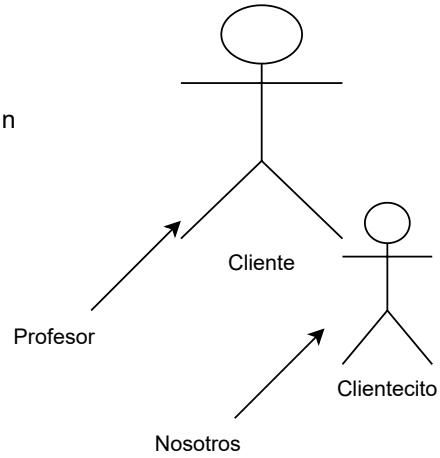
1. Entendimiento del problema

1.1. Definición del cliente

El cliente en esta situación es el profesor, quien pide que se haga un juego el uso de ciertas tecnologías. Sin embargo uno de sus requerimientos, y el más importante, es que seamos nosotros quienes ideemos, diseñemos y hagamos el propio juego. De modo que tenemos dos clientes, el profesor como cliente principal, y nosotros mismos como clientes en un grado inferior pero que a fines practicos se tratará de la misma manera en el contexto del desarrollo del juego base.



1.2. Visión acordada del proyecto

Se va a hacer un videojuego de peleas original, basado en sencillez de mecanica y profundidad de diseño. Usará dos tipos de ataque, un ataque especial y una forma de eludir los ataques, sin forma de moverse por el escenario y usando solo 3 o 4 botones. Debe Tener una base de datos que guarde información del jugador, estadisticas de partidas, etc. Debe ser algo sencillo y de fácil desarrollo pero con la intención de ser llamativo.

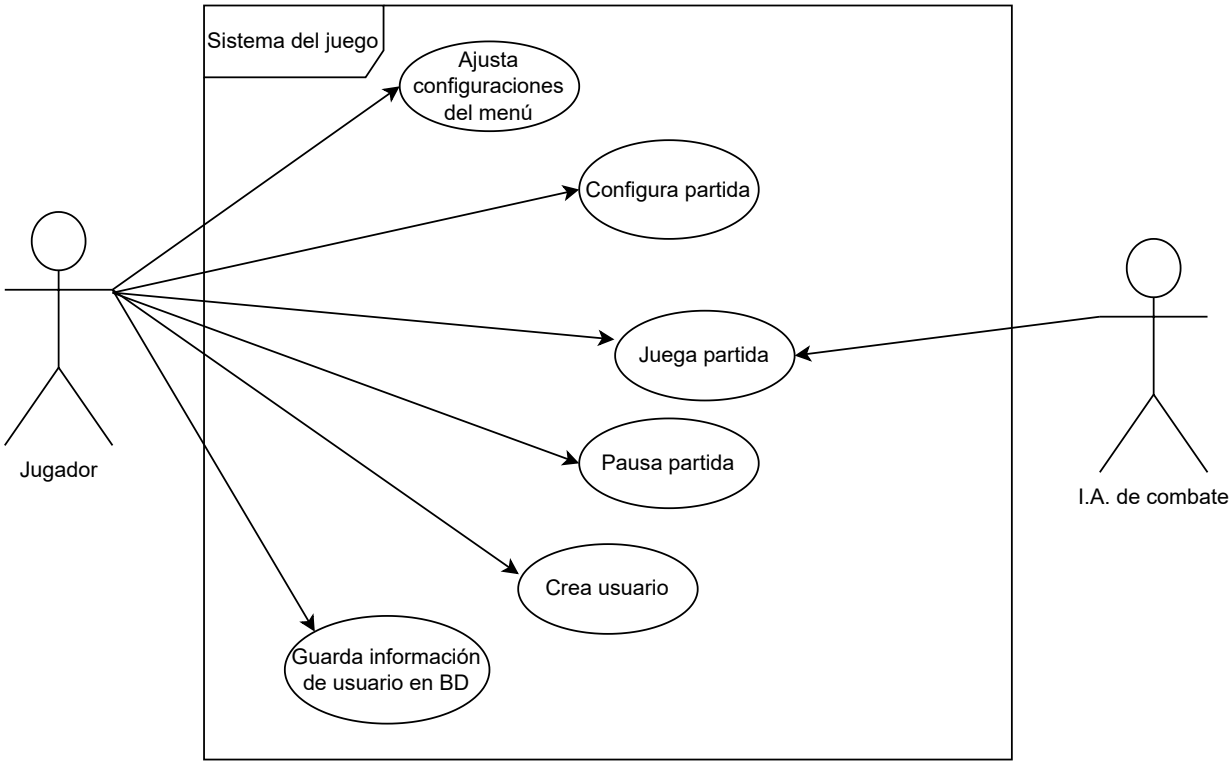
Opcionalmente, si nuestras capacidades nos lo permite, se propuso agregar la posibilidad de crear personajes jugables usando imagenes del jugador por medio de alguna IA. Esta idea está practicamente descartada.

Inicialmente propuesto en Godot o Pygames. Resuelto para hacerse en Godot.

Lista de features

- Al iniciar el juego se presenta un menú donde ajustar sistemas básicos como el sonido, dificultad (Si se implementa), iniciar partida, y la creación de jugador por IA (si se implementa).
- La partida inicia con dos personajes jugables en un escenario, cada uno con un set de movimientos, una barra de vida y un reloj.
- Los personajes deben poder tener un ataque debil, un ataque fuerte, un ataque especial, y una forma de esquivar el ataque enemigo. Considerando la idea de un amague ataquedebil->esquivo.
- Los tipos de ataques deben tener diferencias visuales, sonoras y de tiempo claras que deje claro cual es qué tipo de ataque.
- Los ataques deben ser telegrafiados, predecibles con pistas visuales y de sonido de modo que puedan ser esquivadas por jugadores con habilidad.
- Todo ataque debe ser esquivable en una ventana de tiempo especifica.
- Esquivar debe tener una desventaja, cooldown entre esquivos?.
- Condición de victoria: Tiempo o Agotar la vida del rival.
- Barra especial que cargue el ataque especial según una cantidad de golpes consecutivos?
- Un ataque especial por cada personaje: una combinación automatica de ataques debiles y fuertes. Debe tener una animación y/o efecto visual que indique lo impresionante del ataque.
- Los ataques atenuan el daño si se contrarrestan con otro ataque.
- Debe tener un escenario y música que aviven artisticamente al juego.
- Un sistema de sonido que permita a los personajes hacer ruido según cada acción que hagan (ataques, victoria, derrota, esquivar, etc.)
- Los personajes al iniciar un movimiento quedan en un estado del que no pueden salir hasta que terminen el movimiento, esto para evitar concatenación de movimientos muy rapidamente.
- Los ataques deben tener una forma de saber cuando fueron efectuados, de modo que se sepa qué ataque fue efectuado primero en un caso donde ambos se ataquen a la vez.
- Sistema de rounds: cada partida decidirá al ganador al mejor de 3 rounds o al mejor de 5 rounds.
- La entrada del usuario será por botones en pantalla o teclas del teclado. Soporte para controles?

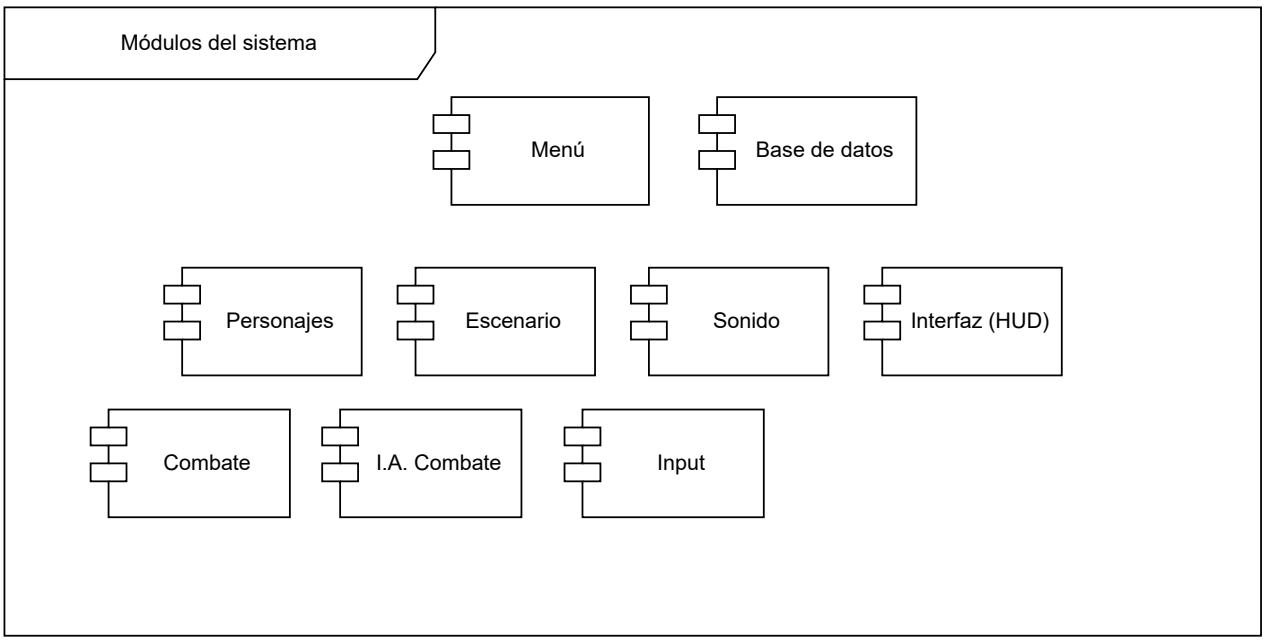
2. Diagrama de Use Case

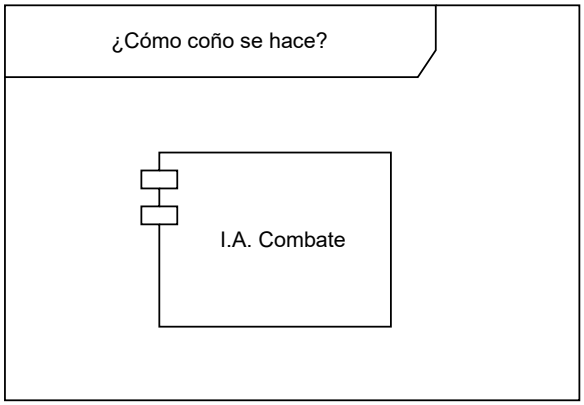


Los actores que actúan en el sistema son el jugador, y las acciones que puede realizar este son las comprendidas acá. Y la I.A., la cual en cierta forma es parte del sistema pero interactua con la partida del mismo modo que el usuario.

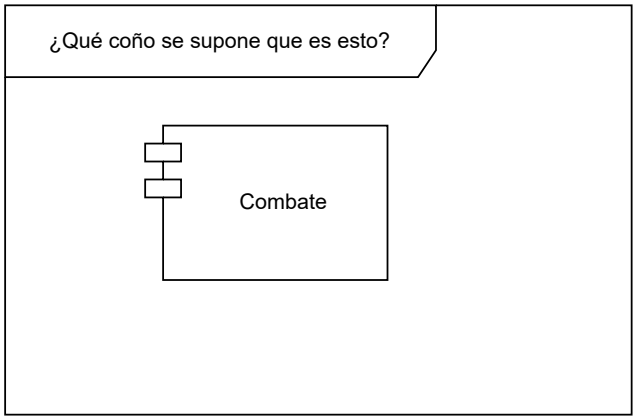
El sistema se puede partir en los siguientes modulos individuales.
Módulos como el menú, base de datos e interfaz no son esenciales para el proyecto y dependen del funcionamiento de otros modulos.

Los tres modulos cuya importancia dicta por donde se empezará el desarrollo están en la siguiente página, junto a la razón de su importancia.

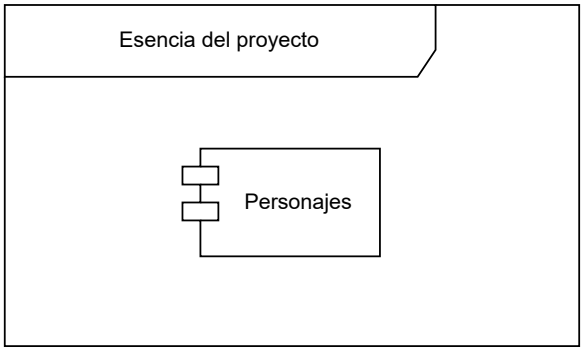




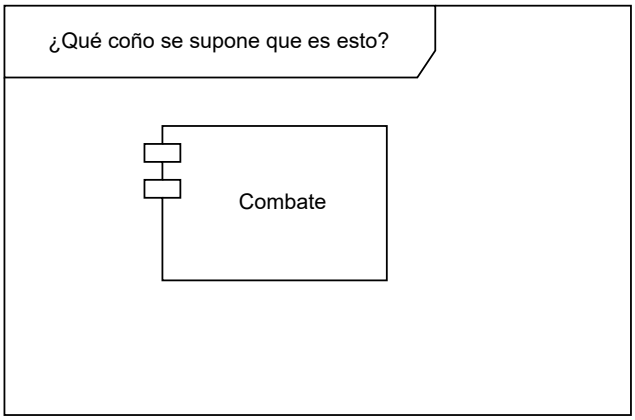
Riesgo: no sé cómo se hace una I.A. de este estilo. Es necesario descubrirlo antes de terminar el desarrollo de modo que las decisiones de diseño e implementación no choquen con la tardía adición de este componente.



Riesgo: No solo debe tenerse un diseño claro de juego, sino también debe entenderse cómo se va a implementar de manera que se tenga, siempre, un orden de acciones que siguen las reglas debidas de juego.



Riesgo: Es un juego de pelea. Es necesario que los personajes sean llamativos, interesantes, satisfactorios de jugar, visualmente placentero y con un haptic feedback bien diseñado.



Riesgo: No solo debe tenerse un diseño claro de juego, sino también debe entenderse cómo se va a implementar de manera que se tenga, siempre, un orden de acciones que siguen las reglas debidas de juego.

Input buffering:
Una vez empieza la ejecución de un ataque o acción el personaje no debería de poder hacer más nada hasta que dicha acción no se vea terminada, ya sea por su culminación o por su cancelación por un ataque enemigo.

Sin embargo, debe haber un sistema que registre cada input y lo almacene por una cierta cantidad de tiempo hasta que caduque o pueda ser utilizada.

Supongamos que tras esquivar el jugador quiere usar un ataque debil, pero pulsa el boton de atacar un frame antes de que el cooldown del input se acabe. Esto resultará en que el usuario sentirá que debió haber atacado, pero el personaje no obedeció su orden.

El buffer guardará la entrada una cantidad de tiempo estipulada y tomará como valida la última entrada si sigue almacenada en el buffer el frame en que el cooldown de input se acabe.

Prioridad de inputs:
El sistema debe poder detectar el tipo de ataque cuando dos se encuentran y decidir qué hacer: darle prioridad a uno, al otro, cancelar ambos, etc.

Secuencia de inputs:
Ya sea para el amague, ya sea para activar el ataque especial, ya sea para cualquier idea futura como los combos. El sistema debe ser capaz de reconocer cadenas de inputs.

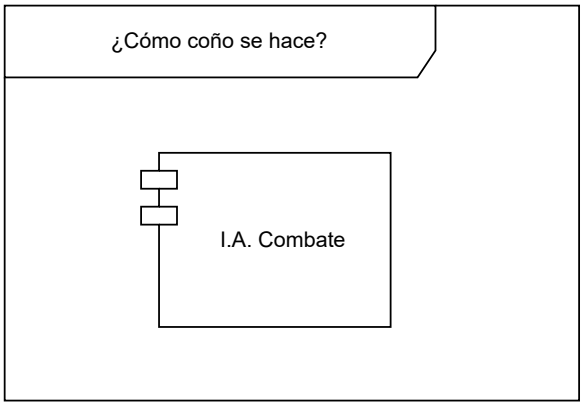
Supongamos que para activar el ataque especial hay que pulsar A+B al mismo tiempo. El sistema debe tener una forma de identificar cuando una cierta combinación de entradas fue introducida, y definir lo que significa que dos botones se presionen al mismo tiempo, tiene que ser exactamente al mismo tiempo o hay una ventana de tiempo en que la entrada es valida?

Este sistema a su vez debe poder ser reseteado o cancelado. Supongamos que una cierta cadena de inputs den como resultado cierta acción, si el personaje es golpeado antes de terminar esa combinación, dicha cadena de inputs debería ser invalidada. Esto sin embargo quizá no sea necesario si no añadimos combos.

Propiedades de Movimientos:
Cada movimiento debe tener un tiempo de ejecución, medido en segundos o frames. Ejecución tras la cual, el movimiento debe poder definir una ventana de tiempo en que puede recibir una segunda instrucción (Esquivar -> amague, Segundo boton -> ataque especial) y cambiar la ejecución por otra.

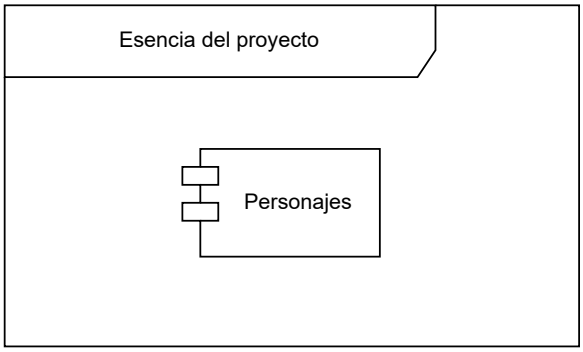
Sin embargo tras cerrar la ventana de espera, no debe poderse ver interrumpida hasta que llegue otro cambio de estado.

Cada movimiento debe tener un cooldown definible tras el cual no se puede recibir una nueva instrucción.



No sé, para despues.

Riesgo: no sé cómo se hace una I.A. de este estilo. Es necesario descubrirlo antes de terminar el desarrollo de modo que las decisiones de diseño e implementación no choquen con la tardía adición de este componente.



Riesgo: Es un juego de pelea. Es necesario que los personajes sean llamativos, interesantes, satisfactorios de jugar, visualmente placentero y con un haptic feedback bien diseñado.

Los personajes son principalmente prueba y error, modificaciones minimas en las animaciones y tiempos de ejecución de acciones hasta que se logre un feel de juego satisfactorio. Pero es necesario tener otros sistemas anteriores para lograrlo.

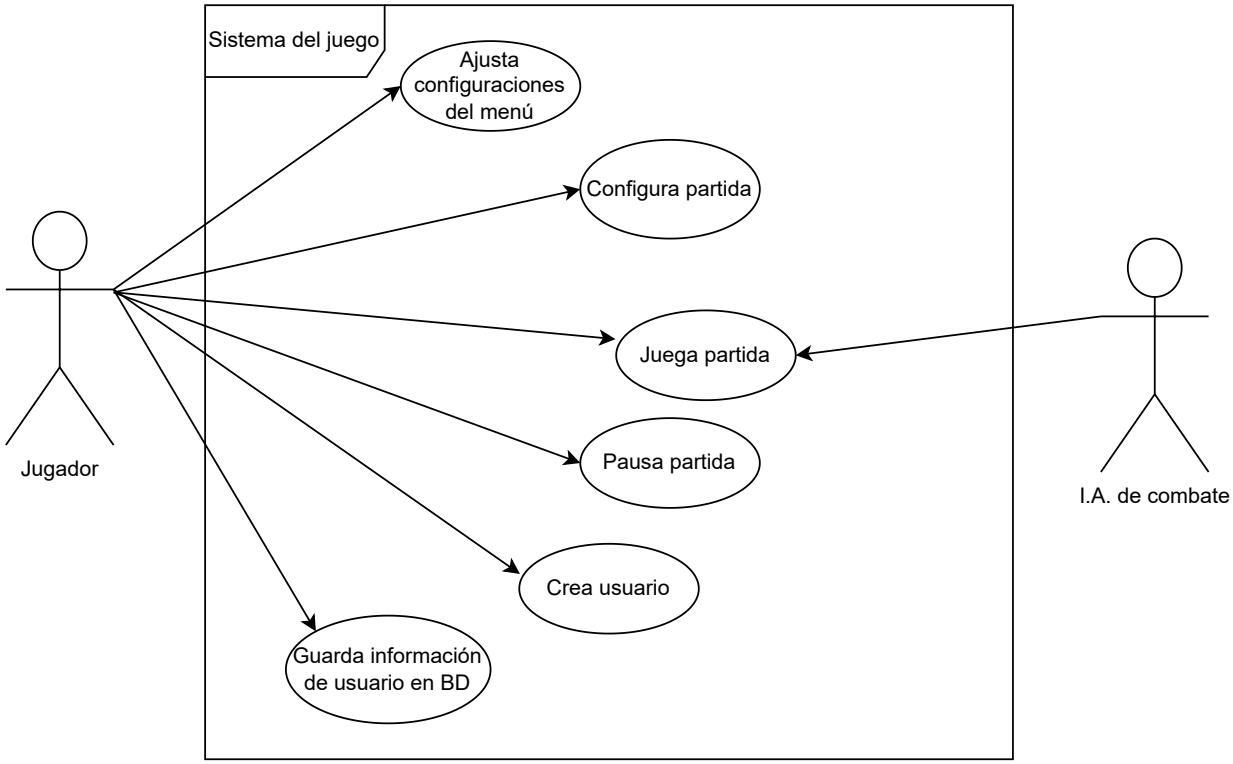
Método de desarrollo

Identificación del mejor método

En el caso de un videojuego creo que, al menos al principio, es más importante centrarse en tener escenarios jugables completos que permitan prontamente las pruebas de juego y luego centrarse en adiciones y caracteristicas modulares que se agregarán por encima del sistema ya funcional. De modo que sugiero optar por empezar el desarrollo con un enfoque en Use Case.

Nos centraríamos en tener una partida bien diseñada y jugable de principio a fin, y luego podré enfocarse en features individuales que dependen de poder jugar el juego para que tengan un sentido.

2. Diagrama de Use Case



Nos enfocaremos en el Use Case de "Juega partida", la cual a su vez puede parecer que cubre "Configura partida", pero la intención es tener una partida jugable, sin importar que ya esté configurada y creada en el propio codigo, algo que el usuario no podría hacer.

Incluso dentro de "juega partida" deberíamos enforcarmos en pura jugabilidad y dejar de lado la interfaz y el sonido por ahora.

El objetivo será tener dos personajes que se peguen en pantalla de la forma en que hemos diseñado en que lo hagan y suceda lo que, según nuestras reglas, debería de suceder.

La idea es tener algo que se pueda jugar y probar, modificar valores hasta obtener un resultado que nos guste.

Este es el comportamiento que deberá, mas o menos, tener el sistema una vez terminado y funcionando según lo planeado. De modo que usaremos este Use Case para determinar qué debe hacerse, y qué comprenderá el desarrollo en esta primera fase temprana del proyecto.

Use Case: Jugar partida

Actores principales: Jugador 1, jugador 2
Pre-condiciones: hay dos jugadores listos para jugar.
Meta: Iniciar y jugar una partida principio a fin hasta que alguien gane.

Main path

1. El sistema inicia ambos personajes en pantalla, uno a cada lado, uno de ellos volteado en su eje Y.
2. El sistema inicia los botones en pantalla y a su vez toma entrada por teclado.
3. El sistema espera una entrada del jugador.
4. Un Jugador ingresa una entrada, por boton o teclado.
5. El sistema almacena dicha entrada en el input buffer.
6. El sistema determina si el personaje está en un estado que puede recibir ordenes.
7. El sistema lee el input buffer.
8. El sistema determina el tipo de acción según la entrada.
9. Si dicha acción es realizable cambia el estado del personaje a "ejecutando X acción".
10. El sistema abre una ventana de tiempo en que puede recibir una nueva entrada para alterar la acción.
11. Se acaba la ventana sin cambios. El sistema empieza a ejecutar la acción, imponiendo un input cooldown.
12. Se acaba el input cooldown, puede recibir nuevas ordenes.
13. Si el personaje sufre daño, recibe un cooldown de acciones corto. Y se resta su propiedad de salud.
14. Sistema chequea que la salud de ambos personajes sea 0.
15. Sistema chequea que el tiempo se haya acabado.
16. Sistema finaliza la partida.
17. Sistema guarda información de partida, usuarios, etc.

Alternate path

6.1 Si el personaje no puede recibir ordenes, mantiene el input en el buffer hasta que caduque o pueda ser ejecutado.

9.1 Si la acción pedida no se puede realizar, no se hace nada, y se borra la entrada del input buffer.

10.1 Si durante el inicio, ventana de tiempo de cambio, o ejecución de una acción, la misma se ve interrumpida de alguna forma por acción del rival, toda acción se cancela y se cambia el estado a uno relevante al contexto.

11.1 El jugador ingresa una entrada que cambia la acción, se cambia el estado de ejecución del personaje.