Primer análisis

Análisis sin Performance Profiler

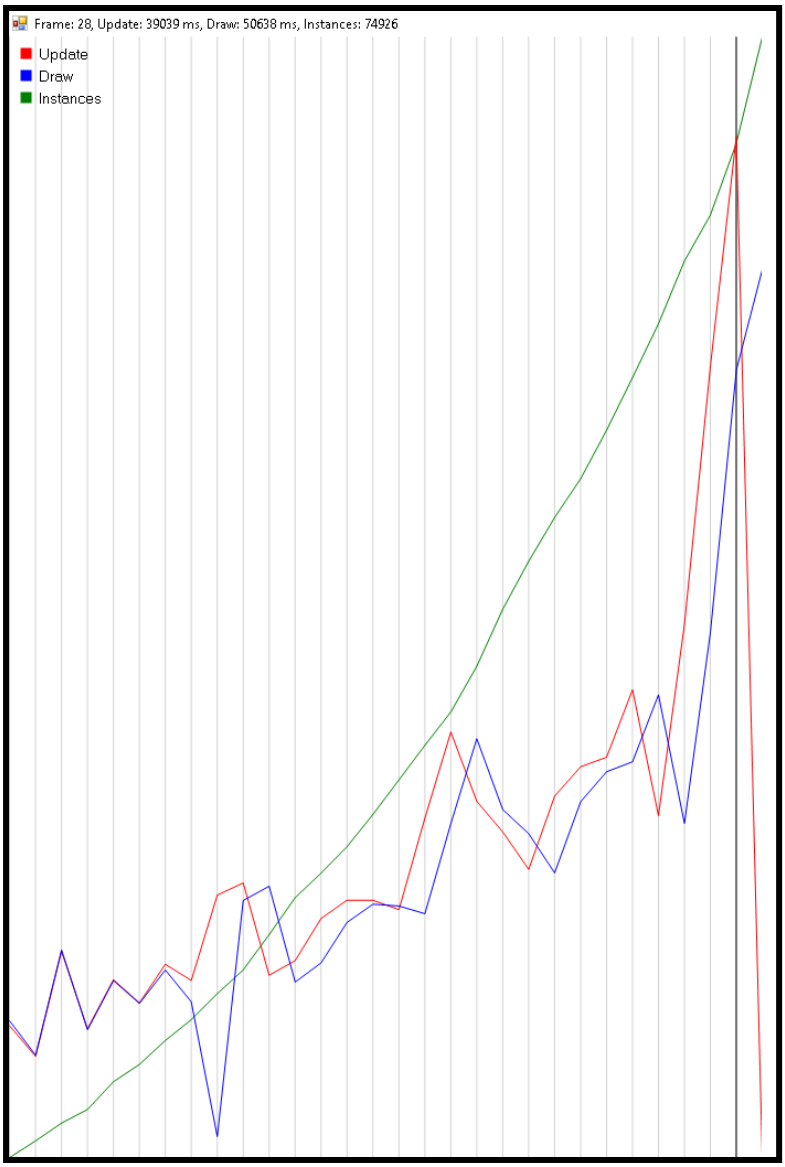
Lo primero que notamos al iniciar es que carga el fondo y la nave (Jugador), luego empieza a mostrar en el título de la ventana del juego las mediciones establecidas.

Al cabo de varios segundos notamos las estrellas generadas y luego aparece una nave enemiga por debajo de la nave del jugador.

Otro caso que se puede apreciar es la lentitud de cómo se desplaza hasta quedar casi congelado dejando en ejecución por varios minutos.

Teniendo varios minutos en proceso, no estamos obteniendo otra aparición de naves enemigas y estrellas que se pudo notar desde comienzo de la ejecución.

Gráficos sin utilizar el Performance Profiler en Debug



	A	B	C	D
1	frame	update	draw	instances
2	0	1224	252	15
3	1	7763	7038	1247
4	2	12786	12123	2557
5	3	23025	22420	3562
6	4	29417	28760	5612
7	5	38242	37541	6891
8	6	45921	45188	8659
9	7	55511	54497	10195
10	8	64298	62226	12116
11	9	77318	63269	13875
12	10	90940	76020	16482
13	11	99983	89479	19208
14	12	109756	98186	21020
15	13	121619	107847	22979
16	14	134384	119514	25354
17	15	147146	132083	27910
18	16	159447	144563	30463
19	17	176282	156658	32929
20	18	197389	173217	36299
21	19	215057	193990	40522
22	20	231206	211248	44059
23	21	245499	227312	47290
24	22	263425	241431	50153
25	23	282808	259080	53740
26	24	302661	278213	57622
27	25	325859	297841	61594
28	26	342819	320788	66239
29	27	369260	337359	69635
30	28	408500	363407	74926
31	29	459138	402446	82776
32				

Desde el primer testeo que aplicamos sin usar un Performance Profiler, al cerrar la aplicación se nos brinda poder analizar está información con las medidas programadas.

Desde este punto de vista, puedo apreciar dos factores que están tomando mucho proceso y son las actualizaciones (Línea roja) que se realizan como también la cantidad de objetos (Línea verde) que se están generando.

Sin entrar a comprobar el código fuente, puedo dar una primera hipótesis de que el primer problema son la cantidad de instancias exagerada, dando lugar a que la CPU esté teniendo cuello de botella ya que no puede procesar tanto los datos.

Con eso puedo pensar que entre el dibujado y las actualizaciones se genera una batalla entre ellos porque no logran recibir los datos que necesitan cada uno, mientras tanto los objetos no paran de generar uno tras otro.

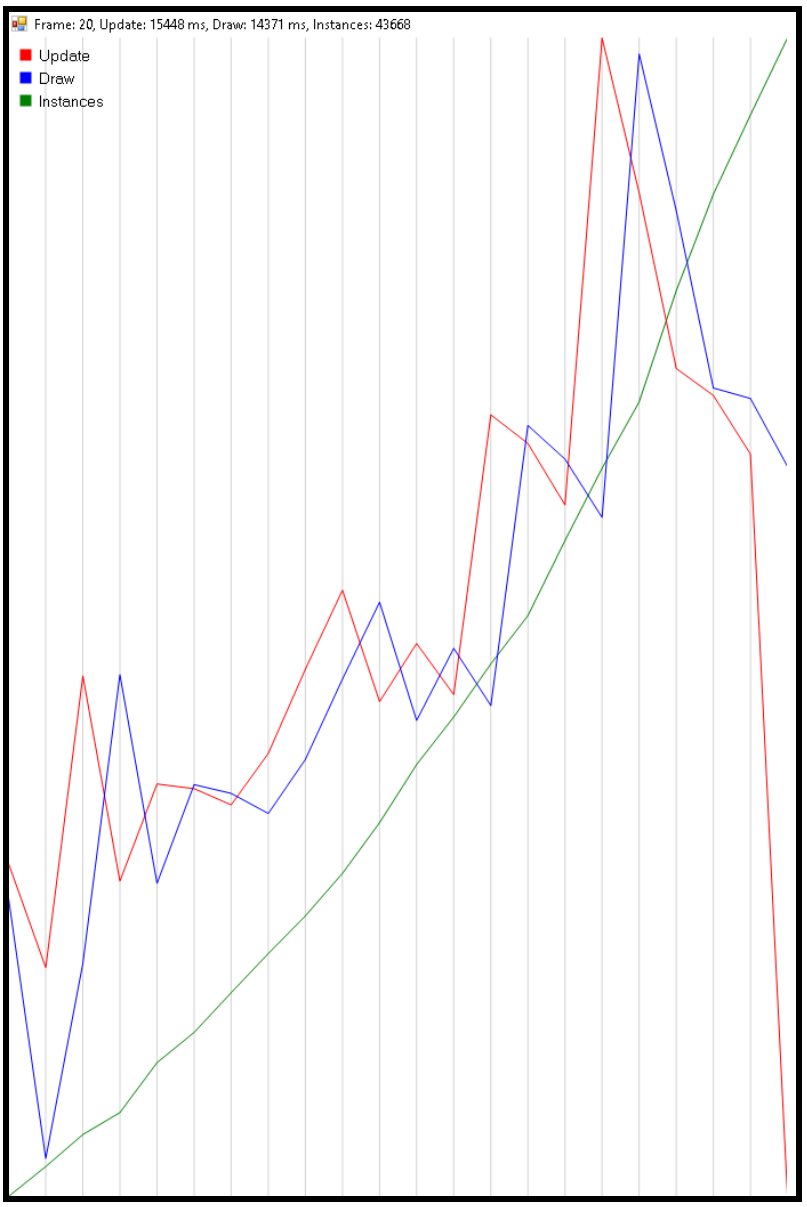
Conclusión, la CPU no recibe los datos en paralelo si no que reciben los datos atrasados y con mucha carga de información en conjunto.

Análisis con Performance Profiler en Release

Al iniciar el juego en Release se obtuvo un mejor resultado comparan en Debug.

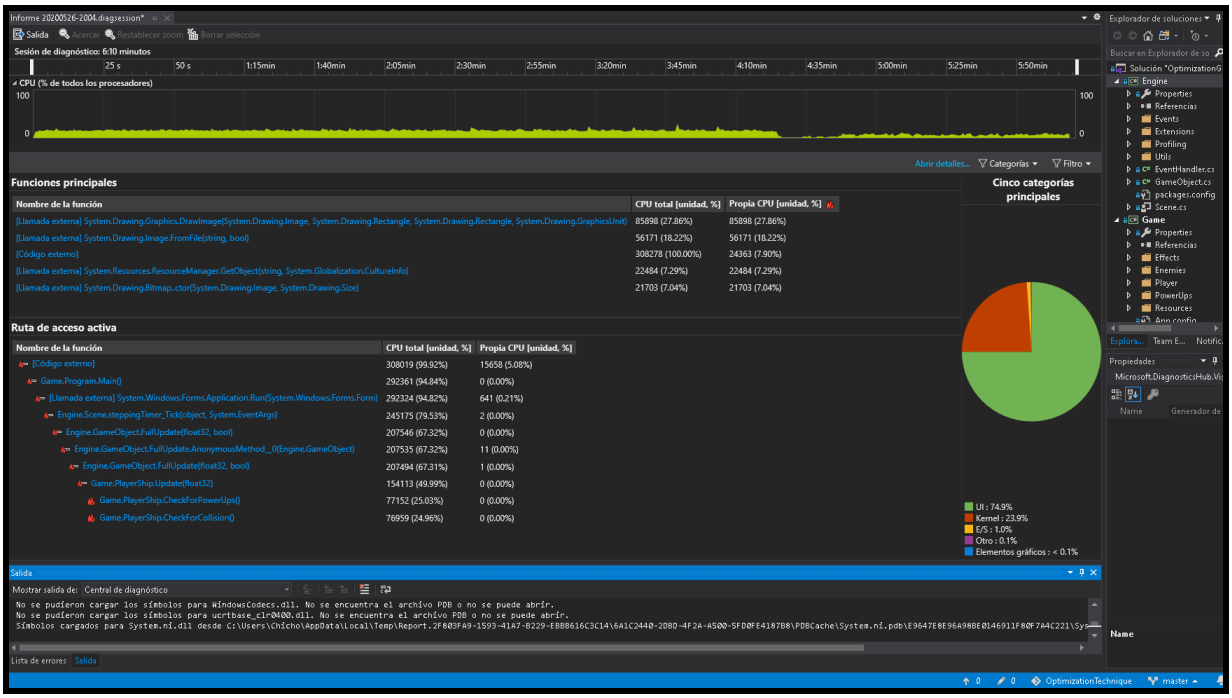
En el gráfico se puede apreciar que desde el inicio de la ejecución las actualizaciones y dibujados van casi de la misma mano hasta que entra en juego las instancia perdiendo el control y obtener resultados como cuellos de botellas. Se podía comprobar como dejaba de responder por unos segundos y luego al transcurrir con mayor cantidad de tiempo del proceso generando los cuelgues que se hacían notar con mayor duración.

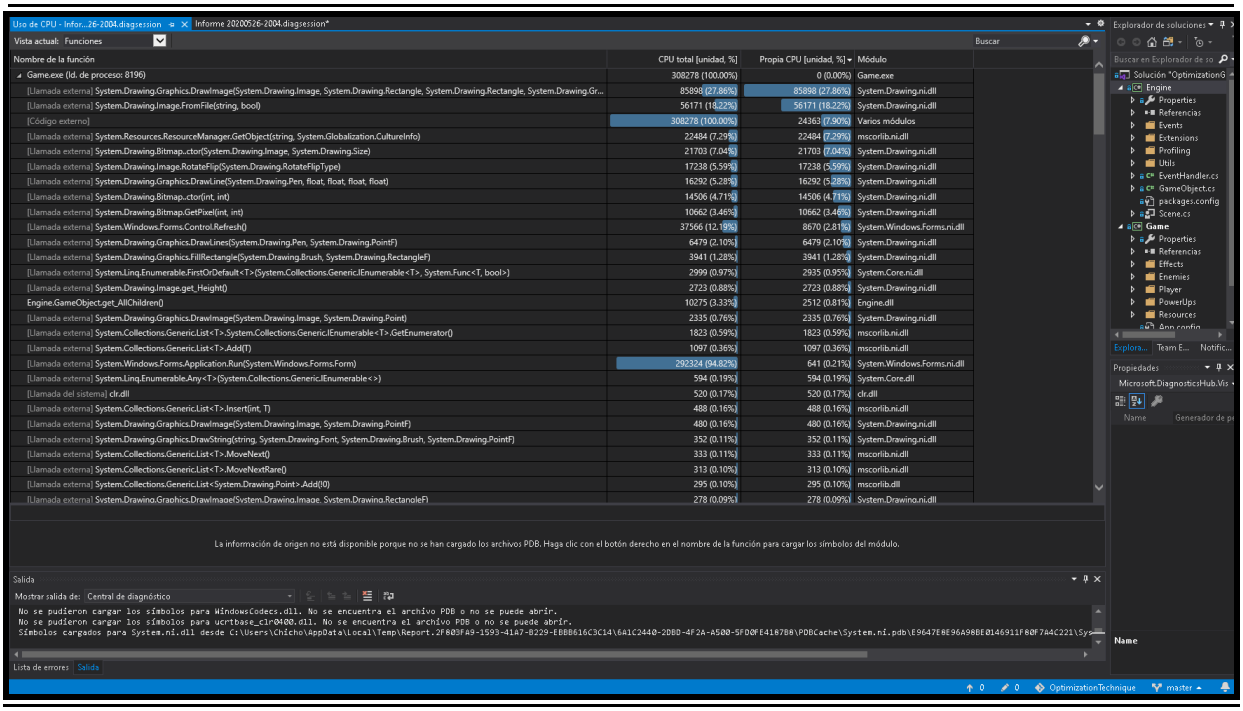
En la siguiente hoja se puede ver las mediciones con Performance Profiler.



	A	B	C	D
1	frame	update	draw	instances
2	0	1071	258	15
3	1	7517	6023	1219
4	2	11950	6763	2510
5	3	22028	11277	3396
6	4	28136	21381	5415
7	5	36124	27446	6635
8	6	44021	35421	8237
9	7	51604	43226	9817
10	8	60184	50642	11335
11	9	70392	59102	13053
12	10	82129	69119	15100
13	11	91710	80626	17448
14	12	102416	89842	19366
15	13	112134	100454	21509
16	14	127265	109959	23456
17	15	141842	124885	26487
18	16	155229	139158	29406
19	17	177653	152303	32084
20	18	197080	174418	36573
21	19	213107	193498	40462
22	20	228614	209148	43668
23	21	242985	224596	46774

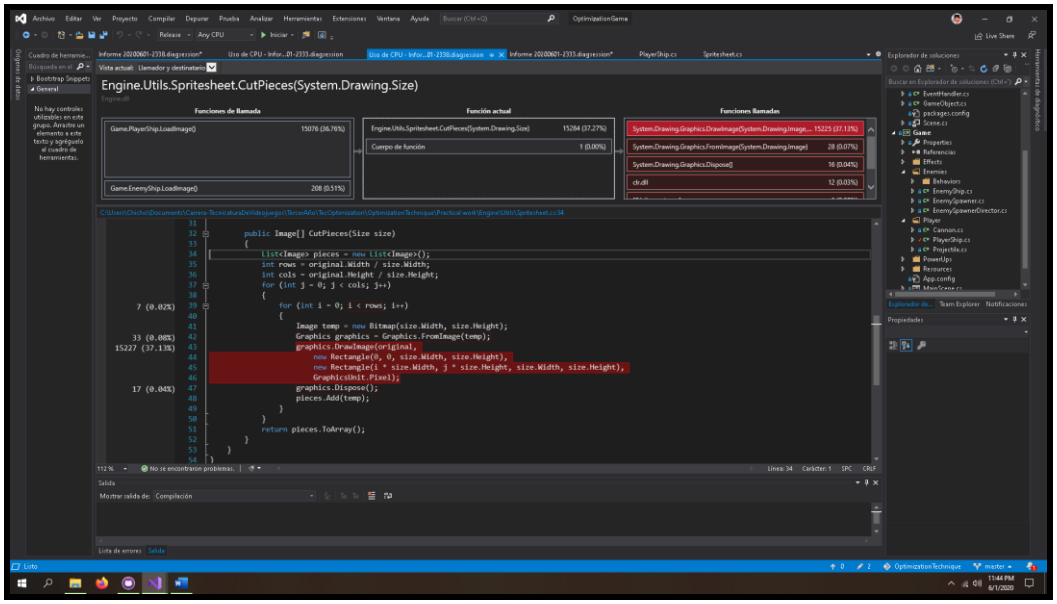
Gráficos generados con el Performance Profiler de Visual Studio





Desde mi punto de vista viendo la lista de uso de mi CPU noto que el mayor problema viene del dibujado y el refresco de la pantalla para dibujar sus posiciones, cargas y bitmap.

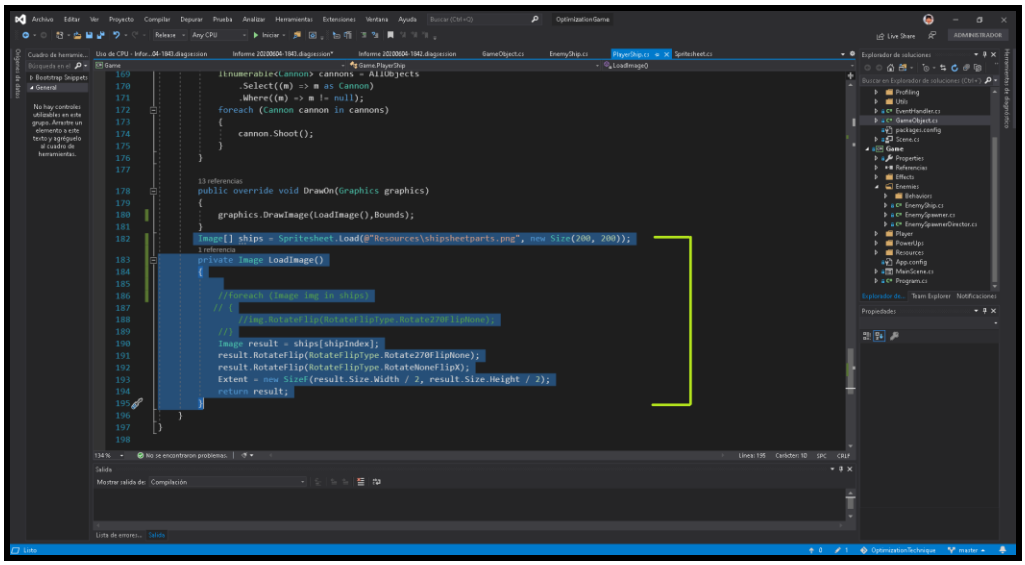
Analizando el código CutPieces



Este código se encarga de recorrer con un ciclo FOR recortando en cada posición una imagen que luego es almacenada en una lista.

Viendo que cada vez que se volvía a ejecutar el médoto LoadImage() en la clase del PlayerShip.cs, dentro de ese código volvia a cargar la imagen y ejecutar nuevamente el corte.

En clase PlayerShip.cs se quitó en el método Update() el método LoadImage(); y en el método LoadIamge(){} se realizó este cambio.



Se colocó la instrucción que se encontraba dentro del bloque del código LoadImage(); como memoria global para así poder cargar la imagen una sola vez.

Esto se logró mejorar el performance mejorando notablemente el rendimiento.