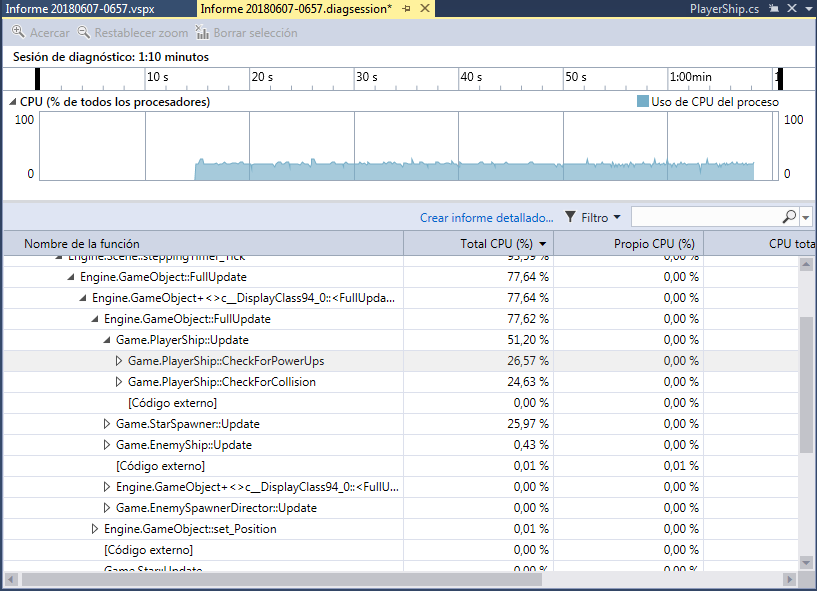
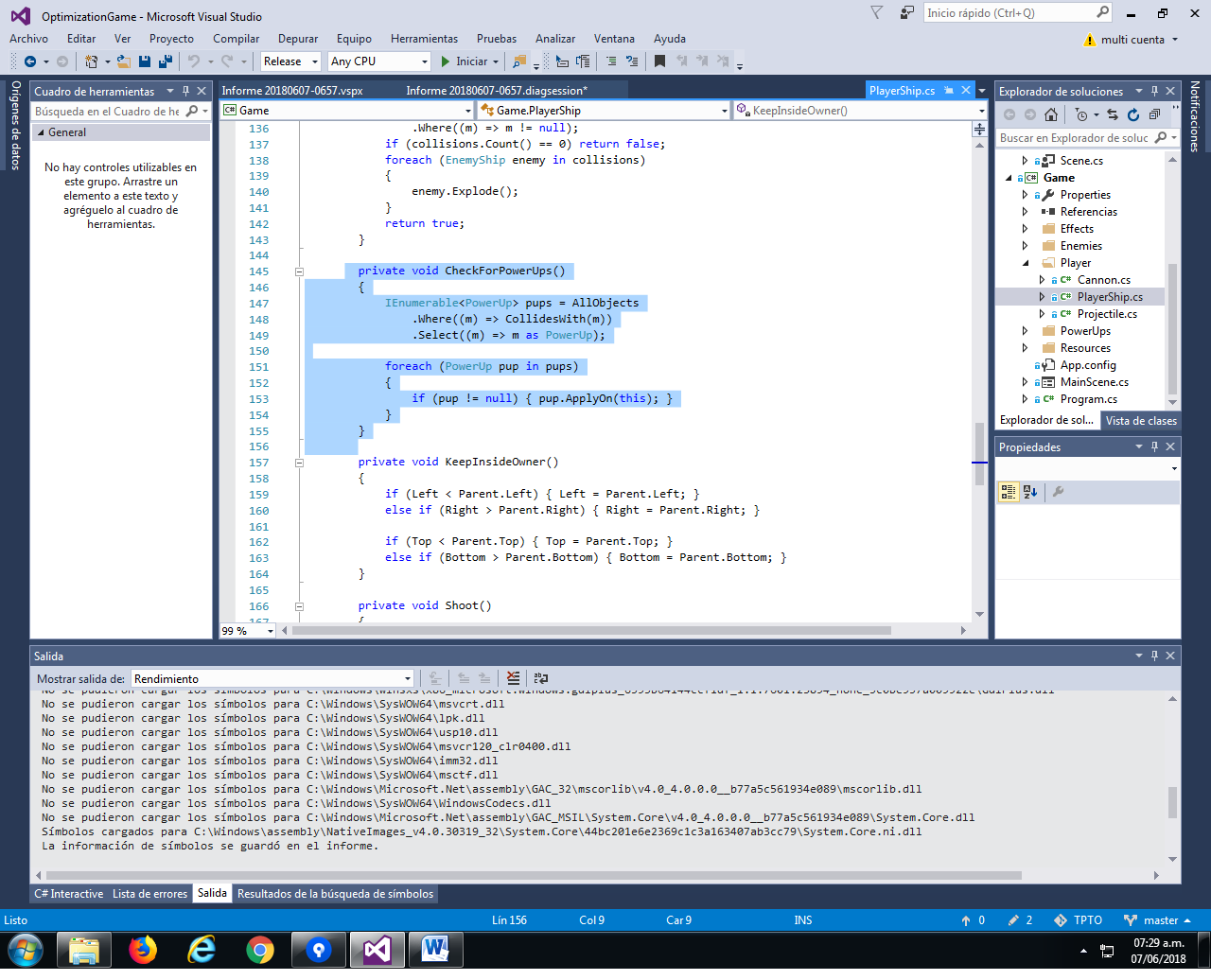
**Juego de tp de TO**

En una run de prueba del juego las mediciones mostraron que en 1 minuto la pantalla se actualizó tan solo 12 veces.  
El generador de perfiles de rendimiento mostró lo siguiente:

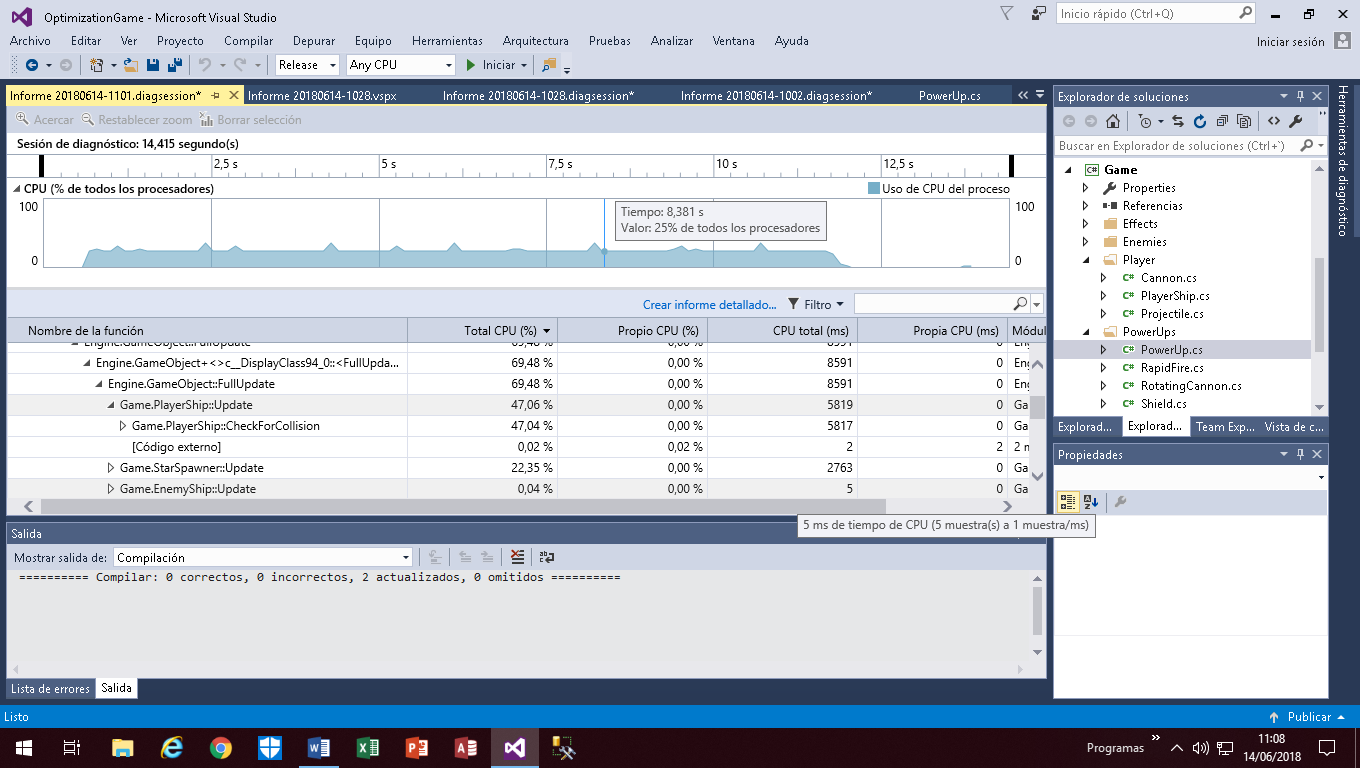


Tras checkear Game.PlayerShip.CheckForPowerUps encontré esto:

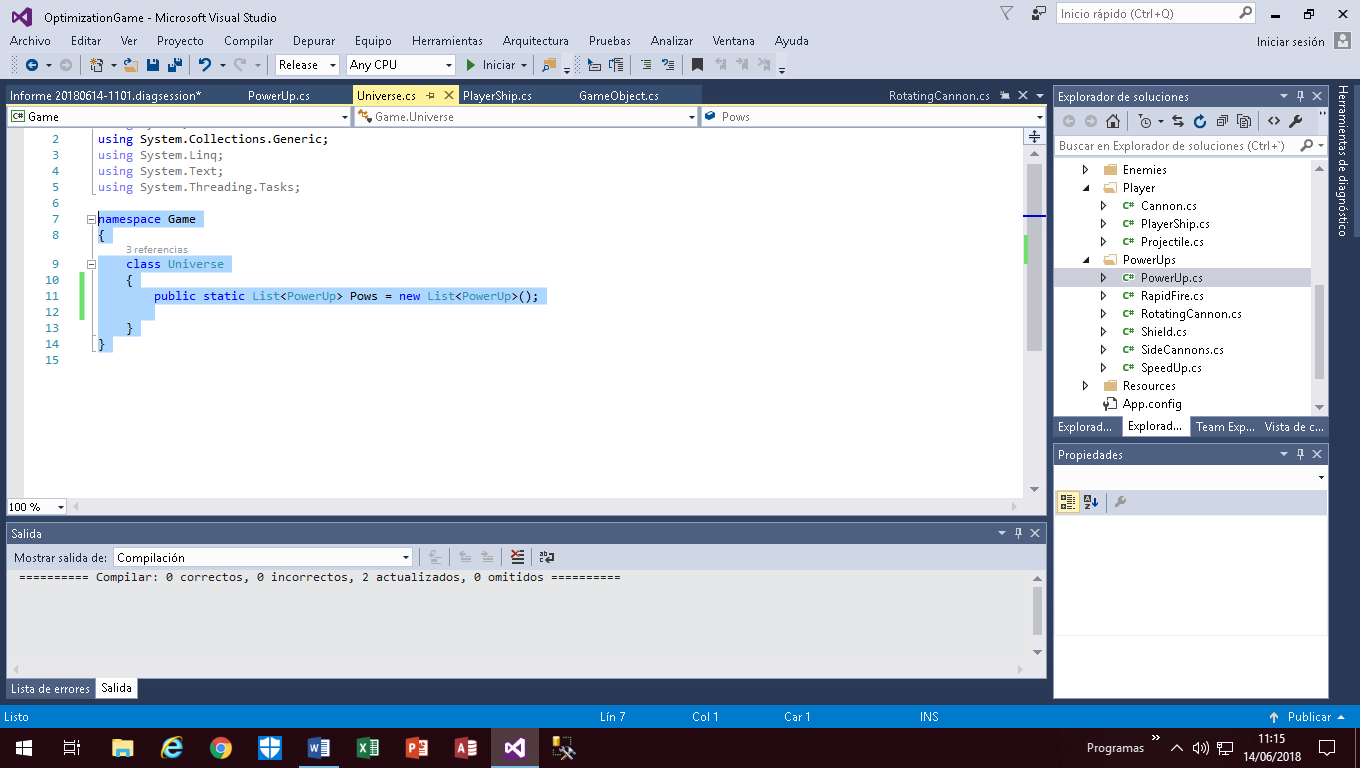


A primera vista el problema parece estar en que se crea una colección de objetos colisionables en cada frame.  
  
Estimo que con un uso de “Spatial partition” debería mitigarse el uso de cpu fuertemente, es decir tener una colección de objetos colisionables por el jugador guardada en memoria a la cual cada objeto generado que sea colisionable se agregar al instanciarse.

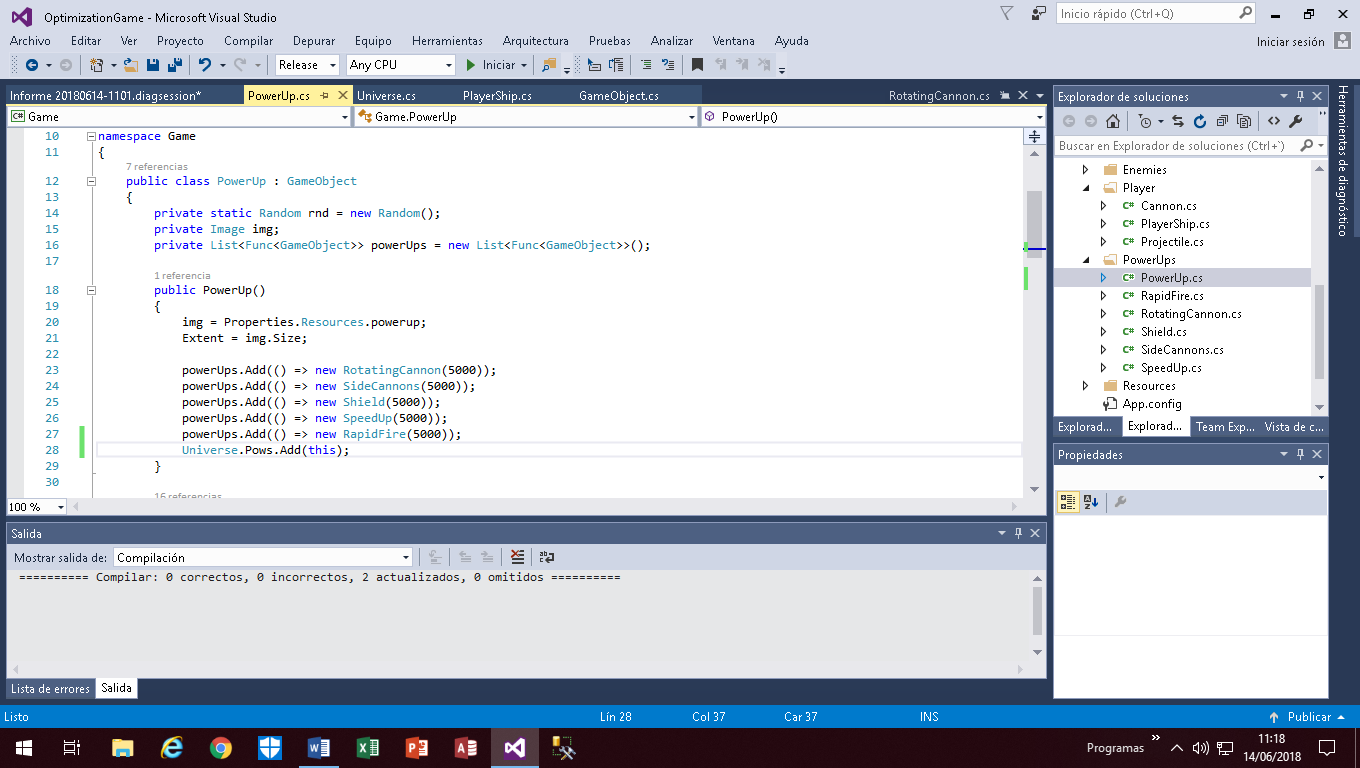
Realice la optmizacion específicamente para la clase power up, es decir cree una lista de power ups global a los que la nave puede acceder para saber si colisiona con ellos y los resultados fueron que el uso de cpu en “checkforpowerups” fue tan baja que no entro en la medición.

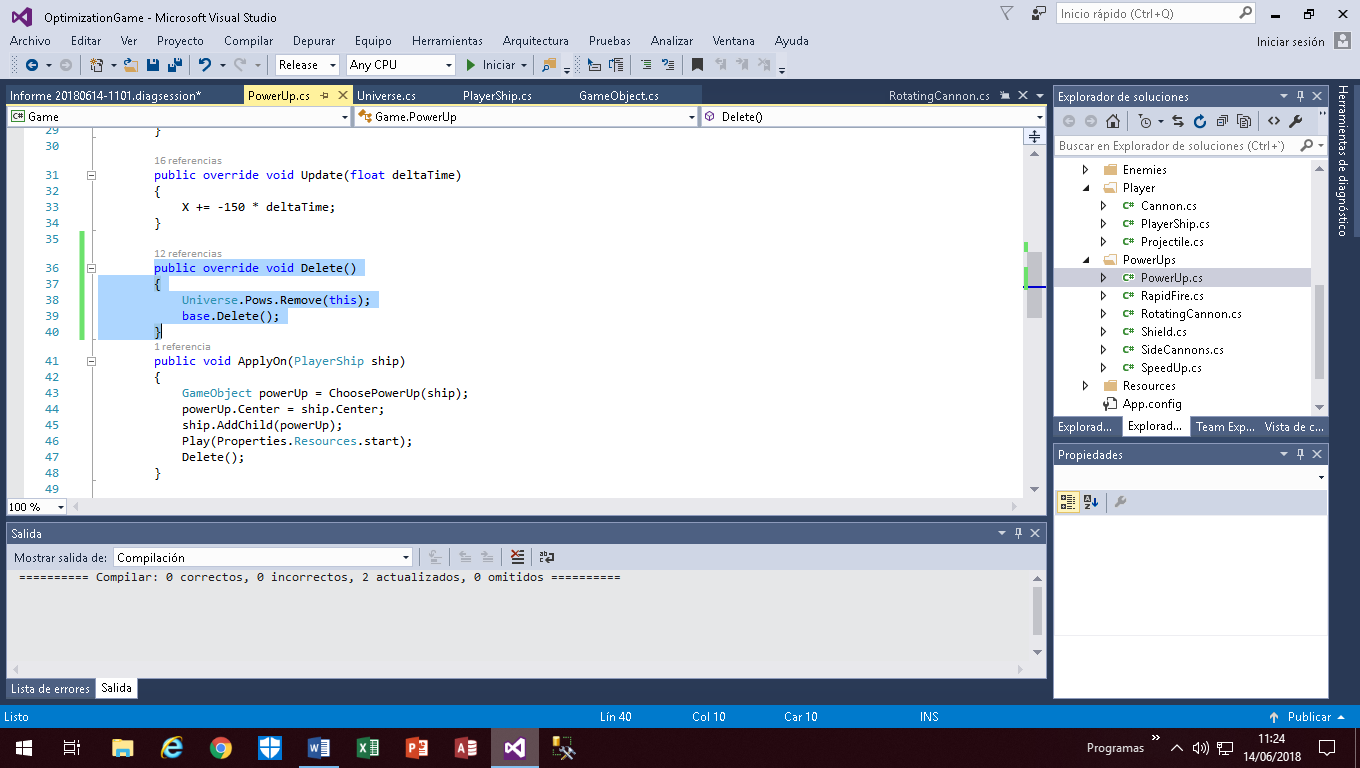


Las siguiente modificacionse al código fueron realizadas:

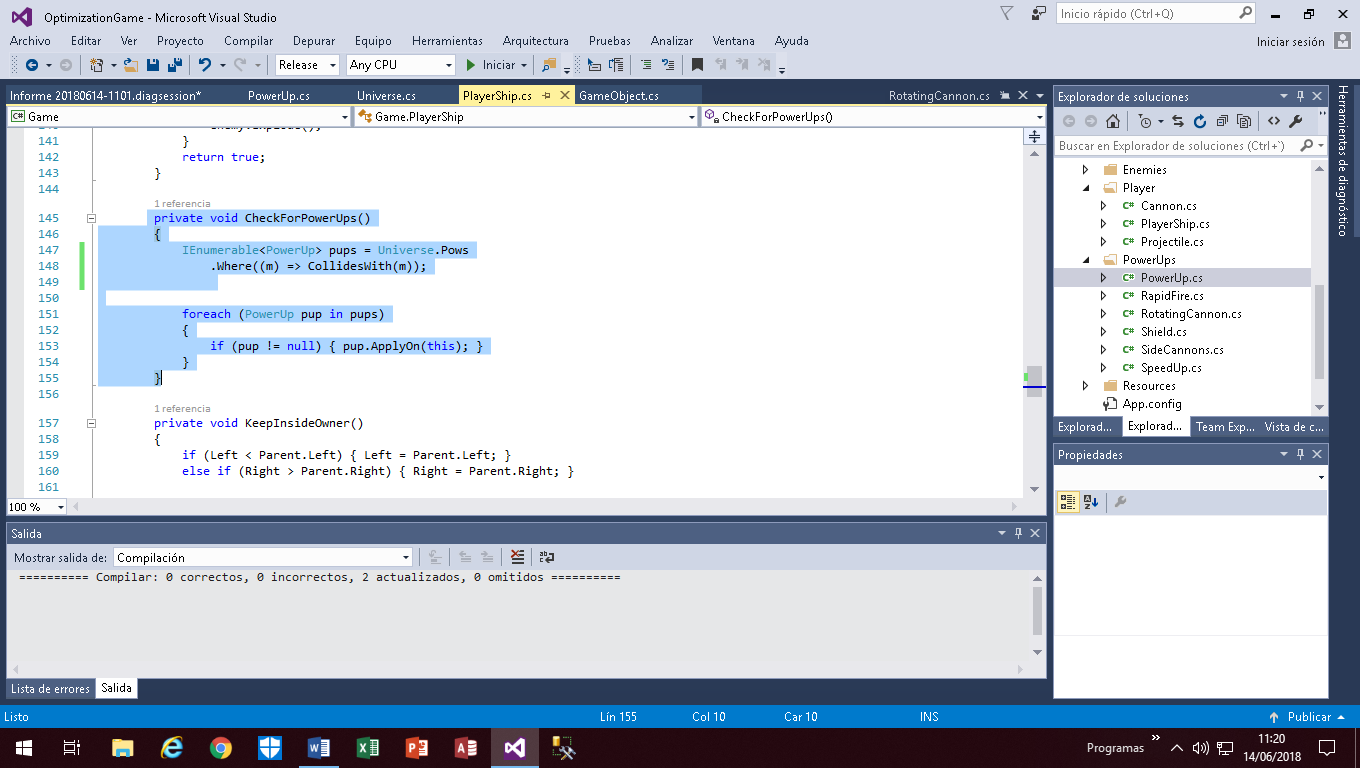


PowerUp:

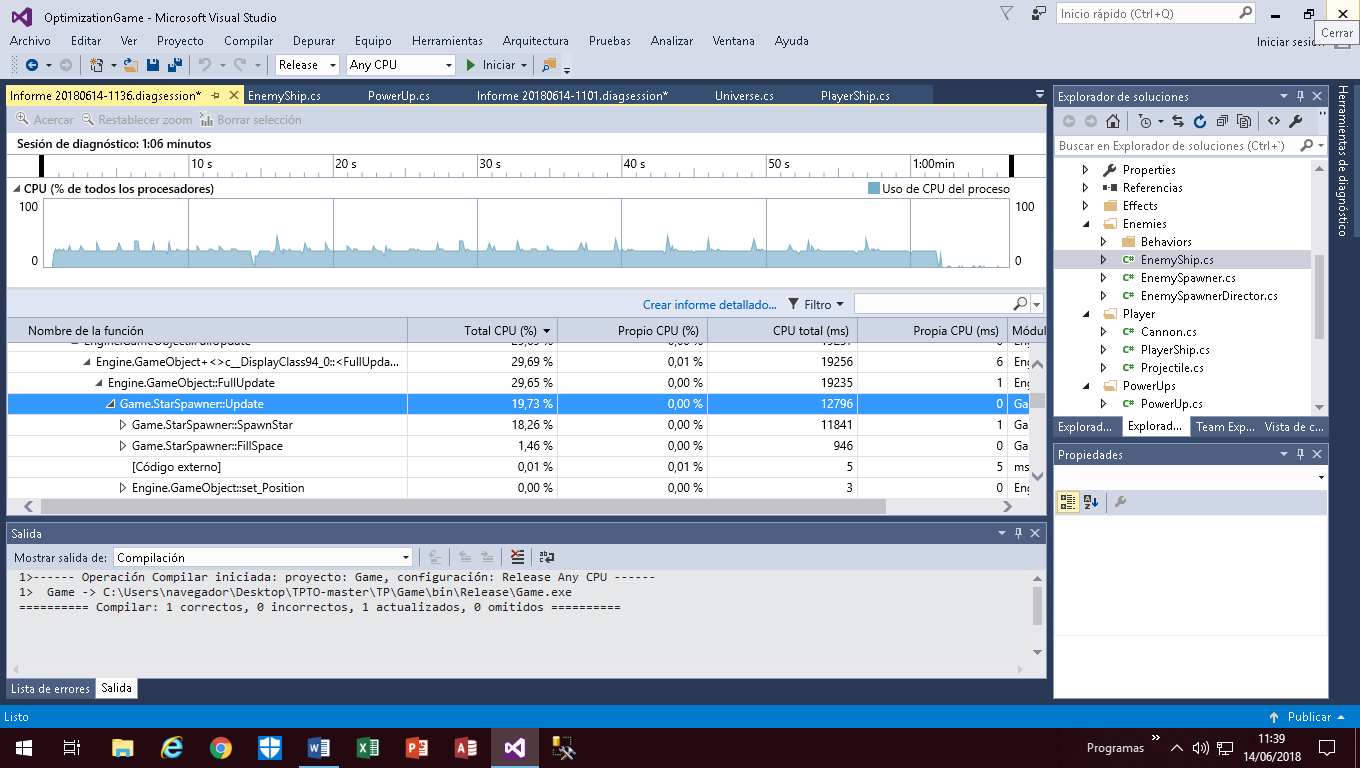




PlayerShip:

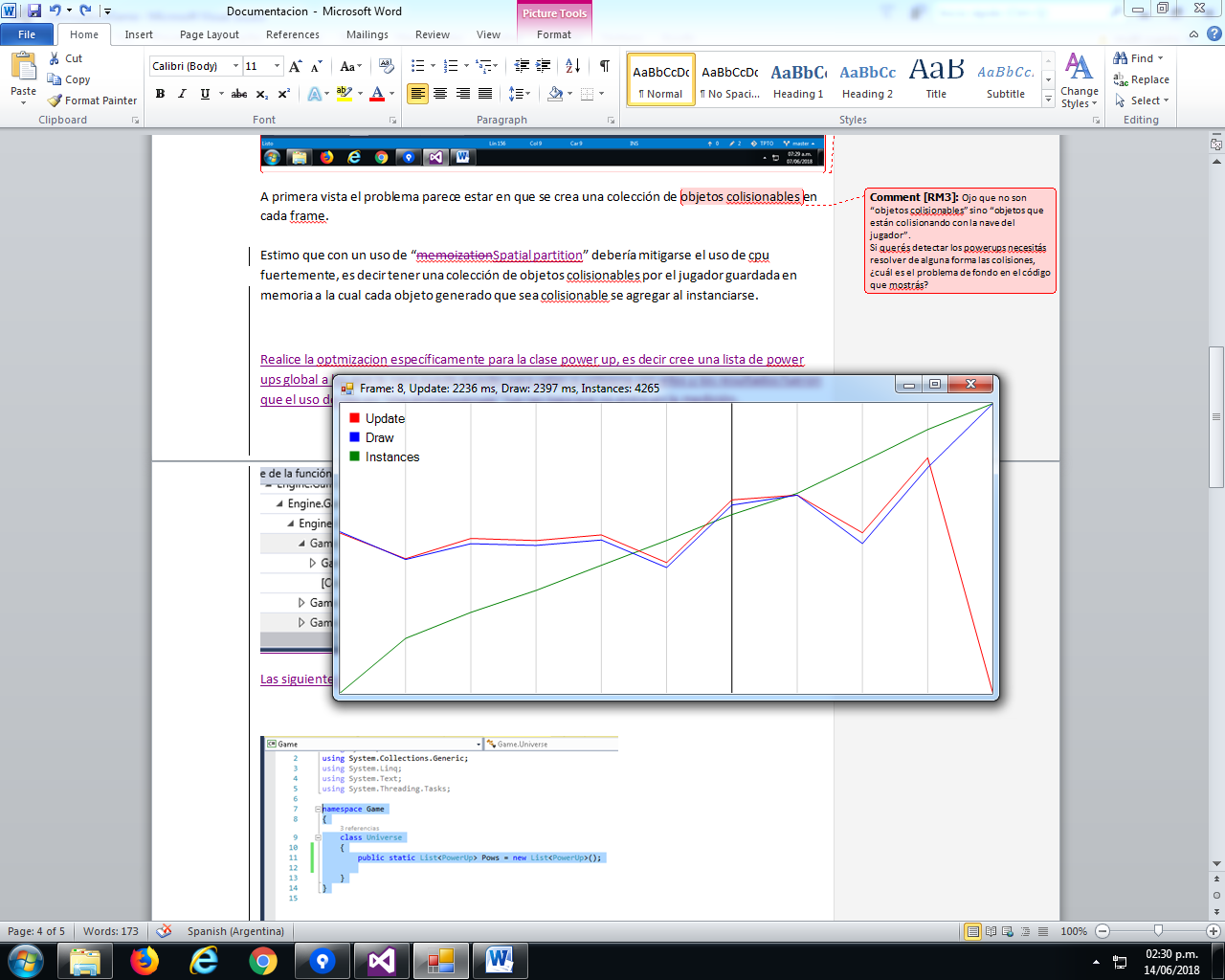


Voy a aplicar la misma optimización a CheckForCollision.

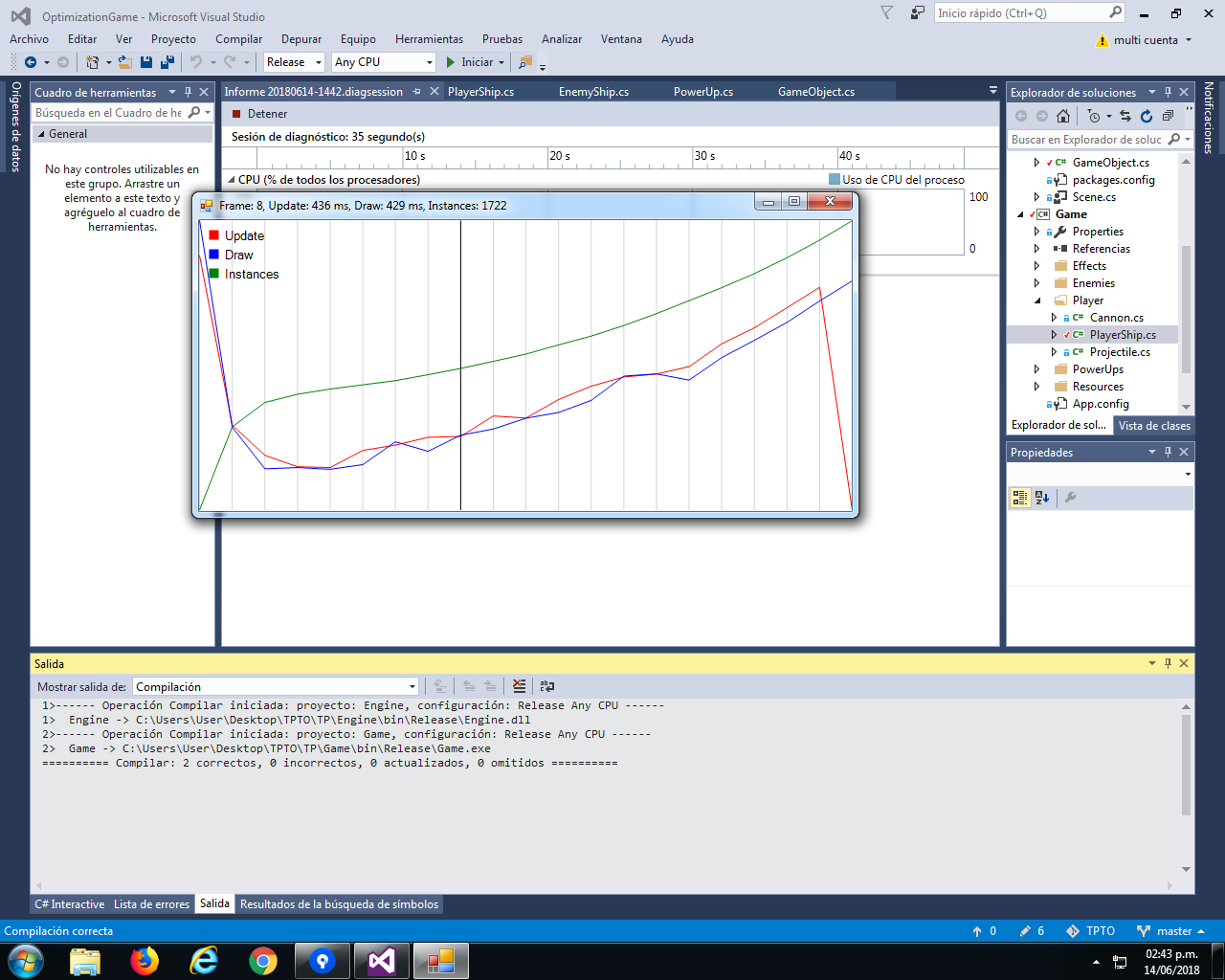


Aproveche que los cambios anteriores los hize en clase y hize la siguiente comparacion contra la version sin modificar en casa y a lo largo de la aplicación la mejora en performance es visible, las mediciones en frame 8 son bastante significativas:

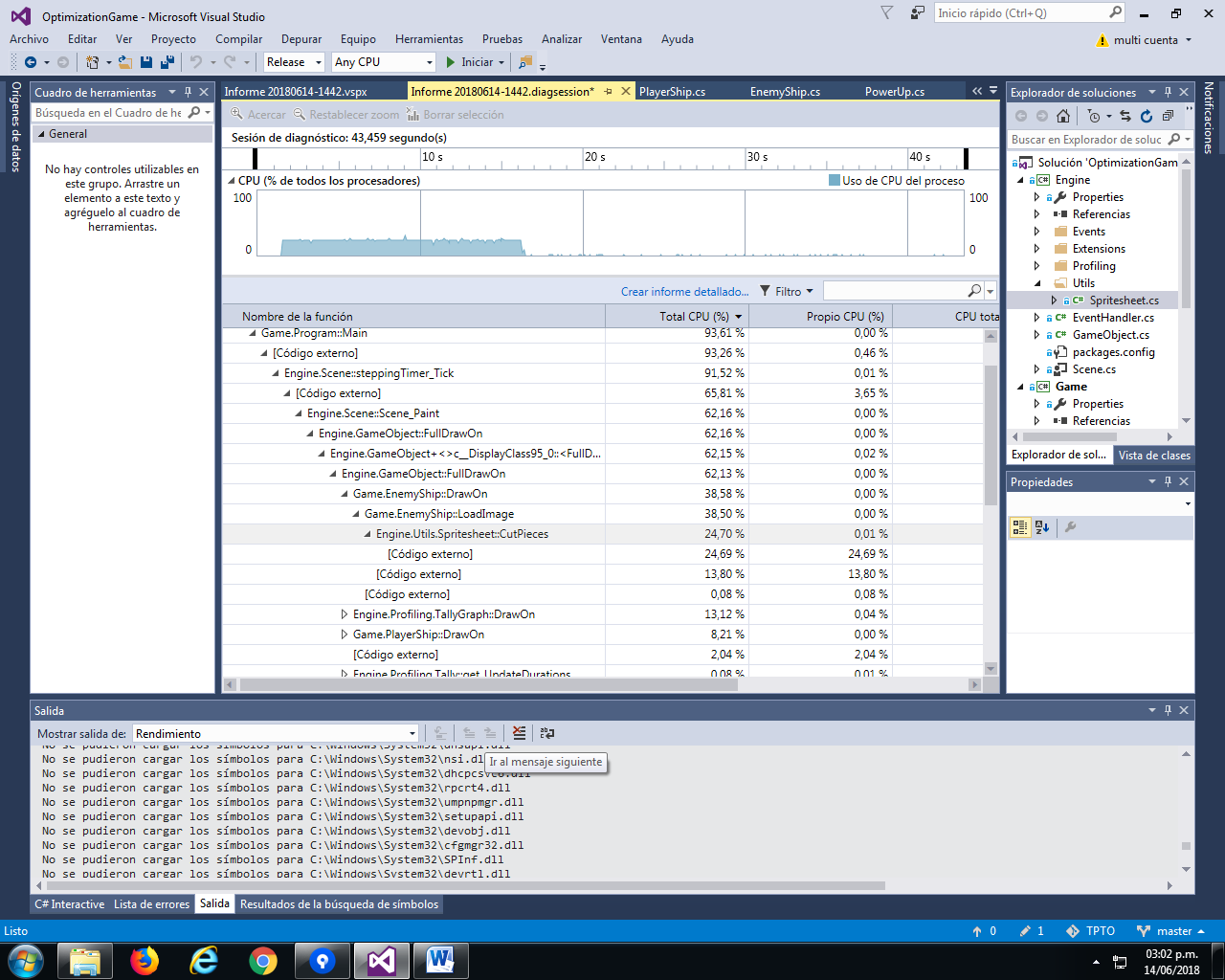
Antes:



Despues:

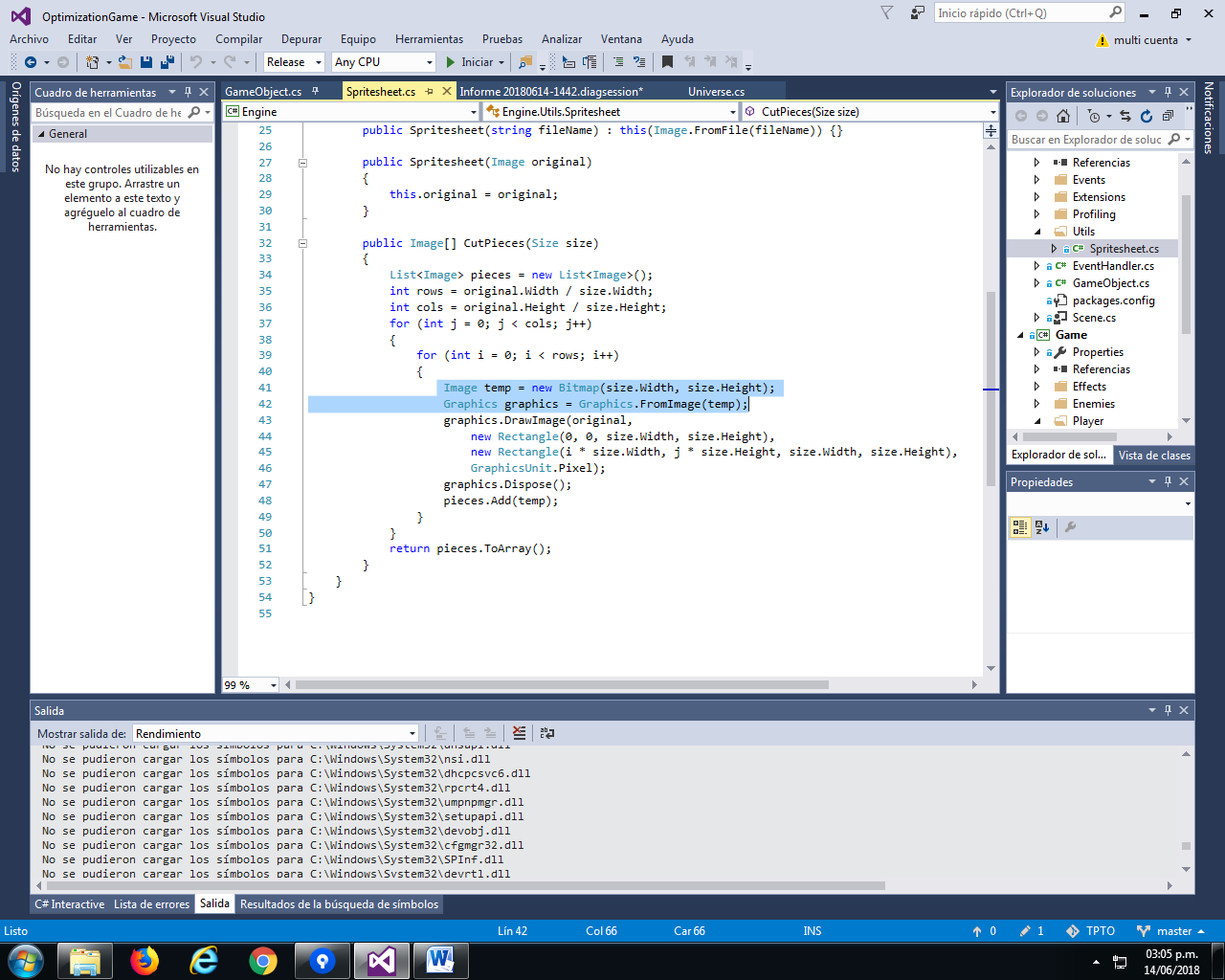


Una nueva medicion muestra el siguiente hotspot:



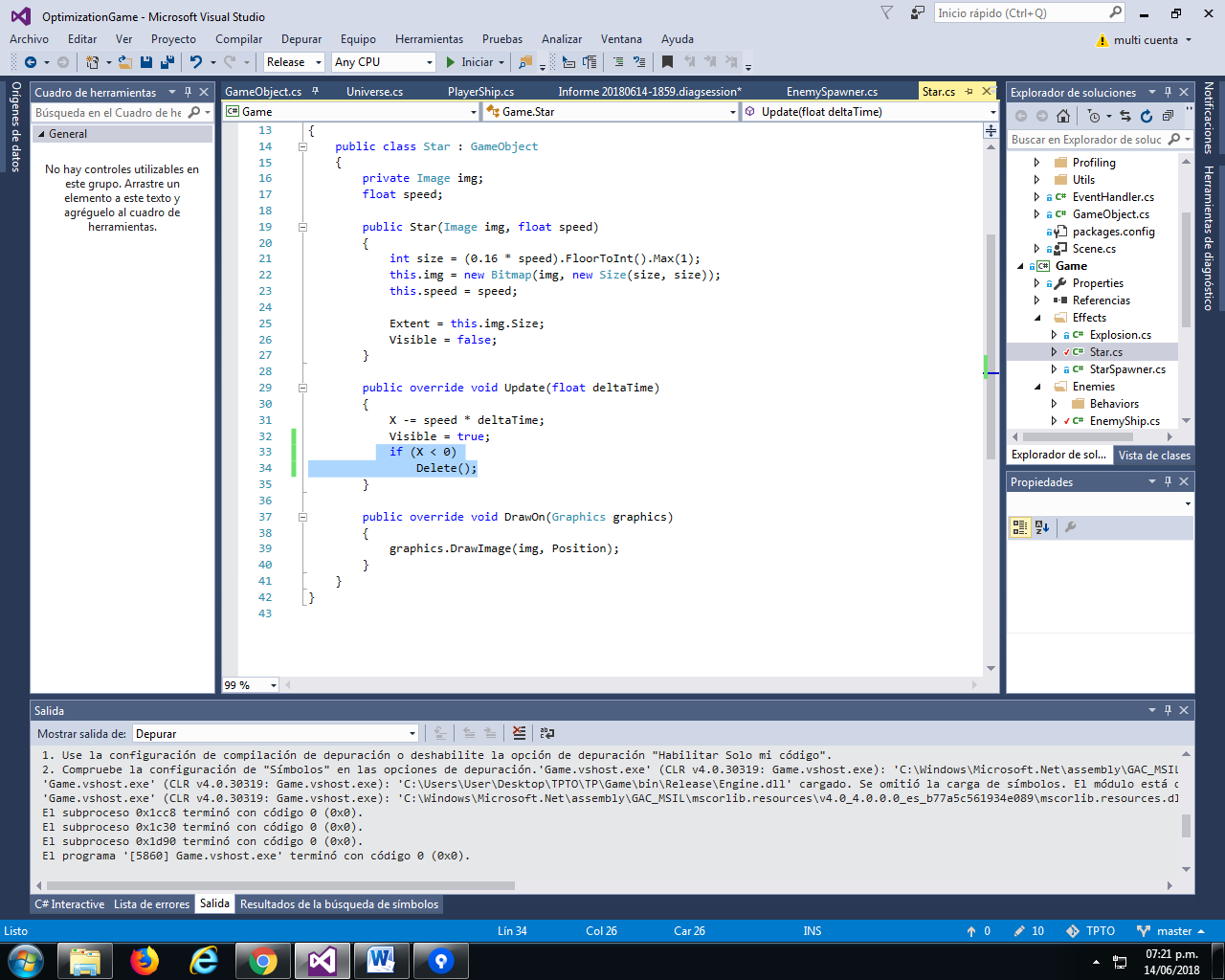
Mi Teoria es que esta parte del método esta realizándose mas veces de la necesaria y que se ganara algo de performance sacándolo del doble ciclo for.

Aun asi el proceso de cortar piezas de la imagen antes de mostrarla y mostrarla por piezas me resulta algo sospechoso, principalmente en los enemigos cuyas piezas no cambian, concidero la posibilidad de tener que hacer un método que reemplaze a este completamente en el caso de los enemigos(espero no tener que hacerlo para no tener que aprender como funciona la graficacion a fondo).



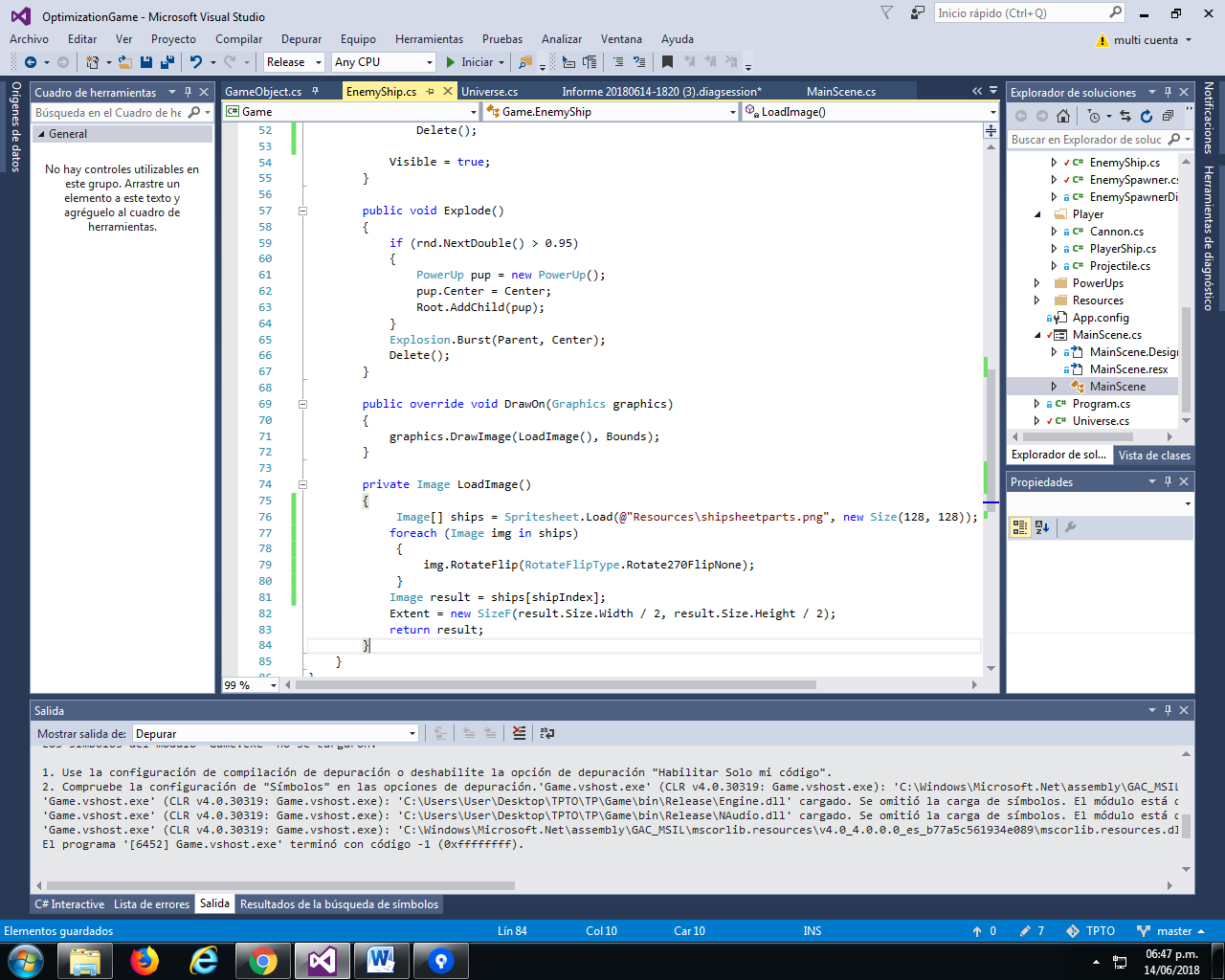
Al hacer eso y además mover la línea graphics .Dispose(); abajo del doble for el resultado fue que todas las naves se ven iguales, sospecho que la clase graphics debe usar algún cache, voy a investigar…

Voy a agregarle a las estrellas la funcionalidad de removerse al salir de la pantalla:



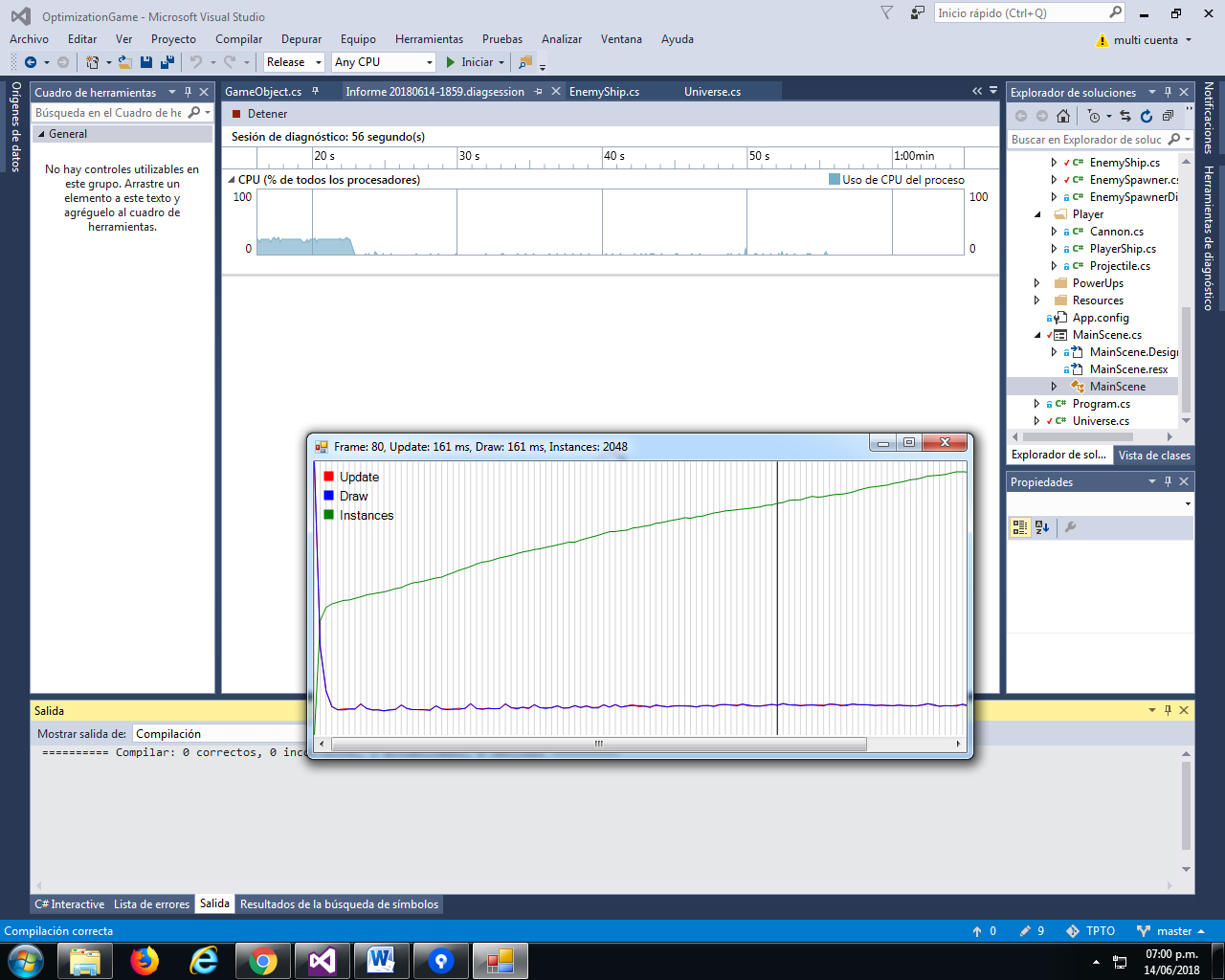
Aplicando lo mismo a naves enemigas consegui que el incremento de instancias sea mucho mas controlado.

La medición anterior también mostro una carga de CPU relevante en el método EnemyShip.LoadImage()

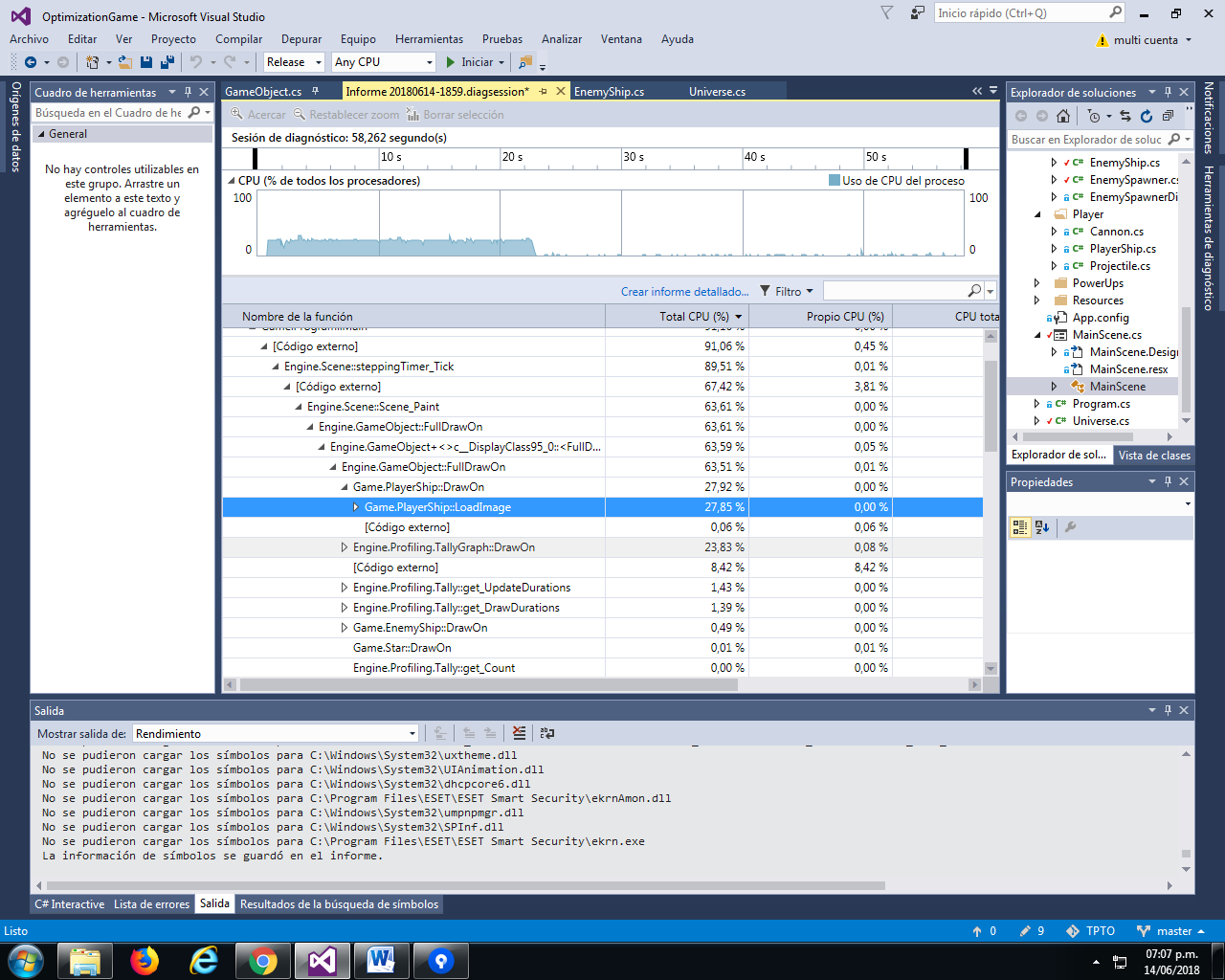


Mi solución a esto es por empezar es tener una colección estática de imágenes a las cuales acceder sin tener que cargar el archivo en cada ciclo de actualización. Seguido de eso delegar la rotación a la clase universo (quien tendrá la lista estática) para hacerlo solo cuando se carga el formulario y nunca mas. El propósito de esto es quitarle responsabilidades al método LoadImage el cual es llamado constantemente.

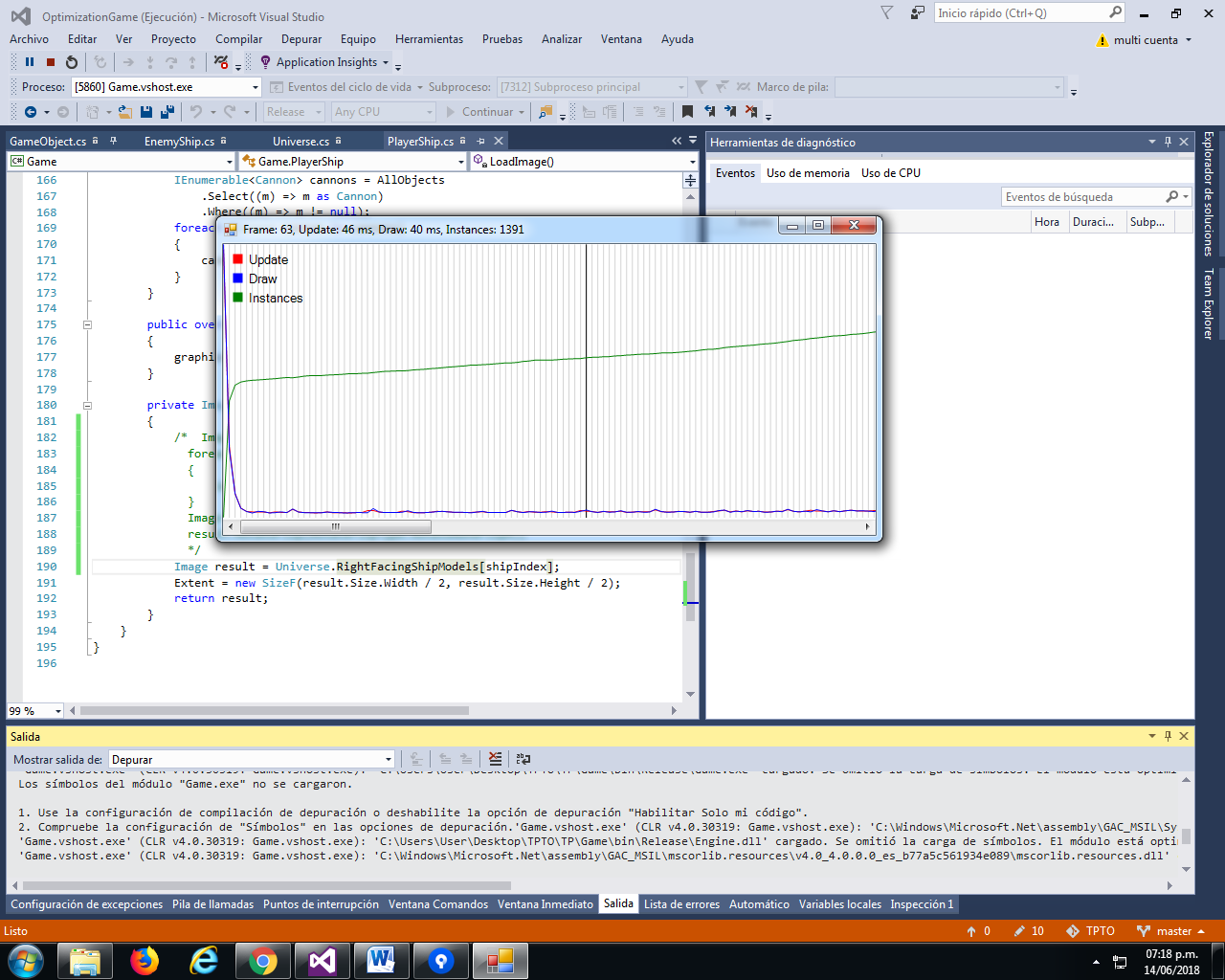
La optimización fue tan buena que el profiler no registra siquiera EnemyShip.LoadImage(), la performance incremento drásticamente:



Uno de los nuevos hotspots es PlayerShip.LoadImage, debería poder ganar performance de la misma forma en ese método.



El resultado fue el esperado:



Cosas para después: 