UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería Informática

Motor de juegos 3D/2D en C++ y OpenGL

3D/2D Game engine made using C++ and OpenGL

Realizado por
Javier Aguilera Puerta

Tutorizado por
Francisco de Asís Rus Mansilla

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2024

Trabajo fin de Grado

E.T.S. INGENIERÍA INFORMÁTICA

Motor de juegos 3D/2D en C++ y OpenGL

3D/2D Game engine made using C++ and OpenGL

Autor:

Javier Aguilera Puerta

Tutor:

Francisco de Asís Rus Mansilla

6 de abril de 2024

En agradecimiento

a mi familia y amigos,
por apoyarme y creer en mí.

a Priyanka,
por estar siempre a mi lado.

Resumen

El mundo de los videojuegos ha demostrado ser, lejos de una moda pasajera, una forma de ocio cada vez más popular, entre jóvenes y adultos por igual, con un futuro brillante. Esto ha creado un gran interés en todos los aspectos relacionados con el desarrollo de juegos, y toda una proliferación de cursos, vídeos, libros y revistas sobre el tópico; así como diferentes herramientas gratuitas y de pago para la creación de los mismos.

Este Trabajo de Fin de Grado (TFG) tiene un doble objetivo, por un lado ofrecer un motor de juegos de código abierto y gratuito, que permita a los usuarios crear juegos en dos dimensiones (2D) o tres dimensiones (3D), abstrayéndoles de toda la complejidad subyacente; y una faceta didáctica, explicando a los usuarios, a través de una serie de videos tutoriales y diagramas, la creación de juegos usando el motor para tal fin.

Durante el desarrollo de este TFG, se estudiará la arquitectura necesaria para la elaboración de un motor de juegos, y se analizarán las soluciones disponibles con sus ventajas y desventajas. Finalmente, se realizará un prototipo del motor de juegos, mostrando diferentes casos de uso del motor, incluyendo un juego de tipo plataforma en 2D.

Palabras Claves

Motor de juegos, videojuegos, OpenGL, GLFW, Entidad Componente Sistema

Abstract

The world of videogames has proven to be, far from a temporary trend, an increasingly popular hobby, both among the young and adults, with a foreseeable brilliant future. This has created a surge of interest in the subject of game development: courses, videos, books and magazines; as well as free and paid tools for their creation.

This Bachelor's Final Project (BFP) serves a dual purpose, firstly to offer a free and open source game engine, that allows the users to create two dimensions (2D) or three dimensions (3D) games, abstracting them of all the complexity behind it; secondly, from a didactive perspective explaining to the users about game development using the game engine for it.

During the development of this BFP, the required architecture for making a game engine will be studied, taking into account the present and past solutions, with their advantages and

disadvantages. Finally, a prototype of the game engine will be developed portraying different use cases of the engine, including a platform game in 2D.

Keywords

Game engine, videogames, OpenGL, GLFW, Entity Component System

Índice general

Índice de figuras

Índice de tablas

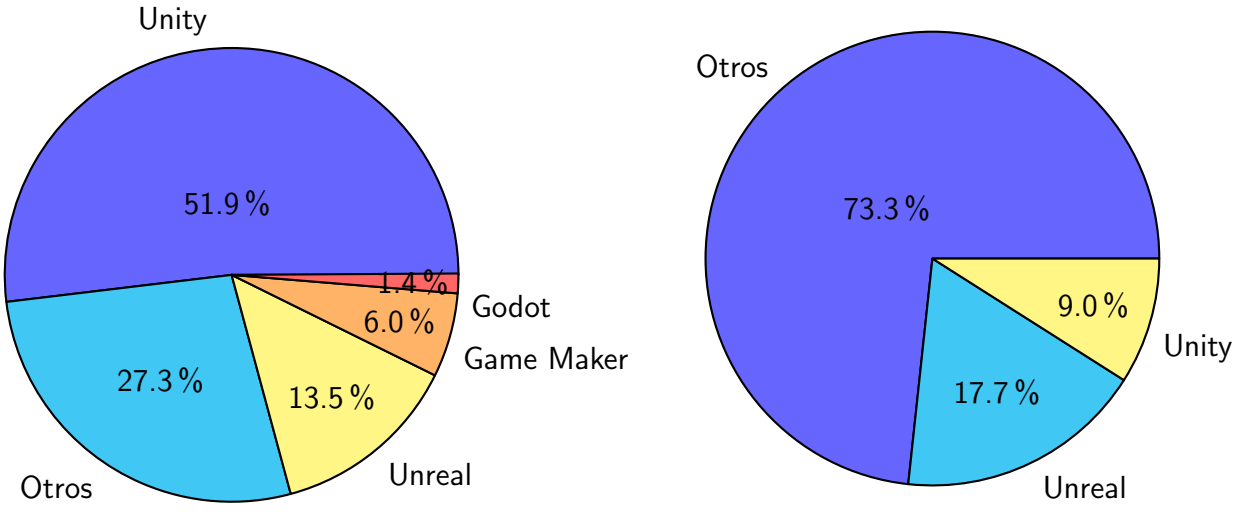
Índice de Códigos

1. Introducción

Los videojuegos, pese a su popularidad, presentan muchas incógnitas para alguien que desee iniciarse en su creación, pudiendo surgir preguntas tales como:

- ¿Cómo organizar el desarrollo de un juego?
- ¿Debería crear mi propio motor de juegos?
- ¿Qué tecnologías usar?
- ¿Qué librerías usar?

En los últimos años, gracias a la popularización de motores como Unity[1], Unreal[2] o Game Maker[3] y sus modelos de negocio, que muchas veces permiten el uso del motor sin licencia, mientras el uso del mismo no sea comercial[4][5][6], han fomentado el llamado fenómeno *indie* en el que individuos o pequeños grupos de amigos se juntan para crear un videojuego. Esto junto a la gran cantidad de tutoriales, vídeos y cursos online ha facilitado mucho esta posibilidad. Como podemos ver en la siguiente Figura 1, el uso de estos motores es muy extendido. [1]



(a) Videojuegos indies

(b) Videojuegos AAA

Figura 1: Distribución de motores usados para videojuegos (septiembre de 2023)

[1] Fuente: *https://gamalytic.com/blog/exploring-the-pc-engine-landscape*

Pero como se ha demostrado tras la reciente polémica con Unity sobre su nueva monetización retroactiva[7], deberían fomentarse más alternativas y opciones de código abierto para que el ecosistema puede seguir creciendo de una forma saludable.

Así mismo estos motores muchas veces vienen sobrecargados con funcionalidades que aumentan su complejidad e incrementan el tiempo para entenderlos o no se ajustan a lo que el usuario necesita. Por suerte la información sobre crear un motor de juegos, con la expansión de internet, se ha hecho cada vez más accesible, pero muchas veces desperdigada.

Este trabajo se propone por lo tanto crear una fuente única con la que un usuario pueda aprender, sin conocimientos previos y desde cero, los pasos a realizar para crear un motor de juegos y el contexto de las decisiones tomadas para formar su propia opinión crítica, así como ofrecer online el fruto de ello con demostraciones para el uso del público general. De tal forma que los usuarios puedan usar el motor para el desarrollo de sus juegos, si así lo desean, y encuentren un set completo de herramientas listo para ello.

1.1. Motivación y Objetivos

La motivación de este proyecto surge a partir de un interés en cómo funcionan por dentro los motores de juegos, así como con la realización de la falta de información en profundidad que cubra en conjunto todas las parte del proceso de realizar uno. El proyecto se orienta dentro un marco didáctico y formativo, con el deseo de ofrecer más opciones gratuitas a las disponibles actualmente. Los objetivos de este proyecto son las siguientes:

O1. Análisis de las soluciones disponibles en el mercado y sus principales diferencias y ventajas en las herramientas que ofrecen.

O2. Estudio de los componentes de un motor de juegos e implementación de forma eficiente.

O3. Desarrollo de un motor de juegos, de código abierto y listo para uso profesional, abstrayendo al usuario toda complejidad a través de una API.

O4. Objetivo didáctico del motor, para ello:

- **O4.1.** Creación de diagramas y videos tutoriales explicando cada parte del motor de juegos y su uso.
- **O4.2.** Programación de diferentes casos de uso del motor y una plantilla ejemplo de juego usando el motor.
- **O4.3.** Elaboración de diagramas y videos tutoriales explicando la creación de un juego de plataforma 2D usando el motor y haciendo uso de la plantilla.

1.2. Metodología

Después de un análisis previo de los componentes necesarios para el motor de juegos y demostraciones necesarias, se organizaron las fases del proyecto usando la herramienta *Trello* Figura 2, y se establecieron fechas concretas para cada fase, en reuniones bimensuales utilizando *Google meets*.

Las fases del proyecto se han enfocado usando la metodología de Prototipo, permitiendo una iteración rápida sobre el feedback dado por el coordinador del proyecto y una validación de los objetivos deseados en cada iteración Figura 3. El proyecto se ha realizado usando *Github* Figura 4 dado que uno de los objetivos era que fuera de código abierto y para mantener un control de versión.

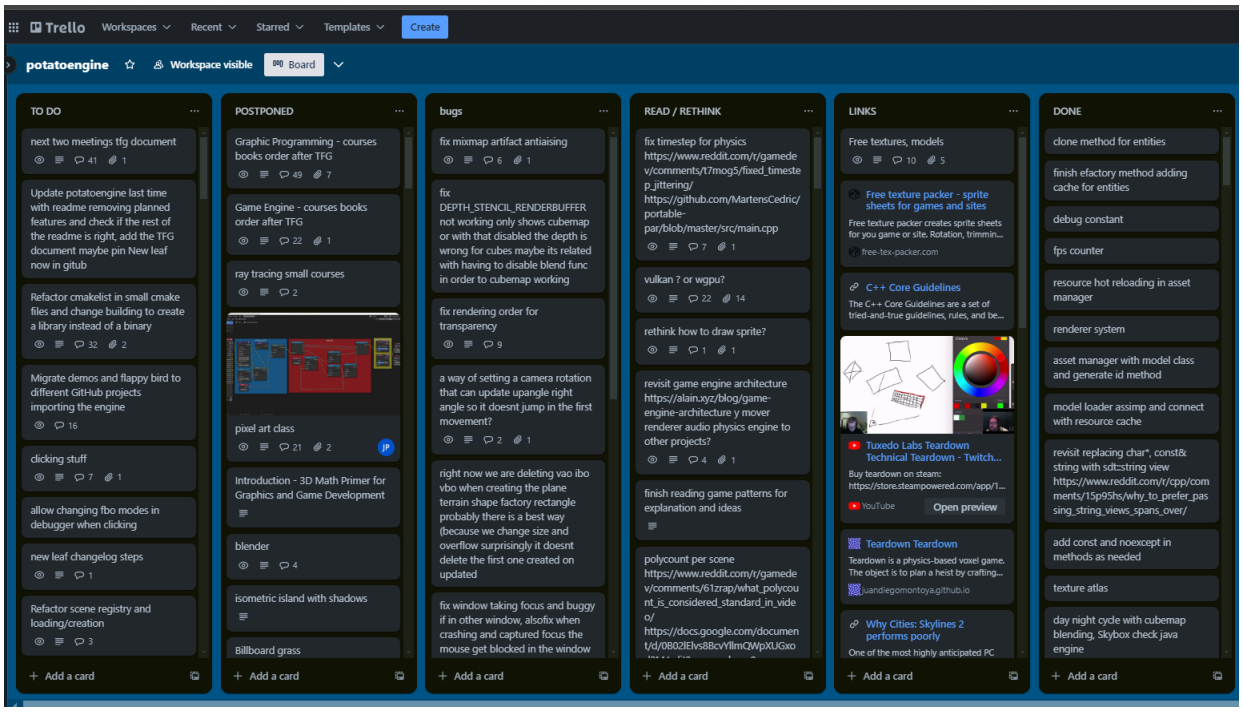


Figura 2: Organización de fases y tareas con Trello

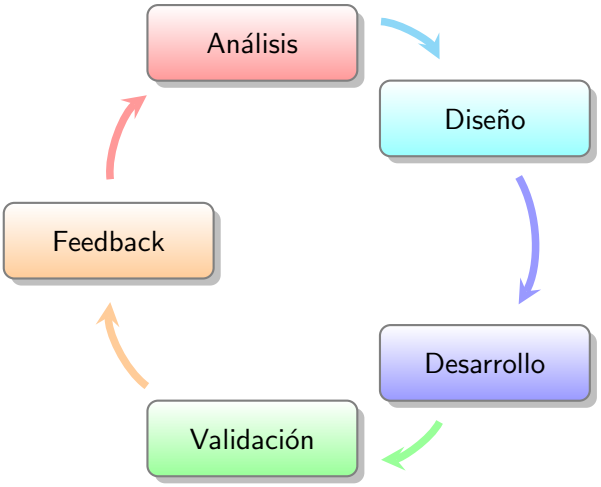


Figura 3: Metodología de Prototipo

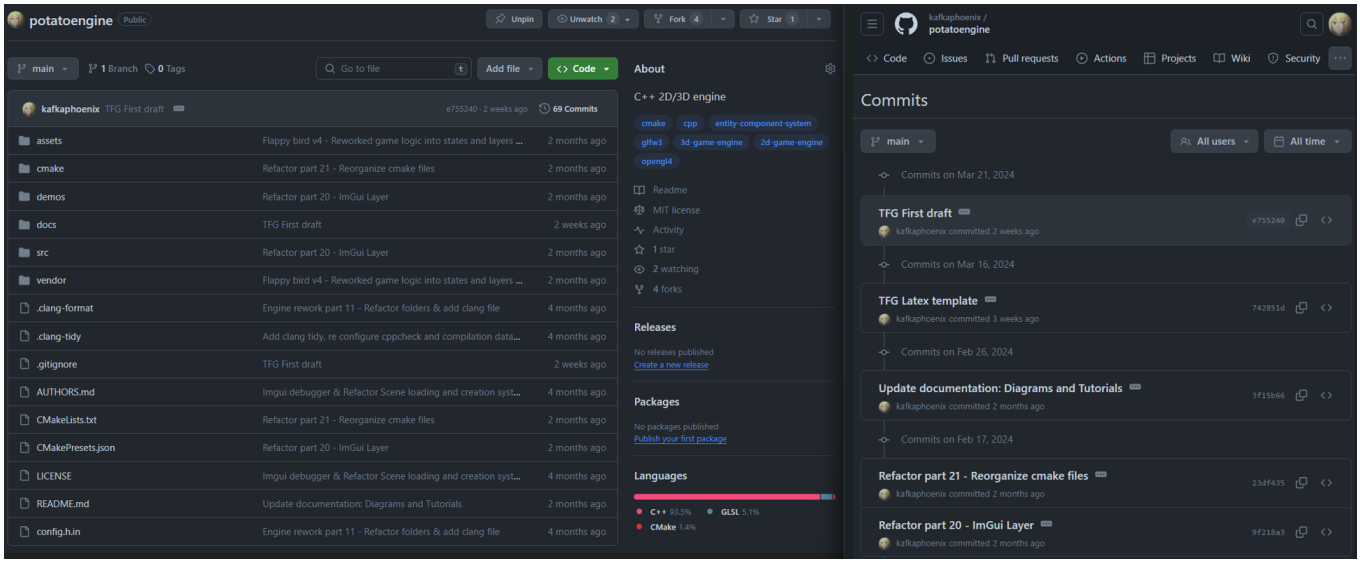


Figura 4: Control de versión con Github

1.3. Estructura de la memoria

La memoria se estructura en una serie de capítulos, en este primero se discute la motivación y objetivos del TFG, así como la metodología que se seguirá, el segundo capítulo tratará sobre el análisis, conclusiones de los trabajos relacionados y tecnologías empleadas / librerías que han ayudado al desarrollo del motor, permitiendo tener una idea clara de que se necesitará de cara a la fase de diseño del prototipo.

Los siguientes tres capítulos comprenderán el diseño, prototipado y validación del motor, explicando en detalle cada componente del motor, decisiones tomadas y comparación entre otras posibles soluciones.

Por último se cierra el documento con las conclusiones del proyecto, reflexiones y líneas futuras de mejoras posibles del motor de juegos.

El proyecto se complementa con una lista de acrónimos, bibliografía y anexos profundizando en partes teóricas.

2. Análisis del trabajo relacionado

Antes de poder hablar sobre el desarrollo de un motor de juegos, se debe entender qué es un motor y analizar qué ofrecen los principales motores que se vieron en la Figura 1.

2.1. ¿Qué es un motor de juegos?

Un motor de juegos se compone de diferentes piezas, pero se pueden ver las principales en la Figura 5.

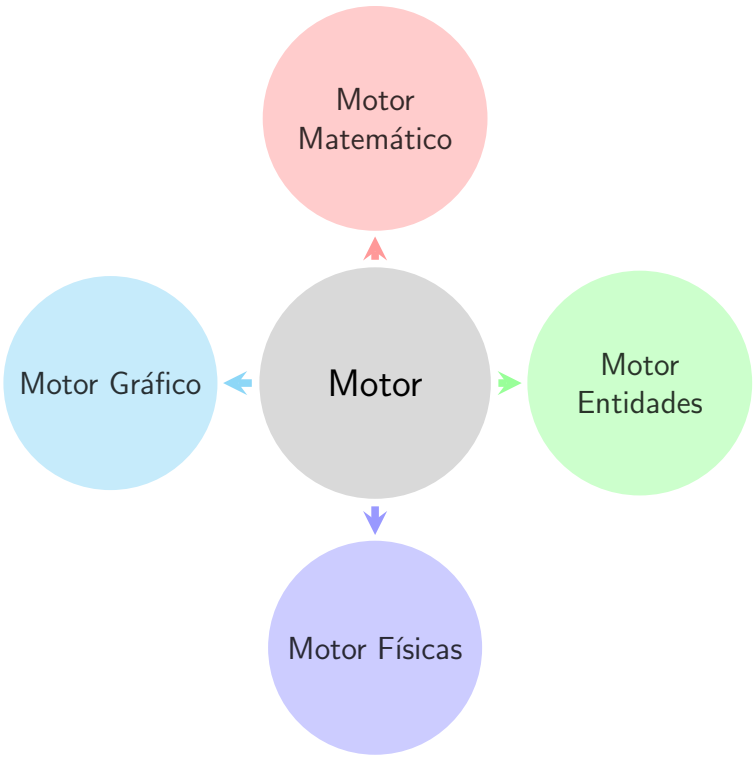


Figura 5: Componentes de un motor

Lo primero que se necesita para poder crear un juego es crear una ventana donde se pueda *visualizar la escena*, de esto se encargará el **motor gráfico**. Esta escena tendrá elementos que se llamarán *entidades* como puede ser el jugador, los árboles o los enemigos, de esto se encargará el **motor de entidades**. Seguidamente esas entidades *reaccionarán al entorno y entre ellas* a través del **motor de físicas**. Y por último estos motores necesitarán hacer uso del **motor matemático** para realizar las diferentes *operaciones algebraicas* que ello requiere.

2.2. Conclusiones del análisis y tecnologías empleadas

Se pueden observar en la Tabla 1 las conclusiones de analizar y comparar las tres opciones más populares de motores en la actualidad Unity[1], Unreal[2] y Game Maker[3] [2],

	Unity	Unreal	Game Maker
Multi-plataforma	Sí	Sí	Sí
Interfaz Interactiva	Sí	Sí	Sí
Documentación	Sí	Sí	Sí
Plantillas	Sí	Sí	Sí
Juegos 2D	Sí	Sí	Sí
Juegos 3D	Sí	Sí	No
Herramientas de Análisis	Sí	Sí	Sí

Tabla 1: Principales características de los motores analizados

Reuniendo toda la información previa, se tiene una idea más clara de lo que esperaría un usuario de un motor:

- Simple de usar
- Extensa Documentación
- Plantillas para una idea inicial del uso del motor
- Herramientas de análisis para mejorar el rendimiento, ver logs e inspeccionar elementos
- Opción de exportar a diferentes plataformas
- Opción de crear juegos tanto en 2D como en 3D

En consecuencia, este proyecto se ha desarrollado usando el lenguaje *C++*, estándar en el desarrollo de videojuegos por su eficiencia, junto a cmake[8] que permite la compilación multiplataforma del proyecto. Además las siguientes librerías se han usado para cubrir las necesidades del motor:

- GLFW[9] y GLAD[10]: Librerías para creación de ventanas y gráficos de alto rendimiento usando OpenGL[11]

[2] Fuente: *https://www.pubnub.com/blog/comparing-game-engines-unity-unreal-corona-gamemaker*

- EnTT[12]: Librería de entidades
- GLM[13]: Librería de matemáticas
- ImGui[14]: Librería para crear la interfaz y menús de la herramienta de análisis
- nlohmann_json[15]: Carga de plantillas de tipo JSON, para definir escenas y entidades
- stb[16]: Carga de imágenes
- Assimp[17]: Carga de modelos
- spdlog[18]: Logs de la aplicación

3. Diseño del Motor

Para las fases de diseño y prototipo se ha hecho uso de las dos herramientas mencionadas en la metodología

En el análisis hablamos sobre qué era un motor de juegos pasemos ahora a hablar de los diferentes componentes en detalle así como decisiones de diseño

4. Prototipado del motor

Para el prototipado TODO Hablar de los refactor y mejoras

```cpp
#include <iostream>

int main() {
  // comentario
  std::cout << "Hola mundo" << std::endl;
  return 0;
}
```

Código 1: Ejemplo

5. Validación del motor

TODO

6. Conclusiones y Líneas Futuras

A lo largo de los capítulos, se han explicado las bases para crear un motor de juegos desde cero, así como las decisiones de diseño tomadas. No sólo siendo una alternativa real y gratuita, sino también código abierto permitiendo así a los usuarios expandirlo según sus necesidades.

La solución planteada abstrae la complejidad de los diversos componentes de un motor de juegos, además de ofrecer un entorno completo acompañado de herramientas para analizar errores, logs y métricas.

El proyecto ha logrado por tanto los objetivos planteados, sirviendo su función de creación de videojuegos, pero también acompañado por diagramas y vídeos didácticos, así como ejemplos de caso de uso, permitiendo una entrada al mundo de creación de videojuegos en sólo unas pocas líneas de código.

La demostración del juego de plataformas en 2D sirve como una plantilla para saber dónde empezar y entender todas las funcionalidades que ofrece el motor.

Como líneas futuras para el proyecto, el motor podría ser expandido para incluir las siguientes mejoras:

- **Importar el proyecto como librería auto-instalable**: La forma de trabajar con el motor actualmente requiere descargarlo como parte del proyecto que los usuarios quieran hacer para construir el ejecutable, al exportar el motor como librería podría ser añadido automáticamente a la hora de generar el ejecutable del proyecto del usuario, reduciendo el número de ficheros aunque esto también limitaría la opción de modificar el motor si fuera necesario.

- **Más plantillas**: Se podría añadir otra plantilla demostrando un juego de tipo 3D. También podría expandirse la actual introduciendo Inteligencia Artificial (IA) de enemigos, más niveles o menú de opciones.

- **Editor**: Una nueva herramienta que permitiría a los usuarios crear el juego de forma visual e interactiva, en lugar de forma programática.

- **Mejorar herramienta de Análisis**: Una utilidad para consultar el rendimiento de la

aplicación (Profiler) permitiría a los usuarios encontrar los cuellos de botella en sus aplicaciones, facilitando la optimización de código y el uso eficiente de la memoria, la CPU y la GPU.

- **Lenguaje de Scripting**: En la actualidad, los principales motores comerciales permiten usar lenguajes de programación para scripting, siendo los más populares Lua y Python. Estos lenguajes de programación no requieren de compilación al ser interpretados permitiendo de una forma sencilla iterar sobre el prototipo, además suelen ser lenguajes más accesibles para los usuarios.
- **Serialización**: Actualmente no es posible guardar un juego, sería una mejora, ya que los usuarios podrían guardar el estado de la escena y sus entidades.

Acrónimos

2D two dimensions. III, IV

2D dos dimensiones. III, 3, 8, 17

3D three dimensions. III

3D tres dimensiones. III, 8, 17

BFP Bachelor's Final Project. III

ECS Entidad Componente Sistema. I–III, V

IA Inteligencia Artificial. 17

OO Orientada a Objetos. I–III

TFG Trabajo de Fin de Grado. III, 6

Bibliografía

[1] "Unity." [Online]. Available: https://unity.com

[2] "Unreal." [Online]. Available: https://www.unrealengine.com/en-US

[3] "Game Maker." [Online]. Available: https://gamemaker.io/en

[4] "Unity pricing." [Online]. Available: https://unity.com/pricing

[5] "Unreal pricing." [Online]. Available: https://www.unrealengine.com/en-US/license

[6] "Game Maker pricing." [Online]. Available: https://gamemaker.io/en/get

[7] "Unity polémica." [Online]. Available: https://es.wired.com/articulos/unity-pierde-confianza-entre-desarrolladores-pese-a-retractarse-de-polemicas-cuotas

[8] "cmake." [Online]. Available: https://cmake.org/

[9] "GLFW." [Online]. Available: https://github.com/glfw/glfw.git

[10] "GLAD." [Online]. Available: https://github.com/Dav1dde/glad.git

[11] "OpenGL." [Online]. Available: https://www.opengl.org/

[12] "EnTT." [Online]. Available: https://github.com/skypjack/entt.git

[13] "GLM." [Online]. Available: https://github.com/g-truc/glm.git

[14] "ImGui." [Online]. Available: https://github.com/ocornut/imgui.git

[15] "nlohmann_json." [Online]. Available: https://github.com/nlohmann/json.git

[16] "stb." [Online]. Available: https://github.com/nothings/stb.git

[17] "Assimp." [Online]. Available: https://github.com/assimp/assimp.git

[18] "spdlog." [Online]. Available: https://github.com/gabime/spdlog.git

[19] "Evolve your hierarchy." [Online]. Available: https://cowboyprogramming.com/2007/01/05/evolve-your-heirachy

[20] "ECS future of mmog." [Online]. Available: https://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1

[21] "Unity Dots ECS." [Online]. Available: https://unity.com/ecs

[22] "Unreal Mass ECS." [Online]. Available: https://docs.unrealengine.com/5.0/en-US/overview-of-mass-entity-in-unreal-engine

[23] "Data locality." [Online]. Available: https://gameprogrammingpatterns.com/data-locality.html

Anexo 1 - Arquitectura Entidad Componente Sistema

En los últimos años, la arquitectura de tipo Entidad Componente Sistema (ECS) es un paradigma de datos cada vez más popular dentro el sector de los videojuegos[19][20]. Los dos principales motores de juego comerciales, Unity[1] y Unreal[2], han añadido gradualmente soporte para esta arquitectura[21][22].

Arquitectura ECS vs OO

La arquitectura de tipo Orientada a Objetos (OO), es un paradigma que agrupa los datos en clases, las cuales contienen información, en forma de parámetros, y lógica, en forma de métodos, haciendo uso de herencia para clases relacionadas.

Como se observa en la Figura 6, las clases *Enemigo* y *Jugador* heredan de una clase padre *Entidad*, añadiendo posteriormente nuevos métodos y variables que puedan necesitar, pero la clase *Moneda*, pese a presentar unas variables y lógica parecida no puede heredar de la clase *Entidad* al no encajar en la estructura de datos. De la misma forma si se creará un nuevo enemigo de tipo Moneda, sería imposible heredar de ambos a la misma vez, teniéndose que duplicar aún más parámetros y lógica. Esta situación no hace si no empeorar a medida que se añaden más clases, creando una jerarquía cada vez más rígida y compleja, e incrementando la dificultad de la lógica y su mantenibilidad.

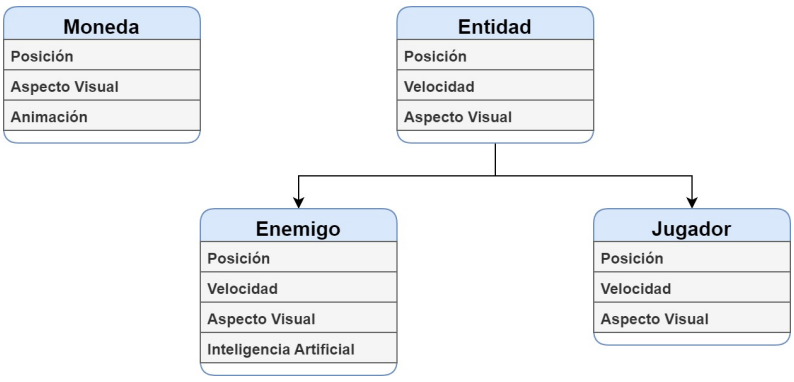


Figura 6: Arquitectura OO

La arquitectura ECS en contraste, favorece la composición en lugar de la herencia, fomentando la reusabilidad del código. Como se observa en la Figura 7 el concepto de clase se deja de lado, para hablar de tres términos:

- **Entidades**: Los actores, agrupando una serie de componentes.
- **Componentes**: Datos.
- **Sistemas**: Lógica que actúa sobre los componentes.

Comparando el mismo ejemplo de antes, se puede ver como en lugar de hablar en términos de clase *Jugador*, *Enemigo* o *Moneda*, se habla de entidades que tendrán uno u otro componente si es necesario.

Lo que define si una entidad usa un tipo de lógica o no es el tipo de componentes que se le añaden, presentando dos ventajas inmediatas, mayor flexibilidad a la hora de ser modificadas y toda la lógica relacionada con un tipo de datos se encuentra en el mismo lugar facilitando la refactorización.

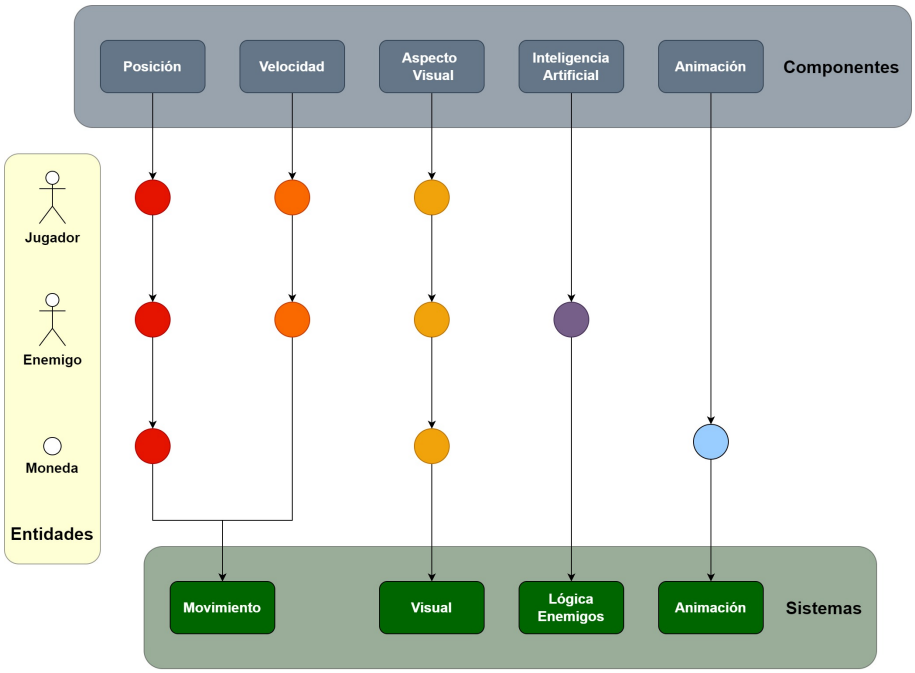


Figura 7: Arquitectura ECS

Por lo tanto se puede resumir las ventajas de la arquitectura ECS con respecto a la OO en los siguiente puntos:

- **Eficiencia en memoria**: En la lógica de un sistema al iterar sobre los componentes todos estarán juntos en memoria[23].
- **Reusabilidad**: Evita la duplicación de código.
- **Mantenibilidad**: Mayor mantenibilidad y más fácil refactorizar código al estar los datos y la lógica en un sitio.
- **Menor complejidad**: Al no hacer tanto uso de la herencia, se evitan escenarios complejos como el problema diamante en multi-herencia.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA