

# Arquitectura ECS

Una manera alternativa de hacer videojuegos

# Sobre mí

- Alejandro Pascual Pozo
- ¡Estuve aquí en la fundación!
- WebRTC / WASM / Backend / Rust
- 7 años aficionado al desarrollo de videojuegos

# Sobre esta presentación

- Introducción
  1. Motores de juego
  2. Motores comerciales modernos
- Definición
  3. ¿Qué es ECS?
  4. Ventajas de ECS
- Adopción
  5. ECS en la industria
  6. Programando sobre ECS
- Conclusiones

# 1. Motores de juego

# Todo vale

```
Console.WriteLine("Introduce tu nombre para ganar:");  
  
string name = Console.ReadLine();  
  
Console.WriteLine($"¡Felicidades {name}, has ganado!")
```

# Bucle de juego

```
void Main() {  
    // ...  
    while (true) {  
        // game logic  
        // rendering  
        // sound  
        // ...  
    }  
}
```

# Implementación propia

```
void Main() {  
    var context = new Context(  
        new Renderer(),  
        new AudioPlayer(),  
        // other resources  
    );  
    while (true) {  
        // game loop  
    }  
}
```

# Entidades

```
class WarriorEntity {  
    int health;  
    int regenerationPerSecond;  
    Sprite sprite;  
    Sound sound;  
    void Regenerate(float deltaSeconds) { /* ... */ }  
    void Render(Renderer renderer) { /* ... */ }  
    void Play(AudioPlayer audioPlayer) { /* ... */ }  
}
```

- Bucle de juego infernal
- Entidades muy complejas



# Bucle de juego en entidades

```
class WarriorEntity {  
    int health;  
    int regenerationPerSecond;  
    Sprite sprite;  
    Sound sound;  
    void Update(Context context) { /* ... */ }  
    void Regenerate(float deltaSeconds) { /* ... */ }  
    void Render(Renderer renderer) { /* ... */ }  
    void Play(AudioPlayer audioPlayer) { /* ... */ }  
}
```

- `Update()` muy compleja
- Entidades muy complejas

# Componentes

```
class WarriorEntity {
    HealthComponent health;
    SpriteComponent sprite;
    SoundComponent sound;
    void Update(Context context) { /* ... */ }
}
class HealthComponent {
    int health;
    int regenerationPerSecond;
    void Regenerate(float deltaSeconds) { /* ... */ }
}
class SpriteComponent { /* ... */ }
class SoundComponent { /* ... */ }
```

- `Update()` muy compleja

# Bucle de juego en componentes

```
class WarriorEntity {
    HealthComponent health;
    SpriteComponent sprite;
    SoundComponent sound;
}
class HealthComponent {
    int health;
    int regenerationRate;
    void Update(Context context) { /* ... */ }
    void Regenerate(float deltaSeconds) { /* ... */ }
}
class SpriteComponent { /* ... */ }
class SoundComponent { /* ... */ }
```

- ¿Utilidad de `WarriorEntity` ?

# Componentes dinámicos

```
class Entity { Set<IComponent> components; }  
interface IComponent {  
    void Update(Context context) { /* ... */ }  
}  
  
class HealthComponent : IComponent { /* ... */ }  
class SpriteComponent : IComponent { /* ... */ }  
class SoundComponent : IComponent { /* ... */ }  
  
var leonidas = new Entity();  
leonidas.components.Add(new HealthComponent());  
leonidas.components.Add(new SpriteComponent());  
leonidas.components.Add(new SoundComponent());
```

- Context muy complejo

# Bucle de juego estático

```
void Main() {  
    var context = new Context(  
        new Renderer(),  
        new AudioPlayer(),  
        // other resources  
        new Set<Entity>(),  
    );  
    while (true) {  
        foreach (var entity in context.entities) {  
            foreach (var component in entity) {  
                component.Update(context);  
            }  
        }  
    }  
}
```

## **2. Motores comerciales modernos**

# Oportunidad de negocio

Externalizar el desarrollo del motor

# Bucle de juego flexible

- `Update()` , `LateUpdate()` , `FixedUpdate()`
- `Start()` , `Awake()`
- `OnDestroy()` , `OnApplicationFocus()`



# Componentes

Transform , Mesh/SpriteRenderer , Rigidbody ...

# APIs ergonómicas

- `GetComponent<T>()`
- `FindGameObjectsWithTag(string tag)`
- `RequireComponent(Type requiredComponent)`
- `SerializeField`

# Mucho más

- Gestión de assets
- Escenas
- Serialización / deserialización
- Networking
- Editores: escenas, animaciones, shaders...
- Portabilidad a múltiples plataformas
- Extensiones

# ¿Hay espacio para mejorar?

- Extender las funcionalidades
- Mejorar la arquitectura

# 3. ¿Qué es ECS?

# Ya tenemos dos piezas

- Entidades: colecciones de componentes
- Componentes: piezas combinables
  - Propiedades (datos)
  - Fragmento del bucle de juego (funcionalidad)

# Mirada crítica a EC

¿Deseamos fragmentos del bucle de juego...

- ...en ***todos*** los componentes?
- ...asociados a un ***único*** componente?

# Funcionalidad y datos ¿relación 1:1?

- Nombre: sin funcionalidad
- Inventario de un cofre: sin funcionalidad propia
- Vida y regeneración: ambos necesarios
- Equipo, área y regeneración: los tres necesarios



# Funcionalidades sobre grupos

Solo debe pelear una pareja de guerreros

- Un miembro del equipo rojo
- Un miembro del equipo azul
- La más cercana entre sí

# ¿Fragmentar el bucle de juego?

- No queremos volver al bucle infernal
- Meterlo dentro de las clases era lo más intuitivo
- ¿Por qué no usar funciones de primera clase?

# Sistemas

Reglas que actúan directamente sobre el contexto

```
void Regenerate(Query<Health, Regeneration> query) {  
    foreach ((health, regeneration) in query) {  
        health.Heal(regeneration.rate);  
    }  
}
```

# Definiciones ECS

- Entidades: colecciones de componentes
- Componentes: propiedades (datos)
- Sistemas: reglas *universales*

# Convivir con estos problemas

- ¿De verdad es tan malo?
- ¿Merece la pena empezar de nuevo?

## **4. Ventajas de ECS**

# Ergonomía

- Se acabaron los managers
- Se acabaron los requires
- Se acabaron los nulls
- Se acabaron los singletons

# Rendimiento

- El desacoplamiento no solo nos ayuda a nosotros
- La lógica ha dado un paso atrás
- Ahora es más fácil aplicar ciertas optimizaciones



# Cache prefetching

Demo interactiva

# Eliminemos los punteros

- ~~Referencias intra-entidad~~ → Queries
- ~~Referencias sobre conjuntos~~ → Queries
- ~~Referencias a recursos~~ → Acceso directo
- ~~Referencias a otras entidades~~ → Ids

# Componentes en memoria

# Ordenemos

# It's free!

- Gracias a ECS, no empeora la ergonomía
- El motor gestiona estas optimizaciones

# **5. ECS en la industria**

# Motores propietarios en producción

*Overwatch Gameplay Architecture and Netcode - GDC*

# Unity



# Primera aparición en 2017

*Massive Battle in the Spellsouls Universe - Unite 2017*

# Años en desarrollo

*ECS "Megacity" walkthrough - Unite 2018*

# Mejoras de ergonomía y rendimiento

*ECS "Megacity" walkthrough - Unite 2018*

# **1.0 y producción en 2022**






# Unreal Engine 5

*Large Numbers of Entities with Mass - State of UE 2022*

# Godot

# Otros proyectos de código abierto

# Bevy

- ECS nativo y eficiente 
- Máxima ergonomía 
- Rust 
- 2.5 años de desarrollo 
- Sin editor 



# 6. Programando sobre ECS

# Hello world en Rust

```
fn main() {  
    println!("Hello world!");  
}
```

Hello world!

# Hello world en Bevy

```
fn main() {  
    App::new()  
        .add_startup_system(say_hello_world)  
        .run();  
}  
  
fn say_hello_world() {  
    println!("Hello world!");  
}
```

Hello world!

# Introduciendo el game loop

```
fn main() {  
    App::new()  
        .add_plugins(MinimalPlugins)  
        // CorePlugin, TimePlugin  
        // ScheduleRunnerPlugin  
        .add_system(say_hello_world)  
        .run();  
}  
  
fn say_hello_world() {  
    println!("Hello world!");  
}
```

```
Hello world!  
Hello world!  
...
```

# Nuestro primer componente

```
#[derive(Component)]  
struct Health {  
    current: f32,  
}
```

# Nuestras primeras entidades

```
app.add_startup_system(spawn_entities);  
  
fn spawn_entities(mut commands: Commands) {  
    commands.spawn(Health { current: 100.0 });  
    commands.spawn(Health { current: 50.0 });  
}
```

# Nuestra primera query

```
app.add_system(print_healths);

fn print_healths(query: Query<&Health>) {
    println!("Healths:");
    for health in &query {
        println!("- {}", health.current);
    }
}
```

# Ejemplo vida: código

```
fn main() {
    App::new()
        .add_plugins(MinimalPlugins)
        .add_startup_system(spawn_entities)
        .add_system(print_healths)
        .run();
}

#[derive(Component)]
struct Health {
    current: f32,
}

fn spawn_entities(mut commands: Commands) {
    commands.spawn(Health { current: 100.0 });
    commands.spawn(Health { current: 50.0 });
}

fn print_healths(query: Query<&Health>) {
    println!("Healths:");
    for health in &query {
        println!("- {}", health.current);
    }
}
```



# Ejemplo vida: salida

```
Healths:
```

```
- 50
```

```
- 100
```

```
Healths:
```

```
- 50
```

```
- 100
```

```
...
```

# Introduciendo otro componente

```
#[derive(Component)]  
struct Regeneration {  
    rate: f32,  
}
```

# Modificando nuestro spawner

```
fn spawn_entities(mut commands: Commands) {  
    commands.spawn(Health { current: 100.0 });  
    commands.spawn(  
        Health { current: 50.0 },  
        Regeneration { rate: 1.0 },  
    ));  
}
```

# Queries sobre tuplas

```
app.add_system(regenerate_healths);

fn regenerate_healths(
    mut query: Query<(&mut Health, &Regeneration)>,
) {
    for (mut health, regeneration) in &mut query {
        health.current += regeneration.rate;
    }
}
```

# Ejemplo regeneración: código

```
fn main() { /* ... */ }

#[derive(Component)]
struct Health { /* ... */ }

#[derive(Component)]
struct Regeneration {
    rate: f32,
}

fn spawn_entities(mut commands: Commands) {
    commands.spawn(Health { current: 100.0 });
    commands.spawn((Health { current: 50.0 }, Regeneration { rate: 1.0 }));
}

fn regenerate_healths(mut query: Query<(&mut Health, &Regeneration)>) {
    for (mut health, regeneration) in &mut query {
        health.current += regeneration.rate;
    }
}

fn print_healths(query: Query<&Health>) { /* ... */ }
```

# Ejemplo regeneración: salida

```
Healths:
```

```
- 100
```

```
- 50
```

```
Healths:
```

```
- 100
```

```
- 51
```

```
Healths:
```

```
- 100
```

```
- 52
```

```
...
```

# TimePlugin

```
fn regenerate_healths(  
    mut query: Query<(&mut Health, &Regeneration)>,  
    time: Res<Time>,  
) {  
    for (mut health, regeneration) in &mut query {  
        health.current += regeneration.rate * time.delta_seconds();  
    }  
}
```

# Nuestro primer plugin

```
fn main() {
    App::new()
        .add_plugins(MinimalPlugins)
        .add_plugin(HealthPlugin)
        .add_startup_system(spawn_entities)
        .run();
}

struct HealthPlugin;

impl Plugin for HealthPlugin {
    fn build(&self, app: &mut App) {
        app.add_system(regenerate_healths)
            .add_system(print_healths);
    }
}
```



# Repaso

- Entidad: colección de componentes con ID
- Componente: struct asociado a una entidad
- Recurso: struct global
- Sistema: función ejecutada por el motor
- Query: iterador sobre tuplas de componentes
- Comandos: gestor de entidades y componentes
- Plugin: colección de elementos modularizada

# Más elementos de la arquitectura

- Eventos: `EventWriter<T>` y `EventReader<T>`
- Estados: `on_enter`, `on_exit` y `on_update`
- Condiciones de ejecución arbitrariamente complejas
- Paralelización automática y manual
- Mundos

# Bevy es más que la arquitectura ECS

Pero todo está construido sobre ella: ¡plugins!

# Plugins oficiales

# DefaultPlugins

- HierarchyPlugin: Parent , Children
- TransformPlugin: Transform , GlobalTransform
- RenderPlugin: Camera , Image , Mesh y Shader
- AudioPlugin: Audio
- AssetPlugin: AssetLoader y Assets<T>
- InputPlugin: Input<T>
- DiagnosticsPlugin: Diagnostics
- Y más...

# Conclusiones

# Conclusiones

- Futuro prometedor
- Pick your poison
  - ECS inmaduro en motores maduros
  - ECS maduro en motores inmaduros
- El aprendizaje de EC y ECS es complementario

# Antes de terminar

- [github.com/pascualex/ecs-presentation](https://github.com/pascualex/ecs-presentation)
- [github.com/bevyengine/bevy/tree/main/examples](https://github.com/bevyengine/bevy/tree/main/examples)
- [learn.unity.com/tutorial/entity-component-system](https://learn.unity.com/tutorial/entity-component-system)
- ¿Preguntas?