HARD TRIP

Memoria Técnica

INDICE

0.- HARD TRIP

0.1.- Videojuego de acción/aventura

1.- MEMORIA EXPLICATIVA

En esta memoria explicamos el funcionamiento en detalle de nuestro juego describiendo sus objetivos. Hablaremos de todos los elementos que tenemos y de las funciones de cada personaje.

2.- DESCRIPCIÓN GENERAL

La idea de Hard Trip surgió de varios fragmentos, a priori de autor desconocido, que nos fueron proporcionados en los primeros días de clase. En ellos se narraba una pequeña historia inconexa y sin contexto que tenía que servir para elaborar la trama, la idea principal de un videojuego a realizar conjuntamente entre alumnos de la titulación de Comunicación Audiovisual y alumnos de Ingeniería de Telecomunicaciones. Después de un proceso de *brain-storming* llevado acabo entre todos los miembros del grupo se configuró un primer esbozo de la idea y desarrollo de Hard Trip.

En una segunda sesión se desveló que los fragmentos seleccionados correspondían a la obra del autor portugués José Saramago "A viagem do elefante".

Objetivos:

- Configuración de una historia coherente y jugable.
- Implementar una adecuada jugabilidad.
- Cumplir con los requisitos técnicos exigidos por el cuerpo docente responsable de la coordinación y supervisión del proyecto.
- Fomento de la colaboración multidisciplinar entre distintas titulaciones, en principio muy diferentes.
- Por último, la entrega de un videojuego que contenga todos los requisitos propuestos en la guía docente.

3.- DISEÑO DE JUEGO

3.1.- Qué va a hacer el jugador

El jugador se pondrá en la piel de un caballero de la Guardia real encargada de la protección de la princesa Juana, hija del emperador Carlos V. Durante el camino el jugador tendrá que sortear diversos obstáculos, como por ejemplo bestias o bandidos, así como proteger al cortejo real que acompaña a la princesa Juana. También podrá encontrar pistas por el camino las cuales facilitarán la toma de decisiones y aumentarán las probabilidades de supervivencia.

3.2.- Donde lo va a hacer

La historia se desarrolla en el desfiladero de Brenner, que se sitúa en la actual frontera austro-italiana.

3.3.- Objetivos

Llegar al final del trayecto con el carruaje de la princesa a salvo, después de derrotar a los distintos enemigos que las caravanas se encuentren por el camino. Además deberá recolectar pistas gracias a las cuales aumentarán sus probabilidades de escoger el mejor camino para su supervivencia.

3.4.- Economía del Juego

Cada carreta tendrá una barra de vida que irá bajando con cada ataque. En caso de llegar a cero todas las caravanas de la comitiva real se acabará el juego.

El jugador, por su parte, también tendrá una vida para enfrentarse a los peligros. Si se llegase a cero, se perdería el progreso actual.

En el caso de que se pierda la carreta de la princesa también se acabaría el juego.

3.5.- Condiciones de Victoria

El principal objetivo es llegar al final del desfiladero con la carreta de la princesa a salvo.

DOCUMENTACIÓN TÉCNICA DEL PROYECTO

4.1.- Sinopsis

Tras salir de Bressanone con destino a Viena, la caravana de Juana de Austria, la hija del rey Carlos V, se dispone a atravesar el peligroso desfiladero de Brenner. Este desfiladero es conocido por los muchos peligros que alberga y que han provocado la muerte de muchas expediciones. Ayudándose de las pistas que el jugador encuentra por el camino, nuestros personajes tratarán de atravesar el desfiladero manteniendo a la princesa Juana de Austria a salvo dentro de su carruaje.

4.2.- Aspectos gráficos y estilísticos

Hard Trip es un videojuego en tres dimensiones y con perspectiva en tercera persona.

Contamos con banda sonora creada para el proyecto y assets de desarrolladores de la comunidad.

4.3.- Tratamiento

Hard Trip es un videojuego que mezcla la aventura con la acción, además de contar con combates y recolección de items por el mapa.

4.4.- Controles del Juego

Las teclas que nos permitirán controlar a los personajes o el juego serán las siguientes:

- Movimiento hacia delante: W
- Movimiento hacia atrás: S
- Movimiento hacia izquierda: Q
- Movimiento hacia derecha: E
- Movimiento de la cámara: A y D

- Cerrar y pasar diálogo: Barra espaciadora
- Atacar: Botón izquierdo del ratón
- Defender: Botón derecho del ratón
- Menú principal: Esc

4.5.- Personajes

Los personajes del juego son:

- *Guardaespaldas:*
  La mano derecha de Carlos V. Su misión es asegurar la protección de Juana. Es una persona versada en combate y astuta.
  - *Sus características y capacidades son las siguientes:*
    - *Nuestro avatar.*
    - *Puede desplazarse por el escenario.*
    - *Recoger pistas.*
    - *Interactuar con el entorno.*
    - *Atacar y defenderse de los posibles enemigos.*
    - *Consulta a la princesa el camino a seguir.*

- *Juana:*
  Es la hija del monarca Carlos V. Disfruta del poder que su posición le otorga y no duda en apelar a su rango cuando necesita imponerse a alguien. Es una mujer educada e inteligente, pese a que a veces toma malas decisiones cuando sus caprichos se sobreponen a su sentido común.
  - *Sus características y capacidades son las siguientes:*
    - *IA básica.*
    - *Intervendrá en las decisiones de los caminos.*
    - *Elegirá uno de ellos en función de las pistas que le mostremos.*

4.6.- Objetos

- Pistas. Hojas de un cuaderno que iremos recogiendo a medida que avancemos por el nivel. Nos darán información sobre lo que nos queda de camino.

4.7.- Interfaz

Para información del jugador contaremos con una serie de iconos que nos permitirán visualizar:

- El número de carretas funcionales que nos quedan.
- La vida restante de cada carreta.
- Un indicador de la vida del jugador.
- Indicador que confirma la elección del enemigo al que se va atacar.
- Número de pistas que hemos recogido durante esta fase.
- Multiplicador de daño y velocidad de ataque.

4.8.- Diagrama de navegación

4.9.- Estructura del proyecto (Walkthrough. Documento de Requisitos)

4.9.1 Introducción

Cuando iniciamos el juego nos sale un pequeño video introductorio donde una voz en off, encima de una serie de videos de paisajes, nos explica quienes son los personajes, hacia donde se dirigen y su porqué.

El texto de la voz en off es el siguiente:

"Llega a Bressanone la caravana de María y su marido Maximiliano, Archiduques de Austria, en la que transportan el elefante obsequiado.

Tras descansar dos semanas de su duro viaje por España e Italia, reemprenden el camino hacia Viena.

Su intención es atravesar el desfiladero de Brenner, del que se cuenta que todos los años se cobra unas cuantas víctimas.

Mientras un grupo acaba de preparar al elefante para el viaje que le queda por delante, otro grupo en el que se encuentra María se interna ya en el desfiladero"

4.9.2.- Nivel I : *Fase 1*

TÍTULO DE FASE - Camino a la oscuridad

CONCEPTO DE LA FASE

Descripción de la Fase

Nuestro protagonista, junto con las carrozas que llevan a la princesa, llegan al inicio del desfiladero de Brenner.

Desarrollo de la acción

Nuestro personaje tiene que escoger por que lado del camino ir para iniciar su andadura por el desfiladero, y prepararse para cualquier obstáculo que pueda encontrar durante su trayecto.

DESCRIPCIÓN DEL DECORADO

Hora y Climatología

Día, mediodía (12:00).

Localización

Desfiladero de Brenner

Recorrido por los dos caminos del desfiladero al nivel del suelo:

- Camino derecha: Paisaje natural: es una zona más verde, llena de árboles verdes frondosos.
- Camino izquierda: Paisaje natural: es una zona más tétrica, con árboles secos sin hojas y con troncos por el suelo con hojas secas alrededor.

PERSONAJES / OBJETIVOS

Jugador

Caballero

Punto de comienzo

Principio del escenario: en el centro

Vidas

- Vida jugador
- Vida de las 3 caravanas

Enemigos

Los enemigos que nos encontramos en esta fase pueden ser los siguientes:

Lobos. reaccionan agresivamente a la presencia del personaje principal.

- Vida: 100pts
- Reciben 35pts de daño
- Restan 5pts de vida al jugador
- Restan 10pts de vida a la carretas

Personas. Son bandidos, los cuales pueden actuar de forma abiertamente agresiva o negociar una retribución a cambio de no atacar.

- Los bandidos agresivos:
  - Vida: 100pts
  - Reciben 20pts de daño
  - Restan 20pts de daño al jugador
  - Restan 15pts de daño a las carretas

Objetivo Principal

Llegar al *checkpoint* al final del tramo con el mayor número de carretas vivas y derrotar a los enemigos.

Objetivos secundarios

Buscar las pistas que están por el camino que nos ayudarán en las posteriores fases.

Objetos

Pistas

Sonido

Sonido de lobo.

Música de fondo.

Sonido del escudo.

Sonido de espada.

PRUEBAS DE LA FASE / DESCRIPCIÓN DEL MAPA

Descripción de la mecánica del juego

La partida comienza con el personaje en el centro de la pantalla, se abre un cuadro de diálogo en el que habla con la princesa para escoger el camino (derecha o izquierda), y en función a esto, se enfrentará a un peligro u otro, aunque ambos desenvocan en el mismo punto.

Movimientos del personaje jugador

Desplazamiento lateral izquierda: Q

Desplazamiento lateral derecha: E

Desplazamiento adelante: W

Desplazamiento atrás: S

Armas

El personaje tiene una espada para atacar y un escudo para defenderse. La espada realiza el daño explicado en el apartado enemigos a cada uno de ellos. El escudo lo hace invulnerable durante un tiempo indefinido.

Secuencias animación / Rutinas de los personajes

Cuando ataca al personaje, se observa la animación de bajar la espada.

Si acaban con la vida de una de las caravanas, esta desaparece.

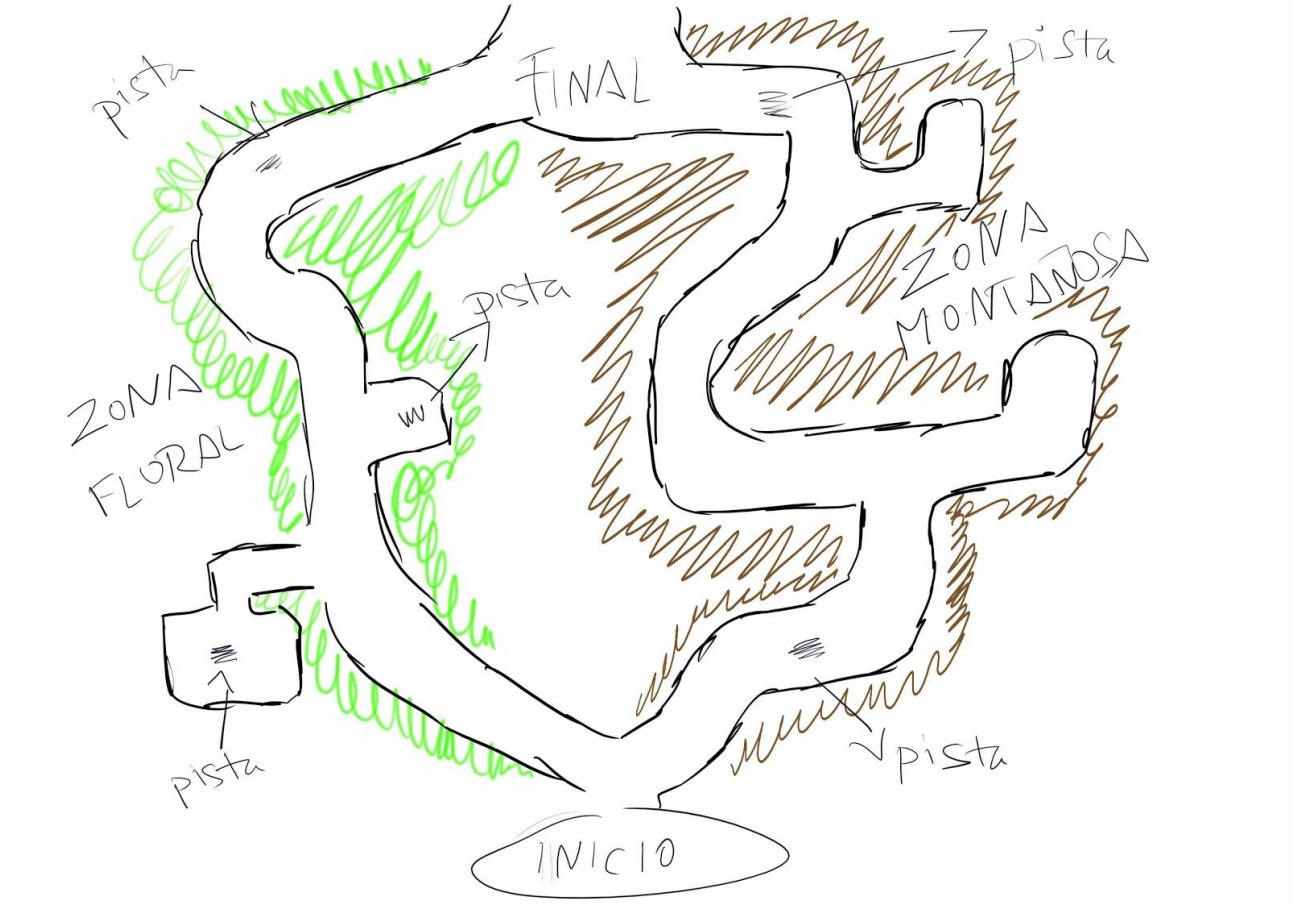
Si el personaje mata a un lobo, animación del lobo cayendo de lado al suelo y desapareciendo al pasar unos segundos.

Si el personaje mata a un bandido, animación del bandido cayendo al suelo y desapareciendo al pasar unos segundos.

Si matan al personaje, se acaba el juego.

MAPA DE LA FASE

** En el juego los caminos están cambiados de sentido, la zona floral está a la derecha y la zona montañosa a la izquierda.



WALKTHROUGH

Estamos en el comienzo del desfiladero de Brenner y nos encontramos con la primera encrucijada. Al ser el principio del juego no contamos con ninguna pista que nos ayude a escoger un camino y deberemos hacerlo por intuición. Los caminos que se nos presentan son dos, uno de ellos una zona floral con apariencia tranquila y agradable, y el otro una zona tétrica con árboles muertos y secos. Ambas zonas tendrán diferentes enemigos a los que habrá que hacer frente para poder avanzar. Además, se podrán encontrar pistas sobre las futuras encrucijadas, que funcionan como ayuda para la elección de los caminos que descubriremos a medida que progresemos en el juego.

- **Camino derecha:** Si el jugador decidió ir por la zona más verde, llena de árboles verdes frondosos,encontrará un camino que a primera vista parece tranquilo y bonito, pero en el que nos encontraremos bandidos y lobos a lo

largo del camino. La primera pista la encontramos en la primera desviación que tiene el camino hacia la derecha, en una pequeña zona que no tiene salida, y que está protegida por un grupo de enemigos, en ella pone: "*Quien con bestias anda, a aullar aprende*". Tras coger la pista, o no, dependiendo si te desviaste del camino, continuamos el viaje hasta llegar a la siguiente desviación que hay a la izquierda donde estará la segunda pista de este lugar, otra vez protegida por enemigos, en la que pone: "*Cuando el zorro escucha gritar a la liebre, siempre llega corriendo, pero no para ayudarla*". para llegar al final de la fase tendremos que seguir todo el sendero, ya que no habrá más desvíos.

- **Camino izquierda:** Si el jugador decide ir por la zona más tétrica, con árboles secos sin hojas y con troncos por el suelo con hojas secas alrededor, se encontrará con el mismo tipo de enemigos que en el otro camino, bandidos y lobos. La primera pista la encontraremos tras pasar el primer grupo de enemigos y al llegar a un cruce. Si decides ir por la izquierda, llegarás a una zona donde habrá un grupo de bandidos defendiendo una pista, en la que pone: "*Quien con bestias anda, a aullar aprende"*, si decides ir por la derecha continuarás el camino enfrentandote a otro grupo de enemigos. Más adelante encontrarás a la izquierda, en un pequeño rincón la última pista: "*Cuando el zorro escucha gritar a la liebre, siempre llega corriendo, pero no para ayudarla*" que también está protegida. Una vez cogida, o no, tienes que seguir avanzando un poco más hasta llegar al final.

DIÁLOGO

Cuando empieza el juego hay un pequeño diálogo entre nuestro personaje y la princesa. Un diálogo que puede estar en español, gallego o inglés dependiendo del idioma escogido para el juego.

Español:

- "Princesa, hemos llegado al desfiladero de Brenner. Dicen que en este lugar ya han perecido muchas otras expediciones. ¡Preparaos para lo peor mi señora!"

- "No tengo miedo alguno. Sé que vos me protegeréis. Muchos otros me han guardado, pero nunca tan bien como vos.Roguémosle al Señor y avancemos. ¿Qué camino prefiere vuestra merced?"

- "Creo que el mejor camino para seguir será el izquierdo. ¿Que opináis vos?"

Inglés:

- "Princess, we have arrived to the Brenner's Gorge. It is said that many others expeditions have perished in this place. Get ready for the worst!"

- "I'm not afraid. I know you will protect me. Many others have protected me, but never as well as you. Pray the Lord and go on. Which way do my lady prefer?"

- "I think the best path would be the one of the left-side"; What do you think?"

Galego:

- : Princesa, chegamos ao desfiladeiro de Brenner. Disque neste lugar pereceron moitas outras expecicións. Preparádevos para o peor miña señora!

- Non teño medo. Sei que vostede me protexerá. Moita outra xente me gardou pero ninguén o fixo tan ben coma vostede. Preguemoslle ao Señor e sigamos. Que camiño prefire a vosa mercé?

- Creo que o mellor camiño será ir cara a esquerda. Que opina vostede?

4.9.2 Nivel I : Fase 2

TÍTULO DE FASE - The First Election

CONCEPTO DE LA FASE

Descripción de la Fase

Nuestro protagonista ha superado el primer tramo del desfiladero, junto con las caravanas que llevan a la princesa. Llegan a otra bifurcación, la cual va a dar lugar a una zona nevada, en la cual debe volver a elegir que camino seguir.

Desarrollo de la acción

Nuestro personaje tiene que escoger por qué lado del camino ha de ir para continuar por el desfiladero, para ello le consulta a la princesa, y prepararse para cualquier obstáculo que pueda encontrar durante su trayecto.

DESCRIPCIÓN DEL DECORADO

Hora y Climatología

Media tarde (16:00).

Localización

Desfiladero de Brenner

Recorrido por los dos caminos del desfiladero al nivel del suelo:

- Camino derecha: Paisaje natural: es una zona nevada con árboles frondosos y plagada de bandidos.
- Camino izquierda: Paisaje natural: es una zona nevada, pero más inhóspita, en la que abundan manadas de lobos que pueden atacar a la comitiva.

PERSONAJES / OBJETIVOS

Jugador

Caballero

Punto de comienzo

Principio del escenario: en el centro

Vidas

- Vida jugador
- Vida de las caravanas

Enemigos

Los enemigos que nos encontramos en esta fase pueden ser los siguientes:

Lobos. reaccionan agresivamente a la presencia del personaje principal.

- Vida: 100pts
- Reciben 35pts de daño
- Restan 5pts de vida al jugador
- Restan 10pts de vida a la carretas

Personas. Son bandidos, los cuales pueden actuar de forma abiertamente agresiva o negociar una retribución a cambio de no atacar.

Los bandidos agresivos:

- Vida: 100pts
- Reciben 20pts de daño
- Restan 20pts de daño al jugador
- Restan 15pts de daño a las carretas

Objetivo Principal

Llegar al final del trayecto con la caravana de la princesa a salvo, después de derrotar a los enemigos que fueron apareciendo por el camino.

Objetivos secundarios

Buscar las pistas que están por el camino que nos ayudarán en las posteriores fases.

OBJECTOS / RECURSOS

Pistas

Sonido

Sonido de lobo.

Música de fondo.

Sonido del escudo.

Sonido de espada.

PRUEBAS DE LA FASE / DESCRIPCIÓN DEL MAPA

Descripción de la mecánica del juego

Jugador en el centro de la pantalla, se procede a escoger el siguiente camino para enfrentarnos a los peligros que este nos presente.

Movimientos del personaje jugador

Desplazamiento lateral izquierda: Q

Desplazamiento lateral derecha: E

Desplazamiento adelante: W

Desplazamiento atrás: S

Armas

El personaje tiene una espada para atacar y un escudo para defenderse. La espada realiza el daño explicado en el apartado enemigos a cada uno de ellos.

Secuencias animación / Rutinas de los personajes

Cuando atacan al personaje o este ataca, se observa la animación de bajar la espada.

Si acaban con la vida de una de las caravanas, esta desaparece.

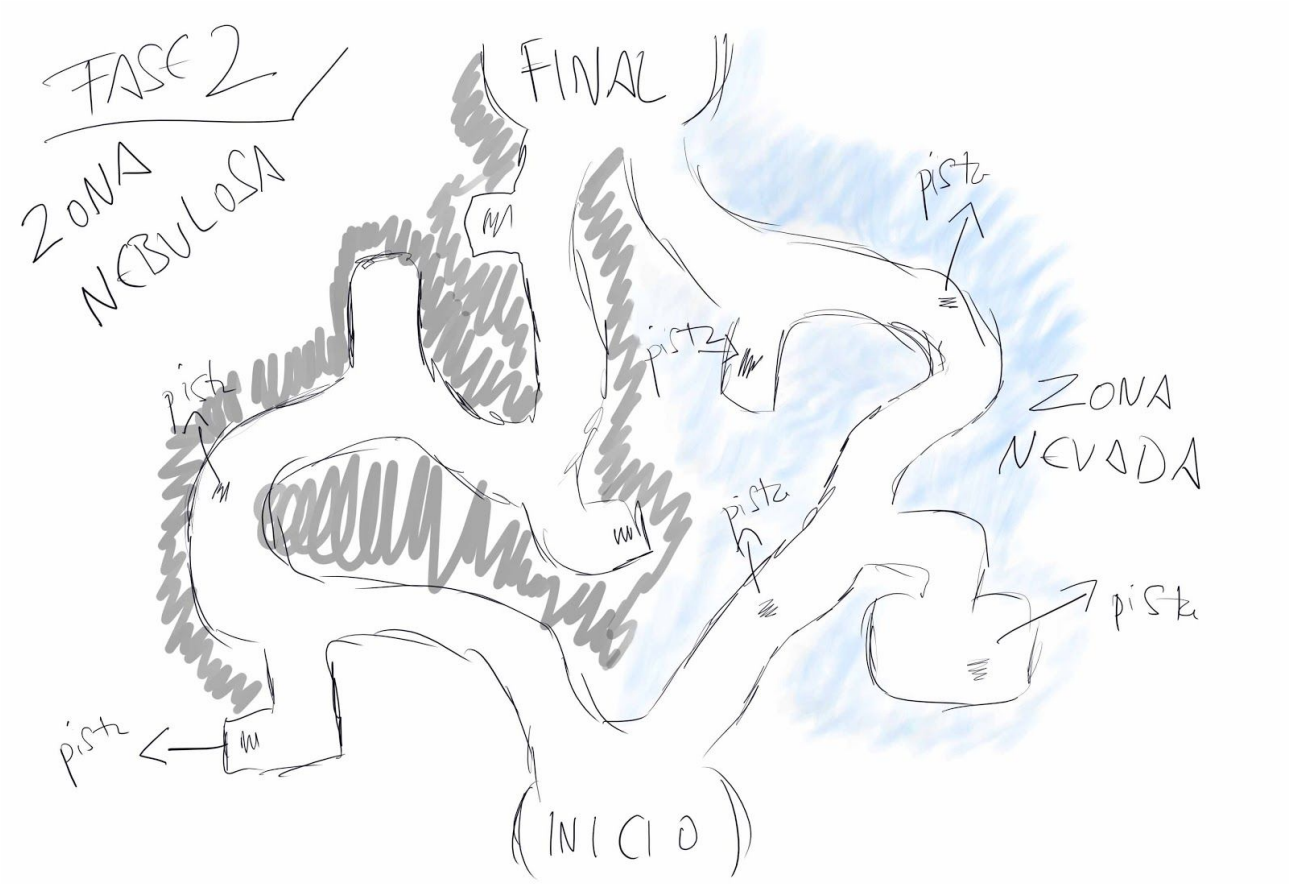
Si el personaje mata a un lobo, animación del lobo cayendo de lado al suelo y desapareciendo al pasar unos segundos.

Si el personaje mata a un bandido, animación del bandido cayendo al suelo y desapareciendo al pasar unos segundos.

Si matan al personaje, se acaba el juego.

MAPA DE LA FASE

**En el juego los caminos están cambiados de sentido, la zona nebulosa está a la derecha y la nevada a la izquierda.



WALKTHROUGH

Nos encontramos en la segunda fase del primer nivel,en la segunda encrucijada. Dependiendo de las pistas que recogimos en la anterior fase, la princesa nos mandará por un lado o por otro.

- **Camino derecha:** Si la princesa decide ir por la derecha, nos encontraremos una zona nevada con árboles frondosos y plagada de bandidos. Cerca del principio aparece una densa niebla que reduce la visión, por lo que solo veremos los elementos que estén cerca del jugador. El primer grupo de bandidos está casi al principio del trayecto, una vez deshechos de ellos, podemos encontrar la primera pista si vamos a la derecha en el primer desvío, allí estará la pista "*Si hurtas y das, te librarás*", como siempre protegida por un grupo de bandidos. Continuaremos nuestro camino, sin la niebla, y encontraremos otros grupos de enemigos con el que tendremos que luchar. Mas adelante habrá otras desviaciones del camino principal, el primero a la izquierda y el segundo a la derecha, donde encontraremos la segunda pista de esta fase, "*De la noche en la espesura, hasta la nieve es oscura*". Posteriormente llegaremos al final.
- **Camino izquierda:** Si la princesa decide ir por la zona nevada de la izquierda, nos encontraremos un lugar más inhóspito, en el que abundan los grupos de lobos que pueden atacar a las caravanas. En el primer giro a la izquierda nos encontraremos con la primera pista, "*Si hurtas y das, te librarás*", la cual estará protegida por un grupo de lobos. Este es un trayecto más recto por lo que una vez que hayamos pasado la curva de nuestro recorrido encontraremos otro

desvío, cerca del final, donde estará la última pista de esta fase, "*De la noche en la espesura, hasta la nieve es oscura*" protegida por los enemigos de la zona. Una vez cogida, llegaremos al final.

Diálogos

Cuando llegamos al final de la primera fase, empieza la segunda con la princesa escogiendo el siguiente camino.

Dependiendo del camino que escoja dira una opción o otra.

Español:

- "Creo que la mejor opción será elegir el camino de la izquierda"
- "Creo que la mejor opción será elegir el camino de la derecha"

Inglés:

- "I think the best path would be the one of the left-side"
- "I think the best path would be the one of the right-side"

Galego:

- "Creo que a mellor opción será elixir o camiño da esqueda"
- "Creo que a mellor opción será elixir o camiño da dereita"

4.9.2 Nivel I *: Fase 3.*

TÍTULO DE FASE - The Second Election

CONCEPTO DE LA FASE

Descripción de la Fase

Nuestro protagonista ha superado el segundo. Ahora salen del tramo nevado y se internan en un tramo baldío.

Desarrollo de la acción

Nuestro personaje tiene que escoger por qué lado del camino ha de ir para continuar por el desfiladero, para ello le consulta a la princesa, y prepararse para cualquier obstáculo que puedan encontrarse en este último tramo.

DESCRIPCIÓN DEL DECORADO

Hora y Climatología

Media tarde (18:00).

Localización

Desfiladero de Brenner

Recorrido por los dos caminos del desfiladero al nivel del suelo:

- Camino derecha: Paisaje natural: es una zona un poco nevada caracterizada por su sensación de frialdad, llena de árboles frondosos con lobos en los alrededores.
- Camino izquierda: Paisaje natural: es una zona desértica, caracterizado por el abundante número de árboles secos, llenos de bandidos.

PERSONAJES / OBJETIVOS

Jugador

Caballero

Punto de comienzo

Principio del escenario: en el centro

Vidas

- Vida jugador
- Vida de las caravanas

Enemigos

Los enemigos que nos encontramos en esta fase pueden ser los siguientes:

Lobos. reaccionan agresivamente a la presencia del personaje principal.

- Vida: 100pts
- Reciben 35pts de daño
- Restan 5pts de vida al jugador
- Restan 10pts de vida a la carretas

Personas. Son bandidos, los cuales pueden actuar de forma abiertamente agresiva o negociar una retribución a cambio de no atacar.

Los bandidos agresivos:

- Vida: 100pts
- Reciben 20pts de daño
- Restan 20pts de daño al jugador
- Restan 15pts de daño a las carretas

Objetivo Principal

Llegar al final del trayecto con la caravana de la princesa a salvo, después de derrotar a los enemigos que fueron apareciendo por el camino.

Objetivos secundarios

Buscar las pistas que están por el camino que nos ayudarán en las posteriores fases.

OBJECTOS / RECURSOS

Pistas

Sonido

Sonido de lobo.

Música de fondo.

Sonido del escudo.

Sonido de espada.

PRUEBAS DE LA FASE / DESCRIPCIÓN DEL MAPA

Descripción de la mecánica del juego

Jugador en el centro de la pantalla, se procede a escoger el siguiente camino para enfrentarnos a los peligros que este nos presente.

Movimientos del personaje jugador

Desplazamiento lateral izquierda: Q

Desplazamiento lateral derecha: E

Desplazamiento adelante: W

Desplazamiento atrás: S

Armas

El personaje tiene una espada para atacar y un escudo para defenderse. La espada realiza el daño explicado en el apartado enemigos a cada uno de ellos.

Secuencias animación / Rutinas de los personajes

Cuando atacan al personaje o este ataca, se observa la animación de bajar la espada.

Si acaban con la vida de una de las caravanas, esta desaparece.

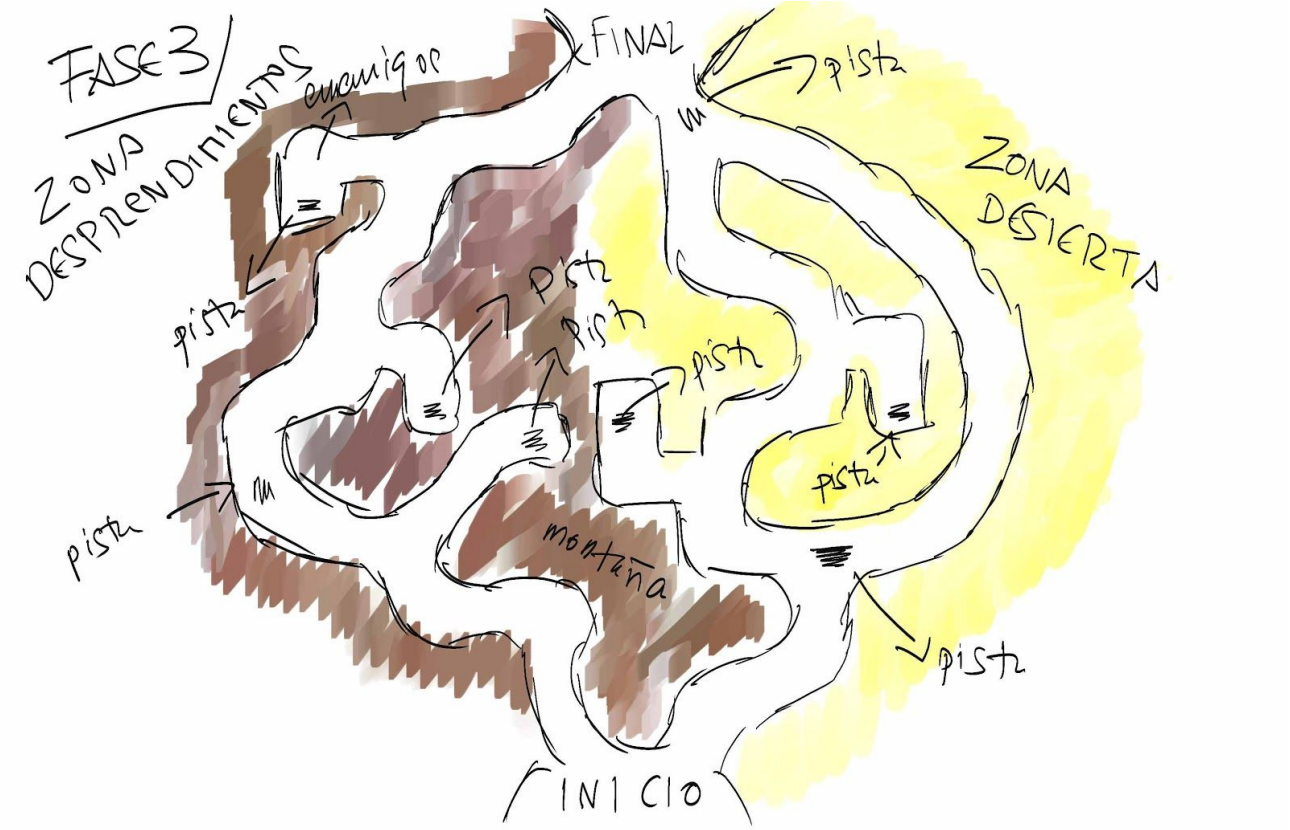
Si el personaje mata a un lobo, animación del lobo cayendo de lado al suelo y desapareciendo al pasar unos segundos.

Si el personaje mata a un bandido, animación del bandido cayendo al suelo y desapareciendo al pasar unos segundos.

Si matan al personaje, se acaba el juego.

MAPA DE LA FASE

**En el juego los caminos están cambiados de sentido, la zona nebulosa está a la derecha y la nevada a la izquierda.



WALKTHROUGH

Nos encontramos en la última fase del primer nivel, en la tercera encrucijada, que dependiendo de las pistas que recogimos en la anterior fase, la princesa nos escogerá un lado u otro.

- **Camino derecha**: Si la princesa decide ir por la zona más fría, llena de árboles frondosos,nos encontraremos con un camino más largo que los anteriores y con enemigos más difíciles de derrotar, en este caso solo habrá lobos. En la primera desviación a la izquierda estará la primera pista, "*Cuidate de la niebla, comandante. Podría ocultar lo que ni los propios dioses osan mirar*" defendida por un grupo de lobos. Si seguimos avanzando nos encontraremos con más enemigos que intentarán evitar que lleguemos al final. Por el camino habrá un giro a la izquierda que no nos llevará a ningún lado y otro a la derecha más adelante en el que estará la última pista, "*La blancura de la nieve hace al cisne negro*" donde también habrá lobos, si avanzamos un poco más llegaremos al final del primer nivel.

- **Camino izquierda:** Si la princesa decide ir por la zona desértica, nos encontraremos un primer grupo de bandidos antes de llegar a una bifurcación. en la que tendremos que escoger si ir por la izquierda o por la derecha. Si escogemos la derecha, el camino será todo recto, sin desviaciones, llena de enemigos y sin tener la posibilidad de coger pistas. Si por el contrario escogemos el camino de la izquierda podremos encontrar las dos pistas de la fase en cada desviación del camino. La primera pista será "*Cuidate de la niebla, comandante. Podría ocultar lo que ni los propios dioses osan mirar*" y la

segunda,"*La blancura de la nieve hace al cisne negro*", todas protegidas por un grupo de bandidos. Una vez pasado los dos desvíos que tiene ese camino, avanzamos hasta llegar al final del nivel 1, donde se nos guardará la partida.

DIÁLOGOS

Cuando llegamos al final de la segunda fase , empieza la tercera con la princesa volviendo a escoger el siguiente camino.

Dependiendo del camino que escoja dira una opción o otra:

Español:

- "Creo que la mejor opción será elegir el camino de la izquierda"
- "Creo que la mejor opción será elegir el camino de la derecha"

Inglés:

- "I think the best path would be the one of the left-side"
- "I think the best path would be the one of the right-side"

Galego:

- "Creo que a mellor opción será elixir o camiño da esqueda"
- "Creo que a mellor opción será elixir o camiño da dereita"

MENSAJES FINALES

- Cuando se acaba la fase, se acaba el primer nivel y aparece:

Español:

"¡Enhorabuena! Has superado el nivel. Puedes seguir explorando el mapa si quieres. ¿Volver al menú?"

Inglés:

"Congrats! You have exceeded the level. You can keep exploring the map if you like. Back to the main menu?"

Galego:

"Parabéns! Superaches o nivel. Podes seguir explorando o mapa se queres. ¿Volver ao menú?"

- Si se escoge la opción de no volver al menú, y exploras el mapa , aparecerá un jefe final que tendrás que derrotar. Una vez le ganes, se muestra el siguiente mensaje:

Español:

"Enhorabuena! Has superado el verdadero desafío.Ya no tenemos nada más que ofrecerte. ¿Volver al menú?"

Inglés:

"Congrats! You have exceeded the true challenge. We do not have any more to offer. Back to the main menu?"

Galego:

"Parabéns! Superaches o verdadeiro desafio. Non temos nada máis que che ofrecer. Volver ao menú?"

- Si pierdes, surge una escena donde te pone: "Game Over" y te da la opción de volver al menú para luego volver a cargar la partida y volver a enfrentarte al jefe.

4.9.2 Nivel II : Fase 1

TÍTULO DE FASE - MI QUERIDA TEMPESTAD

CONCEPTO DE LA FASE

Descripción de la Fase

Nuestro protagonista ha superado el primer tramo del desfiladero, junto con las caravanas que llevan a la princesa, llegan a otra bifurcación. Esta anocheciendo y empieza una fuerte tormenta, lo que trae consigo lluvias y fuertes rachas de viento.

Desarrollo de la acción

Nuestro personaje tiene que escoger por qué lado del camino ha de ir para continuar por el desfiladero, y prepararse para cualquier obstáculo que pueda encontrar durante su trayecto.

DESCRIPCIÓN DEL DECORADO

Hora y Climatología

Anochecer, media tarde (20:00).

Localización

Desfiladero de Brenner

Recorrido por los dos caminos del desfiladero al nivel del suelo:

- Camino derecha: Paisaje natural: es una zona boscosa llena de charcas, que enlentecen la comitiva real.
- Camino izquierda: Paisaje natural: es una zona más inhóspita, en el que existen pequeños remolinos y fuertes rachas de viento que ralentizan y afectan a la estabilidad de la caravana.

PERSONAJES / OBJETIVOS

Jugador

Caballero

Punto de comienzo

Principio del escenario: en el centro

Vidas

- Vida jugador
- Vida de las caravanas

Enemigos

Los enemigos que nos encontramos en esta fase pueden ser los siguientes:

Lobos. reaccionan agresivamente a la presencia del personaje principal.

- Vida: 100pts
- Reciben 35pts de daño
- Restan 5pts de vida al jugador
- Restan 10pts de vida a la carretas

Personas. Son bandidos, los cuales pueden actuar de forma abiertamente agresiva o negociar una retribución a cambio de no atacar.

Los bandidos agresivos:

- Vida: 100pts
- Reciben 20pts de daño
- Restan 20pts de daño al jugador
- Restan 15pts de daño a las carretas

Los bandidos negociadores, si fallan las negociaciones atacan primero al jugador.

- Vida: 100pts
- Reciben 20pts de daño
- Restan 25

Indígenas. Habitantes de las aldeas vecinas que se encuentran en el desfiladero en busca de alimentos:

- Vida:150pts
- Reciben: 15 de daño.
- Restan: 30pts.

Objetivo Principal

Llegar al final del trayecto con la caravana de la princesa a salvo, después de derrotar a los enemigos que fueron apareciendo por el camino.

Objetivos secundarios

Buscar las pistas que están por el camino que nos ayudarán en las posteriores fases.

OBJECTOS / RECURSOS

Pistas

Sonido

Sonido de aviso, cuando nos atacan a la caravana.

Sonido de lobo.

Voces en otro idioma – aviso de que nos advierten de la presencia de los indígenas.

Sonido de osos.

Música de fondo

Sonido de espada

Sonido de escudo

PRUEBAS DE LA FASE / DESCRIPCIÓN DEL MAPA

Descripción de la mecánica del juego

Jugador en el centro de la pantalla, se procede a escoger el siguiente camino para enfrentarnos a los peligros que este nos presente.

Movimientos del personaje jugador

Desplazamiento lateral izquierda: A-W

Desplazamiento lateral derecha: D-W

Desplazamiento adelante: W

Desplazamiento atrás: S

Armas

El personaje tiene una espada para atacar y un escudo para defenderse. La espada realiza el daño explicado en el apartado enemigos a cada uno de ellos.

Secuencias animación / Rutinas de los personajes

Cuando atacan al personaje o este ataca, se observa la animación de bajar la espada.

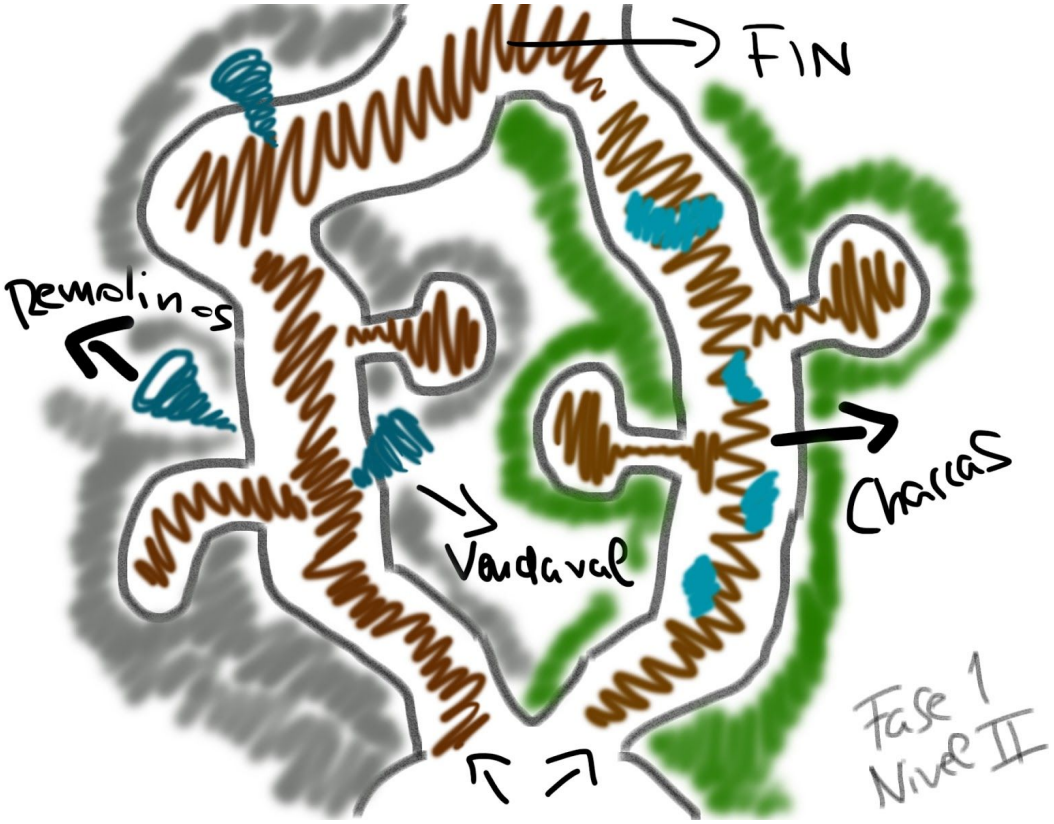
Si acaban con la vida de una de las caravanas, esta desaparece.

Si el personaje mata a un lobo, animación del lobo cayendo de lado al suelo y desapareciendo al pasar unos segundos.

Si el personaje mata a un bandido, animación del bandido cayendo al suelo y desapareciendo al pasar unos segundos.

Si matan al personaje, se acaba el juego.

MAPA DE LA FASE



WALKTHROUGH

Nos encontramos en la primera fase del segundo nivel del juego. La princesa escogerá el camino por donde iremos según las pistas recogidas en el anterior nivel:

- **Camino derecha:** Si vamos por el camino de la derecha nos encontraremos una zona boscosa llena de charcas, que enlentecen la comitiva real. Los primeros enemigos que nos encontraremos son los bandidos negociadores, donde tendremos que darle una parte de vida de nuestra caravana para que nos dejen avanzar, si rechazamos su propuesta nos atacarán violentamente. Una vez superados podremos desviarnos a la izquierda para encontrar la primera pista, *"No combatas la tormenta. Solo atraviésala"* protegida por indígenas. Una vez que proseguimos el camino encontraremos un grupo de lobos impidiendo que nos desviemos a la derecha para coger la última pista de la fase que pone:*"Algunas veces, para poder ver la luz, hay que arriesgar la oscuridad."*. Una vez que cogemos la pista o no, seguimos el camino para encontrarnos al último grupo de enemigos, que están justo antes del final, que son los indígenas. Una vez que nos deshacemos de ellos llegamos al final de la primera fase.

- **Camino izquierda:** El camino de la izquierda es una zona inhóspita, en el que existen pequeños remolinos y fuertes rachas de viento que ralentizan y afectan a la estabilidad de la caravana. En este trayecto tendremos que tener cuidado con los desprendimientos de roca que se producen. El primer enemigo al que nos tendremos que enfrentar es a los lobos, una vez vencidos podremos desviarnos a la izquierda donde se encuentra la primera pista:*"No combatas la tormenta. Solo atraviésala"*, pero antes de llegar a ella tendremos que vencer a un grupo de indígenas que la están protegiendo. Si continuamos el camino podremos girar a la derecha para coger la última pista, "*Algunas veces, para poder ver la luz, hay que arriesgar la oscuridad.*", pero antes tendremos que vencer a un grupo de bandidos que están delante de la pista. Después de coger la pista, o no, continuamos el trayecto donde nos encontraremos con un grupo de bandidos negociadores, donde tendremos que decidir si negociar con ellos o no. Antes de llegar al final tendremos que vencer a un grupo de indígenas.

DIÁLOGOS

Cuando llegamos al final del primer nivel, empieza el segundo con la princesa volviendo a escoger el siguiente camino.

Dependiendo del camino que escoja dira una opción o otra:

Español:

- "Creo que la mejor opción será elegir el camino de la izquierda"
- "Creo que la mejor opción será elegir el camino de la derecha"

Inglés:

- "I think the best path would be the one of the left-side"
- "I think the best path would be the one of the right-side"

Galego:

- "Creo que a mellor opción será elixir o camiño da esqueda"
- "Creo que a mellor opción será elixir o camiño da dereita"

Nivel II : Fase 2

TÍTULO DE FASE - ARRIEROS SOMOS, Y LA MUERTE ENCONTRAREMOS

CONCEPTO DE LA FASE

Descripción de la Fase

Nuestro protagonista ha superado el primer tramo del desfiladero, junto con las caravanas que llevan a la princesa, llegan a otra bifurcación. Esta anocheciendo y empieza una fuerte tormenta, lo que trae consigo lluvias y fuertes rachas de viento.

Desarrollo de la acción

Nuestro personaje tiene que escoger por qué lado del camino ha de ir para continuar por el desfiladero, y prepararse para cualquier obstáculo que pueda encontrar durante su trayecto.

DESCRIPCIÓN DEL DECORADO

Hora y Climatología

Anochecer, media tarde (20:00).

Localización

Desfiladero de Brenner

Recorrido por los dos caminos del desfiladero al nivel del suelo:

- Camino derecha: Poblado abandonado: es una zona llena de casas semidestruidas, con muchos cadáveres a su alrededor.
- Camino izquierda: Paisaje natural: es una zona nevada llena de fragmentos de rocas, en las que se producen frecuentes aludes.

PERSONAJES / OBJETIVOS

Jugador

Caballero

Punto de comienzo

Principio del escenario: en el centro

Vidas

- Vida jugador
- Vida de las caravanas

Enemigos

Los enemigos que nos encontramos en esta fase pueden ser los siguientes:

Lobos. Reaccionan agresivamente a la presencia del personaje principal.

- Vida: 100pts
  - Reciben 35pts de daño
  - Restan 5pts de vida al jugador
  - Restan 10pts de vida a la carreta

Personas. Son bandidos, los cuales pueden actuar de forma abiertamente agresiva o negociar una retribución a cambio de no atacar.

Los bandidos agresivos:

- Vida: 100pts
- Reciben 20pts de daño
- Restan 20pts de daño al jugador
- Restan 15pts de daño a las carretas

Los bandidos negociadores, si fallan las negociaciones atacan primero al jugador.

- Vida: 100pts
- Reciben 20pts de daño
- Restan 25

Indígenas. Habitantes de las aldeas vecinas que se encuentran en el desfiladero en busca de alimentos:

- Vida:150pts
- Reciben: 15 de daño.
- Restan: 30pts.

Naturaleza. Se producen desprendimientos y aludes.

- Restan: 20pts

Osos. Reaccionan violentamente al paso de las caravanas.

- Vida: 250pts

- Reciben: 20pts
- Restan: 30pts

Objetivo Principal

Llegar al final del trayecto con la caravana de la princesa a salvo, después de derrotar a los enemigos que fueron apareciendo por el camino.

Objetivos secundarios

Buscar las pistas que están por el camino que nos ayudarán en las posteriores fases.

OBJECTOS / RECURSOS

Pistas

Sonido

Sonido de aviso, cuando nos atacan a la caravana.

Sonido de lobo.

Voces en otro idioma – aviso de que nos advierten de la presencia de los indígenas.

Sonido de osos.

Música de fondo

Sonido de espada

Sonido de escudo

PRUEBAS DE LA FASE / DESCRIPCIÓN DEL MAPA

Descripción de la mecánica del juego

Jugador en el centro de la pantalla, se procede a escoger el siguiente camino para enfrentarnos a los peligros que este nos presente.

Movimientos del personaje jugador

Desplazamiento lateral izquierda: A-W

Desplazamiento lateral derecha: D-W

Desplazamiento adelante: W

Desplazamiento atrás: S

Armas

El personaje tiene una espada para atacar y un escudo para defenderse. La espada realiza el daño explicado en el apartado enemigos a cada uno de ellos. El escudo lo hace invulnerable durante un tiempo indefinido.

Secuencias animación / Rutinas de los personajes

Cuando ataca al personaje, se observa la animación de bajar la espada.

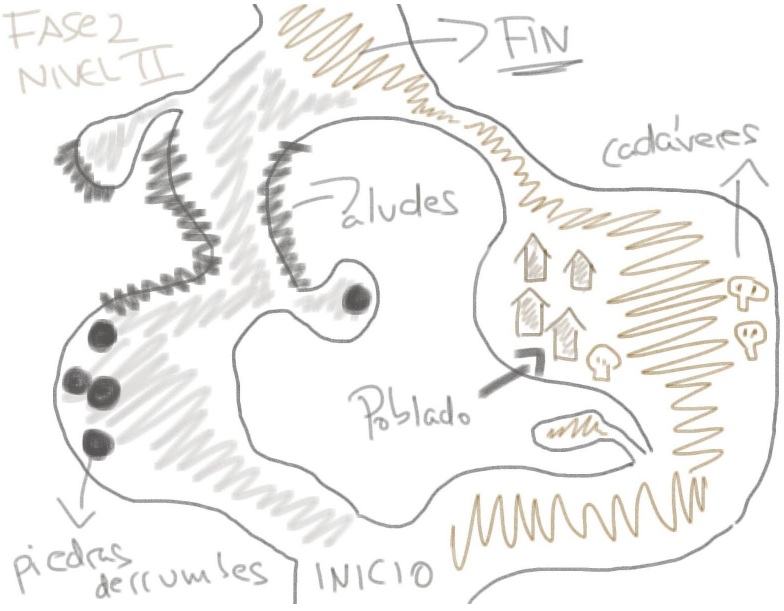
Si acaban con la vida de una de las caravanas, esta desaparece.

Si el personaje mata a un lobo, animación del lobo cayendo de lado al suelo y desapareciendo al pasar unos segundos.

Si el personaje mata a un bandido, animación del bandido cayendo al suelo y desapareciendo al pasar unos segundos.

Si matan al personaje, se acaba el juego.

MAPA DE LA FASE



WALKTHROUGH

Nos encontramos en la última fase del juego, donde la princesa nos dirá el camino por donde tendremos que ir:

- **Camino derecha:** Este camino se caracteriza por ser una zona llena de casas semidestruidas, con muchos cadáveres a su alrededor. Al principio del trayecto nos encontraremos con un grupo de indígenas que intentarán evitar que lleguemos a un poblado abandonado, que está un poco más adelante. Tras derrotarlos y avanzar, a la izquierda hay un pequeño desvío sin salida donde habrá la primera pista, *"Así como vuestra madre os acompañó en vuestro nacimiento, yo estaré ahí el día de vuestra muerte"*, situada al lado de un grupo

de osos dormidos que se despertarán para atacarte en el momento que te aproximes a la pista. Una vez vuelta al camino principal, llegaremos al poblado abandonado. Aquí tendremos que estar atentos porque se producen desprendimientos de piedras que pueden quitarnos vida, además de dos grupos de indígenas que nos atacarán. Al lado de la última casa abandonada que hay en el poblado se encuentra la última pista, "*De frente un precipicio, detrás un lobo*". Una vez cogida, salimos del poblado y antes del llegar al final nos encontraremos con un grupo de osos y indígenas, más fuertes que a los que nos enfrentamos anteriormente, que impedirán a toda costa que lleguemos al final.

- **Camino izquierda:** Es una zona nevada llena de fragmentos de rocas, en las que se producen frecuentes aludes durante todo el trayecto, lo que tendremos que estar alerta todo el tiempo. Al principio del camino, nos encontramos con el primer grupo de enemigos que son lobos, una vez derrotados seguimos avanzando y llegamos a una parte donde el camino es más ancho, en esta parte nos encontraremos con osos y lobos. A la izquierda del camino, donde hay un montón de piedras, estará la primera pista.*"Así como vuestra madre os acompañó en vuestro nacimiento, yo estaré ahí el día de vuestra muerte*"**.** Cuando salgamos de ese camino podemos desviarnos a la derecha, pero solo es un callejón sin salido donde habrá un grupo de osos esperando para atacarnos. Si proseguimos el camino veremos un grupo de lobos tapando la desviación de la izquierda, donde se encuentra la última pista, "*De frente un precipicio, detrás un lobo*". Antes de llegar al final nos tendremos que enfrentar a un grupo mixto de lobos y osos, más fuertes que los anteriores que intentarán evitar que lo consigamos, una vez derrotados llegaríamos al final del juego.

DIÁLOGOS

Cuando llegamos al final de la primera fase del nivel dos, empieza la última fase del juego con la princesa volviendo a escoger el siguiente camino.

Dependiendo del camino que escoja dira una opción o otra:

Español:

- "Creo que la mejor opción será elegir el camino de la izquierda"
- "Creo que la mejor opción será elegir el camino de la derecha"

Inglés:

- "I think the best path would be the one of the left-side"
- "I think the best path would be the one of the right-side"

Galego:

- "Creo que a mellor opción será elixir o camiño da esqueda"
- "Creo que a mellor opción será elixir o camiño da dereita"

Cuando se acaba la fase llegamos al final del juego, donde se nos mostrará una pequeña cinematica en la que aparecerá nuestro jugador y la princesa:

- Princesa, hemos conseguido llegar al final del desfiladero de Brenner. Ha sido un camino duro pero lo hemos logrado.

- No dudaba de vos, ha sido muy valiente durante este largo viaje. Me encargaré personalmente sea recompensado cuando lleguemos a nuestro destino.

- Agradecido mi señora. Su gratitud es mi mayor regalo.

FIN DEL JUEGO

MENSAJES FINALES

Después de la cinemática final se muestran los créditos. Posteriormente volveriamos al menú principal.

4.10.- Lenguajes de programación

El videojuego está programado en C#.

4.11.- Licencias de los productos finales

Los assets y la música son libres de copyright. En el caso de la música del menú principal y el tema principal son obras originales de Myriam Cortizo Pyres que nos ha cedido los derechos sobre ellas para este proyecto.

Principales fuentes para la música libres de copyright:

- https://tabletopaudio.com/
- https://freesound.org/

5.- PROGRAMACIÓN

OBJETOS USADOS

ESCENA 1: MENU

En esta primera escena se nos muestra el menú principal de nuestro juego y se nos da la opción de comenzar un juego nuevo, continuar uno ya empezado, salir del juego o cambiar el idioma de los textos.

Objetos: La siguiente lista enumera los distintos objetos presentes en la escena en el orden en el que se encuentran organizados. Cabe destacar que, en este caso, ninguno de ellos es generado mediante código.

- **Main camera**: Cámara principal a través de la cual el jugador puede observar la escena.
- **Directional Light**: Única iluminación de la escena.
- **Canvas**
    - Panel: Elemento en el que se presentan los distintos botones del menú
        - Background: Imagen que se utiliza como fondo de la escena.
        - ButtonNewGame: Botón que nos lleva a la opción de juego nuevo. La función que se ejecuta al pulsar este botón se encuentra en el script del objeto MenuController.
        - ButtonExit: Botón que nos lleva a la opción de salir del juego. La función que se ejecuta al pulsar este botón se encuentra en el script del objeto MenuController.
        - ButtonLoadGame: Botón que nos lleva a la opción de cargar una partida ya guardada. La función que se ejecuta al pulsar este botón se encuentra en el script del objeto MenuController.
        - ButtonLanguage. Botón que nos permite cambiar de idioma los textos del juego. La función que se ejecuta al pulsar este botón se encuentra en el script del objeto MenuController.
- **EventSystem**: Objeto que se crea conjuntamente con el canvas para la utilización de objetos UI en la escena.

- **MenuController**: Objeto en el que se encuentra el script que controla las funciones de los distintos botones del menú.

Objetos DontDestroyOnLoad:

- **Musica**: Objeto en el que se reproduce la música de la escena.
- **Flags**: Objeto encargado de trasladar la información de las opciones escogidas en el menú a las siguientes escenas.

ESCENA 2: IntroVideo

Esta escena muestra el vídeo introductorio que podemos ver cuando iniciamos una partida nueva.

Objetos:La siguiente lista enumera los distintos objetos presentes en la escena en el orden en el que se encuentran organizados.

- **Main camera**: Cámara principal a través de la cual el jugador puede observar la escena.
- **Directional Light**: Única iluminación de la escena.
- **Introduccion1**: En este objeto se encuentra el vídeo en sí, que se reproduce al iniciar la escena.
- **IntroVideoController**: objeto en el que se encuentra el script que controla que cuando se inicie la escena se lanze el vídeo y que una vez acabado se pase a la siguiente escena.
- **EventSystem**: Objeto que se crea conjuntamente con el canvas para la utilización de objetos UI en la escena.
- **Canvas**
    - Text: Texto en el que se nos indica que pulsando espacio se puede saltar el vídeo introductorio.

Objetos DontDestroyOnLoad:

- **Flags**: Objeto procedente de la escena Menu que no se destruye.
- **Musica**: Objeto procedente de la escena Menu que no se destruye.

ESCENA 3: LoadingScreen

En esta escena previa al juego en sí, se nos muestran los controles que se emplean en él.

Objetos:La siguiente lista enumera los distintos objetos presentes en la escena en el orden en el que se encuentran organizados.

- **Main camera**: Cámara principal a través de la cual el jugador puede observar la escena.
- **Directional Light**: Única iluminación de la escena.
- **Canvas**
    - Image: Imagen que se utiliza como fondo de la escena en la que se nos muestran los controles.
    - Text: Texto en el que se nos dice que pulsando E saltamos a la siguiente escena.
- **EventSystem**: Objeto que se crea conjuntamente con el canvas para la utilización de objetos UI en la escena.

- **MenuController**: Objeto en el que se encuentra el script MenuButton.

Objetos DontDestroyOnLoad:

- **Flags**: Objeto procedente de la escena Menu que no se destruye.
- **Musica**: Objeto procedente de la escena Menu que no se destruye.

ESCENA 4: Alpha

Escena en la que se nos presenta el primer nivel del juego.

Objetos:La siguiente lista enumera los distintos objetos presentes en la escena en el orden en el que se encuentran organizados.

- **Character**: Objeto que representa a nuestro personaje en la escena.
    - Bip001: Conjunto de objetos que conforman el cuerpo y esqueleto del personaje.
    - WK_HeavyIntantry: Elemento en el que se encuentra la textura del avatar.
    - Main camera: Cámara principal a través de la cual el jugador puede observar la escena.
    - AttackRange: Objeto en el que se maneja el área de acción para el ataque del personaje.
    - ShieldSound: Elemento que contiene el sonido que se produce al utilizar el escudo.
    - CombatSound: Objeto que contiene el sonido que se produce al atacar con la espada.
    - VictorySound: Objeto que contiene el sonido que se activa al superar el último desafío.
    - Shield: Icono de un escudo que aparece sobre la cabeza del personaje cuando lo utilizamos.
- **Controllers**
    - ConfigController: Objeto que contiene el script ConfigController.
    - PrincessController: Objeto que contiene el script PrincessController.
    - DialogController: Objeto que contiene el script DialogController.
    - InterfaceController: Objeto que contiene el script InterfaceController.
    - ItemController: Objeto que contiene el script ItemController.
    - CaravanController: Objeto que contiene el script CaravanController.
    - EnemyController: Objeto que contiene el script EnemyController.
    - BossController: Objeto que contiene el script BossController.
    - LanguageController: Objeto que contiene el script LanguageController.
    - SaveController: Objeto que contiene el script SaveController.
    - CinematicController: Objeto que contiene el script CinematicController.
    - LightContoller: Objeto que contiene el script LightController.
- **Interface**: Objeto en el que se encuentran los distintos elementos de la interfaz de usuario (UI).
    - Canvas
        - Dialogue: Objeto en el que se encuentran los elementos del cuadro de diálogo.
            - DialogPanel: Panel en el que se muestran los diálogos.
                - Background: Imagen de fondo del panel.
                - Title: Texto en el que se muestra el título del diálogo.

                - Description: Texto en el que aparece diálogo propiamente dicho.
                    - LeftButton: Botón izquierdo empleado para las decisiones de los diálogos.
                    - RightButton: Botón derecho empleado para las decisiones de los diálogos.
            - HUD: Objeto en el que se encuentran todos los elementos relacionados con las barras de vida de los distintos personajes del juego.
                - BossHealthBar: Barra de vida del jefe final.
                - HealthBar: Barra de vida de nuestro personaje.
                - CarriageStat: Objeto en el que se encuentran las imágenes con la vida de las caravanas
                    - Carriage_1: Vida de la caravana delantera.
                    - Carriage_2: Vida de la caravana de la princesa.
                    - Carriage_3: Vida de la caravana trasera.
                - Inventory: Indicador de total de pistas de la fase actual.
                - CombatCombo: Indicador del multiplicador de combate. Se gana 0.1 al dar un golpe, pierdes la mitad de las cargas al recibir daño.
        - EventSystem: Objeto que se crea conjuntamente con el canvas para la utilización de objetos UI en la escena.
- **Environment**: En este objeto se encuentran los elementos relacionados con el entorno.
    - Directional Light: Luz que se ha empleado para crear el mapa de luces del nivel.
    - Directional Light: Luz no estática del nivel.
- **Items**: Objeto en el que se encuentran todos aquellos que se pueden recoger en el nivel.
    - Clues: En este elemento se encuentran todas las pistas que se pueden conseguir en el nivel.
- **Miscelaneous**: Conjunto de objetos que por sus características no se podían añadir en otro grupo.
    - Target Fase 1-1: Objetivo de la caravana cuando se elije al principio el camino de la izquierda.
    - Target Fase 1-2: Objetivo de la caravana cuando se elije al principio el camino de la derecha.
    - Target Fase 1-3: Objetivo de la caravana que le lleva a la primera intersección en la que la AI decide.
    - Target Fase 2-1: Objetivo de la caravana cuando la AI elige en la primera intersección el camino de la izquierda.
    - Target Fase 2-2: Objetivo de la caravana cuando la AI elige en la primera intersección el camino de la derecha.
    - Target Fase 2-3: Objetivo de la caravana que le lleva a la segunda intersección en la que decide la AI.
    - Target Fase 3-1: Objetivo de la caravana cuando la AI elige en la segunda intersección el camino de la izquierda.
    - Target Fase 3-2:Objetivo de la caravana cuando la AI elige en la segunda intersección el camino de la derecha.
    - Target Final: Objetivo de la caravana que la lleva al final del nivel.
- **Caravan**: Objeto en el que se agrupan las tres caravanas del juego.

    - Carriage-1: Caravana delantera.
        - Modelo: Elemento que contiene todas las partes del modelo de la caravana.
    - Carriage Princesa: Caravana en la que viaja la princesa.
        - Modelo: Elemento que contiene todas las partes del modelo de la caravana.
    - Carriage 3: Caravana trasera.
        - Modelo: Elemento que contiene todas las partes del modelo de la caravana.
- **NPC**: Objeto en el que se agrupan los personajes no jugador exceptuando a las caravanas.
    - Wolfs: Objeto en el que se encuentran todos los lobos del nivel organizados según el camino en el que aparecen.
        - Camino 1- 2: Lobos que aparecen en el camino de la derecha del primer conjunto de caminos.
        - Camino 1-1: Lobos que aparecen en el camino de la izquierda del primer conjunto de caminos.
        - Camino 2-1: Lobos que aparecen en el camino de la izquierda del segundo conjunto de caminos.
        - Camino 3-2: Lobos que aparecen en el camino de la derecha del último conjunto de caminos.
    - Bandits: Objeto en el que se encuentran todos los bandidos del nivel organizados según el camino en el que aparecen.
        - Camino 1- 2: Bandidos que aparecen en el camino de la derecha del primer conjunto de caminos.
        - Camino 1-1: Bandidos que aparecen en el camino de la izquierda del primer conjunto de caminos.
        - Camino 2-2: Bandidos que aparecen en el camino de la derecha del segundo conjunto de caminos.
        - Camino 3-1: Bandidos que aparecen en el camino de la izquierda del último conjunto de caminos.
    - BossWolf: Jefe final que aparece en el claro al final del nivel.
    - BossMinion: Uno de los lobos que aparecen para ayudar al jefe en modo difícil.
    - BossMinion: El segundo lobo que aparece para ayudar al jefe en modo difícil.
- **Terreno Nivel 1**: Objeto en el que se agrupan todos los elementos del primer nivel que incluye: los cinco terrenos que conforman el nivel, la niebla que aparece en el terreno de nieve y las piedras.
- **Cinematic**: En este elemento se encuentran las cámaras que se emplean para la cinemática in-game del nivel.
- **BossStuff:** Contienen los gameObjects relacionados con el combate contra el jefe. Los dos últimos son propios únicamente del modo difícil.
    - BossDarkWall: Es el muro de niebla negra que impide escapar del combate contra el jefe. Hace 5 de daño cada 0.01 segundos, lo que provoca una muerte casi instantánea si se intenta atravesar.
    - IceAreas-Even: Se compone de grupo de áreas de fuego (Sí, de fuego) pares que aparecen durante la lucha contra el jefe en modo difícil. Hacen 5 de daño cada segundo que el avatar permanece sobre él.

    - IceAreas-Odd Se compone de grupo de áreas de fuego (Sí, de fuego) impares que aparecen durante la lucha contra el jefe en modo difícil. Hacen 5 de daño cada segundo que el avatar permanece sobre él.

Objetos DontDestroyOnLoad:

- **Flags**: Objeto procedente de la escena Menu que no se destruye.
- **Musica**: Objeto procedente de la escena Menu que no se destruye.

ESCENA 5: level2

A esta escena se accede tras completar el primer nivel.

Objetos:La siguiente lista enumera los distintos objetos presentes en la escena en el orden en el que se encuentran organizados.

- **Directional Light**: Única iluminación de la escena.
- **Terrain**: Terreno por el que nos movemos en la escena.
- **Controllers**
    - PositionController: Objeto que contiene el script PositionController.
- **Positions**: Objeto en el que se encuentran las distintas posiciones en las que empiezan las caravanas y el personaje cuando entran a la escena.
    - PositionCaravan1: Posición de la caravana delantera.
    - PositionCaravan2: Posición de la caravana de la princesa.
    - PositionCaravan3: Posición de la caravana trasera.
    - PositionPlayer: Posición inicial del personaje.

Objetos DontDestroyOnLoad:

- **Character**: Objeto procedente de la escena Alpha.
- **Controllers**: Objeto procedente de la escena Alpha
- **Interface**: Objeto procedente de la escena Alpha.
- **Caravan**: Objeto procedente de la escena Alpha.
- **Flags**: Objeto procedente de la escena Menu que no se destruye.
- **Musica**: Objeto procedente de la escena Menu que no se destruye.

ESCENA 6: GameOver

Escena a la que se accede al morir o perder la caravana de la princesa.

Objetos:La siguiente lista enumera los distintos objetos presentes en la escena en el orden en el que se encuentran organizados.

- **Main camera**: Cámara principal a través de la cual el jugador puede observar la escena.
- **Directional Light**: Única iluminación de la escena.
- **Canvas**
    - Image: Imagen que se utiliza como fondo de la escena.
    - Text: Texto en el que se nos dice que pulsando Esc salimos de la escena.
- **EventSystem**: Objeto que se crea conjuntamente con el canvas para la utilización de objetos UI en la escena.
- **GameOverController**: Objeto que contiene el script GameOverController.
- **GameOverAudio**: Objeto que contiene la pista de audio que se escucha al entrar en la escena.

Objetos DontDestroyOnLoad:

- **Flags**: Objeto procedente de la escena Menu que no se destruye.
- **Musica**: Objeto procedente de la escena Menu que no se destruye.

SCRIPTS

- Controllers:
    - BossController: Clase que sirve para controlar la activación del modo difícil del jefe final en caso de escribir: "cavitec" durante la cinemática. También sirve para hacer aparecer al propio jefe.
        - checkKeyByCode: método público que cambia el estado en función de la tecla pulsada y si esta es válida.
        - setEstadoByKey: método público que sirve para cambiar a otro estado en función del estado anterior y la última tecla pulsada.
    - CaravanController: Clase a la que se recurre para actuar sobre los *gameObjects* que se corresponden con las carrozas.
        - ChangeTarget: método público que sirve para cambiar el objetivo al que se dirige toda la caravan.
        - StopCaravan: método público que detiene a toda la caravana.
        - MoveCaravan: método público que reinicia el movimiento de toda la caravana.
        - DisableCaravan: método público que desactiva todas las carretas.
    - CinematicController: Clase que se utiliza para gestionar el correcto funcionamiento de la cinemática.
        - Update: Si ha comenzado la cinemática, se encarga de hacer los cambios de cámara apropiados de manera secuencial. Desactiva primero la cámara del jugador, lo mueve a la posición adecuada y después, una vez acabada la cinemática, la reactiva.
        - startCinematic: método público que activa la cinemática.
    - ConfigController: Esta clase sirve principalmente como archivo de configuración. Existen una serie de parámetros y flags que alteran el funcionamiento del juego, como puede ser la vida de los enemigos o la fase en la que nos encontramos. La mayoría de parámetros son estáticos, lo que nos permite acceder a ellos desde cualquier otra clase.
        - Awake: método público que inicializa algunos parámetros.
        - Update: método público con el que se comprueba si se pulsó ESC para abrir la ventana que permite volver al menú.
    - DialogController: Clase que nos permite manipular el sistema de diálogos.
        - cluepopup: método público que crea una ventana de diálogo con el texto de una pista. Se activa al recoger una de ellas.

        - changePrincessDialogue: método público que sirve para gestionar el primer diálogo del juego, en el que hablan el Avatar y la Princesa.
        - leftPath: método público que se activa al contestar SÍ en el primer diálogo del juego. Pone activos los enemigos del camino de la izquierda.
        - RightPath: método público que se activa al contestar NO en el primer diálogo del juego. Pone activos los enemigos del camino de la derecha.
        - princessDialogPopUp: método público que se utiliza para crear una ventana de diálogo con un mensaje de la princesa indicando el camino que ha decidido.
        - endGameDialog: método público que informa de que el nivel ya se ha completado. Permite pasar al siguiente nivel o enfrentarse al jefe.
        - trueEndGameDialog: método público método público que se utiliza para crear una ventana de diálogo informando de la derrota del jefe y que permite volver al menú principal o seguir explorando el mapa.
        - exitToMenuDialog: método público que crea una ventana de diálogo al pulsar la tecla ESC que permite volver al menú principal.
        - sayYes: método público que gestiona responder afirmativamente a una ventana de diálogo según el tipo del mismo.
        - sayNo: método público que gestiona responder negativamente a una ventana de diálogo según el tipo del mismo.
        - Update: método público que permite comprobar si se ha pulsado la tecla Espacio para poder avanzar o salir de un diálogo.
    - DontDestroyController: Clase que se utiliza para prevenir la destrucción de los controllers en ciertas escenas.
        - Start: Comprueba el nombre de la escena para saber si ha de destruir los controllers o no.
    - EnemyController: Clase que permite hacer que aparezcan o desaparezcan los enemigos de un camino.
        - SpawnEnemiesByPath: método público que permite hacer que aparezcan los enemigos de un camino en función de un número de camino(Izq o Dcha) y la fase actual.
    - GameOverController: Clase que se utiliza en la escena de GameOver.
        - Update: método que carga el menu al pulsar ESC.
    - InterfaceController: Clase que permite controlar el funcionamiento de la interfaz o HUD.
        - dealDamageCharacter: método público que reduce la barra de vida en función de la cantidad proveída

        - dealDamageCarriage: método público que reduce el contador de vida de la caravana en función de la cantidad proveída y el id de la caravana.
        - dealDamageToBoss: método público que reduce la barra de vida del jefe en función de la cantidad proveída.
        - resetLife: método público que permite devolver al estado original a la barra de vida del personaje.
    - IntroVideoController: Clase que se encarga de comprobar la finalización del vídeo de introducción.
        - Update: Comprueba constantemente si el vídeo ha terminado, para pasar consecuentemente a la siguiente escena.
    - ItemController: Clase donde se almacenan los textos de las pistas.
        - Clue: método público que llama a dialogController con el texto de la pista indicada.
    - LanguageController: Clase que sirve para gestionar las cadenas de texto en función del idioma.
        - Awake: Método público en el que se generan todas las estructuras con los textos según idioma y tipo
        - getTitleById: Devuelve un título (Los encabezados de los diálogos) en función del ID y del idioma configurado en el ConfigController
        - getTextById: Devuelve un texto, según el ID y el idioma actual.
        - getIntroById: Devuelve el texto introductorio en función del ID y el idioma actual.
        - getMenuOptionById: Devuelve una opción del menú, en función del ID y el idioma actual.
    - LightController: Clase que sirve para gestionar el cambio de luces.
        - SwapLight: Método que sirve para alternar el estado de las luces entre activadas y desactivadas.
    - PositionController: Clase que se utiliza para gestionar la posición en la que aparecerán el jugador y las caravanas en el cambio al segundo nivel.
        - Start: Coloca a las caravanas y al jugador en la posición predefinida.
    - PrincessController: Clase donde se gestiona la IA. Se almacena aquí el vector con los valores de Q-Learning para decidir que camino es realmente más difícil para el jugador.
        - updateQmatrix: Sirve para actualizar el vector de valores Q en función de unos parámetros del combate. Se llama por cada unidad que mata el jugador.
        - makeDecision: Se decide el camino a seguir en función de las pistas obtenidas y los valores en la matriz.
        - loadAI: Se llama para cargar los valores previos de la AI al cargar la partida.

- SaveController: Clase que sirve para gestionar el guardado y la carga de partidas.
  - SavePoint: Guarda una partida. Los parámetros que se le pasan son los valores que se guardarán. Son la posición del jugador y las caravanas, la fase actual, los valores de la IA…
  - LoadSavePoint: Se carga una partida en función de los valores almacenados con anterioridad.
- SceneController: Clase básica que sirve para cargar el menú después de los créditos iniciales.
  - Update: Se comprueba si ya es el momento de cambiar a la siguiente escena y se efectúa la transición si es necesario.

- NPCs:
  - Unit: Es una clase abstracta en la que se modela el concepto de unidad como cualquier entidad que se pueda ver envuelta en un combate dentro del juego. Dentro de la misma existen multitud de métodos para dar soporte al sistema de combate. A continuación una explicación del proceso a nivel conceptual:
    - Las unidades pasan por diversos estados durante el combate: esperando al siguiente ataque, animación de ataque, ataque en sí, en espera, muerto y cadáver.
    - Si la unidad no está en combate, esta se mantiene quieta a la espera de que una unidad enemiga pase cerca.
    - Si una unidad pasa lo suficientemente cerca como para activar el *collider AggroRange* la unidad previamente en espera pasará al estado *Waiting_for_attack* y perseguirá a la unidad que ha entrado en el área.
    - Después, si el enemigo de esta unidad se encuentra de su alcance, es decir, dentro del *collider AttackRange* ésta se detendrá y podrá atacarla si su temporizador de ataque ha llegado a cero.
    - Cuando se cumplen las dos condiciones anteriores, la unidad intentará realizar un ataque. Pasará al estado *Attacking* y ejecutará la animación de ataque. Acabado un segundo temporizador, se tratará de hacer daño al enemigo.
    - Si el enemigo no se estaba defendiendo (sólo puede el jugador) recibirá el daño asociado a nuestro valor, restándole la armadura del objetivo y posteriormente se pasa otra vez al estado *Waiting_for_attack.*
    - Durante todo este ciclo, la unidad reinicia su marcha en caso de que su objetivo se aleja de su rango de ataque *AttackRange*.
    - Si en algún momento, los puntos de vida de la unidad llegan a 0. Esta pasa al estado *Dead,* se ejecuta una animación de muerte y después de la misma pasa al estado *Corpse*. Una vez en ese estado, la unidad permanece activa durante unos segundos y posteriormente desaparece con su desactivación. Durante estos

dos estado, nunca se ejecuta el ciclo anterior. En caso contrario, seríamos testigos de macabras persecuciones.

- Si por otro lado, tenemos éxito en el combate y ya no quedan más unidades contra las que luchar, saldremos de combate, recuperaremos toda la salud y volveremos al estado *Idle.*
- En consecuencia a este ciclo y particularidades, existen numerosas funciones para darle soporte. Se ha intentado que los nombres de los métodos sean lo suficientemente descriptivos para aliviar un poco su compresión.

- Bandit
- Carriage
- Wolf
- Player: En esta y en las anteriores clases se implementan los siguientes métodos que son producto de heredar de la clase Unit.
  - getAttackDelay: método protegido que devuelve el valor de tiempo de preparación de ataque se usa como referencia.
  - getCombatAttackWindow: método protegido que devuelve la duración de la ventana de ataque que se usará como referencia

  Los siguientes son particulares de la clase Player:
  - GetInterfaceController: método protegido que sirve para manipular la interfaz desde el jugador. Principalmente para modificarla al recibir daño.
  - checkPlayerAttack: método protegido para comprobar si el jugador ha enviado una orden de ataque en las condiciones correctas
  - checkPlayerDefend: método protegido para comprobar si el jugador ha enviado la orden de defender en las condiciones adecuadas.
  - playerSwapTargets: método protegido para comprobar si el jugador ha enviado la orden de cambio de objetivo en las condiciones correctas.
- BossSimpleLogic: Clase que se encarga del correcto funcionamiento de los eventos del combate contra el jefe en modo Difícil y la invocación de muros de niebla en modo Normal. (El modo Difícil se activa escribiendo CAVITEC durante la cinemática)
  - Cuando el jefe llega a 1500 de vida invoca a dos lobos alfa para ayudarle
  - Al llegar a 1000 de vida se activan las primeras áreas de fuego.
  - Al llegar a 500 de vida se activan todas.

- CombatColliders:
  - AggroRange: Clase para controlar el área de búsqueda de enemigos.

        - OnTriggerEnter: método privado que sirve para agregar un unidad enemiga a lista de unidades en combate
        - OnTriggerExit: método privado análogo pero para quitar la unidad que haya salido
    - AtackRange: Clase para gestionar el área de ataque de la unidad.
        - OnTriggerEnter: método privado que sirve para agregar la unidad enemiga a la lista de unidades en rango de ataque
        - OnTriggerExit: método privado para eliminar una unidad enemiga de la lista de unidades en rango de ataque.

- Player:
    - PlayerMovement: Clase que sirve para controlar el movimiento del jugador.
        - Update: método privado para comprobar la pulsación de la teclas de movimiento.
        - Move: método privado para gestionar el movimiento del jugador.
        - Animating: método privado para poner las animaciones de movimiento o de estar quieto si procede.
- Otros:
    - Camino: Clase para guardar los datos sobres los caminos que existen. Se guarda el tipo de enemigos que hay, la fase y si el de la izquierda o el de la derecha.
    - CaravanTargetTrigger: Clase que sirve para redirigir a la caravana por una serie de WayPoints.
        - OnTriggerEnter: método privado que al ser llamado se cambia el WayPoint actual de la caravana al siguiente. En caso de ser el último se activa el diálogo de fin de juego.
    - Clue: Clase que se encarga de las pistas.
        - OnTriggerEnter: Cuando es llamada compara si el objeto que colisiona es el jugador y en ese caso llama a ItemController, desactiva la pista y aumenta el contador de pistas.
    - DecisionPrincessTrigger: Clase que se utiliza para gestionar la llegada de la caravana a un punto de decisión de camino, llamar a la princesa para que decida el camino a seguir y guardar la partida
        - OnTriggerEnter: Al ser llamada, guarda la partida y llama a la IA de la princesa para decidir el camino a seguir. Se activan los enemigos de cada camino en cada caso y se desactivan los del resto.
        - load: función para informar a esta clase de que se ha cargado la partida, por lo que la decisión del camino ya está hecha de antemano.

- DontDestroy: Clase para evitar la destrucción de un GameObject rápidamente.
  - Update: Comprueba la escena y destruye el objeto o no en función de la misma.
- Flags: Clase que sirve para pasarle a la escena del primer nivel algunos datos de configuración que se eligen en el menú.
  - setLanguage: Informa del lenguaje que se ha elegido en el menú
  - setLoad: Establece el flag de carga de partida. Por lo que se ha de tener en cuenta de que esta se trata de una partida cargada.
  - getLaguage y getLoad: devuelven estos dos flags respectivamente.
- GoAndNavigate: Clase que se encarga de darle un objetivo al componente NavMeshAgent.
  - Start: Guarda el componente NavMeshAgent en una variable.
  - Update: Si el componente está activado, cambia el destino por el que se encuentra en la variable Target.
- IceDamageArea: Clase que sirve para controlar el funcionamiento de las áreas de fuego(en su momento eran de hielo) que aparecen en el modo difícil del jefe final.
  - OnTriggerEnter: Se añade la unidad enemiga que ha entrado a la lista de unidades dentro.
  - OnTriggerExit: Función análoga a la anterior pero elimina a un enemigo de la lista.
  - Update: Hace daño a cada uno de los enemigos que permanecen dentro del área cada *timeToDamage.*
- PhaseTarget: Clase que únicamente tiene las variables que guardan el camino(path) y la fase(phase) en la que se encuentra.
- Rotator: Como indica su nombre se utiliza para hacer rotar sobre sí mismo un objeto (en nuestro caso las pistas).

- Menu
  - MenuButton: Clase que se encarga de las funciones de los botones de la escena Menu.
    - NewGame: Método público que nos lleva a la escena del vídeo introductorio
    - ExitGame: Método público que permite salir del juego.
    - LoadGame: Método público que, mediante la utilización de la clase Flags, nos lleva a la escena Alpha indicando que se está cargando una partida.
    - Language: Método público que permite cambiar el texto del idioma de la escena Menu, e indicar mediante la utilización de la clase Flags que se ha cambiado de idioma.
  - DontStopMusic: Clase que se encarga de que la música no se detenga al pasar a otra escena.

- Awake: Evita que el objeto se destruya al entrar en otra escena.
- Update: Si la escena en la que se encuentra es Alpha o IntroVideo el objeto en el que está el script se desactiva.

- ControlsContinue: Clase que se utiliza para que cuando nos encontramos en la escena LoadingScreen podamos pasar a la siguiente escena pulsando E.

SCRIPTS AI

Dado que en cada encrucijada hay dos posibles caminos para continuar, se decide que la princesa sea la que decida por qué camino debemos ir. Para este propósito es necesario dotarla de inteligencia.

Con el fin de aumentar la dificultad y el entretenimiento, la princesa elige en cada situación el camino más complicado en función a los enemigos de cada tramo.

Este poder de decisión se modela como una adaptación de Q-Learning.

Contamos con un valor de dificultad para cada enemigo. Cada vez que llegamos a una encrucijada se elige con una probabilidad del 80% el camino más complicado, salvo que hayas conseguido alguna pista, las cuales reducen la probabilidad de elegir el peor camino.

Estos valores de dificultad se actualizan al acabar cada combate, teniendo en cuenta el daño hecho a los enemigos, el daño recibido por el avatar, el daño recibido por la caravana y el tiempo necesario para derrotarlos. A la hora de calcular cuanto incrementar el valor, se premia el daño recibido por el avatar y la caravana y se penaliza cuanto daño reciben los enemigos, ponderando estos cálculos con los valores máximos o mínimos posibles en el tiempo de combate.

6.- ASSETS

Todos los assets que hemos utilizado son libres.

- No navegables:
  - Avatar:https://assetstore.unity.com/packages/3d/characters/toon-rts-units-demo-69687
  - Caravana: https://free3d.com
  - Bandidos: Brute-Mixamo https://www.mixamo.com/#/?page=1&query=brute&type=Character
  - Lobos: LowPoly Fantasy Monsters Pack Ver1.0_DEMO https://assetstore.unity.com/packages/3d/characters/creatures/lowpoly-fantasy-monsters-pack-ver1-0-demo-98393
  - Espada: https://assetstore.unity.com/packages/3d/sword-of-arc-17319
  - Escudo:https://assetstore.unity.com/packages/3d/props/weapons/shield-61351
  - Efectos y partículas: https://assetstore.unity.com/packages/vfx/particles/elemental-free-11158

- Libro: Book Pack
  https://free3d.com/3d-model/book-pack-93349.html
- Árboles:
  https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977
  https://assetstore.unity.com/packages/3d/environments/fantasy/green-forest-22762
  https://assetstore.unity.com/packages/3d/vegetation/free-desert-plants-32410

- Piedras
  https://assetstore.unity.com/packages/3d/environments/landscapes/rock-pack-19856

- Texturas:
  https://assetstore.unity.com/packages/2d/textures-materials/floors/rocky-texture-pack-40697
  https://assetstore.unity.com/packages/2d/textures-materials/grass-and-flowers-pack-1-17100
  https://assetstore.unity.com/packages/2d/textures-materials/floors/pbr-ground-materials-1-dirt-grass-85402
  https://assetstore.unity.com/packages/3d/environments/fantasy/green-forest-22762
  https://assetstore.unity.com/packages/2d/textures-materials/qs-materials-nature-pack-grass-vol-2-32020
  https://assetstore.unity.com/packages/2d/textures-materials/6-free-textures-90953

- Navegables:
  - Hierba
    https://assetstore.unity.com/packages/2d/textures-materials/6-free-textures-90953
    https://assetstore.unity.com/packages/2d/textures-materials/qs-materials-nature-pack-grass-vol-2-32020

  - Flores
    https://assetstore.unity.com/packages/2d/textures-materials/grass-and-flowers-pack-1-17100
  - Terreno

7.- ANEXO I: Scripts

Siguen el orden presentado en su sección:

- BossController

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;
```

```
public class BossController : MonoBehaviour {

    public GameObject Boss;
    public CaravanController caravanController;
    public InterfaceController interfaceController;
    public DialogController dialogController;

    private EstadoKey estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;

    enum EstadoKey
    {
        ESTADO_KEY_PRESSED_NONE,
        ESTADO_KEY_PRESSED_C,
        ESTADO_KEY_PRESSED_A,
        ESTADO_KEY_PRESSED_V,
        ESTADO_KEY_PRESSED_I,
        ESTADO_KEY_PRESSED_T,
        ESTADO_KEY_PRESSED_E
    }

            // Use this for initialization
            void Start ()
    {

            }

            // Update is called once per frame
            void Update ()
    {
        if (!ConfigController.endGame)
            return;

        switch(estado)
        {
            case EstadoKey.ESTADO_KEY_PRESSED_NONE:
                ////Debug.Log("En estado No");
                if (Input.GetKeyDown(KeyCode.C))
```

```
            estado = EstadoKey.ESTADO_KEY_PRESSED_C;
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString.Length);
        }
        break;
    case EstadoKey.ESTADO_KEY_PRESSED_C:
        //Debug.Log("En estaod C");
        if (Input.GetKeyDown(KeyCode.A))
            estado = EstadoKey.ESTADO_KEY_PRESSED_A;
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
        }
        break;
    case EstadoKey.ESTADO_KEY_PRESSED_A:
        //Debug.Log("En estaod A");
        if (Input.GetKeyDown(KeyCode.V))
            estado = EstadoKey.ESTADO_KEY_PRESSED_V;
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
        }
        break;
    case EstadoKey.ESTADO_KEY_PRESSED_V:
        //Debug.Log("En estaod V");
        if (Input.GetKeyDown(KeyCode.I))
            estado = EstadoKey.ESTADO_KEY_PRESSED_I;
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
        }
        break;
    case EstadoKey.ESTADO_KEY_PRESSED_I:
```

```
        //Debug.Log("En estaod I");
        if (Input.GetKeyDown(KeyCode.T))
            estado = EstadoKey.ESTADO_KEY_PRESSED_T;
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
        }
        break;
    case EstadoKey.ESTADO_KEY_PRESSED_T:
        //Debug.Log("En estaod T");
        if (Input.GetKeyDown(KeyCode.E))
            estado = EstadoKey.ESTADO_KEY_PRESSED_E;
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
        }
        break;
    case EstadoKey.ESTADO_KEY_PRESSED_E:
        //Debug.Log("En estaod E");
        if (Input.GetKeyDown(KeyCode.C))
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //spawnBoss();
            ConfigController.hardMode = true;
            //dialogController.hardModeOn();
        }
        else if (Input.inputString.Length != 0)
        {
            estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
            //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
        }
        break;
    default:
        break;
```

```
    }

        }

public void checkKeyCode(KeyCode keyCode)
{
    if (Input.GetKeyDown(keyCode))
        setEstadoByKey(keyCode);
    else if (Input.inputString.Length != 0)
    {
        estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
        //Debug.Log("Se pulsó otra tecla: " + Input.inputString);
    }
}

public void setEstadoByKey(KeyCode keyCode)
{
    switch(keyCode)
    {
        case KeyCode.C:
            if (estado == EstadoKey.ESTADO_KEY_PRESSED_E)
            {
                //spawnBoss();
                ConfigController.hardMode = true;
                //dialogController.hardModeOn();
                return;
            }

            estado = EstadoKey.ESTADO_KEY_PRESSED_A;
            break;
        case KeyCode.A:
            estado = EstadoKey.ESTADO_KEY_PRESSED_V;
            break;
        case KeyCode.V:
            estado = EstadoKey.ESTADO_KEY_PRESSED_I;
            break;
        case KeyCode.I:
            estado = EstadoKey.ESTADO_KEY_PRESSED_T;
```

```
                break;
            case KeyCode.T:
                estado = EstadoKey.ESTADO_KEY_PRESSED_E;
                break;
            case KeyCode.E:
                estado = EstadoKey.ESTADO_KEY_PRESSED_C;
                break;
            default:
                estado = EstadoKey.ESTADO_KEY_PRESSED_NONE;
                break;

        }
    }


    public void spawnBoss()
    {
        //Debug.Log("Spawneando Boss");
        Boss.SetActive(true);
        caravanController.disableCaravan();
        interfaceController.bossHealthBar.gameObject.SetActive(true);
                                interfaceController.bossHealthBar.gameObject.GetComponentInChildren<Text>().text     =
LanguageController.getTitleById(7);
        AudioSource[] audios = GameObject.FindWithTag("Player").GetComponents<AudioSource>();

        foreach (AudioSource audio in audios)
            audio.enabled = false;
    }
}
```

- CaravanController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class CaravanController : MonoBehaviour {

   private GameObject[] Caravan;
```

```
private void Start()
{
    //AVISO: Solo si se utiliza el mismo tag para todos los carruajes.
    Caravan=GameObject.FindGameObjectsWithTag("carriage");
}

public void ChangeTarget(Transform newTarget)
{
    for(int i=0; i<Caravan.Length; i++)
    {
        if(!Caravan[i].GetComponent<NavMeshAgent>().enabled)
            Caravan[i].GetComponent<NavMeshAgent>().enabled= true;

        Caravan[i].GetComponent<GoAndNavigate>().Target = newTarget;
    }
}

public void StopCaravan()
{
    for (int i = 0; i < Caravan.Length; i++)
    {
        Caravan[i].GetComponent<NavMeshAgent>().enabled = false;
    }
}

public void MoveCaravan()
{
    for(int i = 0; i < Caravan.Length; i++)
    {
        Caravan[i].GetComponent<NavMeshAgent>().enabled = true;
    }
}

public void disableCaravan()
{
    for(int i = 0; i < Caravan.Length; i++)
    {
        Caravan[i].SetActive(false);
```

```
        }
    }

}
```

- CinematicController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class CinematicController : MonoBehaviour
{
    public Camera camera1;
    public Camera camera2;
    public Camera camera3;
    public Camera camera4;
    public Camera cameraPlayer;
    public GameObject boss;
    public GameObject player;
    public GameObject playerPosition;
    public GameObject HUD;
    public static bool start = false;
    public static bool done = false;
    private const float fixedTime = 1.8f;
    public float time = 1.8f;
    public AnimationPhase aPhase = AnimationPhase.ANIMATION_PHASE_1;

    public enum AnimationPhase
    {
        ANIMATION_PHASE_1,
        ANIMATION_PHASE_2,
        ANIMATION_PHASE_3,
        ANIMATION_PHASE_4,
        ANIMATION_PHASE_END
    }

        // Use this for initialization
```

```
        void Start ()
{

        }

        // Update is called once per frame
        void Update ()
{
    if (!start)
        return;

    if (done)
        return;

    // Preparativos
    cameraPlayer.gameObject.SetActive(false);
    ConfigController.inCinematic = true;

    // Desactivamos el Aggro del Jefe
    GameObject aggroRange = boss.GetComponentInChildren<AggroRange>(true).gameObject;
    aggroRange.SetActive(false);
    HUD.SetActive(false);

    // Movemos al jugador a la posición
    player.SetActive(false);
    player.transform.position = playerPosition.transform.position;
    player.transform.rotation = new Quaternion(0, 180, 0, 0);
    player.SetActive(true);

    switch (aPhase)
    {
        case AnimationPhase.ANIMATION_PHASE_1:
            camera1.gameObject.SetActive(true);
            time -= Time.deltaTime;
            if (time < 0)
            {
                camera2.gameObject.SetActive(true);
                camera1.gameObject.SetActive(false);
```

```
            aPhase = AnimationPhase.ANIMATION_PHASE_2;
            time = fixedTime;
        }
        break;
    case AnimationPhase.ANIMATION_PHASE_2:
        time -= Time.deltaTime;
        if (time < 0)
        {
            camera3.gameObject.SetActive(true);
            camera2.gameObject.SetActive(false);
            aPhase = AnimationPhase.ANIMATION_PHASE_3;
            time = fixedTime;
        }
        break;
    case AnimationPhase.ANIMATION_PHASE_3:
        time -= Time.deltaTime;
        if (time < 0)
        {
            camera4.gameObject.SetActive(true);
            camera3.gameObject.SetActive(false);
            aPhase = AnimationPhase.ANIMATION_PHASE_4;
            time = fixedTime;
        }
        break;
    case AnimationPhase.ANIMATION_PHASE_4:
        time -= Time.deltaTime;
        if (time < 0)
        {
            camera4.gameObject.SetActive(false);
            aPhase = AnimationPhase.ANIMATION_PHASE_END;
            start = false;
            time = fixedTime;
            cameraPlayer.gameObject.SetActive(true);
            aggroRange.SetActive(true);
            ConfigController.inCinematic = false;
            HUD.SetActive(true);
        }
        break;
```

```
            default:
                break;

        }
            }

    public static void startCinematic()
    {
        start = true;
        done = false;
    }

}
```

- ConfigController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ConfigController : MonoBehaviour {

    // Valores de configuración
            public static int characterMaxLife = 100;
            public static int carriageMaxLife = 100;
            public static int banditAgressiveMaxLife = 100;
            public static int wolfMaxLife = 100;
            public static int characterSpeed = 5;
            public static int carriageSpeed = 3;
            public static int wolfSpeed = 4;
            public static int banditAgressiveSpeed= 4;
    public static LanguageController.Language lenguaje = LanguageController.Language.LANGUAGE_ENGLISH;

    // Flags
    public static int phase = 1;
    public static bool firstDialog = true;
    public static bool endGame = true;
    public static bool inCinematic = false;
```

```
    public static bool hardMode = false;
    public static bool enhancedCombatMode = true;

    // Misc
    public DialogController dialogController;
    public static int clueTotal = 0;

    // Technical values
    public static int combatUpdateFrames = 1;

    private void Update()
    {
        if (firstDialog)
            return;

        if (Input.GetKeyDown(KeyCode.Escape))
        {
            dialogController.exitToMenuDialog();
        }

    }
    // Use this for initialization
    void Awake ()
    {
        firstDialog = true;
        endGame = true; //TODO cambiar a false
        lenguaje = (LanguageController.Language)Flags.getLenguage();
            }

}
```

- DialogController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
```

```
public class DialogController : MonoBehaviour
{
    public CaravanController caravanController;
    public BossController bossController;
    public GameObject panel;
    public Text title;
    public Text description;
    private GameObject[] buttons;
    public Transform LeftTarget;
    public Transform RightTarget;
    public SaveController SaveController;
    public bool init;
    public int phraseAux = 0;
    private DialogType type = DialogType.DIALOG_TYPE_PRINCESS_START;

    enum DialogType
    {
        DIALOG_TYPE_NONE,
        DIALOG_TYPE_PRINCESS_START,
        DIALOG_TYPE_CLUE,
        DIALOG_TYPE_PRINCESS_PATH_DECISION,
        DIALOG_TYPE_END_GAME,
        DIALOG_TYPE_EXIT_GAME,
        DIALOG_TYPE_TRUE_END_GAME
    }

    public List<string> introDialogue;

    // Use this for initialization
    void Start()
    {

        introDialogue = LanguageController.getIntroById(1);
        buttons = GameObject.FindGameObjectsWithTag("button");
        buttons[0].SetActive(false);
        buttons[1].SetActive(false);
        buttons[0].GetComponentInChildren<Text>().text = LanguageController.getMenuOptionById(0);
```

```
    buttons[1].GetComponentInChildren<Text>().text = LanguageController.getMenuOptionById(1);

    if (Flags.getLoad())
    {
        SaveController.LoadSavePoint();
        init = false;
        ConfigController.firstDialog = false;
        buttons[0].SetActive(false);
        buttons[1].SetActive(false);
        panel.SetActive(false);
    }
    else
    {
        StartCoroutine(changePrincessDialogue(introDialogue[0], "avatar"));
        init = true;
    }


}

// Update is called once per frame
void Update()
{

    if (init)
    {

        if (Input.GetKeyDown(KeyCode.Space))
        {
            //Debug.Log ("incrementing phrase");
            phraseAux++;
            if (phraseAux < introDialogue.Count)
            {

                //Debug.Log("inside phase");

                if ((phraseAux % 2) == 1)
                {
```

```
                StartCoroutine(changePrincessDialogue(introDialogue[phraseAux], "princess"));
            }
            else
            {
                StartCoroutine(changePrincessDialogue(introDialogue[phraseAux], "avatar"));
            }

        }
        else
        {

            init = false;
            buttons[0].SetActive(true);
            buttons[1].SetActive(true);
        }

    }


}
else
{
    if (ConfigController.firstDialog)
        return;

    if (Input.GetKeyDown(KeyCode.Space))
    {
        if (type == DialogType.DIALOG_TYPE_PRINCESS_PATH_DECISION)
            caravanController.MoveCaravan();

        buttons[0].SetActive(false);
        buttons[1].SetActive(false);
        panel.SetActive(false);

    }
}
}
```

```
public IEnumerator changePrincessDialogue(string phrase, string character)
{ //SE PUEDE JUNTAR CON EL OTRO CHANGE

    if (character == "princess")
    {
        title.text = LanguageController.getTitleById(2);
    }
    if (character == "avatar")
    {
        title.text = LanguageController.getTitleById(3);
    }
    int letter = 0;
    description.text = "";
    while (letter < phrase.Length)
    {
        description.text += phrase[letter];
        letter += 1;
        yield return new WaitForSeconds(0.02f);

    }

}

public void cluepopup(string clue)
{// se llama desde la pista.

    title.text = LanguageController.getTitleById(1);
    description.text = clue;
    panel.SetActive(true);
    buttons[0].SetActive(false);
    buttons[1].SetActive(false);
}



public void princessDialogPopUp(string text)
{
    type = DialogType.DIALOG_TYPE_PRINCESS_PATH_DECISION;
```

```
        title.text = LanguageController.getTitleById(2);
        description.text = text;
        panel.SetActive(true);
        buttons[0].SetActive(false);
        buttons[1].SetActive(false);
    }

    public void endGameDialog()
    {
        type = DialogType.DIALOG_TYPE_END_GAME;
        title.text = LanguageController.getTitleById(5);
        description.text = LanguageController.getTextById(1);
        panel.SetActive(true);
        buttons[0].SetActive(true);
        buttons[1].SetActive(true);

        ConfigController.endGame = true;
    }

    public void trueEndGameDialog()
    {
        type = DialogType.DIALOG_TYPE_TRUE_END_GAME;
        title.text = LanguageController.getTitleById(6);
        description.text = LanguageController.getTextById(2);
        panel.SetActive(true);
        buttons[0].SetActive(true);
        buttons[1].SetActive(true);

        ConfigController.endGame = true;
    }

    public void exitToMenuDialog()
    {
        type = DialogType.DIALOG_TYPE_EXIT_GAME;
        title.text = LanguageController.getTitleById(4) ;
        description.text = LanguageController.getTextById(3);
        panel.SetActive(true);
        buttons[0].SetActive(true);
```

```
    buttons[1].SetActive(true);

}

public void hardModeOn()
{
    type = DialogType.DIALOG_TYPE_PRINCESS_PATH_DECISION;
    title.text = LanguageController.getTitleById(2);
    description.text = LanguageController.getTextById(6);
    panel.SetActive(true);
    buttons[0].SetActive(false);
    buttons[1].SetActive(false);

}

public void sayYes()
{
    switch(type)
    {
        case DialogType.DIALOG_TYPE_PRINCESS_START:
            ConfigController.firstDialog = false;
            leftPath();
            break;
        case DialogType.DIALOG_TYPE_END_GAME:
        case DialogType.DIALOG_TYPE_TRUE_END_GAME:
            panel.SetActive(false);
            buttons[0].SetActive(false);
            buttons[1].SetActive(false);
            SaveController.SavePoint(ConfigController.phase, 0, 0, 0, 0, 0);
            SceneManager.LoadScene("level2");
            break;
        case DialogType.DIALOG_TYPE_EXIT_GAME:
            SceneManager.LoadScene("Menu");
            break;
        default:
            break;

    }
```

```
}

public void sayNo()
{
    switch (type)
    {
        case DialogType.DIALOG_TYPE_PRINCESS_START:
            ConfigController.firstDialog = false;
            RightPath();
            break;
        case DialogType.DIALOG_TYPE_END_GAME:
            panel.SetActive(false);
            buttons[0].SetActive(false);
            buttons[1].SetActive(false);
            bossController.spawnBoss();
            LightController.SwapLight(false);
            CinematicController.startCinematic();
            SaveController.SavePoint(ConfigController.phase, 0, 0, 0, 0, 0);
            break;
        case DialogType.DIALOG_TYPE_TRUE_END_GAME:
            panel.SetActive(false);
            buttons[0].SetActive(false);
            buttons[1].SetActive(false);
            LightController.SwapLight(true);
            break;
        case DialogType.DIALOG_TYPE_EXIT_GAME:
            panel.SetActive(false);
            buttons[0].SetActive(false);
            buttons[1].SetActive(false);
            break;
        default:
            break;
    }
}

public void leftPath()
{
    caravanController.ChangeTarget(LeftTarget);
```

```
        panel.SetActive(false);
        EnemyController.SpawnEnemiesByPath(1, 1);
    }

    public void RightPath()
    {
        caravanController.ChangeTarget(RightTarget);
        panel.SetActive(false);
        EnemyController.SpawnEnemiesByPath(1, 2);
    }

}
```

- DonDestroyControllers

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class DontDestroyControllers : MonoBehaviour {

        // Use this for initialization
        void Start ()
    {

        if (SceneManager.GetActiveScene().name == "Alpha")
            DontDestroyOnLoad(this.gameObject);

            }

        // Update is called once per frame
        void Update ()
    {
                if(SceneManager.GetActiveScene().name != "Alpha" && SceneManager.GetActiveScene().name != "level2")
        Destroy(this.gameObject);
    }
}
```

- EnemyController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyController : MonoBehaviour {

    public static void SpawnEnemiesByPath(int phase, int path)
    {
#pragma warning disable CS0618 // El tipo o el miembro están obsoletos
        Camino[] Camino = (Camino[])FindObjectsOfTypeAll(typeof(Camino));
#pragma warning restore CS0618 // El tipo o el miembro están obsoletos

        if (Camino.Length == 0)
            Debug.Log("Camino vacío");

        foreach (Camino camino in Camino)
        {
            if (camino.phase == phase && camino.path == path)
            {
                //Debug.Log("Cumpliendo criterios de spawn");
                camino.gameObject.SetActive(true);
            }
            else
                camino.gameObject.SetActive(false);
        }
    }
}
```

- GameOverController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
```

```
public class GameOverController : MonoBehaviour {


        // Update is called once per frame
        void Update () {
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        Debug.Log("Cargando la escena");
        SceneManager.LoadScene("Menu");
    }
        }
}
```

- InterfaceController

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class InterfaceController : MonoBehaviour {

    // Vida del jugador
    public Scrollbar healthBar;
    public int lifeValue;

    // Vida del Boss
    public Scrollbar bossHealthBar;
    public int bossLifeValue;

    // Vida de las caravanas
        public Text[] carriagesText = new Text[3];

    // Total de pistas
    public Text clues;

    // Multiplicador de daño
```

```
public Text damageMultiplier;
public GameObject visualEffect;
public Player player;

// Misc
public Canvas inventoryCanvas;
        public Canvas princessDialogue;
        public Canvas clueDescription;
        public Canvas adviseCanvas;

        void Awake()
        {
                for(int i=0 ; i<3 ; i++){
                        carriagesText[i].text ="100";
                }
        }

private void Update()
{
    clues.text = ConfigController.clueTotal.ToString();
    damageMultiplier.text = player.damageMultiplier.ToString();
    if (player.damageMultiplier >= 1.5f)
        visualEffect.SetActive(true);
    else
        visualEffect.SetActive(false);
}




        //reset character life
        public void resetLife()
{

                lifeValue = ConfigController.characterMaxLife;
    healthBar.size = ConfigController.characterMaxLife;

}
```

```
        //decrement the healthbar for the character
        public void dealDamageCharacter(int damage)
        { //Resta a la vida la entrada damage
                lifeValue = lifeValue - damage;
                if (lifeValue <= 0)
                {
                        lifeValue = 0;
                        Destroy(this.gameObject);
                }
                healthBar.size = lifeValue / 100f;
        }

//decrement the healthbar for the boss
public void dealDamageToBoss(int damage)
{ //Resta a la vida la entrada damage
    if(ConfigController.hardMode)
        damage /= 2;
    bossLifeValue = bossLifeValue - damage;
    if (bossLifeValue <= 0)
    {
        bossLifeValue = 0;
        Destroy(this.gameObject);
    }
    bossHealthBar.size = bossLifeValue / 1000f;

}

//dechrement de life fot the carriages, should indicate the carriage number
public void dealDamageCarriage(int objective, int damage)
{
                int actual = 0;

                Int32.TryParse(carriagesText[objective].text, out actual);

                if(actual>damage){
                        carriagesText[objective].text= (actual-damage).ToString();
                }else{
                        carriagesText[objective].text= "X";
```

```
                }
            }


}
```

- IntroVideoController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class IntroVideoController : MonoBehaviour
{

    public float timeLeft = 41;

    // Update is called once per frame
    void Update()
    {
        timeLeft -= Time.deltaTime;

        if (timeLeft < 0 ||Input.GetKeyDown(KeyCode.Space))
        {
            SceneManager.LoadScene("LoadingScreen");
        }
    }
}
```

- ItemController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ItemController : MonoBehaviour {
```

```
    public int clueNum = 0;
    private string description;
    public DialogController dialogController;
    private GameObject[] caravan;

    private void Start()
    {
        GameObject.FindGameObjectsWithTag("carriage");
    }



    public void Clue(int ID)
    {
        switch (ID)
        {
            case 1:
                description = "Quien con bestias anda, a aullar aprende";
                dialogController.cluepopup(description);
                break;
            case 2:
                    description = "Cuando el zorro escucha gritar a la liebre siempre llega corriendo, pero no para ayudarla";
                dialogController.cluepopup(description);
                break;
            case 3:
                description = "Si hurtas y das, te librarás";
                dialogController.cluepopup(description);
                break;
            case 4:
                description = "De la noche en la espesura, hasta la nieve es oscura";
                dialogController.cluepopup(description);
                break;
            case 5:
                    description = "Cuidate de la niebla, comandante. Podría ocultar lo que ni los propios dioses osan mirar.";
                dialogController.cluepopup(description);
                break;
            case 6:
                description = "La blancura de la nieve hace al cisne negro";
```

```
                dialogController.cluepopup(description);
                break;

            default:
                description = "Esta pista no está funcionando como debería";
                break;
        }
        clueNum++;
    }
}
```

- LanguageController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LanguageController : MonoBehaviour
{
    public enum Language
    {
        LANGUAGE_SPANISH,
        LANGUAGE_ENGLISH,
        LANGUAGE_GALICIAN
    }

    public struct introStruct
    {
        public int id;
        public Language lenguaje;
        public List<String> texto;
    }

    public struct textStruct
```

```
{
    public int id;
    public Language lenguaje;
    public String texto;
}

public static List<introStruct> introList = new List<introStruct>();
public static List<textStruct> titleList = new List<textStruct>();
public static List<textStruct> textoList = new List<textStruct>();

public void Awake()
{
    // Creamos las intros para el primer nivel

    introStruct introStruct = new introStruct();

    // Creamos la intro en Spanish
    introStruct.id = 1;
      introStruct.texto = new List<string>(new string[] {"Princesa, hemos llegado al desfiladero de brenner.\nDicen que en este lugar ya han perecido muchas otras expediciones.\nPreparaos para lo peor mi señora.",
                                "No tengo miedo alguno. Se que vos me protegeréis. Muchos otros me han guardado, pero nunca tan bien como vos.",
                            "Roguémosle al Señor, y avancemos. ¿Qué camino prefiere vuestra merced?",
                        "Creo que el mejor camino para seguir será el izquierdo. ¿Que opinais vos?"});
    introStruct.lenguaje = Language.LANGUAGE_SPANISH;
    introList.Add(introStruct);

    //Creamos la intro en Inglés
    introStruct = new introStruct();

    introStruct.id = 1;
        introStruct.texto = new List<string>(new string[] {"<Inglés>Princesa, hemos llegado al desfiladero de brenner.\nDicen que en este lugar ya han perecido muchas otras expediciones.\nPreparaos para lo peor mi señora.",
                                "No tengo miedo alguno. Se que vos me protegeréis. Muchos otros me han guardado, pero nunca tan bien como vos.",
                            "Roguémosle al Señor, y avancemos. ¿Qué camino prefiere vuestra merced?",
                        "Creo que el mejor camino para seguir será el izquierdo. ¿Que opinais vos?</Inglés>"});
    introStruct.lenguaje = Language.LANGUAGE_ENGLISH;
    introList.Add(introStruct);
```

```
// Creamos los string con los textos normales

//      Primero los titulos de los diálogos
textStruct titleStruct = new textStruct();

// Pista
titleStruct.id = 1;
titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
titleStruct.texto = "PISTA";
titleList.Add(titleStruct);

titleStruct = new textStruct();
titleStruct.id = 1;
titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
titleStruct.texto = "CLUE";
titleList.Add(titleStruct);

// Princesa
titleStruct = new textStruct();
titleStruct.id = 2;
titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
titleStruct.texto = "PRINCESA";
titleList.Add(titleStruct);

titleStruct = new textStruct();
titleStruct.id = 2;
titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
titleStruct.texto = "PRINCESS";
titleList.Add(titleStruct);

// Yo
titleStruct = new textStruct();
titleStruct.id = 3;
titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
titleStruct.texto = "YO";
titleList.Add(titleStruct);
```

```
titleStruct = new textStruct();
titleStruct.id = 3;
titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
titleStruct.texto = "ME";
titleList.Add(titleStruct);

// Menu
titleStruct = new textStruct();
titleStruct.id = 4;
titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
titleStruct.texto = "MENÚ";
titleList.Add(titleStruct);

titleStruct = new textStruct();
titleStruct.id = 4;
titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
titleStruct.texto = "MENU";
titleList.Add(titleStruct);

// Fin de juego
titleStruct = new textStruct();
titleStruct.id = 5;
titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
titleStruct.texto = "FIN DE JUEGO";
titleList.Add(titleStruct);

titleStruct = new textStruct();
titleStruct.id = 5;
titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
titleStruct.texto = "THE END";
titleList.Add(titleStruct);

// Verdadero fin de juego
titleStruct = new textStruct();
titleStruct.id = 6;
titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
titleStruct.texto = "VERDADERO FIN DE JUEGO";
titleList.Add(titleStruct);
```

```
        titleStruct = new textStruct();
        titleStruct.id = 6;
        titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
        titleStruct.texto = "THE TRUE END";
        titleList.Add(titleStruct);

        // Boss Name
        titleStruct = new textStruct();
        titleStruct.id = 7;
        titleStruct.lenguaje = Language.LANGUAGE_SPANISH;
        titleStruct.texto = "Gran Lobo Espadachín";
        titleList.Add(titleStruct);

        titleStruct = new textStruct();
        titleStruct.id = 7;
        titleStruct.lenguaje = Language.LANGUAGE_ENGLISH;
        titleStruct.texto = "Great Wolf SwordMaster";
        titleList.Add(titleStruct);

        // Ahora los propios textos
        textStruct textoStruct = new textStruct();

        // Texto de ganar
        textoStruct.id = 1;
        textoStruct.lenguaje = Language.LANGUAGE_SPANISH;
          textoStruct.texto = "¡Enhorabuena! Has superado el nivel. Puedes seguir explorando el mapa si quieres. ¿Ir al
siguiente nivel?";
        textoList.Add(textoStruct);

        textoStruct = new textStruct();
        textoStruct.id = 1;
        textoStruct.lenguaje = Language.LANGUAGE_ENGLISH;
          textoStruct.texto = "Congrats! You have beaten the level. You can keep exploring the map if you like. Go to
the next level?";
        textoList.Add(textoStruct);

        // Texto verdadero de ganar
        textoStruct.id = 2;
```

```
        textoStruct.lenguaje = Language.LANGUAGE_SPANISH;
          textoStruct.texto = "¡Enhorabuena! Has superado el verdadero desafío.Ya no tenemos nada más que ofrecerte ¿Ir al siguiente nivel?";
        textoList.Add(textoStruct);

        textoStruct = new textStruct();
        textoStruct.id = 2;
        textoStruct.lenguaje = Language.LANGUAGE_ENGLISH;
          textoStruct.texto = "Congrats! You have beaten the true challenge. We do not have any more to offer. Go to the next level?";
        textoList.Add(textoStruct);

        // texto que sale al pulsar esc
        textoStruct.id = 3;
        textoStruct.lenguaje = Language.LANGUAGE_SPANISH;
        textoStruct.texto = "¿Volver al menú principal?";
        textoList.Add(textoStruct);

        textoStruct = new textStruct();
        textoStruct.id = 3;
        textoStruct.lenguaje = Language.LANGUAGE_ENGLISH;
        textoStruct.texto = "Go back to the main menu?";
        textoList.Add(textoStruct);

        // Princesa elige el camino el camino de la izquierda
        textoStruct.id = 4;
        textoStruct.lenguaje = Language.LANGUAGE_SPANISH;
        textoStruct.texto = "Creo que la mejor opción será elegir el camino de la izquierda";
        textoList.Add(textoStruct);

        textoStruct = new textStruct();
        textoStruct.id = 4;
        textoStruct.lenguaje = Language.LANGUAGE_ENGLISH;
        textoStruct.texto = "I think the best path would be the one of the left-side";
        textoList.Add(textoStruct);

        // Princesa elige el camino el camino de la derecha
        textoStruct.id = 5;
        textoStruct.lenguaje = Language.LANGUAGE_SPANISH;
```

```
        textoStruct.texto = "Creo que la mejor opción será elegir el camino de la derecha";
        textoList.Add(textoStruct);

        textoStruct = new textStruct();
        textoStruct.id = 5;
        textoStruct.lenguaje = Language.LANGUAGE_ENGLISH;
        textoStruct.texto = "I think the best path would be the one of the right-side";
        textoList.Add(textoStruct);

        // Princesa elige el camino el camino de la derecha
        textoStruct.id = 6;
        textoStruct.lenguaje = Language.LANGUAGE_SPANISH;
        textoStruct.texto = "Mode díficil activado. Espero que sepas lo que haces";
        textoList.Add(textoStruct);

        textoStruct = new textStruct();
        textoStruct.id = 6;
        textoStruct.lenguaje = Language.LANGUAGE_ENGLISH;
        textoStruct.texto = "Hard Mode On. I hope you know what you are doing.";
        textoList.Add(textoStruct);
    }

    public static String getTitleById(int id)
    {
        foreach(textStruct title in titleList)
        {
            if (title.id == id && ConfigController.lenguaje == title.lenguaje)
                return title.texto;
        }

        return "String not found";
    }

    public static String getTextById(int id)
    {
        foreach (textStruct text in textoList)
        {
            if (text.id == id && ConfigController.lenguaje == text.lenguaje)
```

```
            return text.texto;
        }

        return "String not found";
    }

    public static List<String> getIntroById(int id)
    {
        foreach (introStruct intro in introList)
        {
            //Debug.Log("El id es " + id + " el idioma es " + intro.lenguaje);
            if (intro.id == id && ConfigController.lenguaje == intro.lenguaje)
                return intro.texto;
        }

        return new List<string>(new string[] { "No", "se", "encontro", "la string" });
    }

    public static String getMenuOptionById(int id)
    {
        switch(id)
        {
            case 0:

                if (ConfigController.lenguaje == Language.LANGUAGE_ENGLISH)
                    return "No";
                else
                    return "No";
            case 1:

                if (ConfigController.lenguaje == Language.LANGUAGE_ENGLISH)
                    return "Yes";
                else
                    return "Sí";
            default:

                return "Not found";
        }
```

```
    }
}
```

- LightController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LightController : MonoBehaviour {

    private static GameObject[] lights;
    private void Start()
    {
        lights=GameObject.FindGameObjectsWithTag("Light");

    }

    public static void SwapLight(bool change)
    {
        for(int i=0; i<lights.Length; i++)
        {
            lights[i].SetActive(change);
        }
    }
}
```

- PositionController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PositionController : MonoBehaviour
{
    public GameObject positionCaravan1;
    public GameObject positionCaravan2;
    public GameObject positionCaravan3;
```

```
    public GameObject positionPlayer;

    // Use this for initialization
    void Start()
    {
        GameObject[] caravans = GameObject.FindGameObjectsWithTag("carriage");
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        player.transform.position = positionPlayer.transform.position;
        caravans[0].transform.position = positionCaravan1.transform.position;
        caravans[1].transform.position = positionCaravan2.transform.position;
        caravans[2].transform.position = positionCaravan3.transform.position;
    }

}
```

- PrincessController

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PrincessController : MonoBehaviour {

            private static int damageVsPlayer;
            private static int damageVsCarriage;
            private static int damageVsEnemy;
            private static int combatTime;
            private static int enemyCadence;

            public static float[] Q_Matrix = new float[2];

            private static float[] alphaValues = { 0.5f, 0.2f, 0.1f, 0.5f };

            public static int[] iterations = { 0, 0 };

            public static int[,] ways = {{0,1},{1,0}};
```

```
                                    // Daño base                    // Cadencias                    // Daño hecho al final del combate                    // Tipo de enemigos
        public static void updateQMatrix(int damageVsP, int damageVsC, int damageVsE, float enemyCad, float playerCad, float time, int damageDealVsPlayer, int damageDealVsCarriage, int damageDealVsEnemy, string enemyType){

                int actualEnemy=0;
                float maxPuntuation = 0;
                float minPuntuation = 0;
                float actualPuntuation = 0;
                float weightedValue = 0;

                switch (enemyType) {
                case "wolf":
                        actualEnemy = 0;
                        break;
                case "agressiveBandit":
                        actualEnemy = 1;
                        break;
                default:
                        break;
                }

    //Debug.Log.Log("Los daños son " + damageVsP + " " + damageVsC + " " + damageVsE);
    //Debug.Log.Log("Las cadencias son " + enemyCad + " " + playerCad);
    //Debug.Log.Log("Los tiempos son " + time);
      //Debug.Log.Log("Los daños hechos son " + damageDealVsPlayer + " " + damageDealVsCarriage + " " + damageVsEnemy);
    //Debug.Log.Log("La matriz antes de nada es " + Q_Matrix[actualEnemy]);

                //Calculate MaxPuntuation
                maxPuntuation = (damageVsP/enemyCad) + (2*damageVsC/enemyCad);

                //Calculate MinPuntuation
                minPuntuation = -damageVsE/playerCad;

                //Calculate ActualPuntuation
                actualPuntuation = (damageDealVsPlayer+2*damageDealVsCarriage-damageDealVsEnemy)/time;

                //EscalateValue
```

```
                weightedValue = ((actualPuntuation-minPuntuation)*100)/(maxPuntuation-minPuntuation);

                //the real update of the array
                Q_Matrix[actualEnemy]=
Q_Matrix[actualEnemy]+alphaValues[iterations[actualEnemy]]*(weightedValue-Q_Matrix[actualEnemy]);

                if(iterations[actualEnemy]!=3){
                        iterations[actualEnemy]++;
                }

    //Debug.Log.Log("La matriz ahora es " + Q_Matrix[actualEnemy]);

        }

        public static int makeDecision(int phase, int numberOfClues){

                int probability = 80-numberOfClues*40;

                System.Random rnd = new System.Random ();
                int rand = rnd.Next (1, 100);

                int enemyL = ways [phase - 1, 0];
                int enemyR = ways [phase - 1, 1];

                if (rand <= probability) {
                        if(Q_Matrix[enemyL]<Q_Matrix[enemyR]){
                                return 2;
                        }else{
                                return 1;
                        }
                } else {
                        if(Q_Matrix[enemyL]>Q_Matrix[enemyR]){
                                return 2;
                        }else{
                                return 1;
                        }
                }

        }
```

```
    public static void loadAI(float Qwolf, float Qbandit, int Iterations0, int Iterations1)
    {
        Q_Matrix[0] = Qwolf;
        Q_Matrix[1] = Qbandit;
        iterations[0] = Iterations0;
        iterations[1] = Iterations1;
    }


}
```

- SaveController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class SaveController: MonoBehaviour {
    public  GameObject[] Caravan;
    public  GameObject Character;
    public Transform savepoint2;
    public  InterfaceController InterfaceController;
    public DecisionPrincessTrigger decisionPrincessTrigger;




    public  void SavePoint(int phase, int decision, float qmatrix0, float qmatrix1, int iterations0, int iterations1)
    {
        //guardamos los valores de la AI
        PlayerPrefs.SetInt("Phase", phase);
        PlayerPrefs.SetFloat("QMatrix0", qmatrix0);
        PlayerPrefs.SetFloat("QMatrix1", qmatrix1);
        PlayerPrefs.SetInt("Iterations0", iterations0);
        PlayerPrefs.SetInt("Iterations1", iterations1);
```

```
        //guardamos decision del camino
        PlayerPrefs.SetInt("Decision", decision);

        for (int i = 0; i < Caravan.Length; i++)
            {
                //guarda los puntos de vida de cada caravana
                PlayerPrefs.SetInt("CarriageHealth"+ Caravan[i].GetComponent<Carriage>().getId(),
                    Caravan[i].GetComponent<Carriage>().getHealthPoints());
                //guardamos la posicion de la caravana
                                    PlayerPrefs.SetFloat("PosCar" + Caravan[i].GetComponent<Carriage>().getId() + "X",
Caravan[i].transform.position.x);
                                    PlayerPrefs.SetFloat("PosCar" + Caravan[i].GetComponent<Carriage>().getId() + "Y",
Caravan[i].transform.position.y);
                                    PlayerPrefs.SetFloat("PosCar" + Caravan[i].GetComponent<Carriage>().getId() + "Z",
Caravan[i].transform.position.z);
                                    PlayerPrefs.SetFloat("RotCar" + Caravan[i].GetComponent<Carriage>().getId() + "X",
Caravan[i].transform.rotation.x);
                                    PlayerPrefs.SetFloat("RotCar" + Caravan[i].GetComponent<Carriage>().getId() + "Y",
Caravan[i].transform.rotation.y);
                                    PlayerPrefs.SetFloat("RotCar" + Caravan[i].GetComponent<Carriage>().getId() + "Z",
Caravan[i].transform.rotation.z);
                                   PlayerPrefs.SetFloat("RotCar" + Caravan[i].GetComponent<Carriage>().getId() + "W",
Caravan[i].transform.rotation.w);

        }



        //guardamos posiciones de caravanas y personaje
        PlayerPrefs.SetFloat("PosCharX", Character.transform.position.x);
        PlayerPrefs.SetFloat("PosCharY", Character.transform.position.y);
        PlayerPrefs.SetFloat("PosCharZ", Character.transform.position.z);



    }

    public void LoadSavePoint()
    {
        //Cargamos los valores de la AI
        PrincessController.loadAI(PlayerPrefs.GetFloat("QMatrix0"), PlayerPrefs.GetFloat("QMatrix1"),
```

```
            PlayerPrefs.GetInt("Iterations0"), PlayerPrefs.GetInt("Iterations1"));
        decisionPrincessTrigger.load(true, PlayerPrefs.GetInt("Decision"));
        ConfigController.phase = PlayerPrefs.GetInt("Phase");
        //Character.transform.position = new Vector3(0, 0, 0);

        //Posicion del personaje
        Character.SetActive(false);
        Character.transform.position = new Vector3(PlayerPrefs.GetFloat("PosCharX"),
        PlayerPrefs.GetFloat("PosCharY"), PlayerPrefs.GetFloat("PosCharZ"));

       // Debug.LogError("PosCharX " + PlayerPrefs.GetFloat("PosCharX"));
       // Debug.LogError("PosCharY " + PlayerPrefs.GetFloat("PosCharY"));
       // Debug.LogError("PosCharZ " + PlayerPrefs.GetFloat("PosCharZ"));
        Character.SetActive(true);

        //Posicion de las caravanas
        for (int i = 0; i < Caravan.Length; i++)
        {
            Caravan[i].transform.position = new Vector3(PlayerPrefs.GetFloat("PosCar" + i + "X"),
                PlayerPrefs.GetFloat("PosCar" + i + "Y"), PlayerPrefs.GetFloat("PosCar" + i + "Z"));
            Caravan[i].transform.rotation = new Quaternion(PlayerPrefs.GetFloat("RotCar" + i + "X"),
                                PlayerPrefs.GetFloat("RotCar" + i + "Y"), PlayerPrefs.GetFloat("RotCar" + i + "Z"),
PlayerPrefs.GetFloat("RotCar" + i + "W"));
            int health = PlayerPrefs.GetInt("CarriageHealth" + i);
            Caravan[i].GetComponent<Carriage>().setHealthPoints(health);
           InterfaceController.dealDamageCarriage(i, 100 - health);
            if (health <= 0)
            {
                Caravan[i].SetActive(false);
            }
            else Caravan[i].GetComponent<NavMeshAgent>().enabled = true;



        }
        //Debug.LogError(PlayerPrefs.GetInt("Phase"));
        if (PlayerPrefs.GetInt("Phase") == 2)
        {
            savepoint2.GetComponent<BoxCollider>().enabled = true;
```

```
            savepoint2.GetComponent<SphereCollider>().enabled = false;
        }

    }
}
```

- SceneController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;

public class SceneController : MonoBehaviour {

    public float timeLeft = 22;

        // Update is called once per frame
        void Update ()
    {
        timeLeft -= Time.deltaTime;

        if (timeLeft < 0)
        {
            SceneManager.LoadScene("Menu");
        }
    }
}
```

- Unit

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.SceneManagement;

/*
 *  Hard Trip
 *  Cavitec Games (2017-2018)
 *  Developed by Defu
 *
```

```
 *  En esta clase se modela cualquier unidad o entidad que pueda entrar en combate.
 *  El combate se modela en la función Update de aquí, es lo mejor para evitar problemas de escalabilidad
 *  Los métodos dentro de cada grupo están ordenados alfabeticamente
 *
 */

public abstract class Unit : MonoBehaviour
{
    // Faction types
    public enum Faction
    {
        FACTION_FRIENDLY = 0,          // Tomamos siempre como referencia el jugador, por lo que un ejemplo de
friendly sería la caravan
        FACTION_ENEMY,                 // Enemigos agresivos, bandidos o lobos
        FACTION_NEUTRAL,               // Los bandidos negociadores serían neutrales en un principio.
    }

    // Creature type, puede ser útil para los looteos
    public enum CreatureType
    {
        CREATURE_TYPE_BEAST,           // Lobos
        CRETAURE_TYPE_HUMANOID,        // Bandidos
        CREATURE_TYPE_MISC             // Caravana
    }

    // Diferentes estados durante el combate
    public enum CombatStatus
    {
        COMBAT_STATUS_IDLE = 0,
        COMBAT_STATUS_ATTACKING,
        COMBAT_STATUS_ATTACK_READY,
        COMBAT_STATUS_WAITING_FOR_ATTACK,
        COMBAT_STATUS_DEFENDING,
        COMBAT_STATUS_STUNNED,
        COMBAT_STATUS_DEAD,
        COMBAT_STATUS_CORPSE
    }

    // General characteristics
    protected Faction faction;
    protected int id;
    protected float movementSpeed;
    protected float scale;
    protected CreatureType type;

    // Combat template characterisitcs
    public int attackDamage;           // Daño base de nuestros atques
    public int healthPoints;           // Nuestros puntos de vida
    protected int armor;               // Reducción de daño al recibir un golpe

    // Current Combat characteristics
    protected CombatStatus combatStatus;
    protected int corpseDuration;
    protected float combatAttackDelay;       // Este valor cambia durante el combate, al llegar a cero se ataca y se
resetea al valor de plantilla
    protected float combatAttackWindow;   // Tiempo que dura el ataque
    protected ArrayList unitsInAggroRange;
    protected ArrayList unitsInCombatWith = new ArrayList();
```

```
protected ArrayList unitsInAttackRange;
protected bool inCombat;
protected Unit target;
public float stunnedTime = 0;

// Entrenamiento de la AI
public float combatTotalTime = 0;
public float lastTotalCombatTime = 0;
public bool startCountingTime = false;

public int damageDoneToPlayer = 0;
public int damageDoneToCarriage = 0;
public int damageRecevied = 0;

// Misc
protected bool player;
protected int updateFrames;

// Cosas rápidas para el boss
protected Vector3 startingPosition;
protected Quaternion startingRotation;

// Enhanced Combat
public float damageMultiplier = 1;
public float attackSpeedMultiplier = 1;
public float reducingStacksTime = 2f;

// Getters
public int getArmor()
{
    return armor;
}

// Con este método cogemos el tiempo entre ataques, definido en la clase hija
protected abstract float getAttackDelay();

protected abstract float getCombatAttackWindow();

public int getId()
{
    return id;
}

protected abstract InterfaceController GetInterfaceCotroller();

public Unit getClosestUnitInAggroRange()
{
    // Si está vacío devolvemos null
    if (unitsInAggroRange.Count == 0)
        return null;

    return (Unit)unitsInAggroRange[0];
}

public CombatStatus getCombatStatus()
{
    return combatStatus;
}
```

```
public Faction getFaction()
{
    return faction;
}

public int getHealthPoints()
{
    return healthPoints;
}

public Unit getTarget()
{
    return target;
}

public ArrayList getUnitsInAttackRange()
{
    return unitsInAttackRange;
}

public ArrayList getUnitsInAggroRange()
{
    return unitsInAggroRange;
}

// Setters

public void setCombatStatus(CombatStatus combatStatus)
{
    this.combatStatus = combatStatus;
}

public void setHealthPoints(int health)
{
    this.healthPoints = health;
}

public void setScale(float scale_f)
{
    this.scale = scale_f;
    float scale_x = gameObject.transform.localScale.x * scale;
    float scale_y = gameObject.transform.localScale.y * scale;
    float scale_z = gameObject.transform.localScale.z * scale;
    gameObject.transform.localScale.Set(scale_x, scale_y, scale_z);
}

public void setTarget(Unit target)
{
    this.target = target;
}

// Combat

public void addUnitInAttackRange(Unit victim)
{
    unitsInAttackRange.Add(victim);
}
```

```
public void addUnitInAggroRange(Unit victim)
{
    unitsInAggroRange.Add(victim);
}

public void attackUnit(Unit victim)
{
    //Debug.Log(gameObject.name + " Ataca a " + victim.gameObject.name);

    // Si no se está defencdiendo, le dañamos
    if (victim.getCombatStatus() != Unit.CombatStatus.COMBAT_STATUS_DEFENDING)
        dealDamageToUnit(victim);

    // Aumentamos el multiplicador de daño
    if (this is Player && ConfigController.enhancedCombatMode)
    {
        damageMultiplier += 0.1f;
        attackSpeedMultiplier -= 0.1f;
    }
    playAttackSound();
}

public void chaseUnit(Unit victim)
{
    if (player || !victim)
        return;

    ////Debug.Log(name + ": Persiguiendo a " + victim.name);

    setCurrentAnimation("run");

    NavMeshAgent nav = gameObject.GetComponentInParent<NavMeshAgent>();

    nav.enabled = true;
    nav.SetDestination(victim.gameObject.transform.position);
}

public void dealDamageToUnit(Unit victim)
{
    if(victim.getCombatStatus() == CombatStatus.COMBAT_STATUS_DEAD)
    {
        //Debug.Log("Unidad muerta: " + victim.name + "No se le puede dañar");
        return;
    }

    int dmg = Mathf.RoundToInt(attackDamage * damageMultiplier) - victim.getArmor();

    //Debug.Log("El daño del ataque es" + dmg);

    if (victim.reduceHealthPointsBy(dmg) <= 0)
        victim.setCombatStatus(Unit.CombatStatus.COMBAT_STATUS_DEAD);

    if (victim.isPlayer())
    {
        victim.GetInterfaceCotroller().dealDamageCharacter(dmg);
        damageDoneToPlayer += dmg;
```

```
            Debug.Log("Reducioendo daños");
            victim.damageMultiplier -= (victim.damageMultiplier - 1)/2;
            victim.attackSpeedMultiplier *= 2;
            if (victim.damageMultiplier < 1)
                victim.damageMultiplier = 1;
            if (victim.attackSpeedMultiplier > 1)
                victim.attackSpeedMultiplier = 1;
        }
        else if(victim is Carriage)
        {
            //Debug.Log("Funcion Alex de caravanas, el objetivo es " + victim.getId());
            victim.GetInterfaceCotroller().dealDamageCarriage(victim.getId(), dmg);
        }
        else if(victim is Wolf && ((Wolf)victim).isBoss)
        {
            //Debug.Log("Bajando la vida al Boss " + victim.getId());
            victim.GetInterfaceCotroller().dealDamageToBoss(dmg);
        }

        //Debug.Log("La salud restante de " + victim.gameObject.name + " es " + victim.getHealthPoints());
    }

    public void enterInCombatWith(Unit victim)
    {
       // if (type == CreatureType.CREATURE_TYPE_MISC)
         //   GetComponent<NavMeshAgent>().enabled = false;

        if (unitsInCombatWith.Contains(victim))
        {
            //Debug.Log(gameObject.name + ": Ya estoy en combate con: " + victim.name) ;
            return;
        }

        if(!inCombat && !player)
        {
            //Debug.Log(gameObject.name + ": Mi nuevo target es " + victim.name);
            setTarget(victim);
        }
        else if(isPlayer() && target == null)
        {
            setTarget(victim);
            if (victim.gameObject.transform.Find("TargetIndicator"))
                victim.gameObject.transform.Find("TargetIndicator").gameObject.SetActive(true);
        }

        inCombat = true;
        startCountingTime = true;
        unitsInCombatWith.Add(victim);
        //Debug.Log(gameObject.name + ": Acabo de entrar en combate, preparando mi ataque");
        setCombatStatus(CombatStatus.COMBAT_STATUS_WAITING_FOR_ATTACK);
    }

    public void exitCombatWith(Unit victim)
    {
        unitsInCombatWith.Remove(victim);
        //Debug.Log(gameObject.name + ": Acabo de salir de combate con " + victim.name);

        // No quedan más enemigos contra los que combatir, salimos de combate
```

```
    if (unitsInCombatWith.Count == 0)
        finishCombat();
}

public void finishCombat()
{
    // Reseteamos valores importantes
    inCombat = false;
    target = null;
    reducingStacksTime = 2f;


   // if(type == CreatureType.CREATURE_TYPE_MISC)
     //   GetComponent<NavMeshAgent>().enabled = true;

    if(type != CreatureType.CREATURE_TYPE_MISC)
        stopChasing();

    //Debug.Log(gameObject.name + ": Salgo de combate, esperando a más enemigos");

    if(combatStatus != CombatStatus.COMBAT_STATUS_CORPSE)
        setCombatStatus(CombatStatus.COMBAT_STATUS_IDLE);

    setCurrentAnimation("idle");

    if(player)
        resetHealthPoints();

}

public void handleDeath()
{
    if (player)
        SceneManager.LoadScene("GameOver");
    else if(this is Carriage && id == 1)
        SceneManager.LoadScene("GameOver");

    if (this is Wolf && ((Wolf)this).isBoss)
    {
        ((Wolf)this).GetDialogController().trueEndGameDialog();
        GameObject.Find("VictorySound").GetComponent<AudioSource>().enabled = true;
        AudioSource[] audios = ((Wolf)this).GetComponents<AudioSource>();

        foreach (AudioSource audio in audios)
            audio.enabled = false;
    }

    startCountingTime = false;
    lastTotalCombatTime += combatTotalTime;
    combatTotalTime = 0;

    //finishCombat();
    combatStatus = CombatStatus.COMBAT_STATUS_CORPSE;
    setCurrentAnimation("dead");

    foreach (Unit unit in unitsInCombatWith)
        unit.exitCombatWith(this);
```

```
        // Desactivamos el icono de target
        if (gameObject.transform.Find("TargetIndicator"))
            gameObject.transform.Find("TargetIndicator").gameObject.SetActive(false);

        // Updateamos la matriz de la AI
        if (this is Carriage)
            return;

        string enemytype = "";
        if (this is Wolf)
            enemytype = "wolf";
        else if(this is Bandit)
            enemytype = "agressiveBandit";
                PrincessController.updateQMatrix(attackDamage, attackDamage, attackDamage, combatAttackDelay + combatAttackWindow, 0.85f, lastTotalCombatTime, damageDoneToPlayer, damageDoneToCarriage, damageRecevied, enemytype);
    }

    public void handleLosingTarget()
    {
        // Si seguimos en combate tras perder el target, elegimos otro nuevo
        if (inCombat)
            setTarget(getClosestUnitInAggroRange());
    }

    public void handleUnitDeath(Unit victim)
    {
        // Eliminamos a esa unidad de todas nuestras listas
        exitCombatWith(victim);
        removeUnitInAggroRange(victim);
        removeUnitInAttackRange(victim);
    }

    public bool isInAttackRangeWith(Unit unit)
    {
        if (unitsInAttackRange.Contains(unit))
            return true;
        return false;
    }

    public int reduceHealthPointsBy(int damage)
    {
       return healthPoints -= damage;
    }

    public void reduceCombatAttackDelayBy(int frames)
    {
        //combatAttackDelay = combatAttackDelay -  frames;
        combatAttackDelay -= Time.deltaTime;
    }

    public void removeUnitInAttackRange(Unit victim)
    {
        unitsInAttackRange.Remove(victim);
    }

    public void removeUnitInAggroRange(Unit victim)
    {
```

```
    unitsInAggroRange.Remove(victim);
}

public void resetCombatAttackDelay()
{
    combatAttackDelay = getAttackDelay() + ( Random.value - 0.5f);
    //Debug.Log("Delay: " + combatAttackDelay);
    if (isPlayer())
        combatAttackDelay = getAttackDelay() * attackSpeedMultiplier;
}

public void resetAttackWindow()
{
    combatAttackWindow = getCombatAttackWindow();
}

public void resetHealthPoints()
{
    healthPoints = 100;
    //Debug.Log(name + ":Resetando vida de ");
    GetInterfaceCotroller().resetLife();
}

public void stopChasing()
{
    if (player)
        return;

    NavMeshAgent nav = gameObject.GetComponentInParent<NavMeshAgent>();

    if (nav.enabled)
    {
        // Deshabilitamos el movimiento
        ////Debug.Log("Parando la persecución");
        nav.enabled = false;
        setCurrentAnimation("idle");
    }
    else if(target)
    {
        // Miramos a nuestro objetivo
        Vector3 posicionRelativa = target.gameObject.transform.position - gameObject.transform.position;
        Quaternion rotacion = Quaternion.LookRotation(posicionRelativa);
        transform.SetPositionAndRotation(transform.position, rotacion);
    }

}

public void updateAttackStatus(int framesOffset)
{
    switch(combatStatus)
    {
        case CombatStatus.COMBAT_STATUS_ATTACKING:

            ////Debug.Log(gameObject.name + ": En ventana de ataque!");
            setCurrentAnimation("attack");

            //combatAttackWindow -= framesOffset;
            combatAttackWindow -= Time.deltaTime;
```

```
            ////Debug.Log("El combatAttackWindows es " + combatAttackWindow);

            if (combatAttackWindow <= 0)
            {
                attackUnit(target);

                resetCombatAttackDelay();
                resetAttackWindow();
                combatStatus = CombatStatus.COMBAT_STATUS_WAITING_FOR_ATTACK;
            }

            break;

        case CombatStatus.COMBAT_STATUS_WAITING_FOR_ATTACK:

            setCurrentAnimation("idle");
            reduceCombatAttackDelayBy(framesOffset);

                ////Debug.Log(gameObject.name + ": Mi timer se reduce en " + framesOffset + ", me quedan " +
combatAttackDelay + " para poder volver a atacar");

            // Si es menor o igual que 0, nuestro ataque está listo. Reseteamos el timer
            if (combatAttackDelay <= 0)
            {
                combatStatus = CombatStatus.COMBAT_STATUS_ATTACK_READY;
                //Debug.Log(gameObject.name + ": Mi siguiente ataque está listo!");
            }

            break;

        case CombatStatus.COMBAT_STATUS_ATTACK_READY:

            break;

        case CombatStatus.COMBAT_STATUS_STUNNED:
            setCurrentAnimation("damage");
            stunnedTime -= Time.deltaTime;
            if (stunnedTime < 0)
                setCombatStatus(CombatStatus.COMBAT_STATUS_WAITING_FOR_ATTACK);
            break;

        default:
            break;
    }
}

public void updateMultipliers()
{
    reducingStacksTime -= Time.deltaTime;

    if(reducingStacksTime < 0)
    {
        damageMultiplier -= 0.1f;
        attackSpeedMultiplier += 0.1f;
        if (damageMultiplier < 1)
            damageMultiplier = 1;
        if (attackSpeedMultiplier > 1)
```

```
            attackSpeedMultiplier = 1;
        reducingStacksTime = 2f;
    }
}

// Misc

public bool isPlayer()
{
    return player;
}

public bool isInCombat()
{
    return inCombat;
}

public void hardcoreFinishCombat()
{
    target = null;
    unitsInAttackRange = new ArrayList();
    unitsInCombatWith = new ArrayList();
    inCombat = false;
    combatStatus = CombatStatus.COMBAT_STATUS_IDLE;
    //Debug.Log(name + ": Abandonando combate por fuerza bruta");
}

// Handling animations
public void setCurrentAnimation(string animation)
{
    Animator animator = gameObject.GetComponent<Animator>();
    bool exists = false;

    if (animator)
    {
        // cancelamos todas las demás
        foreach (AnimatorControllerParameter parameter in animator.parameters)
        {
            animator.SetBool(parameter.name, false);
            if (parameter.name == animation)
                exists = true;
        }
        ////Debug.Log(name + ": Poniendo la animación " + animation);
        if(exists)
            animator.SetBool(animation, true);
    }

}

public bool hasAnimationFinished(string animation)
{
    Animator animator = gameObject.GetComponent<Animator>();

    if (!animator)
        return true;


    if (animator.GetCurrentAnimatorStateInfo(0).IsName(animation))
```

```
        return false;

    return true;
}

// Handling Sounds
public void playAttackSound()
{
    if (isPlayer())
    {
        GameObject.Find("CombatSound").GetComponent<AudioSource>().enabled = false;
        GameObject.Find("CombatSound").GetComponent<AudioSource>().enabled = true;
    }
    else if(this is Bandit)
    {
        if (!GetComponent<AudioSource>())
            return;
        GetComponent<AudioSource>().mute = false;
        GetComponent<AudioSource>().enabled = false;
        GetComponent<AudioSource>().enabled = true;
    }else if (this is Wolf)
    {
        if (!GetComponent<AudioSource>())
            return;

        GetComponent<AudioSource>().mute = false;
        GetComponent<AudioSource>().enabled = false;
        GetComponent<AudioSource>().enabled = true;
    }
}

// Funcion principl
public void Update()
{
    updateFrames++;
    if (updateFrames == ConfigController.combatUpdateFrames)
    {
    OldupdateUnit(updateFrames);
    updateFrames = 0;
    }
}

public void OldupdateUnit(int frameOffset)
{

    // Si estamos muertos, desaparecemos
    if (combatStatus == CombatStatus.COMBAT_STATUS_DEAD)
    {
        //Debug.Log(this.gameObject.name + "Ha muerto");
        // Animación de muerte y tras X tiempo desaparecer
        handleDeath();
        return;
    }

    // Manejamos el tiempo de los cadáveres
    if(combatStatus == CombatStatus.COMBAT_STATUS_CORPSE)
    {
        corpseDuration-= frameOffset;
```

```
            if (corpseDuration <= 0)
            {
                gameObject.SetActive(false);
                //Debug.Log("Se ha desactivado + " + name);
            }

            return;
        }

        // Si somos la caravana, no hacemos nada más
        if (type == CreatureType.CREATURE_TYPE_MISC)
            return;



        // Si estamos en combate, comprobamos como va
        if(inCombat)
        {
            // Actualizamos nuestro timer
            updateAttackStatus(frameOffset);
            combatTotalTime += Time.deltaTime;

            if (!target && isPlayer())
            {
                setTarget((Unit)unitsInCombatWith[0]);
                if (target.gameObject.transform.Find("TargetIndicator"))
                    target.gameObject.transform.Find("TargetIndicator").gameObject.SetActive(true);
                return;
            }

            if (getHealthPoints() < 0)
                combatStatus = CombatStatus.COMBAT_STATUS_DEAD;

            if (combatStatus == CombatStatus.COMBAT_STATUS_ATTACKING)
                return;

            // Si nuestro target está muerto, cambiamos al más cercano
            if (target.getCombatStatus() == CombatStatus.COMBAT_STATUS_DEAD ||
                target.getCombatStatus() == CombatStatus.COMBAT_STATUS_CORPSE)
            {
                handleUnitDeath(target);
                handleLosingTarget();
            }

            // Si estamos dentro del rango nos detenemos
            if (isInAttackRangeWith(target))
            {
                if (!isPlayer())
                    stopChasing();

                // Si nuestro ataque está listo, atacamos
                if (combatStatus == CombatStatus.COMBAT_STATUS_ATTACK_READY && !player)
                    combatStatus = CombatStatus.COMBAT_STATUS_ATTACKING;
            }
            // Si no lo estamos, perseguimos al target
                else if (!isPlayer() && getCombatStatus() != CombatStatus.COMBAT_STATUS_STUNNED/*&& type !=
CreatureType.CREATURE_TYPE_MISC*/)
                chaseUnit(target);
```

```
        }
        else if (player)
            updateMultipliers();
    }

    public void updateUnit(int frameOffset)
    {
        // Si estamos muertos, desaparecemos
        if (combatStatus == CombatStatus.COMBAT_STATUS_DEAD)
        {
            //Debug.Log(this.gameObject.name + "Ha muerto");
            // Animación de muerte y tras X tiempo desaparecer
            gameObject.SetActive(false);
            return;
        }









    }
}
```

- Bandit

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Bandit : Unit
{

    // Combat template characterisitcss
    private const float attackDelay = 1.25f;
    public float attackWindow  = 2f;

    // Incializamos el bandido
    void Start()
    {
        // General characteristics
        id = 1;
        type = CreatureType.CRETAURE_TYPE_HUMANOID;
        faction = Faction.FACTION_ENEMY;
        movementSpeed = ConfigController.banditAgressiveSpeed;
```

```
        scale = 8;
        setScale(scale);

        // Combat template characterisitcs
        attackDamage = 15;
        healthPoints = ConfigController.banditAgressiveMaxLife;
        armor = 5;
        attackWindow = 1f;
        //Debug.LogWarning("La duración en frames de la aniamción es " + attackWindow);

        // Current CombatStatus
        combatStatus = CombatStatus.COMBAT_STATUS_IDLE;
        combatAttackDelay = attackDelay;
        combatAttackWindow = (float)attackWindow;
        unitsInAggroRange = new ArrayList();
        unitsInAttackRange = new ArrayList();
        unitsInCombatWith = new ArrayList();
        target = null;
        inCombat = false;

        // Misc
        player = false;
        setAccelerationAndSpeed(30, 25);
        corpseDuration = 1000;

    }

    // Getters
    protected override float getAttackDelay()
    {
        return attackDelay;
    }

    protected override float getCombatAttackWindow()
    {
        return attackWindow;
    }
```

```
    protected override InterfaceController GetInterfaceCotroller()
    {
        return null;
    }

    // Movement Speed
    protected void setAccelerationAndSpeed(float acceleration, float speed)
    {
        NavMeshAgent nav = GetComponent<NavMeshAgent>();

        if (!nav)
            return;

        nav.acceleration = acceleration;
        nav.speed = speed;
    }
}
```

- Carriage

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Carriage : Unit
{

    // Combat template characterisitcss
    private const int attackDelay = 800;
    private const int attackWindow = 30;
    public int carriageId = 0;
    public Unit mytarget;
    public InterfaceController InterfaceController;

    // Use this for initialization
    void Awake()
    {
        // General characteristics
```

```
    id = carriageId;
    //Debug.Log("El id de esta caravana es: " + id);
    type = CreatureType.CREATURE_TYPE_MISC;
    faction = Faction.FACTION_FRIENDLY;
    movementSpeed = ConfigController.carriageSpeed;
    scale = 8;
    //setScale();

    // Combat template characterisitcs
    attackDamage = 0;
    healthPoints = ConfigController.carriageMaxLife;
    armor = 5;

    // Current CombatStatus
    combatStatus = CombatStatus.COMBAT_STATUS_IDLE;
    combatAttackDelay = attackDelay;
    unitsInAggroRange = new ArrayList();
    unitsInAttackRange = new ArrayList();
    unitsInCombatWith = new ArrayList();
    target = null;
    inCombat = false;

    // Misc
    player = false;
}

private void FixedUpdate()
{
    mytarget = getTarget();
}

// Getters

protected override float getAttackDelay()
{
    return attackDelay;
}
```

```
    protected override float getCombatAttackWindow()
    {
        return attackWindow;
    }

    protected override InterfaceController GetInterfaceCotroller()
    {
        return InterfaceController;
    }
}
```

- Wolf

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Wolf : Unit {

    // Combat template characterisitcss
    private float attackDelay = 1.25f;
    public float attackWindow = 0.75f;
    public bool isAplha = false;
    public bool isBoss = false;
    public InterfaceController interfaceController = null;
    public DialogController dialogController = null;

    // Incializamos el lobo
    void Start ()
    {
        // General characteristics
        id = 1;
        type = CreatureType.CREATURE_TYPE_BEAST;
        faction = Faction.FACTION_ENEMY;
        movementSpeed = ConfigController.wolfSpeed;
        scale = 8;
        setScale(scale);
```

```
        // Combat template characterisitcs
        attackDamage = 15;
        healthPoints = 100;
        armor = 5;
        if(isAplha)
        {
            attackDamage = 25;
            armor = 15;
        }
        else if (isBoss)
        {
            attackDamage = 30;
            healthPoints = 1000;
            armor = 15;
            attackDelay = 1.5f;
        }
        attackWindow = 0.75f;

        // Current CombatStatus
        combatStatus = CombatStatus.COMBAT_STATUS_IDLE;
        combatAttackDelay = attackDelay;
        combatAttackWindow = attackWindow;
        unitsInAggroRange = new ArrayList();
        unitsInAttackRange = new ArrayList();
        unitsInCombatWith = new ArrayList();
        target = null;
        inCombat = false;

        // Misc
        player = false;
        corpseDuration = 1000;
    }

    // Getters
    protected override float getAttackDelay()
    {
        return attackDelay;
    }
```

```
    protected override float getCombatAttackWindow()
    {
        return attackWindow;
    }

    protected override InterfaceController GetInterfaceCotroller()
    {
        if(!isBoss)
            return null;
        return interfaceController;
    }

    public DialogController GetDialogController()
    {
        if (!isBoss)
            return null;
        return dialogController;
    }
}
```

- Player

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : Unit {

    // Combat template characterisitcss
    private const float attackDelay = 0.35f;
    public float attackWindow = 0.5f;
    public int currentTargetIndex = 0;
    public InterfaceController InterfaceController;
    int swapTarget = 0;

    // Use this for initialization
```

```
void Start()
{
    // General characteristics
    id = 1;
    type = CreatureType.CRETAURE_TYPE_HUMANOID;
    faction = Faction.FACTION_FRIENDLY;
    movementSpeed = 2;
    scale = 8;
    setScale(scale);

    // Combat template characterisitcs
    attackDamage = 60;
    healthPoints = 100;
    armor = 5;
    attackWindow = 0.5f;

    // Current CombatStatus
    combatStatus = CombatStatus.COMBAT_STATUS_IDLE;
    combatAttackDelay = attackDelay;
    combatAttackWindow = attackWindow;
    unitsInAggroRange = new ArrayList();
    unitsInAttackRange = new ArrayList();
    unitsInCombatWith = new ArrayList();
    target = null;

    // Misc
    player = true;
    InvokeRepeating("checkPlayerDefend", 0f, 0.002f);
    InvokeRepeating("playerSwapTargets", 0f, 0.002f);
    InvokeRepeating("checkPlayerAttack", 0f, 0.002f);
}

// Getters
protected override float getAttackDelay()
{
    return attackDelay;
}
```

```
protected override float getCombatAttackWindow()
{
    return attackWindow;
}
// Misc

protected void checkPlayerAttack()
{
    if (!inCombat)
        return;

    if (!Input.GetKeyDown(KeyCode.Mouse0))
        return;

    if (combatStatus != CombatStatus.COMBAT_STATUS_ATTACK_READY)
        return;

    if (!getTarget())
        return;

    if (!isInAttackRangeWith(getTarget()))
        return;

    combatStatus = CombatStatus.COMBAT_STATUS_ATTACKING;
}

protected void checkPlayerDefend()
{
    if (!inCombat)
        return;

    GameObject shield = gameObject.transform.Find("Shield").gameObject;

    if (!shield)
        return;

    if (Input.GetKey(KeyCode.Mouse1))
    {
```

```
        if (combatStatus != CombatStatus.COMBAT_STATUS_ATTACKING)
        {
            combatStatus = CombatStatus.COMBAT_STATUS_DEFENDING;
            shield.SetActive(true);
            //Debug.Log.Log("Player is defending!");

            if (GameObject.Find("ShieldSound").GetComponent<AudioSource>())
            {
                GameObject.Find("ShieldSound").GetComponent<AudioSource>().enabled = true;
            }

        }

        return;
    }

    if (Input.GetKeyUp(KeyCode.Mouse1))
    {
        combatStatus = CombatStatus.COMBAT_STATUS_WAITING_FOR_ATTACK;
        shield.SetActive(false);
        GameObject.Find("ShieldSound").GetComponent<AudioSource>().enabled = false;
        //Debug.Log.Log("Player is not defending anymore!");
        return;
    }

}

protected void playerSwapTargets()
{
    //Debug.Log.LogWarning("Buscando targets");
    if (!inCombat)
        return;

    // No hay más targets posibles
    if (unitsInCombatWith.Count == 1)
        return;
```

```
if (Input.GetKeyDown(KeyCode.Tab))
    swapTarget = 1;

if (swapTarget == 1)
    if (Input.GetKeyUp(KeyCode.Tab))
        swapTarget = 2;

if (swapTarget != 2)
    return;

//Debug.Log.LogWarning("Hay algo conmigo en combate");
if (getTarget())
{
    //Debug.Log.LogWarning("Quitando el anterior");
    if (getTarget().gameObject.transform.Find("TargetIndicator"))
        getTarget().gameObject.transform.Find("TargetIndicator").gameObject.SetActive(false);
}

// Si es el último, ponemos el primero
if (unitsInCombatWith.IndexOf(getTarget()) == (unitsInCombatWith.Count - 1))
    setTarget((Unit)unitsInCombatWith[0]);
else
    setTarget((Unit)unitsInCombatWith[unitsInCombatWith.IndexOf(getTarget())+1]);

getTarget().gameObject.transform.Find("TargetIndicator").gameObject.SetActive(true);

swapTarget = 0;

/*
//Debug.Log.LogWarning("Poneindo nuevo target");
try
{
    setTarget((Unit)unitsInCombatWith[currentTargetIndex]);
}
catch (ArgumentOutOfRangeException ex)
{
    currentTargetIndex = 0;
    return;
```

```
        }
        // Activamos el icono de target

        currentTargetIndex++;
        // Si es el ultimo de la lista reseteamos el target
        if (currentTargetIndex >= (unitsInCombatWith.Count))
            currentTargetIndex = 0;
            */
    }

    protected override InterfaceController GetInterfaceCotroller()
    {
       return InterfaceController;
    }
}
```

- AggroRange

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AggroRange : MonoBehaviour {

    private void OnTriggerEnter(Collider other)
    {
        Unit colliderOwner = gameObject.GetComponentInParent<Unit>();
        Unit victim = other.gameObject.GetComponent<Unit>();

        // Lo que ha entrado en el área no es una unidad
        if (victim == null)
            return;

        if (colliderOwner.getCombatStatus() == Unit.CombatStatus.COMBAT_STATUS_CORPSE)
            return;

        ////Debug.LogWarning("Entered aggro range " + other.gameObject.name);
```

```
        // Si es el jugador o la caravana...
        if (victim.getFaction() == Unit.Faction.FACTION_FRIENDLY)
        {
            colliderOwner.addUnitInAggroRange(victim);
            colliderOwner.enterInCombatWith(victim);
            victim.enterInCombatWith(colliderOwner);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        Unit colliderOwner = gameObject.GetComponentInParent<Unit>();
        Unit victim = other.gameObject.GetComponent<Unit>();

        // Lo que ha salido del área no es una unidad
        if (victim == null)
            return;

        //Debug.LogWarning("exited aggro range " + other.gameObject.name);

        // Eliminamos a la unidad de la lista
        colliderOwner.removeUnitInAggroRange(victim);

        // Ambas unidades salen de sus mutuos combates
        colliderOwner.exitCombatWith(victim);
        victim.exitCombatWith(colliderOwner);

        // Si el que sale era nuestro target, hay que elegir otro
        if (victim.Equals(colliderOwner.getTarget()))
        {
            //Debug.LogWarning(other.gameObject.name + " ha salido de nuestro área de aggro y era nuestro target");
            colliderOwner.handleLosingTarget();
        }

    }
}
```

- AttackRange

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AttackRange : MonoBehaviour
{

    private void OnTriggerEnter(Collider other)
    {
        Unit colliderOwner = gameObject.GetComponentInParent<Unit>();
        Unit victim = other.gameObject.GetComponent<Unit>();

        if (victim == null)
            return;

        if (victim.getFaction() == Unit.Faction.FACTION_FRIENDLY)
            colliderOwner.addUnitInAttackRange(victim);
        else if (colliderOwner.isPlayer() && victim.getFaction() == Unit.Faction.FACTION_ENEMY)
            colliderOwner.addUnitInAttackRange(victim);

        //Debug.LogWarning(colliderOwner.name + ": Entered attack range " + victim.name);
    }

    private void OnTriggerExit(Collider other)
    {
        Unit colliderOwner = gameObject.GetComponentInParent<Unit>();
        Unit victim = other.gameObject.GetComponent<Unit>();

        colliderOwner.removeUnitInAttackRange(victim);
    }

}
```

- PlayerMovement

```
using UnityEngine;


public class PlayerMovement : MonoBehaviour {



            Vector3 movement;                   // The vector to store the direction of the player's movement.
            Animator anim;                      // Reference to the animator component.
            //Rigidbody playerRigidbody;          // Reference to the player's rigidbody.
            //int floorMask;                      // A layer mask so that a ray can be cast just at gameobjects on the floor layer.
            //float camRayLength = 100f;          // The length of the ray from the camera into the scene.
    public float speed = 35;
    public Camera mainCamera;
    private Vector3 offset;
    private Unit playerUnit;
```

```
bool walking = false;
// Use this for initialization
void Awake ()
        {
    playerUnit = GetComponent<Unit>();
                // Create a layer mask for the floor layer.
                //floorMask = LayerMask.GetMask ("Floor");

                // Set up references.
                //playerRigidbody = GetComponent <Rigidbody> ();
        }

        // Update is called once per frame
        void FixedUpdate ()
        {
    float h = Input.GetAxisRaw("Horizontal");
    float v = Input.GetAxisRaw("Vertical");

    Animating(0,v);
}

private void Update()
{
    walking = false;
    speed = 35;

    if (playerUnit.getCombatStatus() == Unit.CombatStatus.COMBAT_STATUS_ATTACKING)
        return;

    if (ConfigController.firstDialog)
        return;

    if (playerUnit.getCombatStatus() == Unit.CombatStatus.COMBAT_STATUS_DEFENDING)
        speed /= 2;



    if (ConfigController.inCinematic)
```

```
            return;

        if (Input.GetKey(KeyCode.W))
        {
            walking = true;
            transform.position += transform.forward * Time.deltaTime * speed;
        }

        if (Input.GetKey(KeyCode.S))
        {
            walking = true;
            transform.position += -transform.forward * Time.deltaTime * speed;
        }

        if (Input.GetKey(KeyCode.E))
        {
            walking = true;
            transform.position += transform.right * Time.deltaTime * speed;
        }

        if (Input.GetKey(KeyCode.Q))
        {
            walking = true;
            transform.position += -transform.right * Time.deltaTime * speed;
        }

        if (Input.GetKey(KeyCode.A))
        {
            walking = true;
            transform.RotateAround(transform.position, Vector3.up, -200 * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.D))
        {
            walking = true;
            transform.RotateAround(transform.position, Vector3.up, 200 * Time.deltaTime);
        }
    }
```

```
    void Move(float h, float v)
            {
        float x = Mathf.Cos(transform.rotation.eulerAngles.y * 2 * Mathf.PI / 180);
        float z = Mathf.Sin(transform.rotation.eulerAngles.y * 2 * Mathf.PI / 180);

        Debug.Log("El rotation y vale:" + transform.rotation.eulerAngles.y);
        Debug.Log("EL x vale: " + x);
        Debug.Log("EL z vale: " + z);

        movement.Set (x, 0f, z);
        //movement = movement.normalized;
        //playerRigidbody.MovePosition (transform.position + movement);
        transform.position +=  transform.forward  * 3;
    }

            void Animating (float h, float v)
            {
        Unit me = gameObject.GetComponent<Unit>();
        if (walking)
            me.setCurrentAnimation("run");
        if (!walking && me.getCombatStatus() != Unit.CombatStatus.COMBAT_STATUS_ATTACKING)
            me.setCurrentAnimation("idle");
    }
}
```

- Camino

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Camino : MonoBehaviour {

    public int phase = 0;
    public int path = 0;
    public int enemy = 0; // Wolf: 1, BAndit: 2 Híbrido: 3
}
```

- CaravanTargetTrigger

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class CaravanTargetTrigger : MonoBehaviour
{
    public DialogController dialogController;
    public CaravanController caravanController;
    public Transform target;
    public BossController bossController;

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.GetComponent<Carriage>())
            if (gameObject.GetComponent<PhaseTarget>())
                                                    if    (gameObject.GetComponent<PhaseTarget>().phase    ==    3    &&
gameObject.GetComponent<PhaseTarget>().path == 3)
            {
                 dialogController.endGameDialog();
                //bossController.spawnBoss();
                //CinematicController.startCinematic();
                this.gameObject.SetActive(false);
            }

        if (other.gameObject.CompareTag("carriage"))
            caravanController.ChangeTarget(target);
        //this.gameObject.SetActive(false);
    }
}
```

- Clue

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class Clue : MonoBehaviour {

    public int ID;
    public ItemController itemController;
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            itemController.Clue(ID);
            this.gameObject.SetActive(false);
            ConfigController.clueTotal++;
        }
    }
}
```

- DecisionPrincessTrigger

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DecisionPrincessTrigger : MonoBehaviour {

    public CaravanController caravanController;
    public DialogController dialogController;
    public SaveController saveController;
    public Transform targetIzq;
    public Transform targetDcha;
    public string princessText;
    public bool loading;
    public int loadedValue;

    private void OnTriggerEnter(Collider other)
    {

        if (!other.GetComponent<Carriage>())
            return;
```

```
        GameObject[] Caravan = GameObject.FindGameObjectsWithTag("carriage");
        GameObject[] Player = GameObject.FindGameObjectsWithTag("Player");

        Player[0].GetComponent<Player>().hardcoreFinishCombat();

        for (int i = 0; i < Caravan.Length; i++)
        {
            Caravan[i].GetComponent<Carriage>().hardcoreFinishCombat();
        }

        int decision = 0;
        if (!loading)
        {
            //llamamos a la prinsesa
            //System.Random rand = new System.Random();
            Debug.Log("El total de pistas es " + ConfigController.clueTotal);
            decision = PrincessController.makeDecision(ConfigController.phase, ConfigController.clueTotal);
            ConfigController.clueTotal = 0;

                        saveController.SavePoint(ConfigController.phase, decision, PrincessController.Q_Matrix[0],
PrincessController.Q_Matrix[1],
           PrincessController.iterations[0], PrincessController.iterations[1]);

            //ConfigController.phase++;
        }
        else
        {
            decision = loadedValue;
            ConfigController.clueTotal = 0;
        }

        // Desactivamos el flag
        loading = false;
        ConfigController.phase++;
        switch (decision)
        {
            case 1:
                if (targetIzq.GetComponent<PhaseTarget>())
```

```
                    EnemyController.SpawnEnemiesByPath(targetIzq.GetComponent<PhaseTarget>().phase, targetIzq.GetComponent<PhaseTarget>().path);
                caravanController.ChangeTarget(targetIzq);
                princessText = LanguageController.getTextById(4) + "\nQ[0](Lobos)= " + PrincessController.Q_Matrix[0] + "\nQ[1](Bandidos)= " + PrincessController.Q_Matrix[1];
                break;
            default:
                if (targetDcha.GetComponent<PhaseTarget>())
                    EnemyController.SpawnEnemiesByPath(targetDcha.GetComponent<PhaseTarget>().phase, targetDcha.GetComponent<PhaseTarget>().path);
                caravanController.ChangeTarget(targetDcha);
                princessText = LanguageController.getTextById(5) + "\nQ[0](Lobos)= " + PrincessController.Q_Matrix[0] + "\nQ[1](Bandidos)= " + PrincessController.Q_Matrix[1];
                break;
        }

        dialogController.princessDialogPopUp(princessText);
        caravanController.StopCaravan();
        this.gameObject.SetActive(false);
    }

    public void load(bool load, int loadedValue)
    {
        this.loading = load;
        this.loadedValue = loadedValue;

    }
}
```

- DontDestroy

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class DontDestroy : MonoBehaviour
{
```

```
    private void Awake()
    {
        DontDestroyOnLoad(this.gameObject);

    }

    private void Update()
    {
        if (SceneManager.GetActiveScene().name == "Menu" || SceneManager.GetActiveScene().name=="GameOver")
        {
            Destroy(this.gameObject);
        }

    }









}
```

- Flags

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;


public class Flags : MonoBehaviour {
    private static int Lenguage=0;
    private static bool Load=false;


    private void Awake()
    {
        DontDestroyOnLoad(this.gameObject);
    }
```

```
    public static void setLenguage(int ID_lenguage)
    {
        Lenguage = ID_lenguage;
    }
    public static int getLenguage()
    {
        return Lenguage;
    }

    public static void setLoad(bool change)
    {
        Load = change;
    }
    public static bool getLoad()
    {
        return Load;
    }

}
```

- GoAndNavigate

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class GoAndNavigate : MonoBehaviour {

   public Transform Target;
   public NavMeshAgent myNav;

        // Use this for initialization
        void Start ()
    {
        myNav = GetComponent<NavMeshAgent>();
```

```
        }

        // Update is called once per frame
        void Update ()
    {
        if(Target != null)
            if(myNav.enabled)
                myNav.SetDestination(Target.position);

        }
}
```

- IceDamageArea

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class IceDamageArea : MonoBehaviour
{

    // Use this for initialization
    public bool areaActive = false;
    public float timeToActive = 3f;
    public float timeToDamage = 1f;
    public int damage = 10;
    float currentTimeToDamage = 0;

    public ArrayList unitsInside = new ArrayList();

    void Start()
    {
        currentTimeToDamage = timeToDamage;
    }

    // Update is called once per frame
    void Update()
    {
        if (!areaActive)
```

```
    {
        timeToActive -= Time.deltaTime;

        if (timeToActive < 0)
            areaActive = true;
        return;
    }
    else
        currentTimeToDamage -= Time.deltaTime;

    if (currentTimeToDamage < 0)
    {
        foreach (Unit victim in unitsInside)
        {
            Wolf wolf = new Wolf();
            wolf.attackDamage = damage;
            wolf.dealDamageToUnit(victim);
        }
        currentTimeToDamage = timeToDamage;
    }

}

private void OnTriggerStay(Collider other)
{

}

private void OnTriggerEnter(Collider other)
{
    Unit unit = other.GetComponent<Unit>();

    if (unit == null)
        return;

    if (unit.getFaction() == Unit.Faction.FACTION_FRIENDLY)
        unitsInside.Add(unit);
}
```

```
    private void OnTriggerExit(Collider other)
    {
        Unit unit = other.GetComponent<Unit>();

        if (unit == null)
            return;

        if (unit.getFaction() == Unit.Faction.FACTION_FRIENDLY)
            unitsInside.Remove(unit);
    }

}
```

- PhaseTarget

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PhaseTarget : MonoBehaviour {

    public int phase = 0;
    public int path = 0;



}
```

- Rotator

```
using UnityEngine;
using System.Collections;

public class Rotator : MonoBehaviour
{

    void Update()
    {
        transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime);
    }
```

```
}
```

- MenuButton

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MenuButton : MonoBehaviour {

    public Text TextLanguage;
    private void Start()
    {
        Flags.setLenguage(0);
        Flags.setLoad(false);
    }
    public void NewGame(string level)
    {
        Debug.LogWarning("Change to level 1");
        SceneManager.LoadScene(level);
    }

    public void ExitGame()
    {
        Debug.LogWarning("Exiting");
        Application.Quit();
    }
    public void LoadGame()
    {
        Flags.setLoad(true);
        SceneManager.LoadScene("Alpha");
    }
    public void Language()
    {
        if (Flags.getLenguage() == 1)
        {
            Flags.setLenguage(0);
```

```
            TextLanguage.text = "Español";
            Debug.LogError(Flags.getLenguage());

        }
        else
        {
            Flags.setLenguage(1);
            TextLanguage.text = "Inglés";
            Debug.LogError(Flags.getLenguage());
        }
    }
}
```

- DontStopMusic

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class DontStopMusic : MonoBehaviour
{

            // Use this for initialization
            void Awake ()
    {
        DontDestroyOnLoad(this.gameObject);
            }

    private void Update()
    {
        if (SceneManager.GetActiveScene().name == "Alpha" || SceneManager.GetActiveScene().name == "IntroVideo")
            gameObject.SetActive(false);
    }
}
```

- ControlsContinue

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.SceneManagement;

public class ControlsContinue : MonoBehaviour
{

        // Update is called once per frame
        void Update ()
    {

      if (Input.GetKeyDown(KeyCode.E))
          SceneManager.LoadScene("Alpha");

        }
}
```