edix

TRABAJO FINAL DE GRADO

Jorge Sanchez-Alor, Antonio De Gea y Adrián Rodríguez

Grupo 4. Grado: Desarrollo de Aplicaciones Multiplataforma

Resumen

El desarrollo de videojuegos es una industria en constante evolución y con grandes posibilidades de negocio. Teniendo esto en consideración se presentó la posibilidad de aumentar los conocimientos en este ámbito desarrollando un videojuego. No obstante, desarrollar un videojuego presenta retos adicionales en cuanto a requisitos de interfaz de usuario y capacidades multimedia que otro tipo de aplicaciones no presentan (como puede ser una API REST). Para reducir el tiempo de desarrollo y centrarnos en las tareas de programación se optó por crear un juego clónico de Pokémon, reduciendo de esta forma los requisitos de diseño funcionales y gráficos. En el presente documento se mostrará el proceso de desarrollo además de algunos de sus puntos más importantes, junto con toda la información relevante que se solicita.

Keywords

desarrollo, videojuego, diseño, Pokémon, unity, Android, Windows, RPG, clon, C#

Agradecimientos

Se desea hacer una especial mención a Félix por hacer sus clases dinámicas, divertidas y especialmente prácticas. Por habernos enseñado a programar con diversas tecnologías utilizadas actualmente el mercado. Y, por último, por sus explicaciones, las cuales han sido en todo momento claras y concisas. Ha sido un placer cursar las asignaturas que impartías.

Tabla de contenido

1. Introducción

En los últimos años la industria de los videojuegos ha tenido un crecimiento importante, sobre todo en el ámbito de los móviles. Esta industria mueve cada año miles de millones y a día de hoy genera más beneficios que la industria cinematográfica o musical. Es por tanto un sector en plena expansión lleno de posibilidades de negocio.

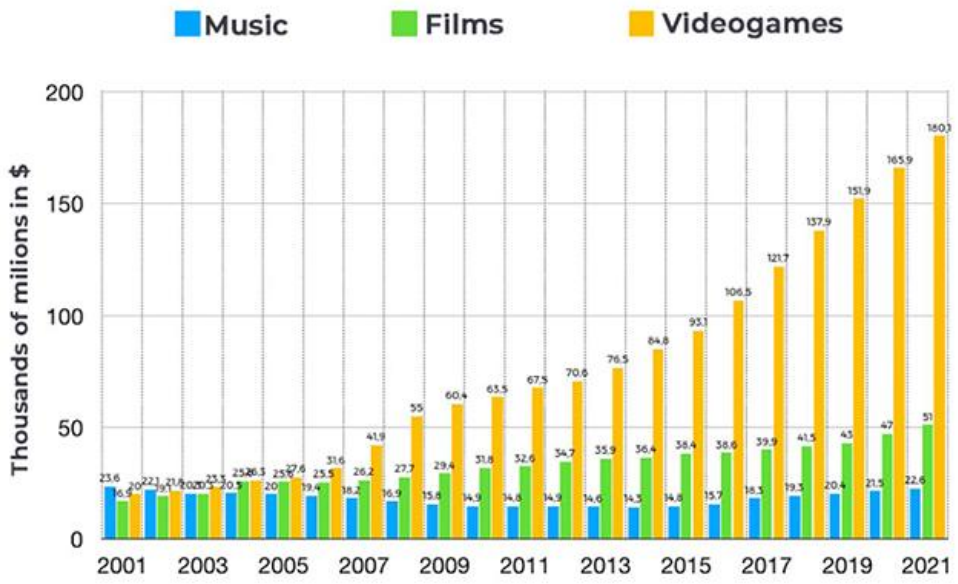

Figura 1. Beneficios anuales de las industrias del Videojuego, Cine y Música.

Por ello, invertir tiempo y esfuerzo en crear aplicaciones de entretenimiento es una actividad interesante de cara a desarrollar una carrera profesional. Sin embargo, el desarrollo de videojuegos presenta los mismos desafíos que cualquier otro tipo de aplicación a las que hay que añadir una especial importancia en el desarrollo de interfaces de usuario y el grafismo.

Teniendo esto en mente, y llegando al objetivo que nos ocupa, se planteó la idea de crear un videojuego como proyecto de fin de grado buscando de esta manera profundizar en las habilidades aprendidas en el módulo y obtener experiencia desarrollando este tipo de aplicaciones multimedia.

Debido a que el tiempo de desarrollo era bastante limitado, se optó por intentar emular un producto existente, en nuestro caso los antiguos juegos de Pokémon de Nintendo para la GameBoyAdvance. De esta forma, podíamos centrarnos en el ámbito de la programación y uso de Unity y reducir el tiempo de diseño en gran medida, ya que se podrían conseguir los sprites las músicas, los sonidos, etc. de diferentes fuentes de internet y que no habría que invertir demasiado tiempo en definir las mecánicas de

juego ya nos vendrían impuestas como requisitos debido a que como ya se ha indicado antes, el objetivo es emular el máximo número posible de éstas.

Así pues, hemos conseguido obtener un prototipo jugable del clon de Pokémon, al que hemos llamado “Poqimon – Learner Version”, en el conseguimos implementar muchas de las mecánicas del juego original, aunque no todas.

A pesar de que el videojuego es en esencia bastante corto, la realidad es que podría expandirse rápidamente ya que la parte más crítica, que es la implementación de las Batallas y el Sistema de Experiencia están implementada casi en su totalidad. Además el proyecto está construido de tal forma que permite al diseñador modificar los parámetros y crear nuevos poqimons y ataques con unos pocos clicks.

En este documento además del resumen y la introducción se presentarán a continuación los módulos formativos que se han empleado para el desarrollo del proyecto, las herramientas y lenguajes utilizados, y los integrantes del grupo y las contribuciones de cada uno. Después se mostrarán las Fases del Proyecto donde se expondrán con mayor detalle cada una de las fases del desarrollo. Seguidamente se expondrán las conclusiones y el margen de mejora, además de los objetivos conseguidos. Por último aparecerán la Bibliografía y los Anexos de los puntos que se han considerado más importantes o complejos.

2. Herramientas y lenguajes utilizados

2.1 Unity

Unity es un motor de videojuegos con todas las herramientas que se necesitan para poder crear uno.

2.2 Visual Studio Code

Editor principal para el desarrollo del proyecto, buena integración con Unity.

2.3 C#

Lenguaje de programación usado por Unity para el desarrollo y la lógica del proyecto. Es un lenguaje orientado a objetos y a su vez orientado a componentes, para poder usar y crear componentes de software.

2.4 Github

Un servicio web con el que se ha podido administrar y gestionar el proyecto gracias a su versión de control que lleva los cambios que se hacen en el código.

2.5 HTML/CSS (SCSS o SASS)

Lenguajes que tienen la funcionalidad de estructurar y aplicar diseños al *landing page*.

2.6 Bootsrap

Librería de CSS que establece una serie de estilos y contenedores útiles y fáciles de usar para la maquetación de la página.

2.7 Typescript (Javascript)

Lenguaje que extiende a JS como superconjunto de este, permitiendo el *tipado*. Utilizado para las funcionalidades del *landing page* (Controlador).

2.8 Angular

Framework de JS con el que se ha desarrollado el *landing page* del videojuego.

2.9 GIMP

Programa usado para el recorte preciso de Sprites para los objetos del proyecto.

2.10 Logoist

Software utilizado para el desarrollo de elementos de la GUI de batalla (y cuadros de diálogos)

3. Módulos formativos aplicados

3.1 Programación

Esta asignatura ha sido usada para todos los scripts de los que está compuesto el videojuego, más adelante se explicarán los lenguajes utilizados.

3.2 Entornos de Desarrollo

Esta asignatura ha sido muy útil para trabajar en equipo. Un buen uso de Github es fundamental para poder llevar un seguimiento del progreso del proyecto. Al ser varias personas en el mismo equipo se ha conseguido unificar el trabajo hecho por todos de una manera simple y eficiente.

3.3 Lenguaje de Marcas

Se ha usado los conceptos básicos de la asignatura para poder tener un Landing Page, es algo adicional que le da más profesionalidad al producto desarrollado.

3.4 Desarrollo de interfaces

Asignatura usada para darle funcionalidad con JavaScript al *Landin Page* creado.

3.5 Programación Multimedia y Dispositivos Móviles

El proyecto se trata de un videojuego por lo que se ha usado de forma extensa esta asignatura, todo lo que tiene que ver con la programación, los recursos, la implementación de archivos multimedia. El lenguaje y las herramientas usadas serán explicadas más adelante.

3.6 Acceso a Datos

Gracias a esta asignatura se ha implementado un sistema de guardado basado en ficheros.

3.7 Inglés

Se ha usado la asignatura de inglés para darle un público más amplio al proyecto, además es una buena práctica ya que los sectores de informática y sobre todo de programación suelen ser entornos donde el inglés está presente en el día a día.

3.8 Programación de servicios y procesos

Esta asignatura ha sido esencial para gestionar bien el código, se han usado eventos, funciones lambda, microservicios, una estructura de código modular para aprovechar el código todo lo posible.

4. Aportación de cada estudiante

4.1 Jorge

Jorge Sánchez-Alor se encargó principalmente de desarrollar toda la parte del control de los NPCs, los diálogos, los entrenadores, el sistema de experiencia y evolución, los objetos base de los Poqimons y sus Movimientos y la pantalla principal.

4.2 Adrián

Adrián Rodríguez Montesinos se ha encargado particularmente del controlador de la batalla, la lógica de ataques por turnos, la lógica de los estados de los poqimons, la captura de los poqimon; así como de la GUI de dicha batalla.

También ha desarrollado el *landing page* de la aplicación.

4.3 Antonio

Antonio de Gea se ha encargado del diseño del mundo y su creación, el control del héroe, las funciones de guardar y cargar, el médico, el inventario, y los objetos base.

Además de lo anterior todos los integrantes han contribuído al testing y corrección de errores de la aplicación.

5. Fases del proyecto

5.1 Mundo

El mundo está diseñado en tres fases: el primer pueblo, donde empieza el juego nada más crear la partida, el camino, aquí es donde están la mayoría de las mecánicas importantes del juego, y el segundo pueblo, donde se puede encontrar al médico.

Lo primero que se tuvo que hacer es buscar sprites para un juego de 2D. Se encontraron varios que daban muchos problemas al cortarlos así que se tuvo que recurrir a photoshop para poder utilizarlas. Una vez obtenidos los sprites deseados se crearon tres layers, una para el fondo, donde se colocó el suelo, los lagos, flores, etc. La siguiente para el césped, en esta layer se le dio interacción para cambiar de escena a la batalla. Y la tercera para los edificios y todos los objetos que necesiten un collider.

Figura 2. Mundo *poqimon*. Primer pueblo.

Figura 3. Mundo *poqimon*. Camino entre pueblos.

Figura 4. Mundo *poqimon*. Segundo pueblo.

5.2 Player

La primera fase del player era encontrar sprites, y hacer las animaciones de movimiento, una vez terminado eso se le fueron dando componentes para poder interactuar con el mundo creado.

El player es el encargado de tener los componentes del equipo Poqimon, de los objetos, del guardado, de interactuarcon los otros NPCs. Es la pieza central del proyecto en la escena fuera de las batallas.

Figura 5. Sprites del player.

5.3 Objetos

Los objetos son recolectables que el player puede coger y guardar en el menú, para ello se crearon tres prefabs con sus imágenes y atributos correspondientes y se les dio un componente llamado “Pickup” que al interactuar con ella enseña un cuadro de texto con el nombre del objeto que se acaba de encontrar. Una vez se ha guardado este objeto desaparece del suelo.

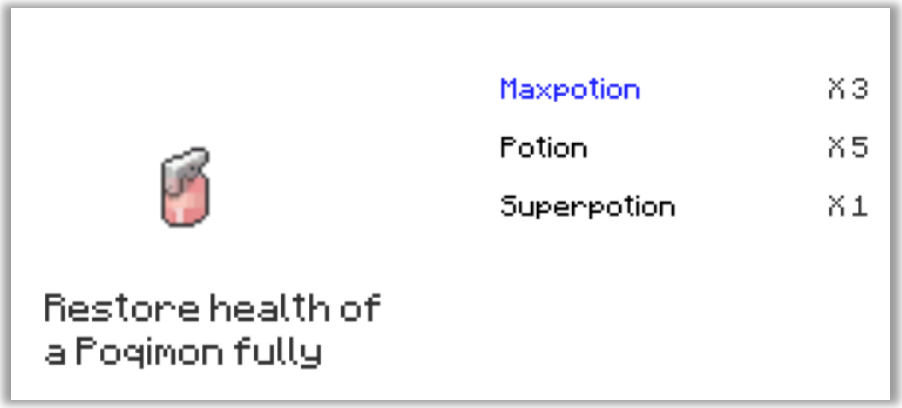

Figura 6. Submenú de objetos.

5.4 Poqimon y movimeintos

Los *poqimon* son la base del videojuego, ya que son las entidades que interaccionan en las batallas, restándose vida a través de sus movimientos. Los *poqimon* tienen una serie de cualidades que influyen en su capacidad para atacar y defenderse, así como una serie de movimientos dependientes de un tipo y unas condiciones.

Existen varios tipos de movimientos:

- movimientos físicos: si aciertan quitan vida, afectan a los atributos ataque y defensa.
- movimientos especiales: si aciertan quitan vida, afectan a los atributos ataque especial y defensa especial.
- movimientos de cambio de estado: no quitan vida. Si aciertan modifican la condición (estado del *poqimon*).

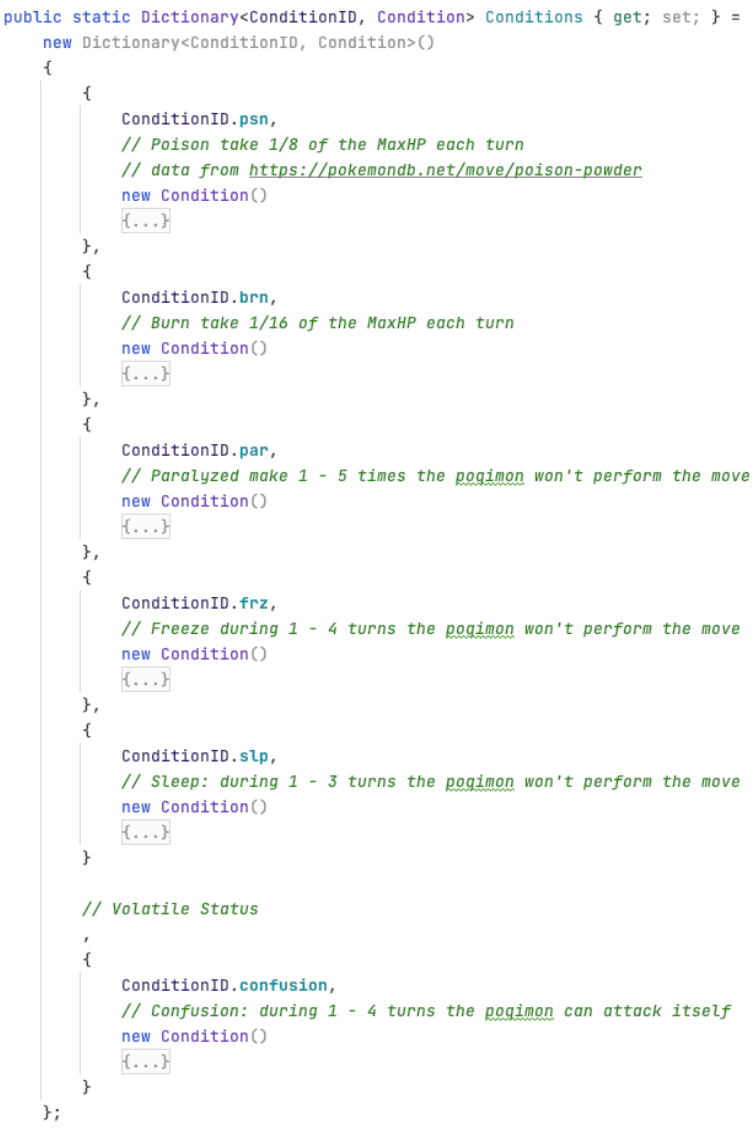

Figura 7. Código del diccionario de movimientos

Estos últimos tienen características individuales, por ello ha habido que crear una clase a modo de Base de Datos con los mismos, en ella, cabe destacar el diccionario de condiciones (de tipo condición) que alberga los datos de los mismos. [Figura 7]

Tipando las clases base de los *poqimon*, *PoqimonBaseObject* y los movimientos *MoveBaseObject*, como *ScriptableObject*; desde la UI de *Unity*, sin necesidad de codificar, podremos crear cuantos movimientos y *poqimon* queramos.

Se han creado un conjunto de varios movimientos que se han asignado a unos cuantos *poqimon*. Un movimiento se asigna a un nivel determinado de un *poqimon*, y si este es alcanzado, el *poqimon* puede aprenderlo. A continuación, se muestran los susodichos:

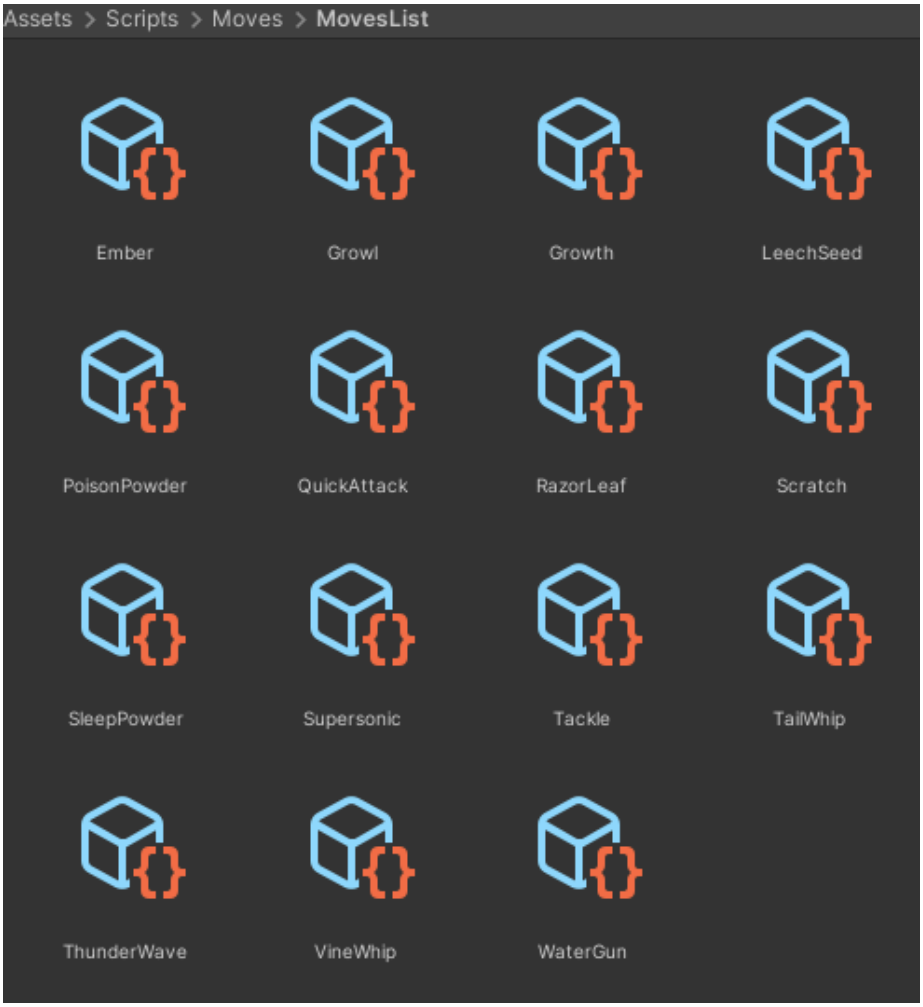

Figura 8. Movelist. Movimientos creados.

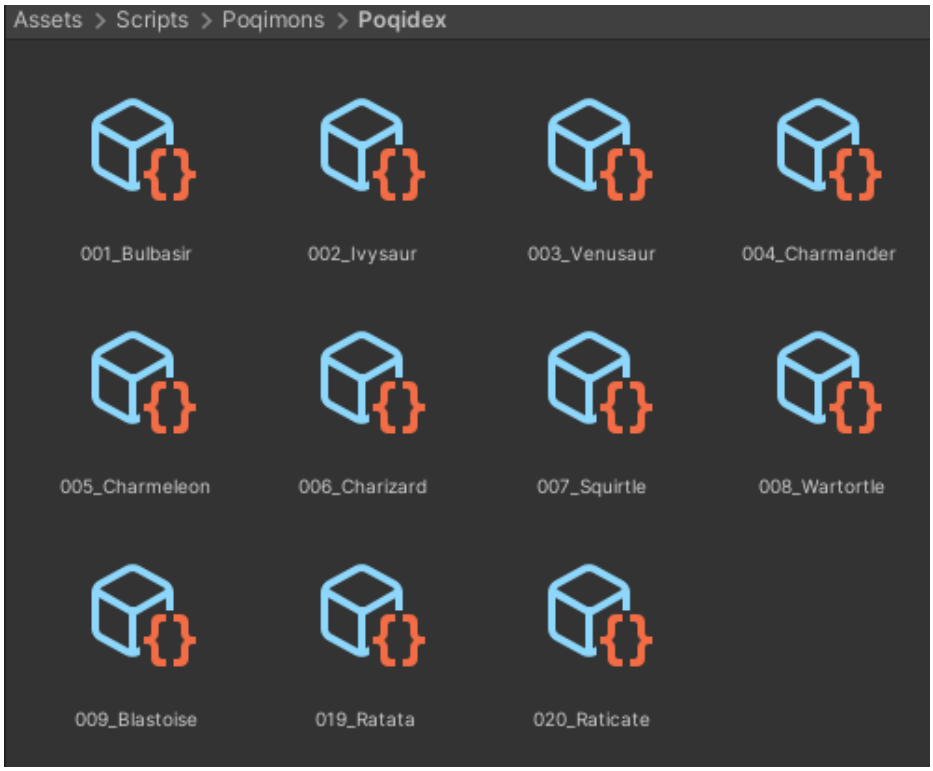

Figura 9. Poqidex. Poqimon creados.

5.5 Sistema de Experiencia

Se ha implementado un sistema de experiencia en el que los poqimons ganan cierta cantidad de experiencia al derrotar enemigos, que depende principalmente del nivel del oponente, la clase de poqimon oponente, y la velocidad de crecimiento del poqimon (fórmulas extraídas de Bulbapedia)

Cuando después de un ataque, o al final del turno del oponente debido a un efecto de estado, un oponente es derrotado, se calcula la cantidad de experiencia que recibirá el poqimon activo del jugador, y se sumará a su experiencia actual. Este efecto se representa mediante una barra que aumenta de tamaño en función de la proporción restante de experiencia para el siguiente nivel. En caso de que se alcance la cantidad de experiencia necesaria para el siguiente nivel, el poqimon, subirá un nivel.

5.5.1 Subida de nivel

Si ocurre una subida de nivel, se recalculan las nuevas estadísticas del poqimon y se actualizan.

5.5.2 Aprender nuevos ataques

Además, si en dicho nivel existiera algún ataque dentro de la lista LearnableMoves del poqimon, se rellenará la lista de ataques actuales Moves con el nuevo elemento. En el caso de que el poqimon ya conozca el número máximo de movimientos, que en nuestro caso es 4, se presentará al usuario un mensaje y un interfaz que le permitirá descartar uno de ellos, ya que los poqimons sólo pueden tener 4 movimientos disponibles en su lista de movimientos.

5.5.3 Evolución

Por último, si al final de la batalla, el nivel del poqimon es igual o superior al indicado en la propiedad Evolution, se disparará la pantalla de evoluciones de los poqimons. Este proceso se expondrá en el siguiente epígrafe

En dicha pantalla, se presentará al jugador un mensaje de que su poqimon está listo para evolucionar. Además, se reproducirá el tema musical de la evolución y la fanfarria de éxito mientras se ejecuta una animación de los sprites de los poqimons que contiene varios sistemas de partículas. Cuando termine, el objeto poqimon se actualizará para usar el nuevo objeto base que contiene los datos de la evolución y se recalcularán las estadísticas del poqimon.

Figura 10. Sistemas de partículas. GUI evolución de un *poqimon*.

5.6 Sistema de batalla

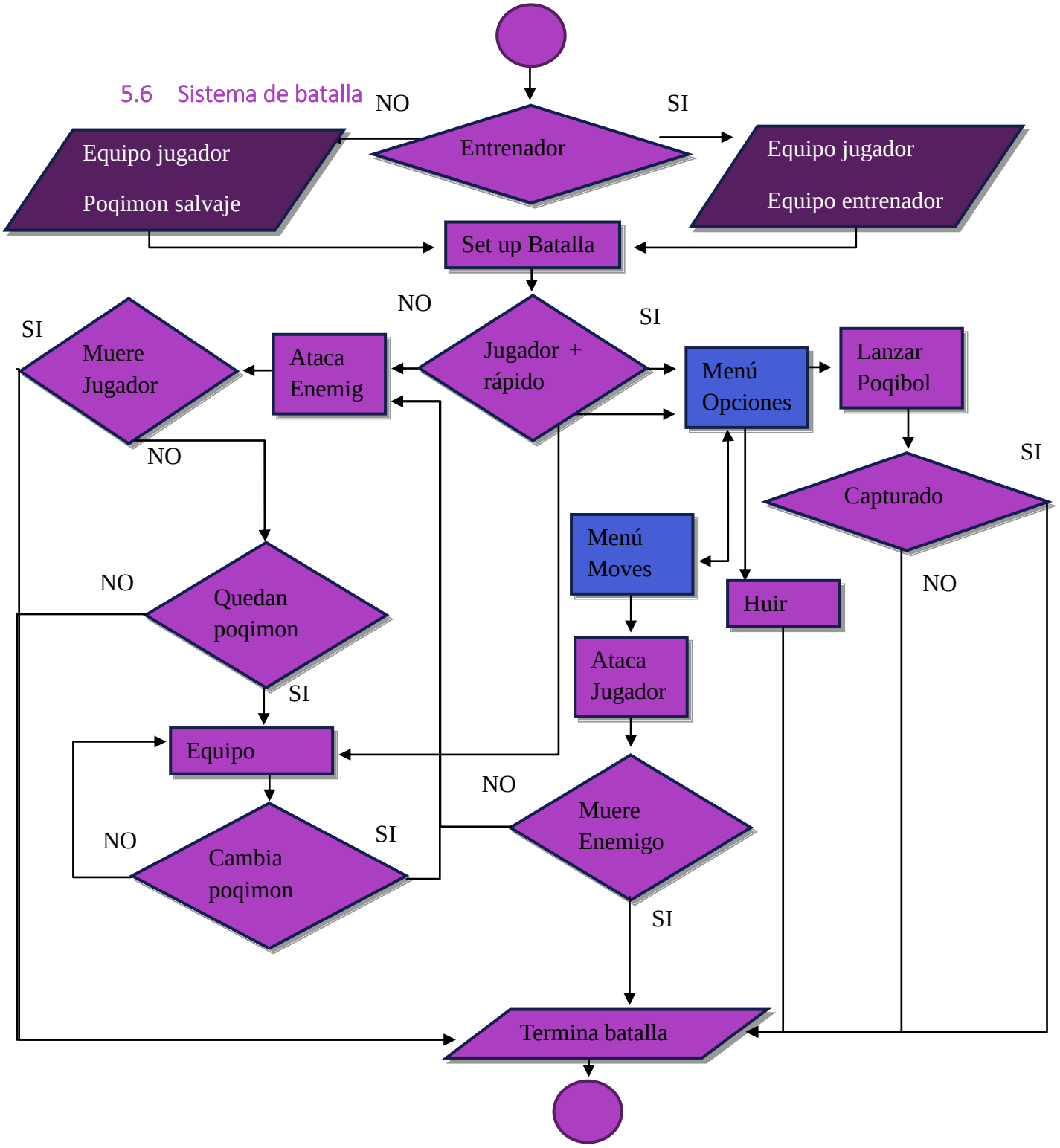

Figura 11. Cuadro opciones battala. Cuadro de diálogo mundo.

El controlador de la batalla es el script mas importante de nuestro proyecto. Alberga toda la lógica del sistema de batalla por turnos, tanto con entrenadores, como con *poqimon* salvajes, la captura de los *poqimon*, la huida, la experiencia, las animaciones durante la batalla, y la vuelta al juego o, si fuera preciso, dar paso a la evolución de un poqimon.

Se ha reflejado mediante un diagrama de flujo un resumen conciso de la lógica de este sistema, caracterizado por su gran modularidad, el intercambio de datos y un rígido sistema a prueba de bugs.

En primer lugar, desde el controlador del juego, tenemos dos opciones de batalla, la de entrenadores y la *poqimon* salvajes. Una vez seleccionada la que corresponde, se carga el *init* (*start*) ese tipo de batalla concreto. Ambos cargan sus datos necesarios, en el caso de entrenador, un conjunto de *poqimon*, y en el caso de batalla salvaje, uno solo; pero los dos hacen uso del mismo set up (con sus condiciones distintivas dentro de dicha función).

Este set up es una *corutina* de C# que inicializa y/o llama a todos lo componentes necesarios para la batalla. Tras ello, se llama a un método que elige el primer turno, dependiendo de la velocidad del *poqimon,* si el jugador es el más rápido entraremos a llamar al menú de opciones, en el que el jugador puede elegir entre seleccionar movimiento, huir, tirar *poqibol* o cambiar de *poqimon.* Si el enemigo es más rápido, se elegirá de forma aleatoria un ataque entre los que tenga (IA enemigo básica).

En el menú de movimientos del jugador, se selecciona el movimiento requerido y se ataca. Si, tras el ataque, el enemigo es debilitado, se gana la batalla; en caso contrario pasa a atacar el enemigo. Si el enemigo debilita al jugador y este no tiene más *poqimon*, se pierde la batalla, en caso contrario, se cambia de *poqimon* y se vuelve al menú de movimientos.

Cuando es un combate con un entrenador la lógica de dicho entrenador es calcada a la del jugador: si se debilita el *poqimon* actual y hay más, continua la batalla, sino pierde el entrenador.

Por último, encontramos la opción de huir, que sale de la batalla, y la opción de tirar *poqibol*, que en caso de capturar, o de no hacerlo (el *poqimon* salvaje escapa), también se sale de la batalla.

5.7 Interfaz gráfica de la batalla (GUI)

La interfaz de la batalla consta de diversos elementos. En primer lugar, un HUD para cada *poqimon*, con su barra de vida (texto de vida y barra de experiencia añadidos en el caso del jugador), el nivel del *poqimon* y su nombre. EN este HUD también aparecerán, durante el combate, en caso de haberlos, condiciones (cada una resaltada con un color).

Además del HUD hay un cuadro de diálogo que a su vez hace de fondo para el menú de ataques. Añadido a este cuadro de dialogo hay un menú de opciones, que a la vez, hace de fondo para la información de cada ataque seleccionado, en el cual, cada tipo de ataque se resalta con un color predeterminado.

Por debajo de todos estos elementos se encuentra el fondo de la batalla, y por encima del mismo, las imágenes de los *poqimon*: el del jugador y el enemigo.

Figura 12. GUI completa batalla

La mayoría de estos elementos se han diseñado a mano mediante diseño vectorial, puesto que no se habían encontrado recursos que cumplieran nuestros requisitos. Aquí se muestran los diseños:

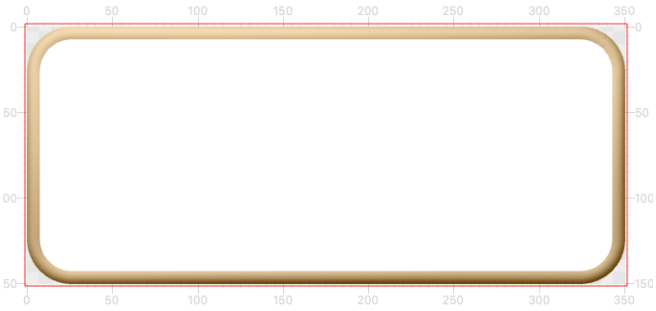

Figura 13. Cuadro opciones battala y Cuadro de diálogo de mundo

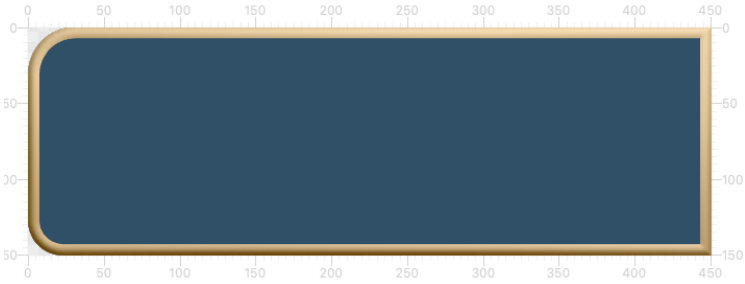

Figura 14. . Cuadro de diálogo batalla

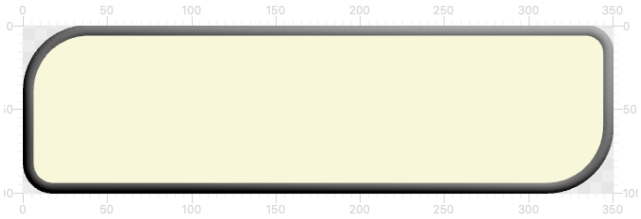

Figura 15. HUD del jugador y del rival

6. Conclusiones

Este videojuego difiere en la dinámica explicada en clase de programación multimedia, ya que no se trata de un juego de plataformas, sino de un RPG con ataque por turnos. Los tres miembros del equipo hemos aprendido mucho sobre la dinámica de desarrollo de este tipo de videojuegos realizando dicho proyecto. No obstante, se ha notado el cambio de dificultad frente a un juego de plataformas, no tanto en la creación de un mundo, sino en el sistema de ataque por turnos, los diferentes ataques con sus diferentes propiedades, las interacciones con otros *NPCs*, los diversos *pokémon*, así como el cambio o captura de estos.

6.1 Objetivos cumplidos y resultados obtenidos

En primer lugar, se ha cumplido que funcione a la perfección el sistema de batallas con las diferentes dinámicas: ataques normales, especiales y de cambios en la condición de los *poqimon*, cambio de *poqimon*, battlas salvajes y entrenadores debilitación de los *poqimon*, subir experiencia al terminar la batalla, evolucionando si se da el caso, y la captura (posible solo con *poqimon* salvajes) y escape de los *poqimon*.

Se ha generado un mundo condiversos NPCs, entre los cuales, cabe destacar los entrenadores y el curandero. Las colisiones, interacciones (batallas y curación) y diálogos emergentes.

Hemos creado una serie de *poqimon*, los cuales tienen una serie de movimientos generados, de diferentes tipos y con diversas particularidades. Consiguiendo que los *poqimon* aprendan los movimientos correspondientes cuando llegan a determinado nivel.

6.2 Mejoras del proyecto

El sistema de batalla actual no permite que se priorice un movimiento más rápido antes que otro, ya que el *player* ataca siempre, a menos que cambie de *poqimon,* y tras él, atacaría el enemigo. Así sucesivamente hasta que termina el combate porque uno de ellos se ha debilitado. Sería interesante plantear una lógica distinta que haga posible esta casuística.

En el sistema de captura, si un *poqimon* no es capturado, sería conveniente que no huya de la batalla, y que siga atacando. Para ello habría que cambiar la lógica del controlador de la batalla.

7. Glosario de términos

NPC

Del inglés (Non Player Character), se refiere a un personaje del videojuego que no está controlado por alguien que está jugando, lo controla el ordenador.

Catching Rate

Rango de números que afecta el resultado de una captura exitosa. Cuanto más alto es el rango de captura, más probabilidades hay de que salga bien.

Tipo de *poqimon*

Son atributos de los poqimons para crear relaciones entre ellos de ventajas y desventajas.

ScriptableObject

Es un contenedor que se puede usar para guardar gran cantidad de datos, independientemente de las instancias de la clase.

SerializableField

Un atributo que marca campos para que Unity pueda guardar y cargar los valores de estos campos.

Dictionary<T>

Un diccionario es una colección genérica que usada generalmente para guardar parejas de “clave/valor”.

8. Bibliografía

8.1 Programación

Documentación oficial lenguaje C# (general)

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/

Func<T> y Action<T> de C#

https://docs.microsoft.com/es-es/dotnet/api/system.func-1?view=net-6.0

https://docs.microsoft.com/es-es/dotnet/api/system.action-1?view=net-6.0

Documentación oficial de unity

https://docs.unity.com/

Masterclasses y apuntes de programación multimedia (unity).

Profesora: Raquel Cerdá. Edix.

Corutinas Unity

https://docs.unity3d.com/es/2018.4/Manual/Coroutines.html

Juego estilo *pokémon*

https://www.youtube.com/watch?v=FL_GLH6M9nM

Angular – landing page del videojuego

https://angular.io/

JavaScript y CSS – landing page del videojuego

https://www.w3schools.com/

8.2 Datos

Datos de los *pokémon*

https://pokemondb.net/pokedex/all

Tipos de *pokémon* (matriz de efectividad)

https://pokemondb.net/type

Ratio de captura de los *pokémon*

https://bulbapedia.bulbagarden.net/wiki/List_of_Pok%C3%A9mon_by_catch_rate

Algoritmo captura de los *pokémon*

https://bulbapedia.bulbagarden.net/wiki/Catch_rate

Estadísticas de los *pokémon*

https://pokemon.fandom.com/wiki/Statistics

Datos de los movimientos

https://pokemondb.net/move/generation/1

Experiencia de los *pokémon*

https://bulbapedia.bulbagarden.net/wiki/Experience

8.3 Recursos

Música del videojuego

https://downloads.khinsider.com/game-soundtracks/album/pokemon-gameboy-sound-collection

Sprites de los *pókemon*

https://veekun.com/dex/downloads

Sprites y spritesheets del mundo *pókemon* y los NPC

https://www.spriters-resource.com/game_boy_advance/pokemonfireredleafgreen/

Librería animaciones

http://dotween.demigiant.com/

Librería de estilos – landing page

https://getbootstrap.com/

Iconos – landing page

https://www.flaticon.es/

Imágenes – landing page

https://pixabay.com/es/

9. Anexos

9.1 ANEXO A – Fórmula del daño e implementación

Fórmula de daño:

$$Damage = \left(\frac{\left(\frac{2 \times Level}{5} + 2 \right) \times Power \times Att/Def}{50} + 2 \right) \times Modifiers$$

Donde:

$Level$: $nivel\ del\ poqimon\ atacante$
$Power$: $el\ poder\ del\ ataque\ empleado$
Att: $la\ estadística\ de\ ataque\ del\ poqimon\ atacante$
Def: $la\ estadística\ de\ defensa\ del\ poqimon\ defensor$
$Modifiers = Critical \times Random \times Type$

Donde:

$Critical$: $2\ si\ el\ poqimon\ realiza\ un\ golpe\ crítico, 1\ en\ caso\ de\ no\ hacerlo$
$Random$: $es\ una\ cantidad\ aleatoria\ entre\ 0.85\ y\ 1$
$Type$: $el\ multiplicador\ por\ tipo, que\ se\ extrae\ de\ la\ tabla\ de\ tipos.$
$Puede\ tomar\ los\ valores\ [0, 0.25, 0.5, 1, 2, 4]$

Dicha fórmula se implemeta en el siguiente método:

```
///<summary>
///Take Damage function (formula from Bulbapedia)
///</summary>
///<param name="move"> The attacking Poqimon move </param>
///<param name="attacker"> The attacking Poqimon object </param>
///<returns> true if Poqimon fainted because of the attack, false
otherwise </returns>
public DamageDetails TakeDamage(Move move, Poqimon attacker)
{
    //Critical Hit
    float critical = 1f;
    if (UnityEngine.Random.Range(0f, 1f) * 100f <= 6.25f)
    {
        critical = 2f;
    }

    //Move Effectiveness
    float type = TypeChart.GetEffectiveness(move.MoveBase.MoveType,
this.PoqimonBase.PoqimonType1) *
```

```
                TypeChart.GetEffectiveness(move.MoveBase.MoveType,
this.PoqimonBase.PoqimonType2);

    var damageDetails = new DamageDetails()
    {
        TypeEffectiveness = type,
        Critical = critical,
        Fainted = false
    };

    //Damage Value
    float d = 0;
    //Random Modifier (plus critical and aeffectiveness)
    float modifiers = UnityEngine.Random.Range(0.85f, 1f) * critical *
type;
    //Level dependency
    float a = (2 * attacker.PoqimonLevel + 10)/250f;
    //Damage Calculation
    float atk = (move.MoveBase.MoveCategory == CategoryType.Special) ?
attacker.SpAttack : attacker.Attack;
    float def = (move.MoveBase.MoveCategory == CategoryType.Special) ?
this.SpDefense : this.Defense;
    d = a * move.MoveBase.MovePower * ((float)atk/ def) + 2;
    int damage = Mathf.FloorToInt(d + modifiers);

    UpdateHP(damage);

    return damageDetails;
}
```

El método calcula si el ataque es crítico, en cuyo caso hace que critical tomo el valor 2.

Luego se calcula el multiplicador del tipo que se extrae del array de tipos.

Se crea el objeto que recibirá el BattleController para poder mostrar los mensajes correspondientes.

Se calcula el daño del ataque.

Se aplica el daño a la vida del poqimon que recibe el ataque.

Se devuelve el objeto damageDetails.