Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

# Desarrollo de videojuegos 3d

Master Universitario en Desarrollo de Videojuegos

# Arcade BlockOut

Juan Siverio Rojas

La Laguna, 27 de junio de 2023

Contenido



# 1.1 Arcade BlockOut

En esta actividad se ha creado el escenario mediante la herramienta de *Modeling de Unreal.*

Concretamente, los objetos que se encuentran dentro de la carpeta llamada *"Escenario BlockOut" d*el outliner, han sido creados mediante esta herramienta. Básicamente hay una única malla por planta, más dos escaleras. Además, se han agregado algunos objetos para el desarrollo del GamePlay descargados de diferentes repositorios.

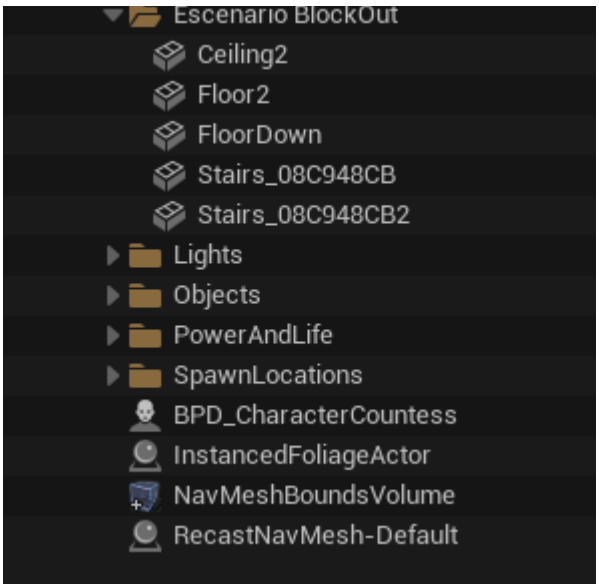


*Ilustración 1.Aspecto del outliner*

## 1.2 Objetivo

En esta tarea, el jugador controla a la condesa, la cual tiene habilitadas varias animaciones. Se desarrollo en un escenario de tres plantas, laberíntico, aunque la planta superior es abierta. El objetivo es destruir a todos los enemigos, buscándolos entre las tres plantas. Se dispone de bolas verdes para recuperar vida, y bolas azules para obtener el poder “Power” que permite destruir los muros agregados durante 10 segundos.

El juego intenta infundir algo de terror, ya que la protagonista (el jugador) es una condesa vampírica, es por eso, que la iluminación elegida es oscura, lo cual es lo lógico ya que el edificio de tres plantas intenta representar un monumento vampírico (intenté asimilarlo al aparecido en la película de Blade), por lo que no debe tener mucha iluminación proveniente del sol, por eso, no tiene ventanas. Las proporciones de las puertas y escaleras son mayores que los personajes, así como la altura de techos, ya que también habitan orcos, cuyo tamaño es bastante mayor al de las vampiras.

La ubicación de las escaleras, necesarias para cambiar de planta, tanto para subir, como para bajar, viene representada por una piedra circular con iluminación, para ser encontrada más fácilmente, ya que de lo contrario puede ser difícil encontrar las salidas.

Hay pasadizos secretos, bolas de poder escondidas.

*Ilustración 2. Marcador de escaleras indicado con la flecha en azul.*

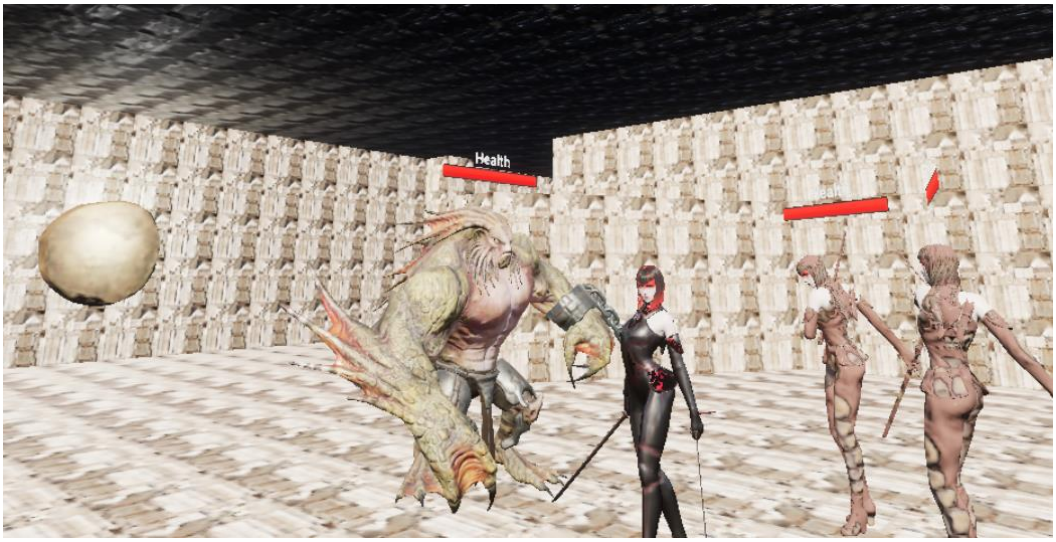


*Ilustración 3. Captura de la planta 1. Jugador siendo atacado.*

### 1.2.1 Características.

- Hay dos tipos de NPC enemigos:

  Condesa (color tierra): Realiza solo un ataque cuando está cerca. Su movimiento es más

lento que el del jugador, pero más rápido que el del orco.

Orcos: Realiza también un solo ataque, pero si alcanza al jugador, quita más vida. Su movimiento es más lento que el de la condesa marrón.

- Todos los personajes son instanciados de forma aleatoria repartidos entre unos "Target Points" ya predefinidos en la escena. La aleatoriedad se refiere tanto al punto en el que nuestro jugador aparece, como a la aparición de orcos o condesas. Esto permite, que no resulte tan repetitiva la escena, comenzando en diferentes posiciones del mapa y teniendo diferente cantidad de orcos y condesas, así como en diferentes posiciones en cada partida.
- Los NPC, utilizan NavMesh y AI, para su movimiento, además apoyado por la máquina de estados para las acciones. Los movimientos de ambos NPC son, que una vez nuestro jugador entra en el campo de visión de la cámara(raytrace), el NPC correrá hacia nosotros. Cuando esté cerca pasará a atacarnos si se comprueba que estamos dentro del ángulo configurado entre el NPC y el personaje.
- El Hud principal, muestra la barra de vida del jugador y la barra de tiempo del poder "Power".

  Un Hud de tipo espacial, situado encima de cada enemigo, muestra la vida de cada uno de estos.
- Hay algunas bolas azules, "Power", las cuales dan tanto al jugador como a los enemigos la posibilidad de "romper" los muros que se encuentran dispersos entre las tres plantas, permitiendo escapar rápidamente (se habilita la simulación física de las muros agregados, no de las paredes de la malla creada con *modeling*). Este poder dura 10 segundos, y se puede ver esa duración en una barra de progreso situada en el Hud principal, así como una emisión de partículas que acompañarán al personaje durante ese tiempo. Cuando finaliza, se vuelve a desactivar la simulación física.
- El jugador (la condesa) es en tercera persona y dispone de dos ataques:
  - Simple -> botón izquierdo del ratón.
  - Fuerte -> botón derecho.
  - Rotación de cámara con el mouse.
  - Movimiento -> WASD.
  - Salto -> barra espaciadora

## 1.3 Estructura

El proyecto está realizado conjuntamente entre C++ y Blueprints.

Las clases C++, comienzan por "CPP_XXX" y las Blueprints *derivadas* de estas clases C++, comienzan por BPD_XXX. El resto, usan la nomenclatura natural de Unreal, BP_XXX (clases blueprint) AB_XXX (Animation Blueprint).

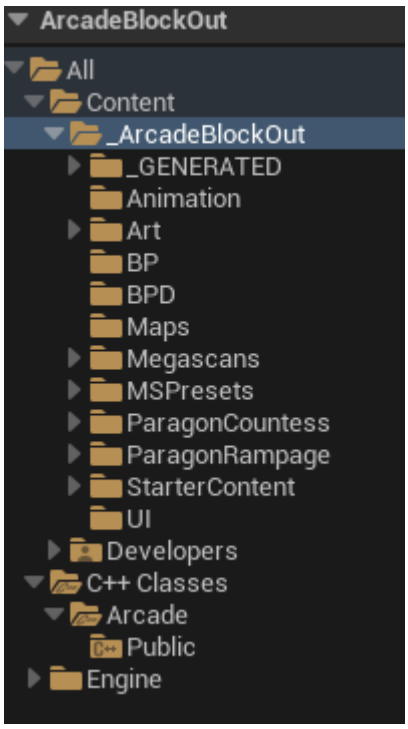


*Ilustración 4.Organización de las carpetas.*


Ilustración 5.Clases C++

Las clases son las siguientes:

- CPP_MyGameMode

  Clase base para BPD_MyGameMode.

  Encargada de Instanciar a los personajes de forma aleatoria, así como recolectar toda la información de los Actores, clasificándolos por etiqueta, clase, ya que me facilita el acceso a ellos desde cualquier punto y con menor coste.

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Enemies")
TArray<TSubclassOf<class ACPP_Enemies>> enemies_;

UPROPERTY(VisibleAnyWhere, BlueprintReadOnly, Category = "ActorsSpawned")
TSet<AActor *> allActors_;

UPROPERTY(VisibleAnyWhere, BlueprintReadOnly, Category = "ActorsSpawned")
TMap<FString, AActor*> actorsByName_;

UPROPERTY(VisibleAnyWhere, BlueprintReadOnly, Category = "ActorsSpawned")
FTransform locationInitialPlayer_;

TMap<FString, TSet<AActor*>> actorsByTag_;
TMap<FString, TSet<AActor*>> actorsByClass_;
```

```
void FindAllActors();
void ShowAllActors();
void ShowAllActorsByTags(FString tag);
void ShowAllActorsByClass(FString clas);

template<class T>
void ShowTheseActors(TArray<T*>& draft);


void DestroyAndupdateActors(AActor* actor);
void SpawnAndupdateActors(AActor* actor);

AActor * FindActorByName(FString name);

TArray<AActor*> FindActorsByTag(FString tag);

template<class T>
TArray<T*> FindActorsByClass(FString clas);
```

*Ilustración 6.Propiedades y Métodos principales CPP_MyGameMode.*

- CPP_CharacterCountess

  Clase base para BPD_CharacterCountess.

  Encargada del movimiento de cámara y personaje, así como de sus ataques. El movimiento de la cámara es libre, pero el movimiento del personaje siempre se orientará hacia donde esté apuntando la cámara.

- BPD_CharacterCountess

  Clase que deriva de CPP_CharacterCountess

  Implementa toda la lógica de daño, vida, power.

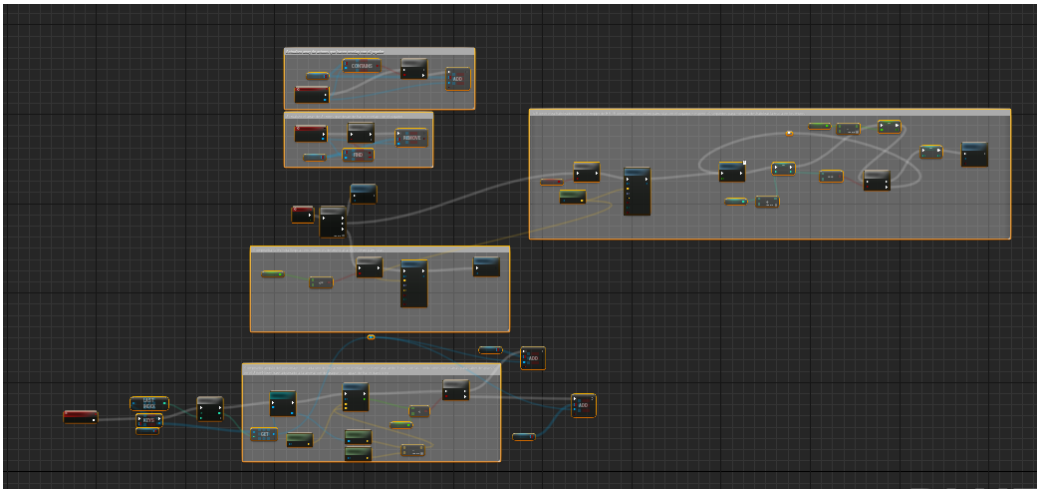

*Ilustración 7.Parte del código blueprints de BPD_CharacterCountess.*

- CPP_Enemies.

  Clase base para CPP_OrcoEnemy y CPP_CountessEnemy.

  Definen variables y funciones como Angle() que permiten calcular en ángulo entre dos vectores, o eventos.

```
public:

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Fight, meta
        bool bFightSoft_;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Fight, meta =
        bool bFightHard_;

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Overlap, me
        bool bOverlap_;

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Fight)
        bool applyDamage_ = false;

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Overlap, me
        AActor* actorOverlap_;

    UPROPERTY(EditAnyWhere, BlueprintReadWrite, Category = Fight, meta = (
    float life_ = 1;

    UPROPERTY(EditAnyWhere, BlueprintReadWrite, Category = Fight, meta = (
    float damage_ = 0.1;

public:

    UFUNCTION(BlueprintCallable)
    float Angle(FVector v1, FVector v2); ///calcula el ángulo entre dos v

    UFUNCTION(BlueprintNativeEvent, BlueprintCallable)
    void UpdateWidget();
```

*Ilustración 8.Propiedades y funciones CPP_Enemies*

Estas a su vez tienen clases blueprints derivadas, BPD_CountessEnemy y BPD_OrcoEnemy.

- CPP_MyAiController.

  Clase base para BPD_MyAiController.

  Encargada del movimiento de los NPC (orcos y condesas). Utiliza raytrace y navmesh para el movimiento.

- BPD_MyAicontroller.

  Implemente la lógica del movimiento de los NPC. Deriva de CPP_MyAicontroller.

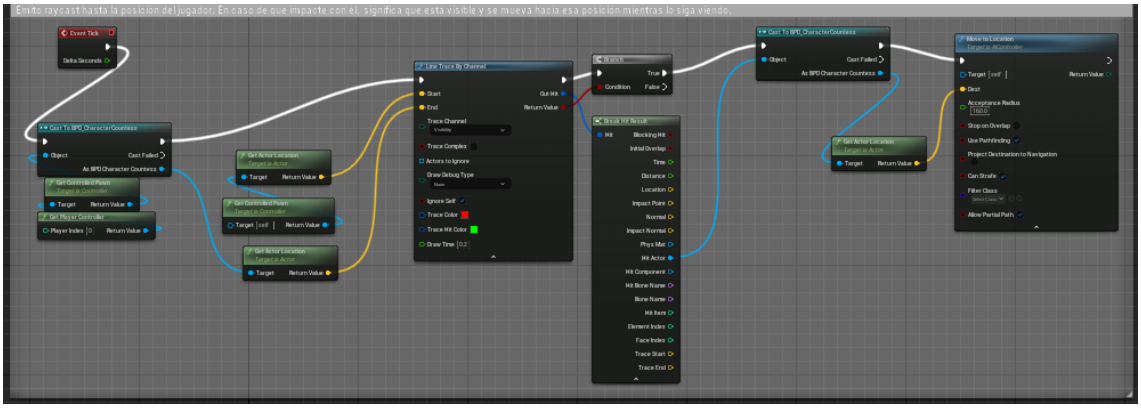

*Ilustración 9.Lógica BPD_MyAIController*

- CPP_PossibleSpawnLocation.

  Clase que deriva de ATargetPoint y que indica una posible localización de spawn de un actor.

- BP_Power.

  Clase Pura blueprints, encargada de llamar a la función power() del jugador para activar la física en los muros.

- BP_Life.

  Clase Pura blueprints, encargada de recargar la barra de vida del jugador.

- BP_AnimAttackNotify.

  Blueprint que deriva de AnimNotifyState, el cual se encarga de aplicar el daño, tanto a los enemigos como al jugador. Esta clase es llamada como un evento aplicado en un momento exacto de la animación, para que el daño se produzca justo en el momento adecuado, cuando el ataque golpea. Así es más realista.

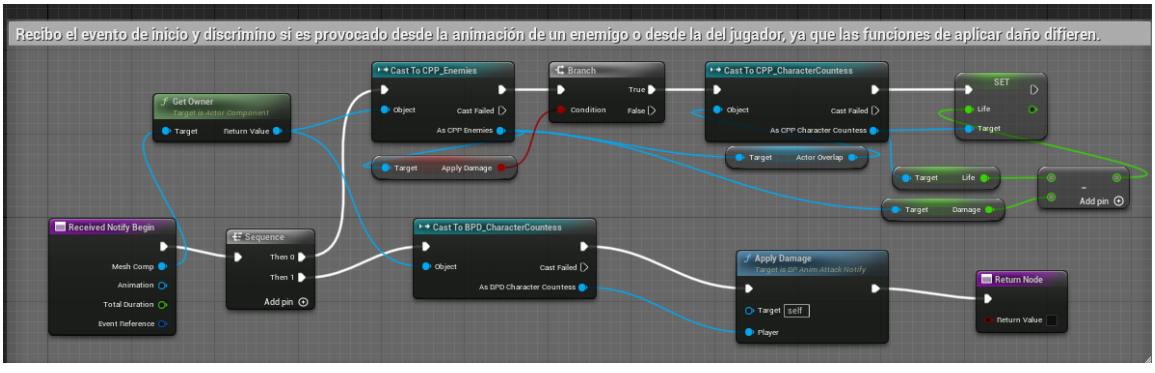


*Ilustración 10. Evento proveniente de la animación.*

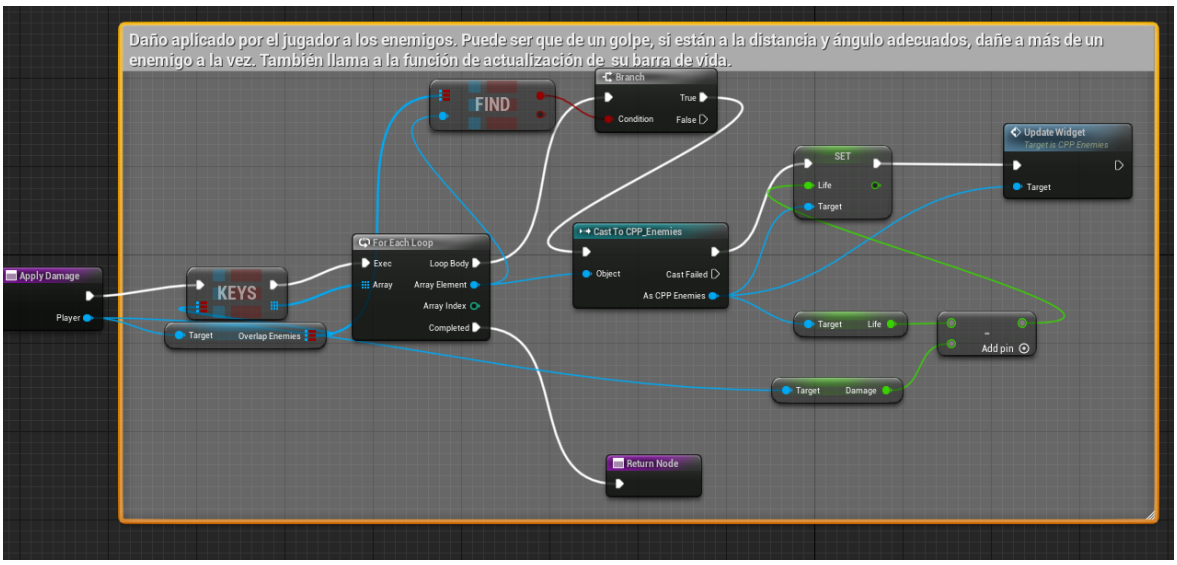


*Ilustración 11.Función para aplicar daño y actualizar Widget.*

- BP_Hud.(Widget)

  Se encarga de la actualización del Hud principal, barra de vida y power.

- BP_Health.(Widget)

  Se encarga de la actualización de la barra de vida de cada personaje.

- AB_CountessEnemy, AB_Countess y AB_OrcoEnemy, clases Animation blueprint, encargadas de almacenar las animaciones y estados de cada personaje.

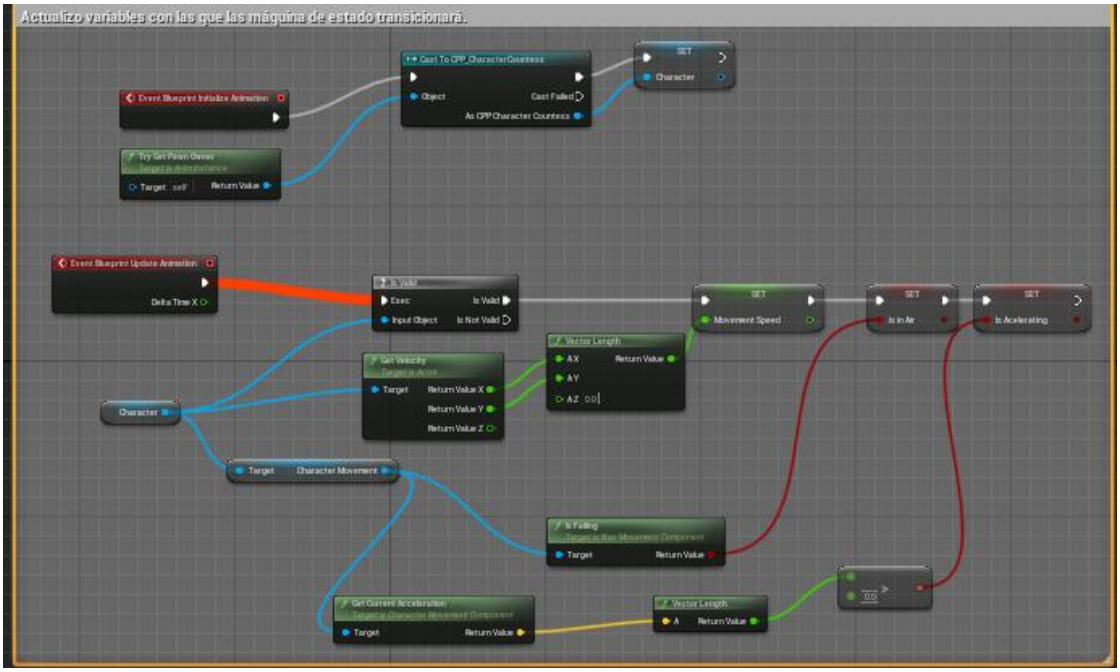


*Ilustración 12.EventGraph Blueprint animación Condesa.*

- Las siguientes clases son menos importantes y algunas no llevan implementación.

  CPP_WALL.

  Todos los muros derivan BPD_Wall que a su vez deriva de esta clase y es la que se busca cuando se quiere activar o desactivar la física en caso de coger el objeto Power.

CPP_Objects y CPP_GameManager, CPP_MyLibrary.

Clases con implementaciones de prueba.