtrl.facingRight)
        {
            // ... instantiate the rocket facing right and set it's velocity to the right.
            Rigidbody2D bulletInstance = Instantiate(rocket, transform.position, Quaternion.Euler(new Vector3(0,0,0))) as Rigidbody2D;
            bulletInstance.velocity = new Vector2(speed, 0);
        }
        else
        {
            // Otherwise instantiate the rocket facing left and set it's velocity to the left.
            Rigidbody2D bulletInstance = Instantiate(rocket, transform.position, Quaternion.Euler(new Vector3(0,0,180f))) as Rigidbody2D;
            bulletInstance.velocity = new Vector2(-speed, 0);
        }

        // Provenance
        playerCtrl.Prov_Shoot();
    }
}
```

# Remover

```
//=============================================================
// Provenance
//=============================================================
void Prov_Death(GameObject player)
{
    PlayerControl playerControl = player.GetComponent<PlayerControl>();
    playerControl.Prov_Death();
}
```

# Remover

```
void OnTriggerEnter2D(Collider2D col)
{
    // If the player hits the trigger...
    if(col.gameObject.tag == "Player")
    {
        // .. stop the camera tracking the player
        GameObject.FindGameObjectWithTag("MainCamera").GetComponent<CameraFollow>().enabled = false;

        // .. stop the Health Bar following the player
        if(GameObject.FindGameObjectWithTag("HealthBar").activeSelf)
        {
            GameObject.FindGameObjectWithTag("HealthBar").SetActive(false);
        }

        // ... instantiate the splash where the player falls in.
        Instantiate(splash, col.transform.position, transform.rotation);
        // ... destroy the player.
        Destroy (col.gameObject);

        //Provenance Death
        Prov_Death(col.gameObject);

        // ... reload the level.
        StartCoroutine("ReloadGame");
    }
    else
    {
        // ... instantiate the splash where the enemy falls in.
        Instantiate(splash, col.transform.position, transform.rotation);

        // Destroy the enemy.
        Destroy (col.gameObject);
    }
}
```

# Rocket

```
//============================================================
// Provenance
//============================================================
void Prov_generateInfluence()
{
    GameObject heroObj = GameObject.FindGameObjectWithTag("Player");
    PlayerControl playerControl = heroObj.GetComponent<PlayerControl>();
    playerControl.prov.GenerateInfluence("Enemy", this.GetInstanceID().ToString(), "Damage", "-1", 1);
}

void Prov_DamageEnemy(GameObject enemy)
{
    Prov_generateInfluence();
    enemy.GetComponent<Enemy>().Prov_Hurt(this.GetInstanceID().ToString());
}
```

# Rocket

```
void OnTriggerEnter2D (Collider2D col)
{
    // If it hits an enemy...
    if(col.tag == "Enemy")
    {
        // ... find the Enemy script and call the Hurt function.
        col.gameObject.GetComponent<Enemy>().Hurt();

        // Call the explosion instantiation.
        OnExplode();

        // Provenance
        Prov_DamageEnemy(col.gameObject);

        // Destroy the rocket.
        Destroy (gameObject);
    }
    // Otherwise if it hits a bomb crate...
    else if(col.tag == "BombPickup")
    {
        // ... find the Bomb script and call the Explode function.
        col.gameObject.GetComponent<Bomb>().Explode();

        // Destroy the bomb crate.
        Destroy (col.transform.root.gameObject);

        // Destroy the rocket.
        Destroy (gameObject);
    }
    // Otherwise if the player manages to shoot himself...
    else if(col.gameObject.tag != "Player")
    {
        // Instantiate the explosion and destroy the rocket.
        OnExplode();
        Destroy (gameObject);
    }
}
```

# Pickup Spawner

```
//============================================================
// Provenance
//============================================================
private void Prov_SpawnAgent()
{
    ExtractProvenance prov = GetComponent<ExtractProvenance>();
    prov.NewAgentVertex("ItemSpawner","");

}

public void Prov_SpawnPickup(string type, string infID, float value)
{
    ExtractProvenance prov = GetComponent<ExtractProvenance>();
    prov.NewEntityVertexFromAgent(type,"");
    prov.GenerateInfluence("Player", infID, type, value.ToString(), 1);

}
```

# Pickup Spawner

```
void Awake ()
{
    // Setting up the reference.
    playerHealth = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerHealth>();

    // Provenance
    Prov_SpawnAgent();
}
```

# Bomb & Health Pickup

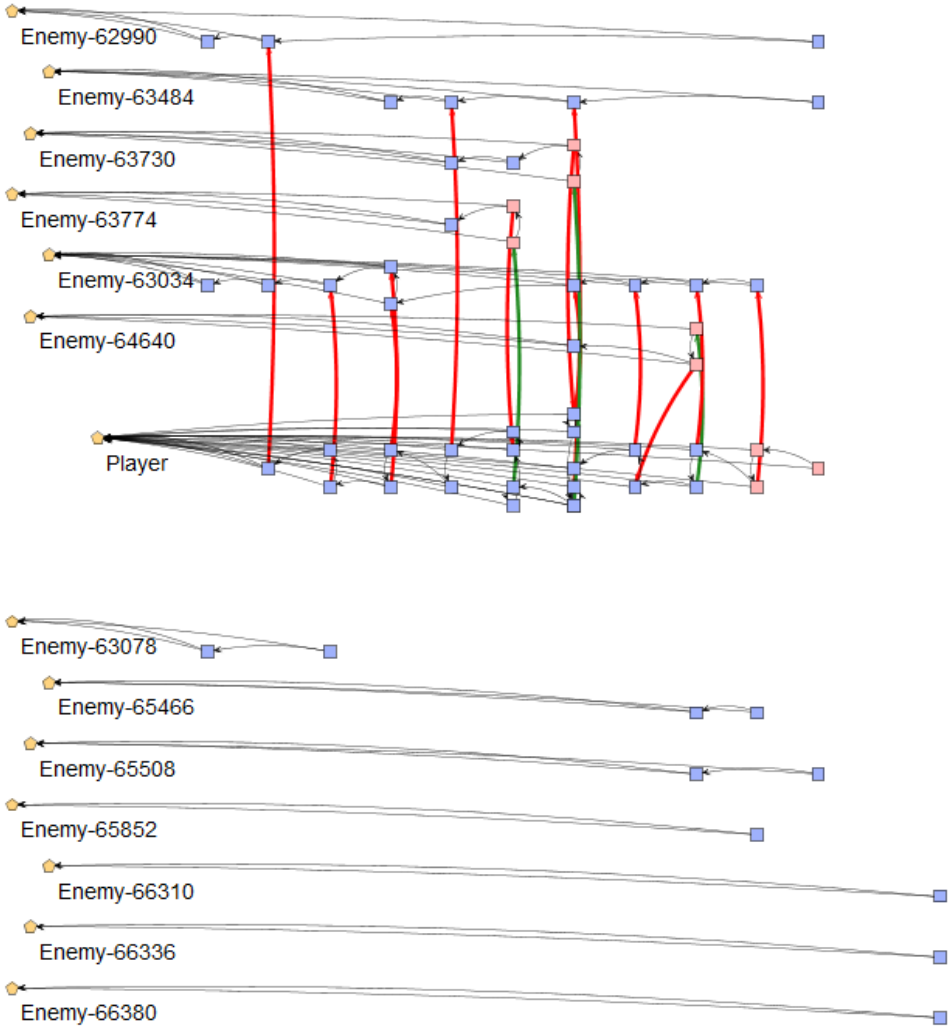
```
//============================================================
// Provenance
//============================================================

private void Prov_SpawnPickup()
{
    PickupSpawner pickupSpawner = GameObject.Find("pickupManager").GetComponent<PickupSpawner>();
    pickupSpawner.Prov_SpawnPickup("Bomb", this.GetInstanceID().ToString(), 1);
}

private void Prov_Pickup()
{
    PlayerControl playerControl = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerControl>();
    playerControl.Prov_PickUp(this.GetInstanceID().ToString());
}
```

```
//============================================================
// Provenance
//============================================================

private void Prov_SpawnPickup()
{
    pickupSpawner.Prov_SpawnPickup("LifeBox", this.GetInstanceID().ToString(), healthBonus);
}

private void Prov_Pickup()
{
    PlayerControl playerControl = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerControl>();
    playerControl.Prov_PickUp(this.GetInstanceID().ToString());
}
```

# Bomb & Health Pickup

```
void Awake()
{
    // Setting up the reference.
    anim = transform.root.GetComponent<Animator>();

    // Provenance
    Prov_SpawnPickup();
}


void OnTriggerEnter2D (Collider2D other)
{
    // If the player enters the trigger zone...
    if(other.tag == "Player")
    {
        // ... play the pickup sound effect.
        AudioSource.PlayClipAtPoint(pickupClip, transform.position);

        // Increase the number of bombs the player has.
        other.GetComponent<LayBombs>().bombCount++;

        // Provenance
        Prov_Pickup();

        // Destroy the crate.
        Destroy(transform.root.gameObject);
    }
    // Otherwise if the crate lands on the ground...
    else if(other.tag == "ground" && !landed)
    {
        // ... set the animator trigger parameter Land.
        anim.SetTrigger("Land");
        transform.parent = null;
        gameObject.AddComponent<Rigidbody2D>();
        landed = true;
    }
}
```

# Provenance Graph Example



Introduction
Provenance in Games
Unity3D Provenance scripts
Unity3D Example
Conclusion

# CONCLUSION

# Conclusion

- Contributions
  - Rich Data Extraction
    - Broader Range of Analysis
  - Cause-and-Effect Relationships
  - Game Provenance Visualization
- Future Work
  - Automatic Inferences
  - Pattern Detection
  - Graph Layouts

# Provenance in Games

**Troy C. Kohwalter,**

*Instituto de Computação – Universidade Federal Fluminense (UFF)*