Kaunas University of Technology

Faculty of Informatics

T120B162 Software Systems Testing

Todžės Warriors

Lab 4. Performance testing

Ugnius Rinkevičius	**Student**
Martynas Kuliešius	**Student**
Nedas Liaudanskis	**Student**
Paulius Osipauskas	**Student**

Prof. Lukas Arlauskas

Table of contents:

Part 1

Objective

The analysis aimed to evaluate CPU and memory usage during the execution of primary test cases for a SignalR-based game application using **Visual Studio's built-in diagnostic tools**. Key areas of focus:

1. Identify methods consuming the most CPU time.
2. Measure memory usage, including identifying the most allocated objects and their impact.

Results

CPU Profiling

- **Top CPU Consumers**:

  `System.Threading.ThreadPoolWorkQueue.Dispatch`:

  - **69.63%** of total CPU usage.
  - Manages threading and task execution within the application, essential for handling concurrent requests.

  `SignalR_Snake.Hubs.SnakeHub.AllPos()`:

  - **29.37%** of total CPU usage.
  - Processes and broadcasts positional updates to all connected clients.

  `SignalR_Snake.Hubs.SnakeHub.Timer_Elapsed()`:

  - **5.96%** of total CPU usage.
  - Handles periodic game state updates.

  `SignalR_Snake.Hubs.SnakeHub.SendDir(float)`:

  - **1.17%** of total CPU usage.
  - Updates player directions from client inputs.
- **Key Observations**:

  `AllPos()` is a significant CPU hotspot, likely due to frequent iterations and data processing for client synchronization.

  Thread pooling consumes a major share of CPU, reflecting the application's reliance on multithreading to manage SignalR communications.

Memory Profiling

- **Memory Usage Growth**:
  - Memory usage increased steadily throughout the test cases.

    - Garbage collection (GC) events were triggered consistently, but optimizations may be needed to reduce memory pressure.
- **Most Allocated Objects**:
    - **SignalR Messaging Objects**:

      `Microsoft.AspNet.SignalR.Messaging.DefaultSubscription`:

        - **~283 MB** inclusive size.
        - Indicates frequent subscriptions/unsubscriptions during gameplay.

      `List<Microsoft.AspNet.SignalR.Messaging.Message>`:

        - **~115 MB** allocated.
        - Represents the primary data structure for SignalR message handling.
    - **String Allocations**:

      Duplicate Strings:

        - Repeated allocations of strings like `"SnakeHub.ALLPOS()"` and `"SENDDIR()"`.
        - Total memory wasted by duplicates: **~5.48 MB**.

Key Insights and Recommendations

CPU Optimization

1. **Optimize `AllPos()`**:
    - Minimize lock usage where possible to avoid contention.
    - Batch position updates for clients to reduce the frequency of broadcast calls.
    - Consider using optimized data structures for handling positional updates.
2. **Thread Management**:
    - Reduce the overhead of thread pooling by optimizing task creation and processing.
    - Use `async/await` to improve asynchronous handling and reduce thread contention.

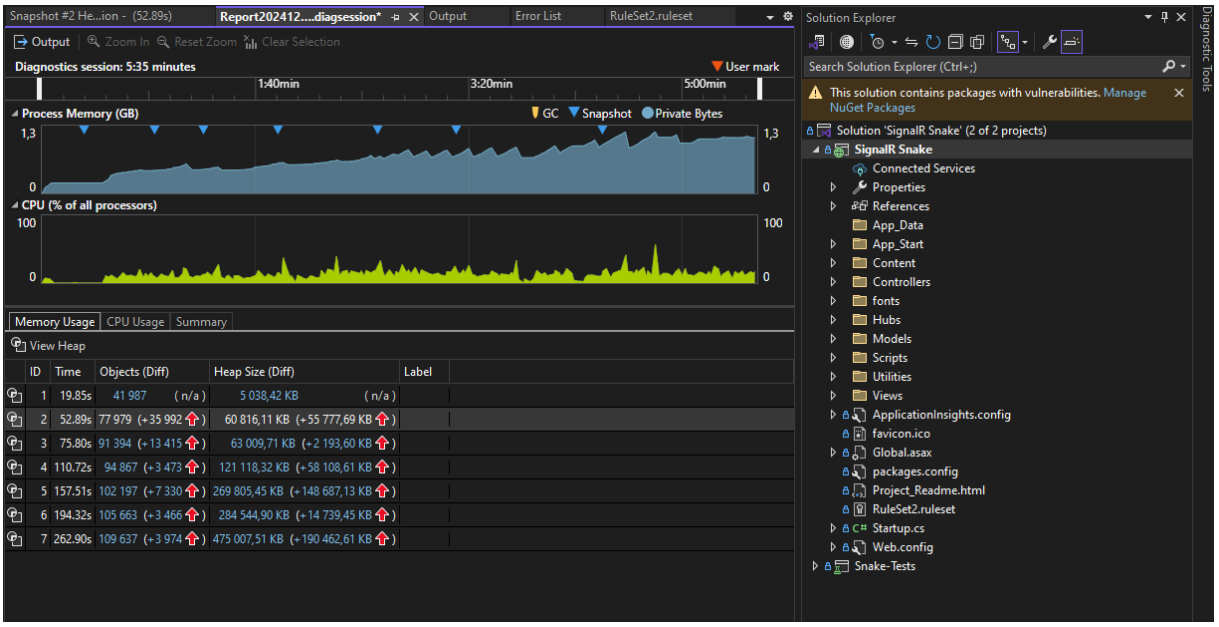
Memory Optimization

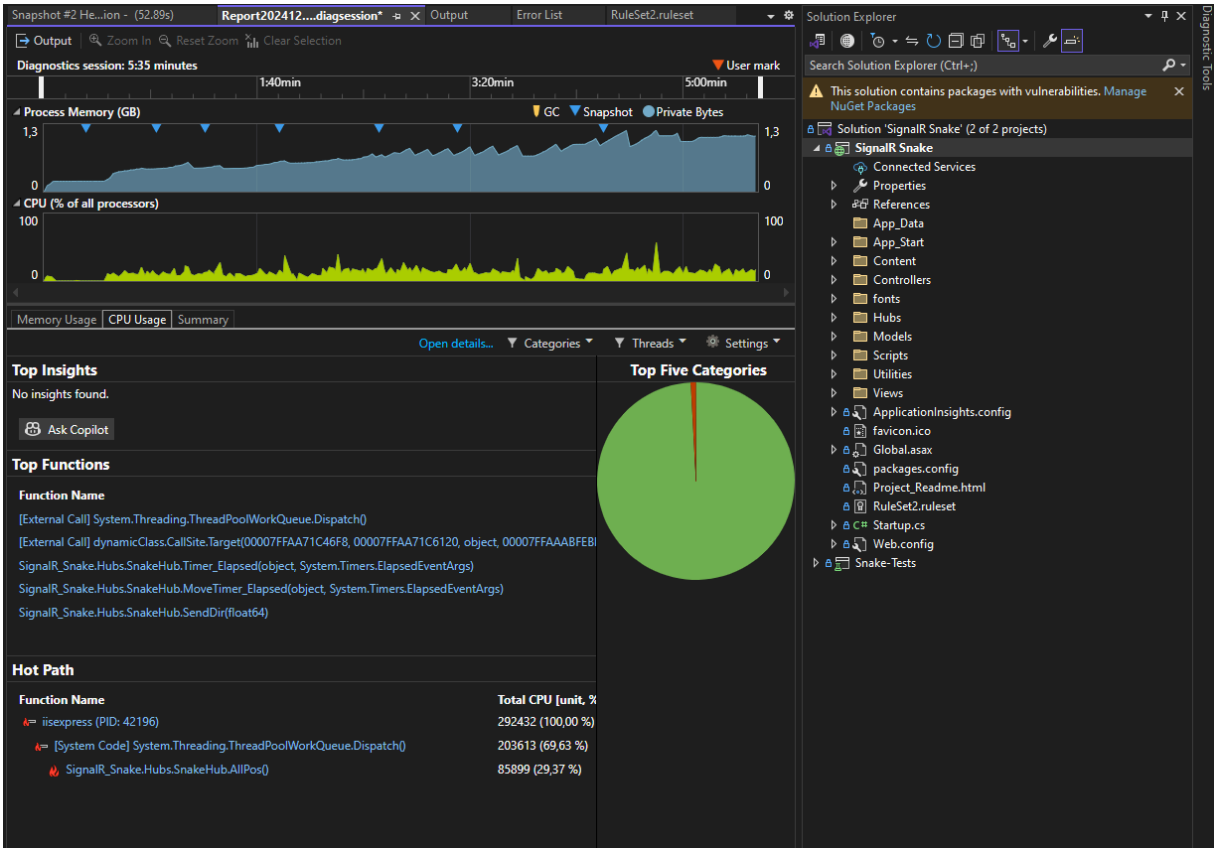
1. **Reduce String Duplication**:
    - Use `String.Intern` to manage repeated strings.
    - Consolidate frequently used strings into constants or enums.
2. **Reuse Objects**:
    - Implement object pooling for frequently created SignalR messaging objects (`DefaultSubscription` and `Message`).
    - Evaluate whether caching strategies can reduce allocations during high-frequency operations.
3. **Garbage Collection**:
    - Monitor and analyze GC activity patterns to ensure it is not disrupting gameplay performance.

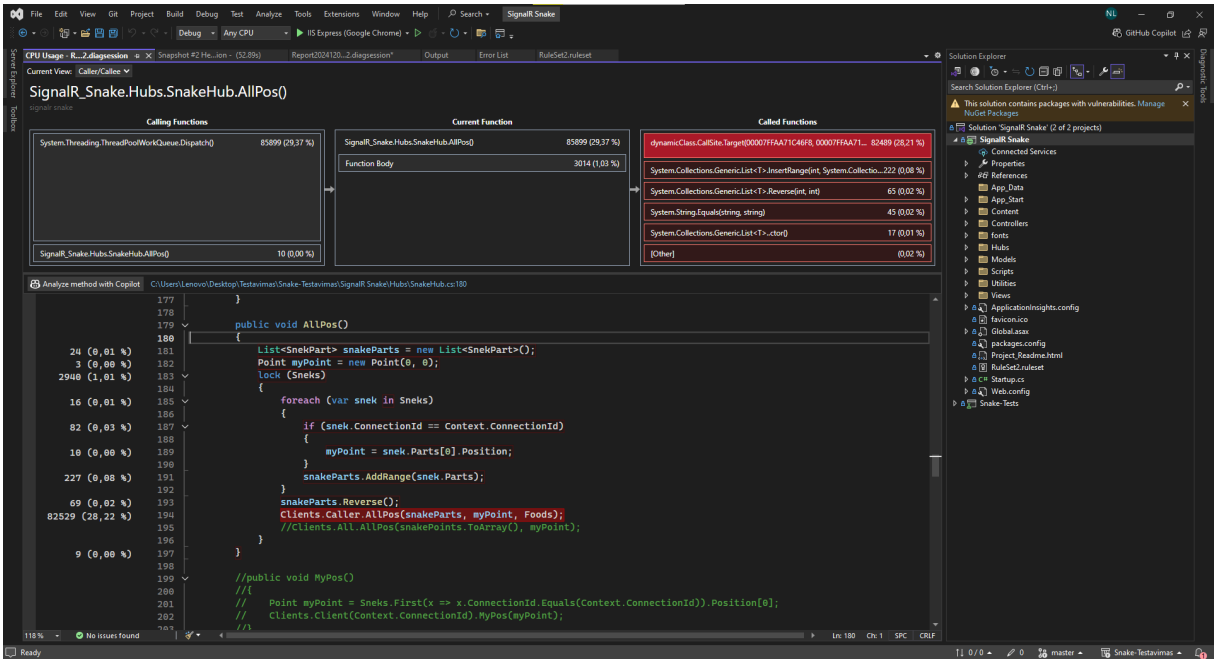
- Consider optimizing the lifetime of objects to reduce memory pressure and GC overhead.

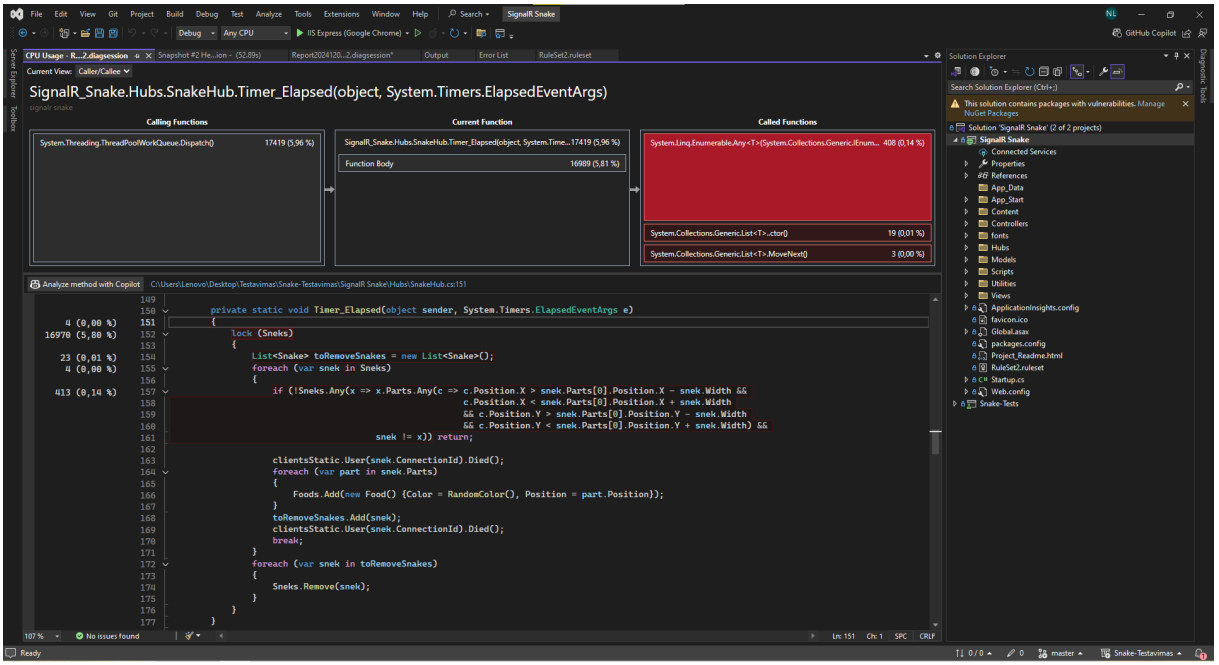
Summary

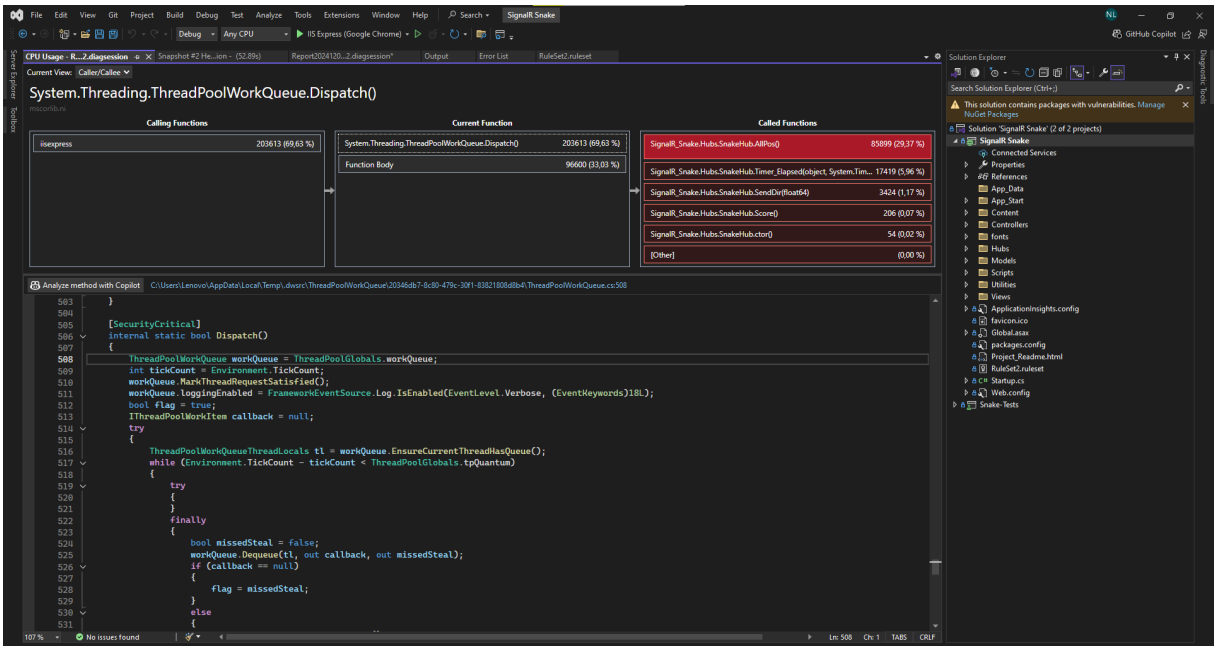
The profiling conducted using **Visual Studio** identified CPU hotspots (`AllPos()` and `ThreadPoolWorkQueue.Dispatch`) and memory inefficiencies (SignalR objects and duplicate strings). By addressing these areas, the application can achieve better performance and scalability, especially under high-concurrency scenarios. The system shows strong potential for improvement with targeted optimizations.




Snapshot #2 He...ion - (52.89s)
Report202412....diagsession*
Output
Error List
RuleSet2.ruleset
Diagnostics session: 5:35 minutes
User mark
1:40min
3:20min
5:00min
Process Memory (GB)
GC
Snapshot
Private Bytes
CPU (% of all processors)
Memory Usage
CPU Usage
Summary
Open details...
Categories
Threads
Settings
Top Insights
No insights found.
Ask Copilot
Top Five Categories
Top Functions
Function Name
[External Call] System.Threading.ThreadPoolWorkQueue.Dispatch()
SignalR_Snake.Hubs.SnakeHub.Timer_Elapsed(object, System.Timers.ElapsedEventArgs)
SignalR_Snake.Hubs.SnakeHub.MoveTimer_Elapsed(object, System.Timers.ElapsedEventArgs)
SignalR_Snake.Hubs.SnakeHub.SendDir(float64)
Hot Path
Function Name
Total CPU [unit, %
iisexpress (PID: 42196)
292432 (100,00 %)
[System Code] System.Threading.ThreadPoolWorkQueue.Dispatch()
203613 (69,63 %)
SignalR_Snake.Hubs.SnakeHub.AllPos()
85899 (29,37 %)
Solution Explorer
Search Solution Explorer (Ctrl+;)
This solution contains packages with vulnerabilities. Manage NuGet Packages
Solution 'SignalR Snake' (2 of 2 projects)
SignalR Snake
Connected Services
Properties
References
App_Data
App_Start
Content
Controllers
fonts
Hubs
Models
Scripts
Utilities
Views
ApplicationInsights.config
favicon.ico
Global.asax
packages.config
Project_Readme.html
RuleSet2.ruleset
Startup.cs
Web.config
Snake-Tests


SignalR_Snake.Hubs.SnakeHub.AllPos()
Calling Functions
Current Function
Called Functions
System.Threading.ThreadPoolWorkQueue.Dispatch() 85899 (29,37 %)
SignalR_Snake.Hubs.SnakeHub.AllPos() 85899 (29,37 %)
Function Body 3014 (1,03 %)
dynamicClass.CallSite.Target(00007FFAA71C46F8, 00007FFAA71... 82489 (28,21 %)
System.Collections.Generic.List<T>.InsertRange(int, System.Collectio... 222 (0,08 %)
System.Collections.Generic.List<T>.Reverse(int, int) 65 (0,02 %)
System.String.Equals(string, string) 45 (0,02 %)
System.Collections.Generic.List<T>..ctor() 17 (0,01 %)
[Other] (0,02 %)
SignalR_Snake.Hubs.SnakeHub.AllPos() 10 (0,00 %)
Analyze method with Copilot
C:\Users\Lenovo\Desktop\Testavimas\Snake-Testavimas\SignalR Snake\Hubs\SnakeHub.cs:180
public void AllPos()
List<SnekPart> snakeParts = new List<SnekPart>();
Point myPoint = new Point(0, 0);
lock (Sneks)
foreach (var snek in Sneks)
if (snek.ConnectionId == Context.ConnectionId)
myPoint = snek.Parts[0].Position;
snakeParts.AddRange(snek.Parts);
snakeParts.Reverse();
Clients.Caller.AllPos(snakeParts, myPoint, Foods);
//Clients.All.AllPos(snakePoints.ToArray(), myPoint);
//public void MyPos()
//{
// Point myPoint = Sneks.First(x => x.ConnectionId.Equals(Context.ConnectionId)).Position[0];
// Clients.Client(Context.ConnectionId).MyPos(myPoint);


SignalR_Snake.Hubs.SnakeHub.Timer_Elapsed(object, System.Timers.ElapsedEventArgs)
Calling Functions
Current Function
Called Functions
System.Threading.ThreadPoolWorkQueue.Dispatch() 17419 (5,96 %)
SignalR_Snake.Hubs.SnakeHub.Timer_Elapsed(object, System.Time... 17419 (5,96 %)
Function Body 16989 (5,81 %)
System.Linq.Enumerable.Any<T>(System.Collections.Generic.IEnum... 408 (0,14 %)
System.Collections.Generic.List<T>..ctor() 19 (0,01 %)
System.Collections.Generic.List<T>.MoveNext() 3 (0,00 %)
Analyze method with Copilot
private static void Timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
lock (Sneks)
List<Snake> toRemoveSnakes = new List<Snake>();
foreach (var snek in Sneks)
if (!Sneks.Any(x => x.Parts.Any(c => c.Position.X > snek.Parts[0].Position.X - snek.Width &&
c.Position.X < snek.Parts[0].Position.X + snek.Width
&& c.Position.Y > snek.Parts[0].Position.Y - snek.Width
&& c.Position.Y < snek.Parts[0].Position.Y + snek.Width) &&
snek != x)) return;
clientsStatic.User(snek.ConnectionId).Died();
foreach (var part in snek.Parts)
Foods.Add(new Food() {Color = RandomColor(), Position = part.Position});
toRemoveSnakes.Add(snek);
clientsStatic.User(snek.ConnectionId).Died();
break;
foreach (var snek in toRemoveSnakes)
Sneks.Remove(snek);
4 (0,00 %)
16970 (5,80 %)
23 (0,01 %)
4 (0,00 %)
413 (0,14 %)
No issues found
Ln: 151 Ch: 1 SPC CRLF
Ready
master
Snake-Testavimas


System.Threading.ThreadPoolWorkQueue.Dispatch()
Calling Functions
Current Function
Called Functions
iisexpress 203613 (69,63 %)
System.Threading.ThreadPoolWorkQueue.Dispatch() 203613 (69,63 %)
Function Body 96600 (33,03 %)
SignalR_Snake.Hubs.SnakeHub.AllPos() 85899 (29,37 %)
SignalR_Snake.Hubs.SnakeHub.Timer_Elapsed(object, System.Tim... 17419 (5,96 %)
SignalR_Snake.Hubs.SnakeHub.SendDir(float64) 3424 (1,17 %)
SignalR_Snake.Hubs.SnakeHub.Score() 206 (0,07 %)
SignalR_Snake.Hubs.SnakeHub.ctor() 54 (0,02 %)
[Other] (0,00 %)
Analyze method with Copilot
[SecurityCritical]
internal static bool Dispatch()
ThreadPoolWorkQueue workQueue = ThreadPoolGlobals.workQueue;
int tickCount = Environment.TickCount;
workQueue.MarkThreadRequestSatisfied();
workQueue.loggingEnabled = FrameworkEventSource.Log.IsEnabled(EventLevel.Verbose, (EventKeywords)18L);
bool flag = true;
IThreadPoolWorkItem callback = null;
try
ThreadPoolWorkQueueThreadLocals tl = workQueue.EnsureCurrentThreadHasQueue();
while (Environment.TickCount - tickCount < ThreadPoolGlobals.tpQuantum)
try
finally
bool missedSteal = false;
workQueue.Dequeue(tl, out callback, out missedSteal);
if (callback == null)
flag = missedSteal;
else
No issues found
Ln: 508 Ch: 1 TABS CRLF

Current View: Call Tree | Expand Hot Path | Show Hot Path | Reset Root

Function Name	Total CPU [unit, %]	Self CPU [unit, %]	Module	Category
iisexpress (PID: 42196)	292432 (100,00 %)	63736 (21,80 %)	Multiple modules	
[External Call] System.Threading.ThreadPo...	203613 (69,63 %)	96600 (33,03 %)	mscorlib.ni	
SignalR_Snake.Hubs.SnakeHub.AllPos()	85899 (29,37 %)	3004 (1,03 %)	signalr snake	
SignalR_Snake.Hubs.SnakeHub.Timer_Elaps...	17419 (5,96 %)	16989 (5,81 %)	signalr snake	
SignalR_Snake.Hubs.SnakeHub.SendDir(floa...	3424 (1,17 %)	3323 (1,14 %)	signalr snake	
SignalR_Snake.Hubs.SnakeHub.Score()	206 (0,07 %)	177 (0,06 %)	signalr snake	
SignalR_Snake.Hubs.SnakeHub.ctor()	54 (0,02 %)	13 (0,00 %)	signalr snake	
SignalR_Snake.Hubs.SnakeHub.MoveTimer_...	8 (0,00 %)	4 (0,00 %)	signalr snake	
SignalR_Snake.Hubs.SnakeHub.NewSnek(str...	3 (0,00 %)	1 (0,00 %)	signalr snake	
[External Call] System.Threading.TimerQueue....	17407 (5,95 %)	1239 (0,42 %)	mscorlib.ni	
[Unwalkable]	2416 (0,83 %)	2416 (0,83 %)		
[External Call] dynamicClass.IL_STUB_ReverseP...	2122 (0,73 %)	2114 (0,72 %)	system.web	
[External Call] dynamicClass.IL_STUB_ReverseP...	2106 (0,72 %)	2106 (0,72 %)	system.web	

CPU Usage - R...52.diagsession | Snapshot #2 He...ion - (52.89s) | Snapshot #1 H...on - (19.85s) | Report2024120...2.diagsession* | Output | Error List | RuleSet2.ruleset

Managed Memory

Compare With Baseline: None

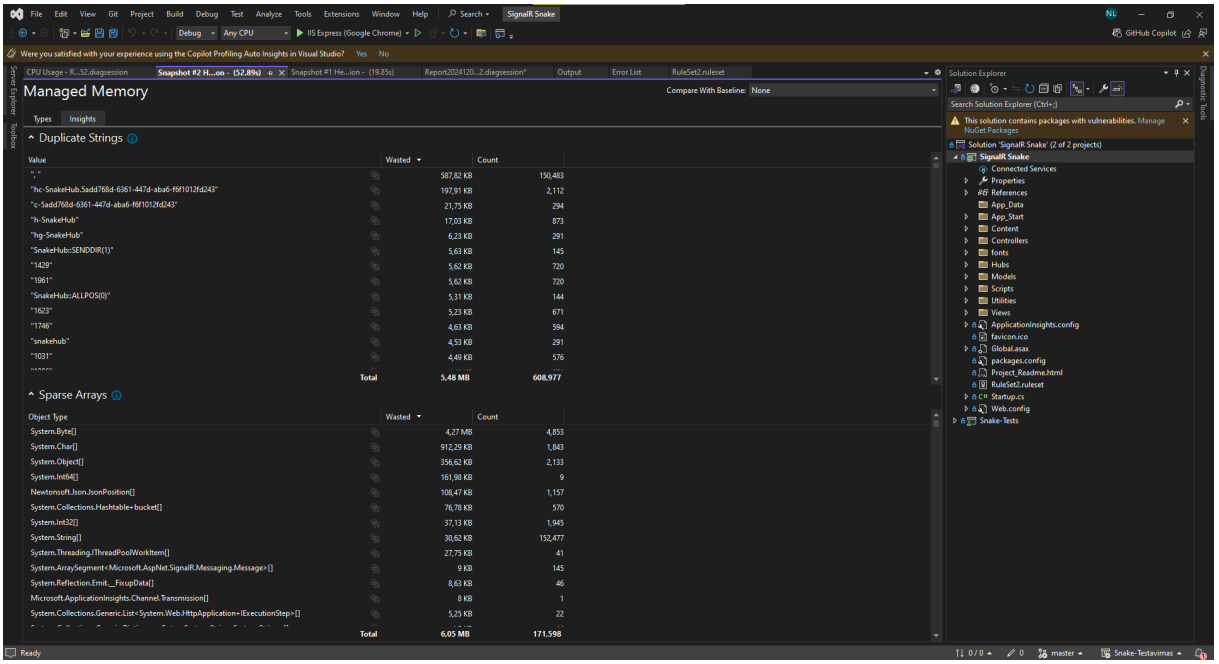
Types | Insights

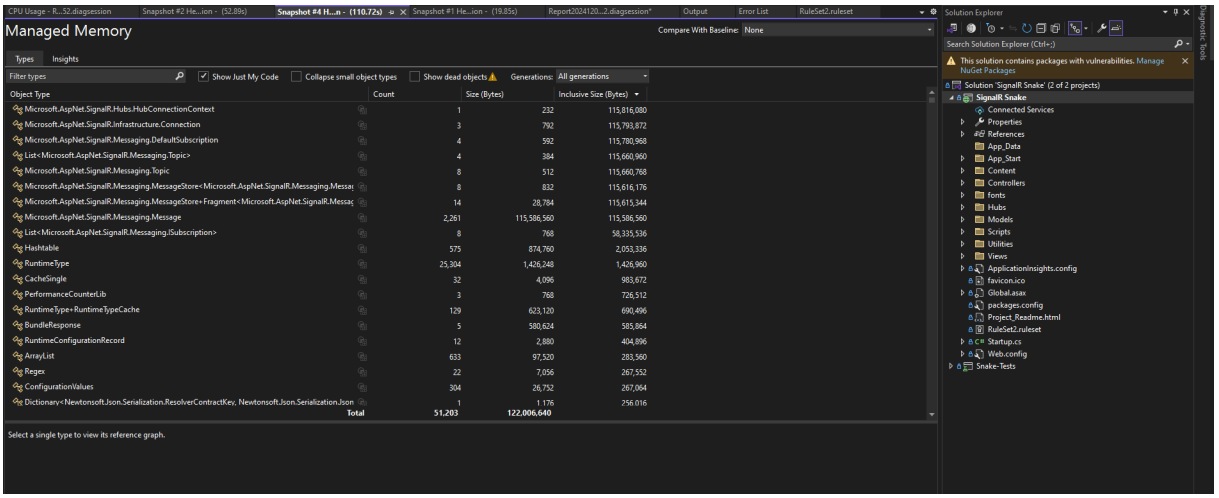
Duplicate Strings

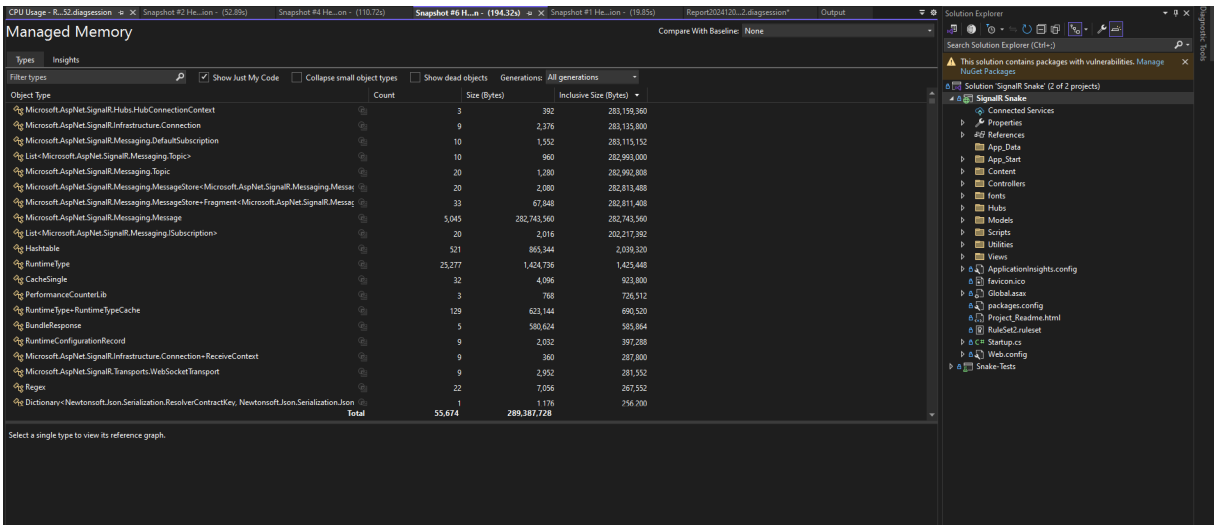
Value	Wasted	Count
"<mobileControls sessionStateHistorySize="6" cookielessDataDictionaryType="System.Web.Mobile.C	39,17 KB	3
"Semicolon insertion rules applied"	33,58 KB	522
"/* NUGET: BEGIN LICENSE TEXT\r\n *\r\n * Microsoft grants you the right to use these script files for	29,8 KB	20
" "	28,43 KB	14,559
"<httpHandlers>\r\n <add path="eurl.axd" verb="*" type="System.Web.HttpNotFoundHandle	25,93 KB	3
"<compilation>\r\n <assemblies>\r\n <add assembly="mscorlib"/>\r\n <ad	24,89 KB	3
"\r\n "	22,36 KB	637
"<healthMonitoring>\r\n <bufferModes>\r\n <add name="Critical Notification" max	17,55 KB	3
"002400000c8000001401000006020000002400005253413100080000010001006l3399aff18ef1a2c2514a273	16,88 KB	16
"\r\n "	15,61 KB	572
":"	12,17 KB	6,232
"t"	11,93 KB	6,107
"e"	11,88 KB	6,086
" "		
	Total 2,76 MB	203,109

Sparse Arrays

Object Type	Wasted	Count
System.Char[]	1,27 MB	29,557
Microsoft.Ajax.Utilities.AstNode[]	616,68 KB	22,870
System.Int32[]	463,5 KB	40,804
Microsoft.Ajax.Utilities.Context[]	424,18 KB	13,574
System.Collections.Hashtable+bucket[]	359,32 KB	4,513
System.Object[]	234,34 KB	6,583
System.Int64[]	199,5 KB	51
System.Collections.Generic.HashSet+Slot<Microsoft.Ajax.Utilities.Lookup>[]	182,88 KB	4,896
System.Collections.Generic.HashSet+Slot<Microsoft.Ajax.Utilities.INameReference>[]	165,77 KB	6,952
System.Byte[]	116,02 KB	2,612
System.Text.RegularExpressions.RegexNode[]	106,63 KB	4,747
System.Collections.Generic.HashSet+Slot<Microsoft.Ajax.Utilities.INameDeclaration>[]	106,27 KB	3,508
System.Collections.Generic.Dictionary+Entry<System.String, Microsoft.Ajax.Utilities.JSVariableField>[	68,81 KB	2,091
	Total 4,94 MB	181,234


Managed Memory
Duplicate Strings
Sparse Arrays


Managed Memory


Managed Memory

Part 2

Tools Used

- **Load Testing**: Apache JMeter
- **Profiler and Resource Monitoring**: Visual Studio Diagnostics Tools

---

Overview

The objective was to evaluate the performance of a .NET web game under stress conditions. The test simulated **1000 simultaneous users** for **5 minutes**, measuring response times, throughput, and resource usage. Additionally, a profiler was attached to analyze CPU and memory utilization during the load test.

Key Observations

1. Response Times and Throughput (JMeter Results)

From the JMeter outputs:

Graphs:

- Response times remained stable, with no significant spikes, indicating that the server handled the load effectively.
- Minimal latency suggests the backend is optimized for quick request handling.

Results Table:

- Sample times for HTTP requests averaged around **5 ms**, with no outliers beyond the acceptable limits.
- Pointer requests were faster, averaging **4 ms**, with some variability but no major delays.

Summary Report:

- **HTTP Requests**:
  - Average response time: **5 ms**
  - Minimum: **2 ms**, Maximum: **113 ms**
  - Throughput: **5.2 requests/second**
- **Pointer Requests**:
  - Average response time: **4 ms**
  - Minimum: **1 ms**, Maximum: **54 ms**
  - Throughput: **5.2 requests/second**
- **Overall Throughput**: **10.5 requests/second**
- **Error Rate**: **0%** for all requests.

Conclusion: The application maintained low and consistent response times with excellent throughput, effectively supporting the simulated 1000 users.

---

2. Resource Utilization (Visual Studio Profiler Results)

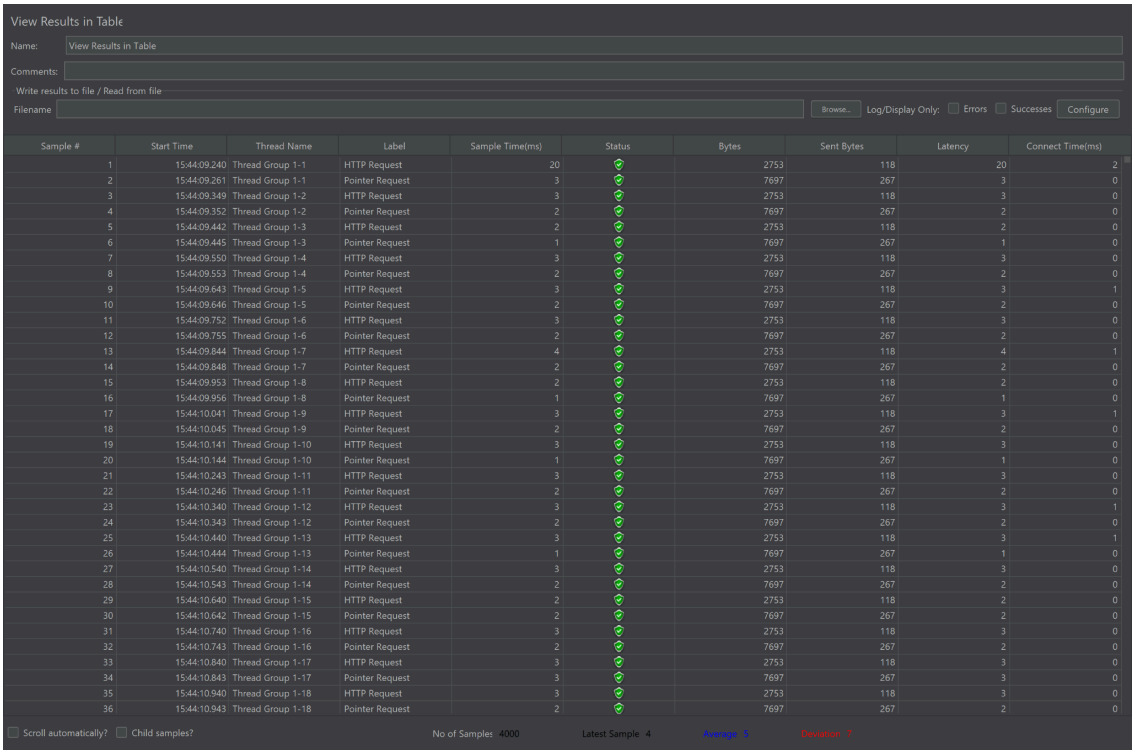
- **CPU Usage**:
  - CPU usage stayed under acceptable thresholds, with only minor peaks during the heaviest loads.
  - This suggests the application efficiently processed incoming requests without straining the processor.
- **Memory Usage**:
  - Memory usage steadily increased over the test duration but plateaued after a certain point.
  - **Garbage Collection**:
    - Regular garbage collection was observed, effectively reclaiming memory and preventing leaks or excessive allocations.
- **Conclusion**: Resource usage was stable and optimized. There were no signs of bottlenecks or memory leaks.

Analysis

- **Performance**: The application demonstrated strong performance with:
  - Consistent response times.
  - High throughput of 10.5 requests per second.
  - No errors across all requests.
- **Stability**:
  - Both CPU and memory usage patterns indicate a well-optimized system capable of handling high loads without significant strain.

Recommendations

1. **Scalability Testing**:
   - Extend the test to simulate higher concurrency levels (e.g., 2000-5000 users) to confirm scalability.
   - Use Apache JMeter's distributed testing mode if required.
2. **Continuous Monitoring**:
   - Deploy real-time monitoring tools like Application Insights (for Azure) or Grafana (with Prometheus) to observe live system performance in production.
3. **Performance Tuning**:
   - Investigate minor response time variations in HTTP and Pointer requests under high load to ensure consistent performance across all endpoints.
4. **Further Profiling**:
   - Attach profilers to measure database performance and investigate specific application logic under heavier loads.



View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: Errors Successes Configure

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	15:44:09.240	Thread Group 1-1	HTTP Request	20		2753	118	20	2
2	15:44:09.261	Thread Group 1-1	Pointer Request	3		7697	267	3	0
3	15:44:09.349	Thread Group 1-2	HTTP Request	3		2753	118	3	0
4	15:44:09.352	Thread Group 1-2	Pointer Request	2		7697	267	2	0
5	15:44:09.442	Thread Group 1-3	HTTP Request	2		2753	118	2	0
6	15:44:09.445	Thread Group 1-3	Pointer Request	1		7697	267	1	0
7	15:44:09.550	Thread Group 1-4	HTTP Request	3		2753	118	3	0
8	15:44:09.553	Thread Group 1-4	Pointer Request	2		7697	267	2	0
9	15:44:09.643	Thread Group 1-5	HTTP Request	3		2753	118	3	1
10	15:44:09.646	Thread Group 1-5	Pointer Request	2		7697	267	2	0
11	15:44:09.752	Thread Group 1-6	HTTP Request	3		2753	118	3	0
12	15:44:09.755	Thread Group 1-6	Pointer Request	2		7697	267	2	0
13	15:44:09.844	Thread Group 1-7	HTTP Request	4		2753	118	4	1
14	15:44:09.848	Thread Group 1-7	Pointer Request	2		7697	267	2	0
15	15:44:09.953	Thread Group 1-8	HTTP Request	2		2753	118	2	0
16	15:44:09.956	Thread Group 1-8	Pointer Request	1		7697	267	1	0
17	15:44:10.041	Thread Group 1-9	HTTP Request	3		2753	118	3	1
18	15:44:10.045	Thread Group 1-9	Pointer Request	2		7697	267	2	0
19	15:44:10.141	Thread Group 1-10	HTTP Request	3		2753	118	3	0
20	15:44:10.144	Thread Group 1-10	Pointer Request	1		7697	267	1	0
21	15:44:10.243	Thread Group 1-11	HTTP Request	3		2753	118	3	0
22	15:44:10.246	Thread Group 1-11	Pointer Request	2		7697	267	2	0
23	15:44:10.340	Thread Group 1-12	HTTP Request	3		2753	118	3	1
24	15:44:10.343	Thread Group 1-12	Pointer Request	2		7697	267	2	0
25	15:44:10.440	Thread Group 1-13	HTTP Request	3		2753	118	3	1
26	15:44:10.444	Thread Group 1-13	Pointer Request	1		7697	267	1	0
27	15:44:10.540	Thread Group 1-14	HTTP Request	3		2753	118	3	0
28	15:44:10.543	Thread Group 1-14	Pointer Request	2		7697	267	2	0
29	15:44:10.640	Thread Group 1-15	HTTP Request	2		2753	118	2	0
30	15:44:10.642	Thread Group 1-15	Pointer Request	2		7697	267	2	0
31	15:44:10.740	Thread Group 1-16	HTTP Request	3		2753	118	3	0
32	15:44:10.743	Thread Group 1-16	Pointer Request	2		7697	267	2	0
33	15:44:10.840	Thread Group 1-17	HTTP Request	3		2753	118	3	0
34	15:44:10.843	Thread Group 1-17	Pointer Request	3		7697	267	3	0
35	15:44:10.940	Thread Group 1-18	HTTP Request	3		2753	118	3	0
36	15:44:10.943	Thread Group 1-18	Pointer Request	2		7697	267	2	0

Scroll automatically? Child samples? No of Samples 4000 Latest Sample 4 Average 5 Deviation 7

Summary Report

Name: Summary Report

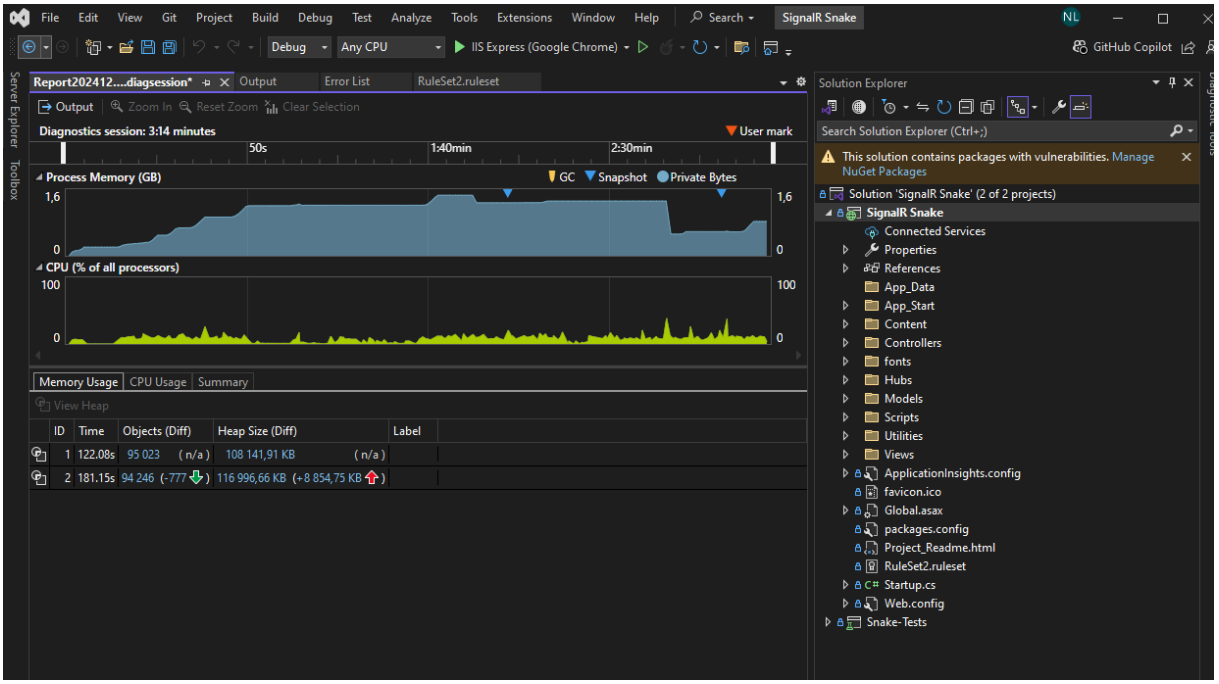
Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	2000	5	2	113	7,42	0,00%	5,2/sec	14,06	0,60	2753,0
Pointer Request	2000	4	1	54	7,84	0,00%	5,2/sec	39,30	1,36	7697,0
TOTAL	4000	5	1	113	7,65	0,00%	10,5/sec	53,36	1,97	5225,0

Include group name in label? Save Table Data Save Table Heade



Summary

The combination of **Apache JMeter** for load testing and **Visual Studio Profiler** for resource analysis proved effective in evaluating the application. The web game successfully supported 1000 users, maintaining low response times, high throughput, and efficient resource utilization. The system is well-optimized for current load requirements and shows readiness for scaling to higher user loads.