Development Test

Documentación

Arquitectura del Juego

La arquitectura utilizada es Component Based.
De esta manera es posible dividir las entidades del juego en componentes independientes y especializados, lo que permite poder utilizar cada componente en objetos diferentes sin tener que duplicar código. Además, permite una mejor legibilidad y mantenibilidad del proyecto.

Aplica principios tanto de Stable Abstraction como de Stable Dependency.
Abstracción para acciones individuales y comportamiento de objetos, como dependencia para los controladores.

Modelo de Red

El modelo de red utilizado es de Autoridad Distribuida.
De esta manera tanto el cliente como el servidor tienen cierta autoridad en distintas acciones, lo que permite tiempos de respuesta mucho más cortos y una mayor fluidez en la jugabilidad.

Por ejemplo, el movimiento del jugador es autoridad del cliente y el servidor solo valida su posición en el mundo, pero algunas acciones como la de reproducir animaciones o aplicar daños son auditadas por el servidor.

Tecnologías Utilizadas

Fishnet

Solución networking para Unity.

Steamworks

Tecnología utilizada para poder manejar un servidor estable, con la posibilidad de crear salas de juego dentro.

Input System de Unity

Permite gestionar los inputs del usuario de manera más eficiente y flexible.
Además permite mapear los inputs de distintos dispositivos o controles en acciones individuales.

Cinemachine

Facilita la composición, el seguimiento y la transición de la cámara del juego.

2D Sprite Editor

Permite editar sprites del juego para distintas funcionalidades.
En este caso se utilizó para configurar el corte de los sprites de UI y que puedan ser estirados sin que afecte a su aspecto.

Profiler y Memory Profiler

Herramientas para el análisis y monitoreo en tiempo real sobre el uso de CPU, GPU, memoria y otros recursos del sistema.

Desafíos en el Desarrollo

El principal desafío fue aprender a utilizar Fishnet eficientemente, aunque realmente no fue un problema porque la documentación que tienen, si bien algunos aspectos no están actualizados, es una buena base.

Otros de los problemas que tuve es que, al estar acostumbrado a usar Photon y los servidores integrados que tiene, fue un poco engorroso buscar la forma de poder utilizar un servidor sin tener que pagarlo.

La solución más sencilla fue la de utilizar Steam, aunque al no tener tantos conocimientos sobre el armado de backend fue un poco difícil al principio.
Terminé siguiendo un tutorial de Youtube, sencillo… hasta que me tope con que la solución de Steamworks te redirigía al juego de prueba de Steam puesto en el id de la aplicación. Esto es porque la versión que se utilizaba en el video era mucho más vieja que la última en release, y sobre la última en release no hay nada de documentación al respecto.

Utilizar una versión más vieja desembocó en el último problema que tuve, que fue en el intentar hacer la predicción del lado del cliente, la cual no logré.

Fishnet proporciona una documentación que explica bastante bien la teoría de la predicción del cliente, de hecho tienen dos formas de hacerlo:

- En la primera, el ejemplo con el cual lo explican es únicamente compatible con los componentes del controlador de ellos mismos, y si bien se podía adaptar… rompía demasiado la estructura con la que estaba trabajando y requería re-adaptar algunos sistemas (error mio por no haberlo investigado desde el principio y utilizar el controlador de ellos). A pesar de todo eso, la solución que brindaban no era del todo eficiente, y aparentemente consumía muchos recursos de conexión que impactaban en la performance.

- La segunda opción que ofrecen, es una versión experimental que facilita esta predicción. Acá desemboca el problema que mencionaba anteriormente, y es que la versión de Fishy.Steamworks que estaba utilizando rompía dependencias y modificaba scripts core de fishnet.
  Por lo que si quería hostear en Steam, no iba a poder hacer uso de esa forma de predicción.