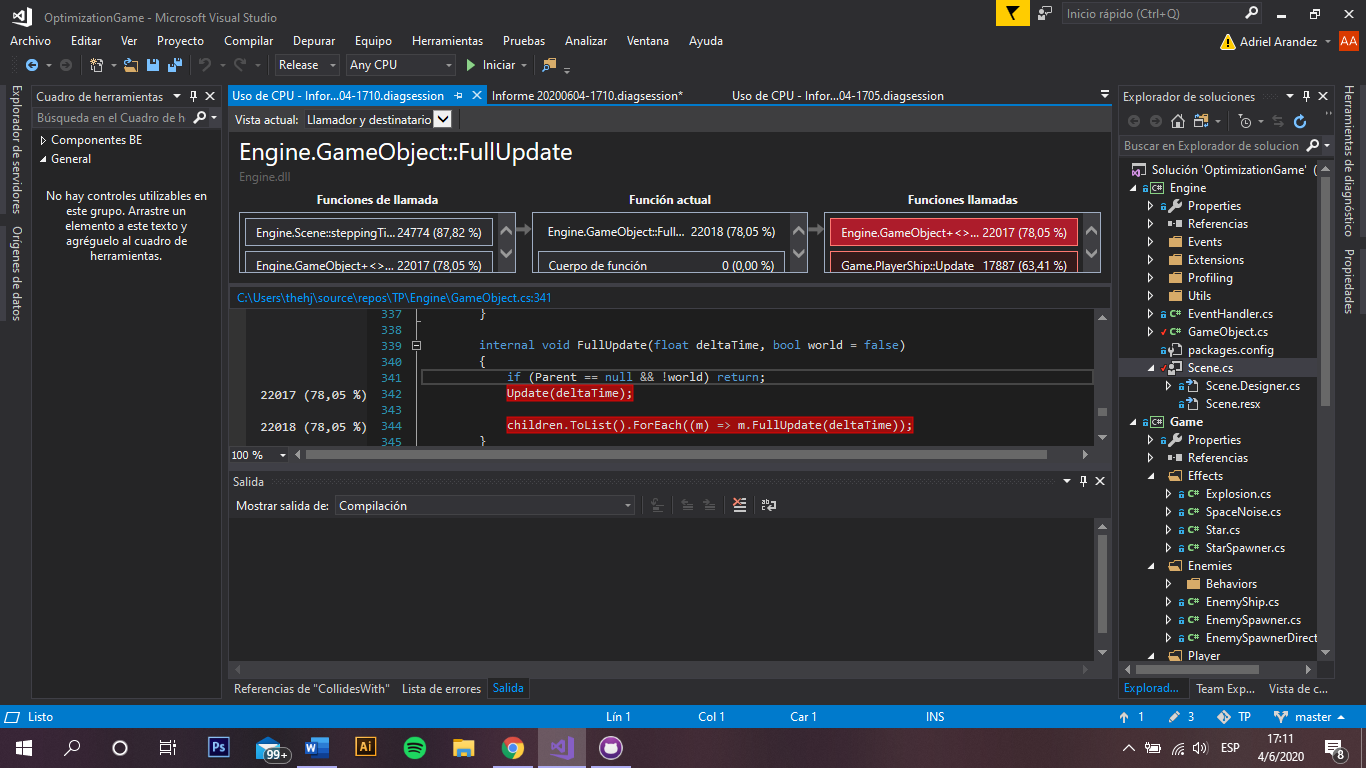
**Trabajo Práctico Técnicas de Optimización Arández Adriel:**

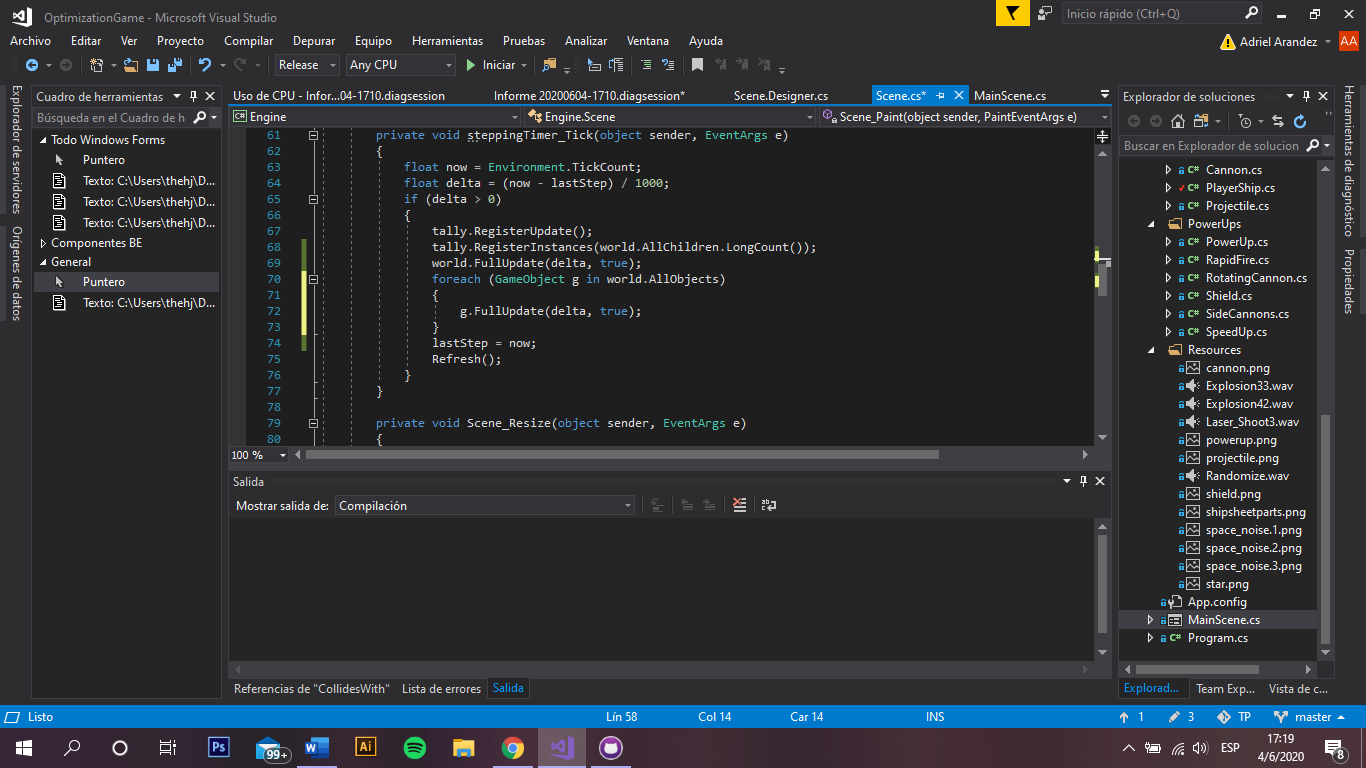
Primera medición:

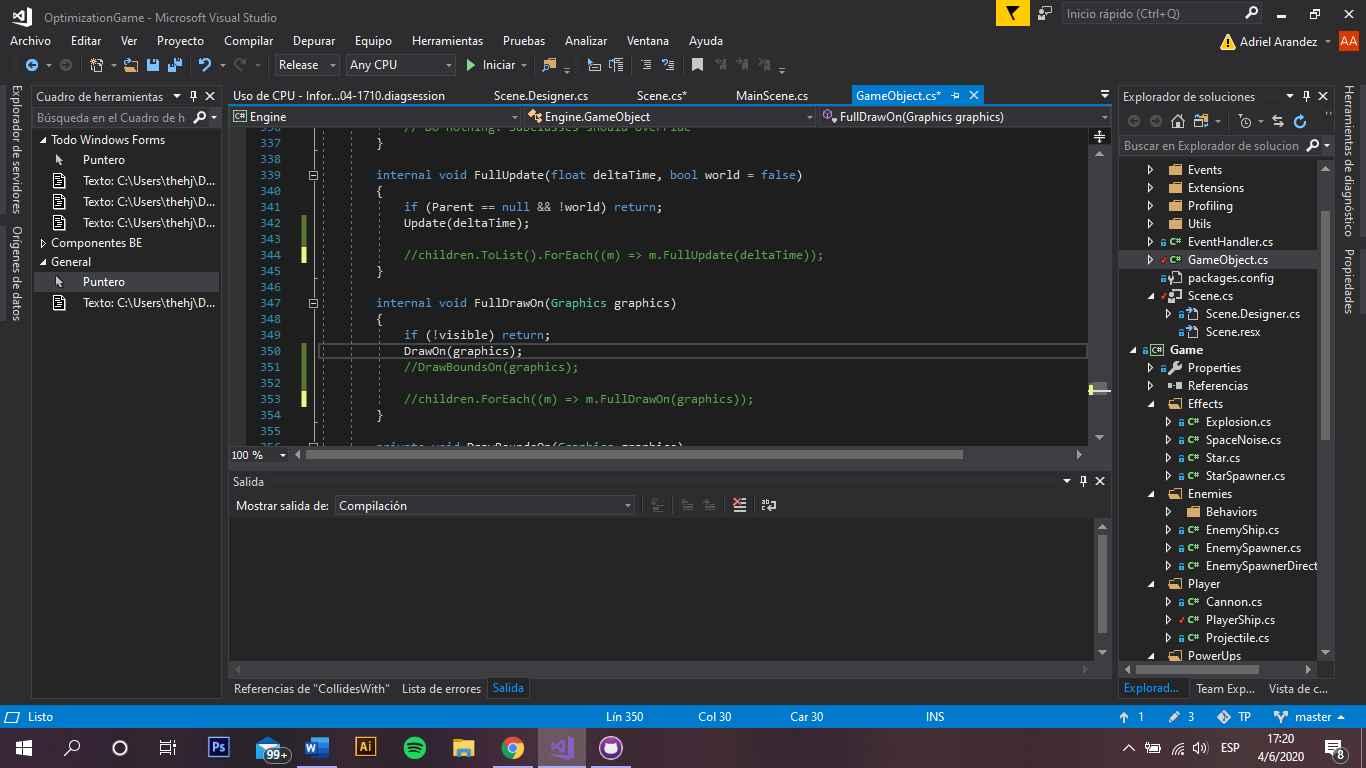


En la primer medición se puede ver que el método FullUpdate() está consumiendo gran parte de la capacidad de proceso, esto en parte debido a la larga cadena que de *children* encadenados que posee cada GameObject en base al objeto *world* que los engloba y que en cada una se ejecutaba el método que por cada objeto hacía que se ejecutase en sus *children*. Lo mismo sucedía con el método FullDrawOn(). En teoría si solo se ejecutara un Update por cada objeto el rendimiento mejoraría.

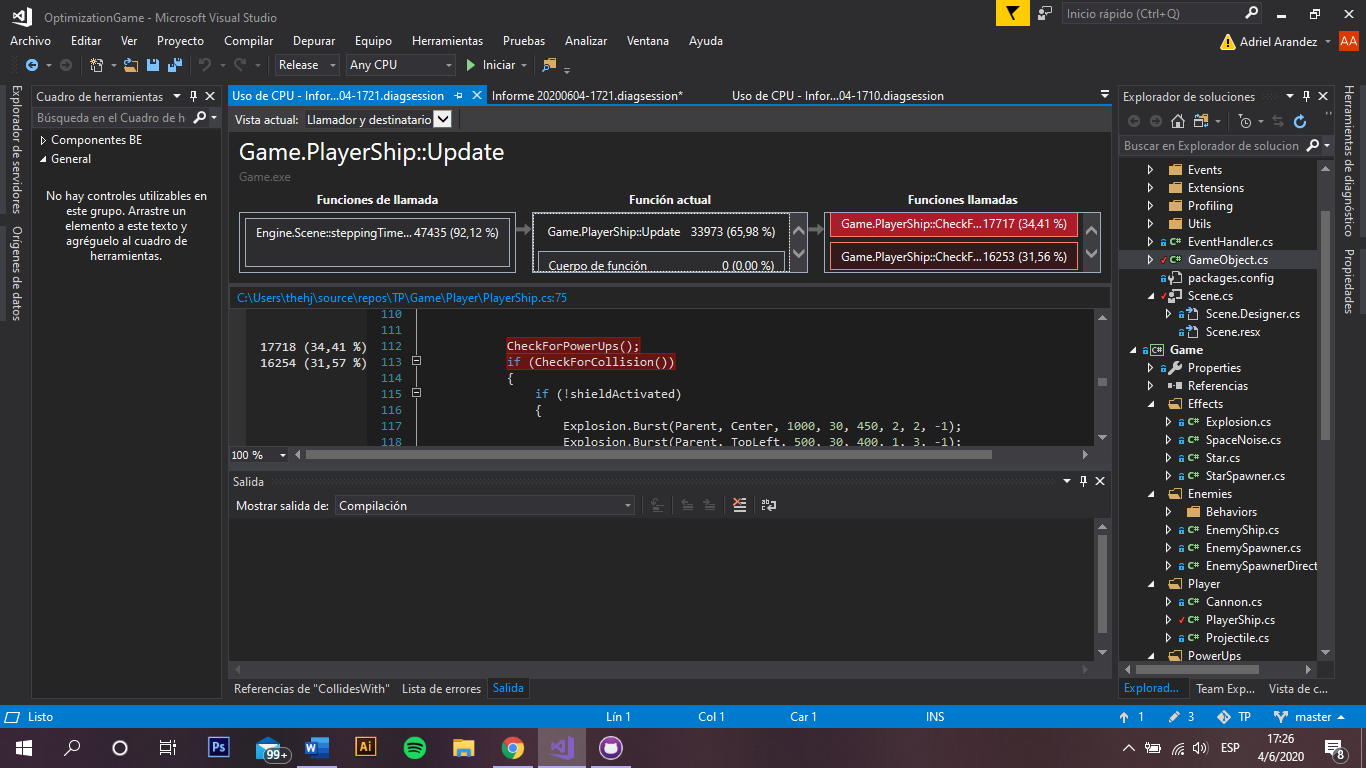
Resolución:

La elección fue reemplazar esto por ejecutar un FullUpdate y el FullDrawOn de manera individual por cada GameObject desde la escena principal.





Segunda medición:

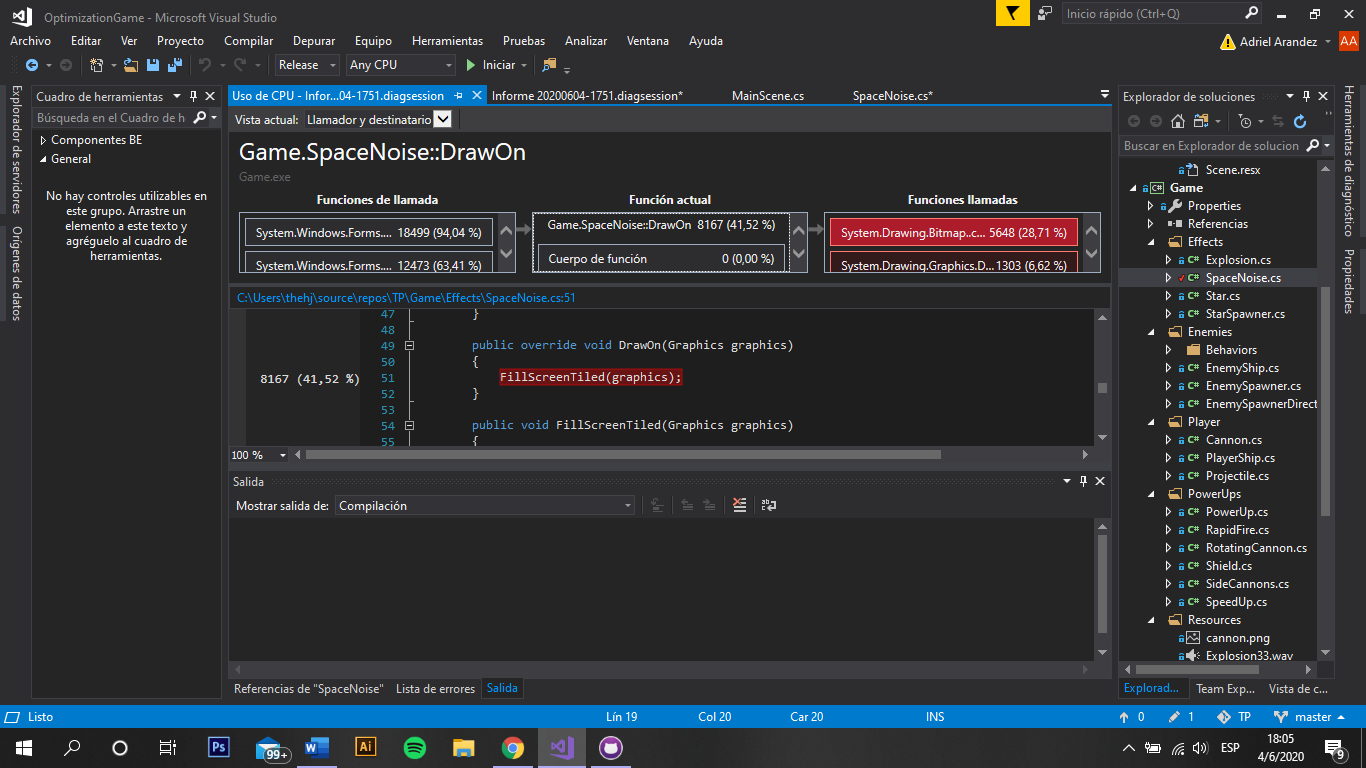


En la segunda medición se puede notar como los métodos CheckForPowerUps() y CheckForCollision() contenidos en el Update de la clase PlayerShip ocupan gran parte del procesamiento. El motivo es probablemente la manera en la cual se obtiene y compara con otros objetos para confirmar sus colisiones, ya que, primero compara su colisión con todos los objetos en la escena y luego determina si son o no correspondientes al tipo de objeto buscado. Analizar primero su tipo y luego su estado de colisión debería mejorar la ejecución.

Resolución:

Luego de modificar el método que recolectaba todos los objetos y luego analizaba si eran del tipo requerido, invirtiendo el orden, el rendimiento aumentó.

Tercera medición:

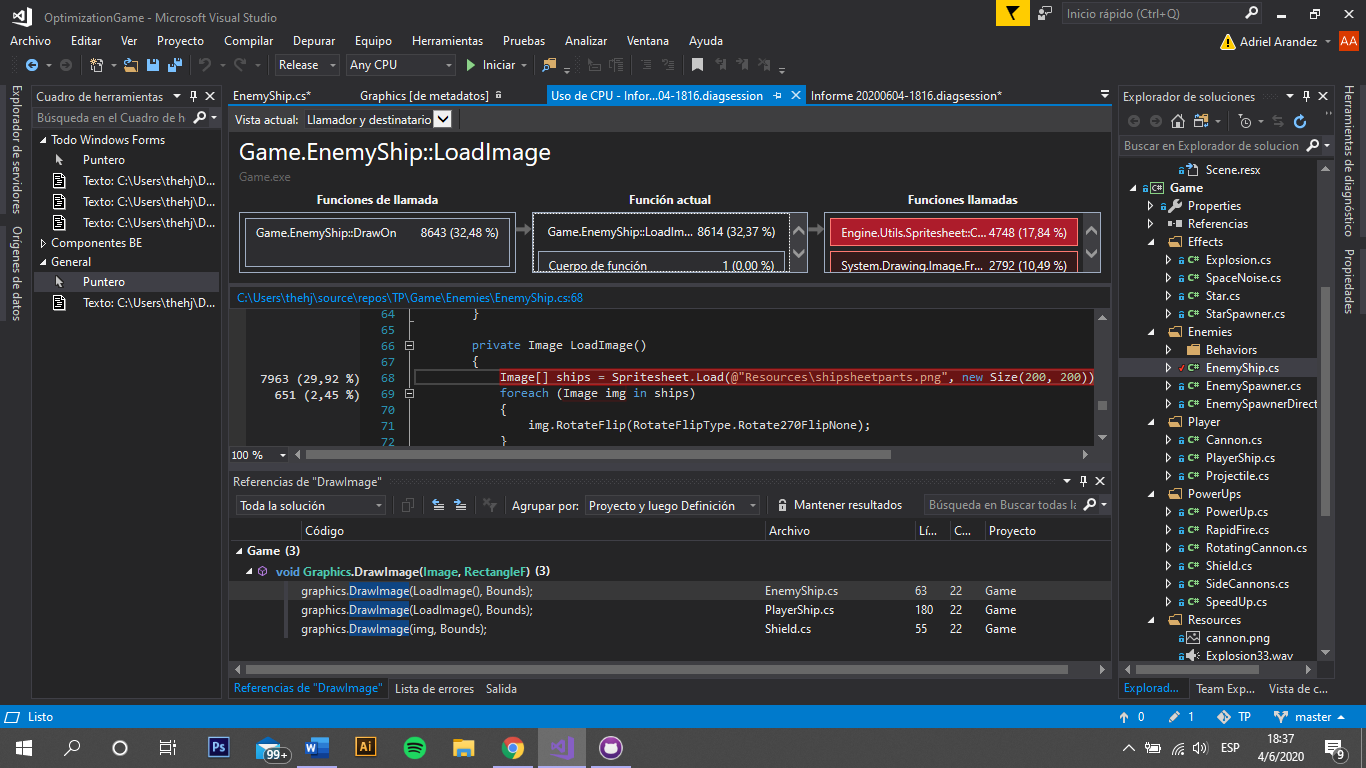


La tercera medición dio como resultado que el método FillScreenTiled() consumía mucho, mas puntualmente la utilización de Bitmap, el cual es creado en cada actualización de dibujado, tener uno ya creado probablemente mejorara la performance.

Resolución:

Crear un bitmap que no debe redibujarse en cada actualización aumenta la velocidad de ejecución del programa.

Cuarta medición:

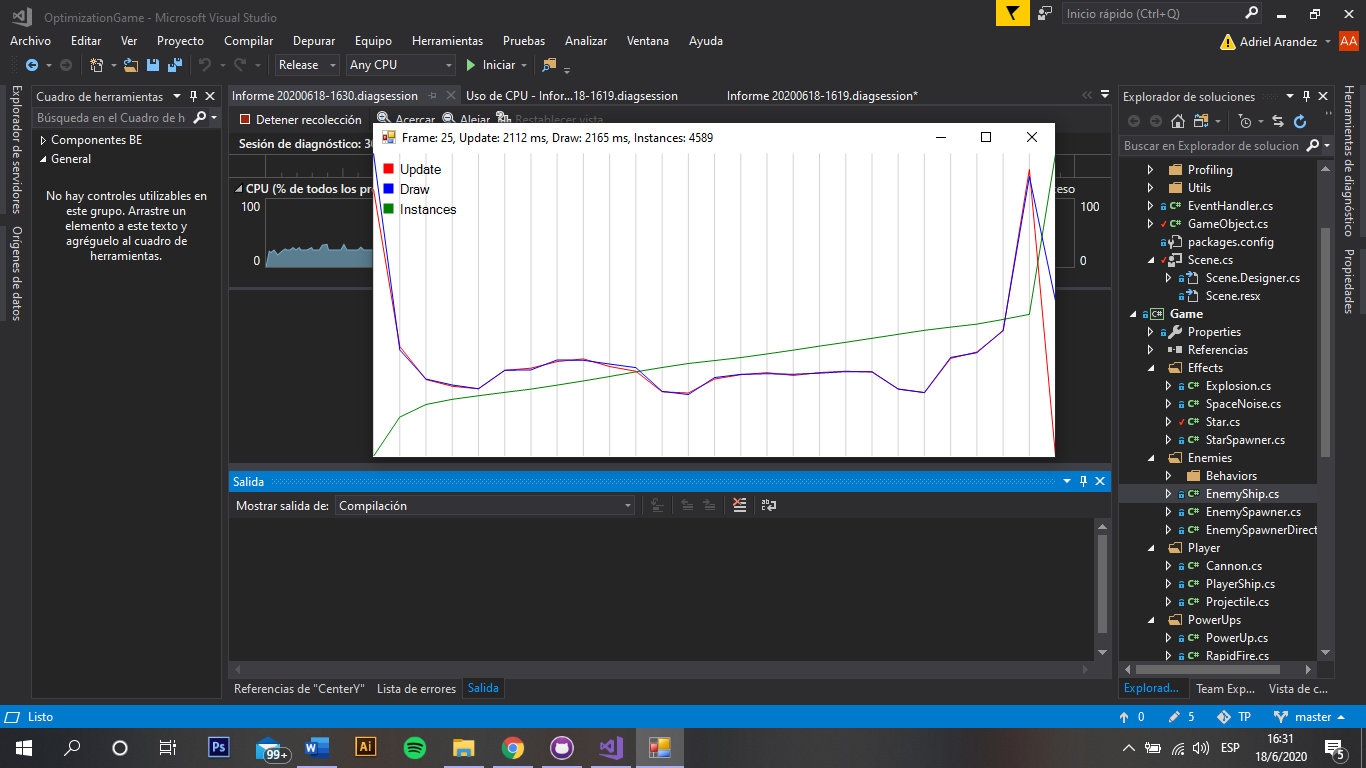


En esta medición es visible como el método LoadImage es poco eficiente ya que en cada actualización debe recargar la imagen correspondiente a cada nave enemiga antes de dibujarla, esto podría ser mejorado si la imagen fuera cargada y almacenada previamente, para ser utilizada en cada redibujado.

Resolución:

Guardar la imagen y no cargarla en cada redibujado mejoró el rendimiento.

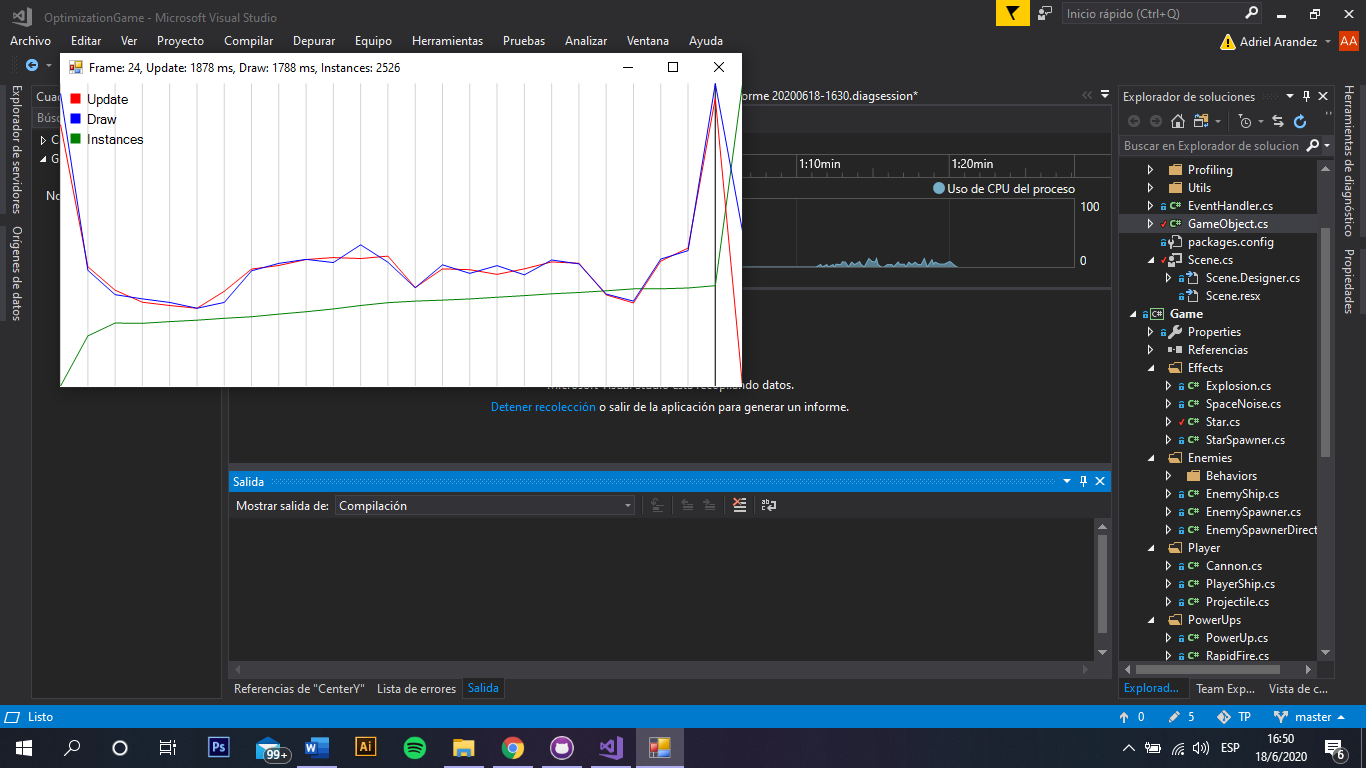
Quinta medición:



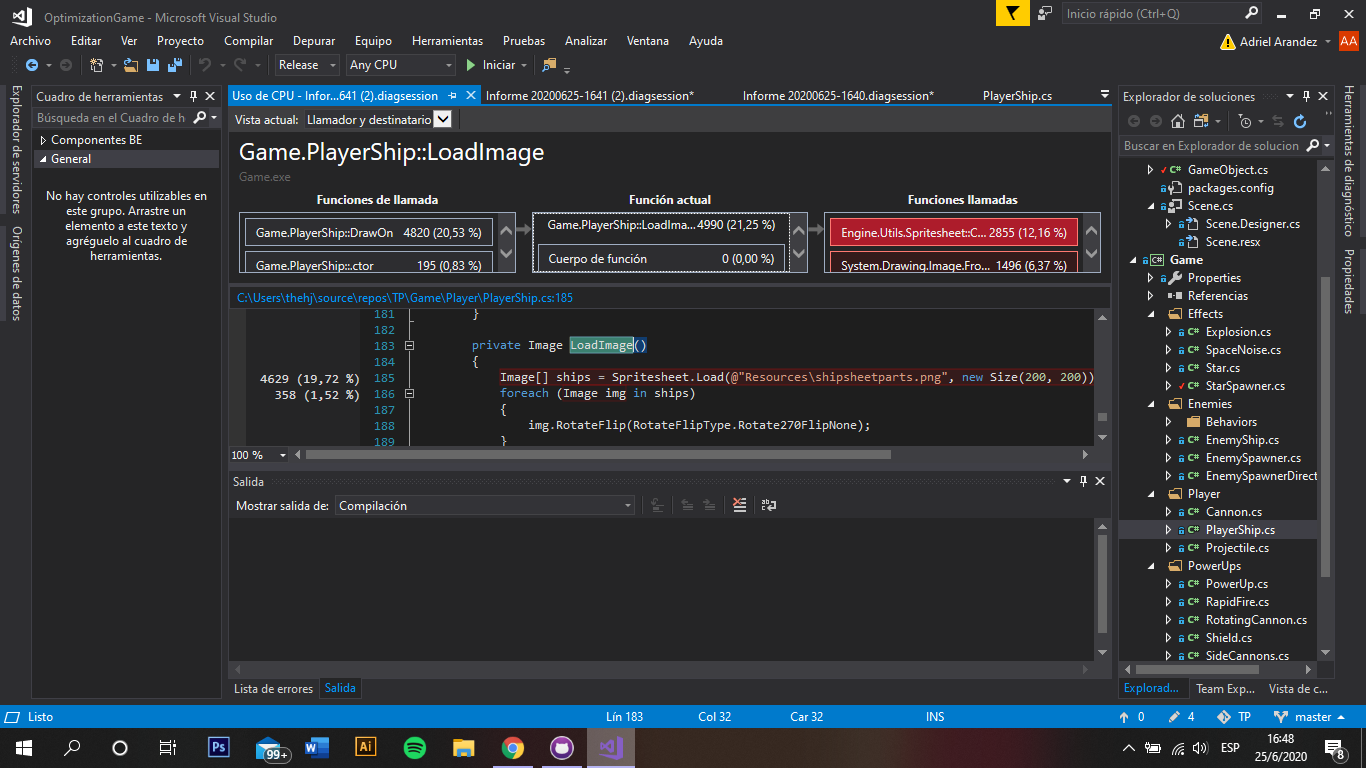
En esta medición puede verse claramente que la cantidad de instancias dentro de la aplicación va constantemente en aumento, ocupando memoria innecesaria y probablemente realizando procesamientos que no serán útiles. Para solucionarlo la idea es destruir todos los objetos en los cuales su X sea menos a 0, ósea que estén fuera de la zona visible.

Resolución:

Al principio la solución fue eliminar desde el FullUpdate los Gameobjects cuya X fuera inferior a 0, esto resultó bien excepto por el hecho que un tipo de nave enemiga precisaba de la información de su parent, pero ésta era destruida al llegar a 0. La solución entonces, para no alterar el funcionamiento, fue destruir los Gameobject al llegar a -5 y ejecutar el código de las naves enemigas solo si su parent existía. La cantidad de instancias se redujo considerablemente.



Sexta medición:



En esta medición, así como n la cuarte se puede ver que cada vez que se va a dibujar la nave del jugador se carga desde los recursos, para optimizarlo lo ideal sería hacerlo solo una vez.

Resolución:

Luego de realizar el cambio mencionado, cargando la imagen solo al principio, esa parte del código ya no ocupa tantos recursos.