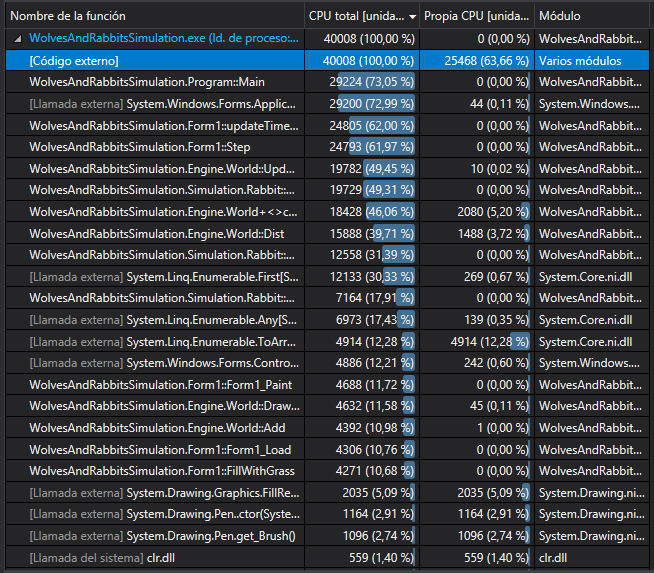
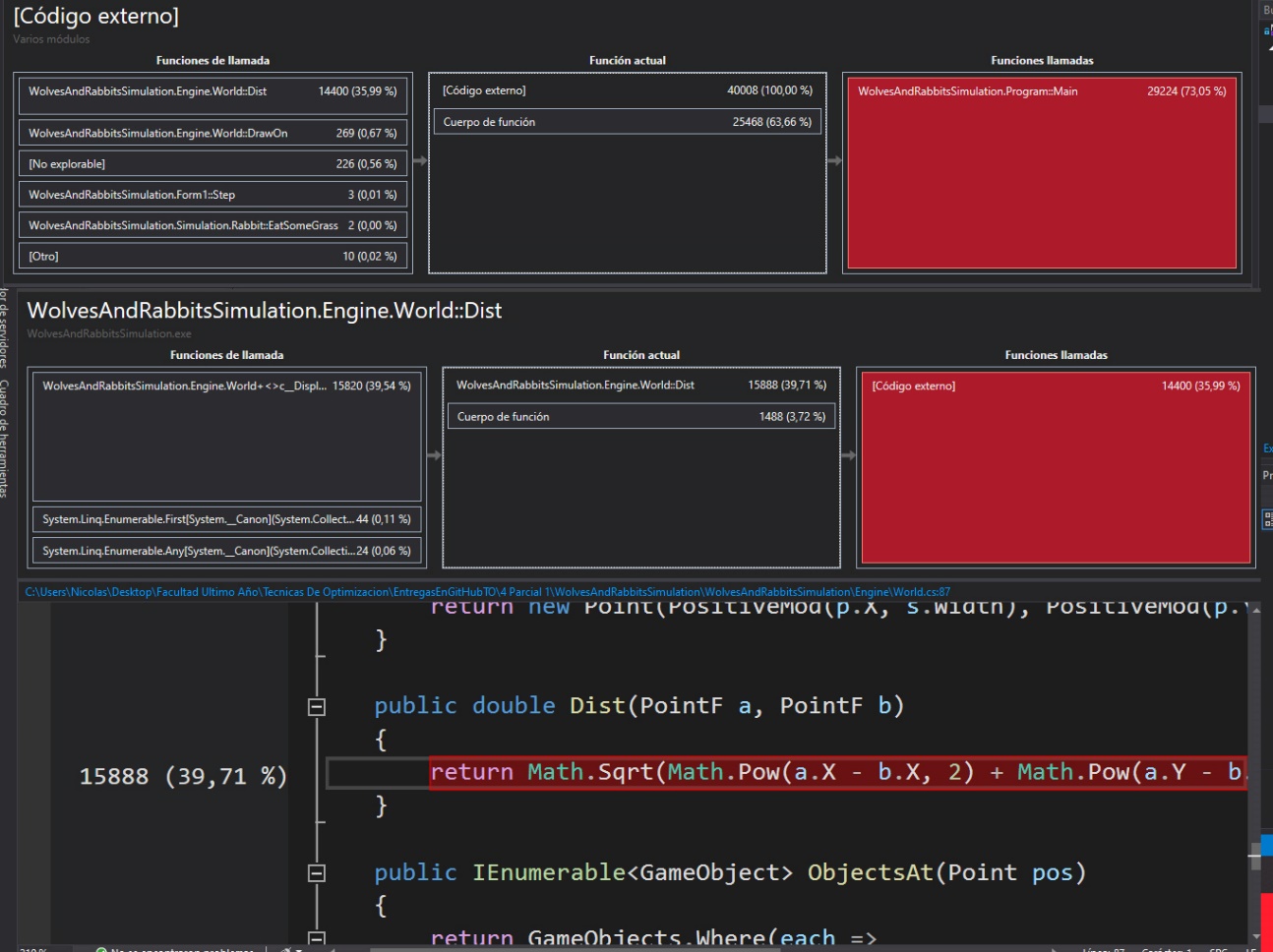
Parcial 1

Tendremos un código de una pequeña aplicación que simula el comportamiento de un grupo de conejos de forma simplificada.  
  
Existen 2 tipos de entidades: Rabbit, que representa a los conejos, y Grass, que representa al pasto que los conejos pueden comer. Los conejos se grafican con un cuadrado de color blanco y el pasto se grafica cuadrado cuyo color depende de cuán crecido esté el mismo.  
  
En cada frame, los conejos se moverán al azar con un por el escenario intentando comer el pasto que encuentren en sus inmediaciones y reproducirse con cualquier conejo que encuentren cerca. La reproducción sólo será posible si el conejo está en edad reproductiva y tiene suficiente comida. Cada conejo puede dar hasta 30 crías por frame. Si la cantidad de comida del conejo se acaba o si el conejo sobrevive luego de una cierta cantidad de frames, las chances de morir del mismo se incrementan. La simulación se reinicia si la cantidad de conejos vivos en algún momento llega a cero.  
  
Esta implementación está construida de forma intencionalmente ineficiente. Pueden modificar cualquier parte del código siempre y cuando mantengan la misma funcionalidad.  
  
Consignas  
1. Realizar mediciones de performance que permitan diagnosticar el problema.  
2. Escribir qué problemas encontró en la simulación en función de las mediciones.  
3. Proponer una solución al problema encontrado, describirla en sus palabras con el mayor detalle posible.  
4. Implementar la solución planteada haciendo los cambios en el código que sean necesarios.  
5. Validar la implementación realizando nuevas mediciones. ¿Se resolvió el problema? En caso negativo, ¿qué otras soluciones alternativas se le ocurren?  
  
PUNTOS EXTRA: Una vez resuelto el problema de optimización, incorporar una tercera entidad a la simulación: los lobos (Wolf, en inglés). Los lobos deberían deambular por el escenario buscando conejos para comer.

HIPOTESIS 1

1. Mediciones Tomadas





1. El código externo utiliza muchos recursos, y está ralentizando el programa, este es llamado por el método Dist, que a su vez es llamado por el método ObjectAT.

Revisando el código varias veces encontré que ObjectAT, recorría el vector con todos los elementos (rabit y grass) cada vez que un conejo quería comer, o reproducirse.

Otro problema que destaque es que tenemos un vector con todos los elementos dentro de la aplicación, y esto trae ciertos problemas al momento de recorrer el vector, ya que accedo a más elementos de los que quiero.

(Un Ejemplo de esto último: Si quiero que el conejo busque las posiciones de la Grass, también voy a recorrer los conejos).

Y un último problema que encontré fue que en el Forms, se encuentran 2 eventos que llaman a Step(), (el método Step() tiene la funcionalidad de llamar a diversas funciones para actualizar el programa).

Entre los eventos encontramos “updateTimer\_Tick” que llama a Step() cada cierto tiempo y “Form1\_Click” que también se encarga de llamar a la función Step(), este último está de más, solo con el“updateTimer\_Tick” debería bastar.

1. Principalmente lo que propongo son 3 cosas.

La primera es almacenar los rabits y la Grass en estructuras separadas y de mejor accesibilidad.

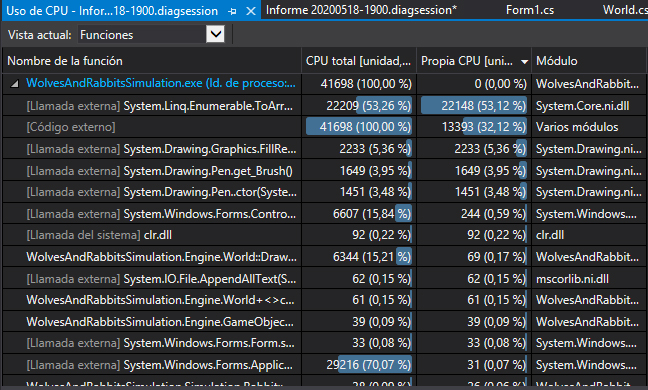
Lo segundo es modificar el método ObjectAT para que no realice tantos cálculos y el conejo pueda acceder mediante su posición, a el bloque de Grass que está pisando para comerlo directamente.

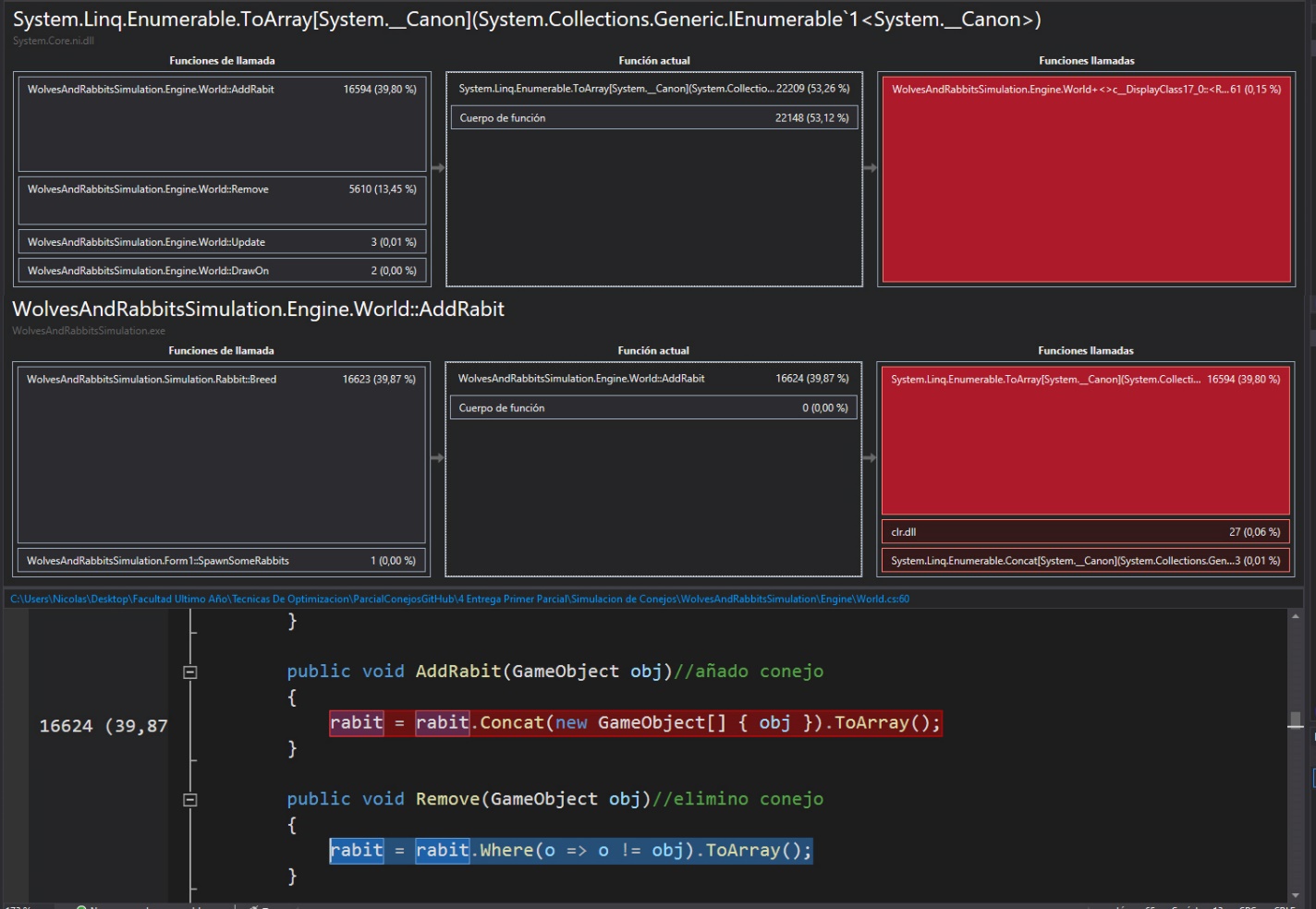
Lo tercero es Eliminar el “Form1\_Click” y dejas solamente el “updateTimer\_Tick”.

1. Los cambios Fueron implementados al código.
2. La implementación realizada fue positiva el programa mejoro mucho en los fps y el programa en si avanza más rápido en relación tiempo. El programa decae por otros problemas (que optimizare en la Hipotesis 2).

Hipotesis 2

1. Mediciones Tomadas





1. El Problema encontrado es que la librería de Linq consume muchos recursos y tiempo de la CPU (53,12%).

En estos recursos consumidos se puede destacar 2 funciones donde se encuentra principalmente el problema, la función AddRabit y Remove.

Estas 2 funciones son las que utilizan Linq.

1. La manera en el que se añaden los conejos a los vectores y se concatenan en “private GameObject[] rabit = new GameObject[0];” es muy lenta.

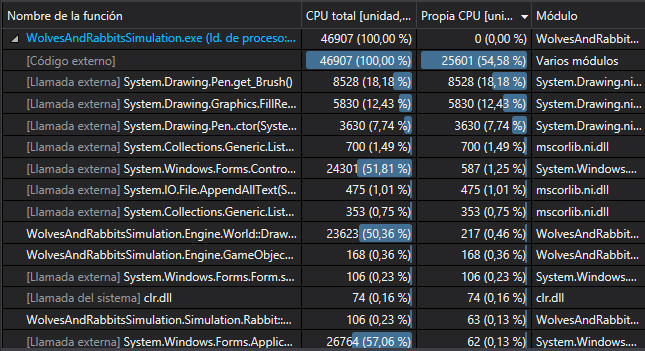
La solución que propongo es cambiar la estructura de datos que almacena a los conejos por una Lista y cambiar los métodos Add y Remove para que sean mas eficientes y no tengan que recorrer toda la estructura de datos para añadir o eliminar un elemento (Rabit).

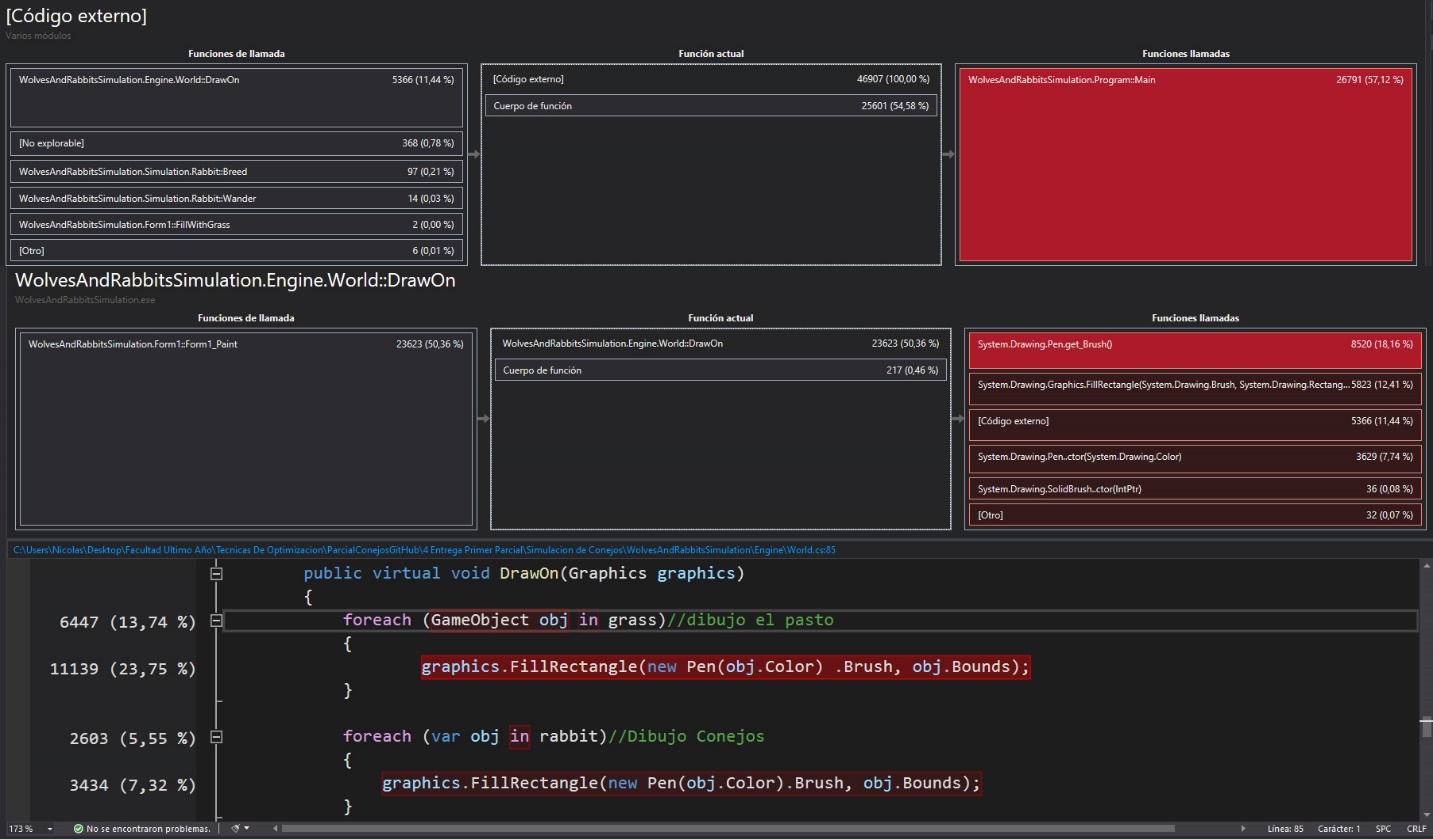
1. Los cambios Fueron implementados al código.
2. La implementación realizada fue positiva el programa mejoro mucho, ya no se cuelga al momento en el que los conejos se reproducen en masa. Por otro lado, el programa mejoro notablemente, si lo comparamos con la primera versión, pero surgió otro problema, al avanzar en el programa la memoria colapsa y se termina la ejecución tirando este error 🡪 OutOfMemoryException

Este error de memoria se arreglará en la (Hipotesis 3).

Hipotesis 3

1. Mediciones Tomadas





Otra medición tomada, es mirar el stopwatch que viene en el código por defecto, que muestra cómo se ralentiza los fps y frames del programa a medida que avanza, hasta saltar el error OutOfMemoryException.

1. El problema principal que encuentro es en la función DrawOn, ya que dibuja constantemente cada ves que se actualiza el programa (en la función Refresh()), hasta que se ralentiza el programa y luego colapsa con el error OutOfMemoryException ).

Este error se debe a que dibujar tanto consume toda la memoria hasta que el programa colapsa.

1. La solución que planteo es guardar los diferentes Sprites de la Grass y los conejos en una estructura de datos, y en ves de dibujar constantemente, accedo a la estructura y traigo los Bitmaps (Sprite)
2. Los cambios fueron implementados a media, ya que no logre guardar los sprites de Grass como bitmaps, pero a cambio logre guardar un vector con Pens, con todos los colores del Grass y rabbit.
3. Aunque la implementación deseada no se pudo implementar al 100%, el guardar los Colores redujo mucho la carga en la memoria, así que la implementación fue positiva.