



## Age of Empires Toolkit

### Group Members

Tran Trong Nghia	22BI13332
Nguyen Thanh Nam	22BI13325
Tran Duc Duong	22BI13117
Nguyen Tuan Dung	22BI13107
Truong Dai An	22BI13008
Vu Hung Anh	22BI13045
Doan Trung Kien	BI12-226

### Group Project Report

on January 9, 2025

#### Supervisor

Prof. Huynh Vinh Nam

# Abstract

Age of Empires is one of the most iconic real-time strategy (RTS) game franchises, renowned for its depth in strategy, historical accuracy, and community-driven expansions. However, managing and customizing game scenarios, optimizing strategies, and balancing units can be complex and time-consuming for players, modders, and developers. To address these challenges, this project introduces a specialized Age of Empires Toolkit that leverages modern software techniques to simplify game modification, map design, and strategic analysis. This toolkit incorporates features such as unit and resource balancing using Dll and reverse engineering. By enhancing accessibility and efficiency in AoE customization, the toolkit aims to empower the AoE community in creating more engaging and personalized gameplay experiences.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Table of Contents</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context and Motivation . . . . .	1
1.2 Objectives . . . . .	2
<b>2 Methodology</b>	<b>3</b>
2.1 Architecture . . . . .	3
2.1.1 DLL Injection . . . . .	3
2.1.2 Attach Process . . . . .	4
2.2 User Interface . . . . .	5
2.3 Operations . . . . .	5
2.3.1 Toggle steroids . . . . .	6
<b>3 Research</b>	<b>12</b>
3.1 Tools . . . . .	12
3.2 Memory Scanning . . . . .	12
3.3 Retrieve Multi-Level Pointers . . . . .	15
3.4 Modify Memory Bytes . . . . .	16
3.5 Reversing Game Logic . . . . .	17
3.5.1 Game Settings Reversing . . . . .	17
3.5.2 Game Logic Reversing . . . . .	18
<b>4 Results</b>	<b>24</b>

# Chapter 1

## Introduction

### 1.1 Context and Motivation

Real-time strategy (RTS) games (in this case, Age of Empires) have long been popular for their strategic depth and immersive gameplay. Because these aspects are as complex as they are addictive, both gamers and developers face challenges that limit their ability to fully explore and maximize the potential of the game.

Players often engage in tasks such as resource management, base building, and combat, which require a combination of planning and quick decision-making. To succeed, they must have a deep understanding of the game, including the strengths and weaknesses of each civilization, the advantages of various map types, and decisions on which units or upgrades to invest resources in. This process is time-consuming, prone to human error, and not everyone is willing to dedicate such significant effort to mastering the game. Unfortunately, despite the game's widespread appeal, there is limited support for tools that help players analyze their gameplay, optimize strategies, and better understand the game's mechanics.

As players often push the game to its limits to discover optimal strategies for both offline campaigns and competitive multiplayer, developers face the monumental task of planning ahead and testing all possible scenarios. Balancing an RTS game like Age of Empires is particularly challenging due to the many interconnected elements in each match. Even a minor change to a unit's stats, the cost of an upgrade, or the layout of a map can have ripple effects that impact the entire game ecosystem.

To address these challenges, developers require a professional support tool that not only assists in testing but also provides in-depth analytics to evaluate how changes affect gameplay balance. For players, the same tool could serve as an invaluable resource to help them better understand the game, develop strategies, and improve their overall performance.

Our AoE Toolkit main goal is to solve all those problems. Its features include gameplay analysis of what player have archived, total control of game elements for developers, and user-friendly interfaces for all users. By addressing the needs of both sides, the toolkit enhances the Age of Empires experience, making the game more accessible, balanced, and enjoyable for everyone involved.

## 1.2 Objectives

The main objectives are expected to be achieved in this group project:

1. Develop a toolkit as a software that has user-friendly ui, and is easy-to-use for both player and developer wanting to test new ideas
2. Implement and conduct experiments of the changing of data flow in the game structure

# Chapter 2

## Methodology

### 2.1 Architecture

#### 2.1.1 DLL Injection

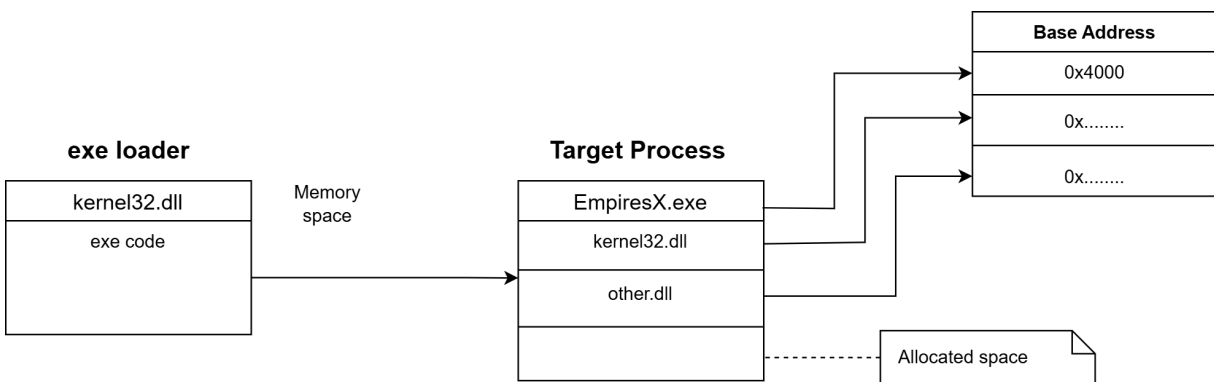


Figure 1: Dll Injection architecture

- Call WinAPI **OpenProcess()** to retrieve handles to a target Process, these Handles will be used in others API.

- Allocate space for the preparation of the injection. The amount of space on target process will be equal to the string "Path/to/dll". This string will be pass to LoadLibraryA() API.

API used: **VirtualAllocEx()**

- From exe loader , use CreateRemoteThroed() API to create a thread inside Target process, which does the following:

- Create a thread that runs inside the target process.
- Create RemoteThread() take several arguments.

Important Arguments:

- Handle the target process.
- `LPThread_Start_Routine`: A task that executes when thread is created.

⇒ Set `LPThread_Start_Routine` to **LoadLibraryW()**.

⇒ Once a thread is created inside the target process, it set to load a library which is a dll.

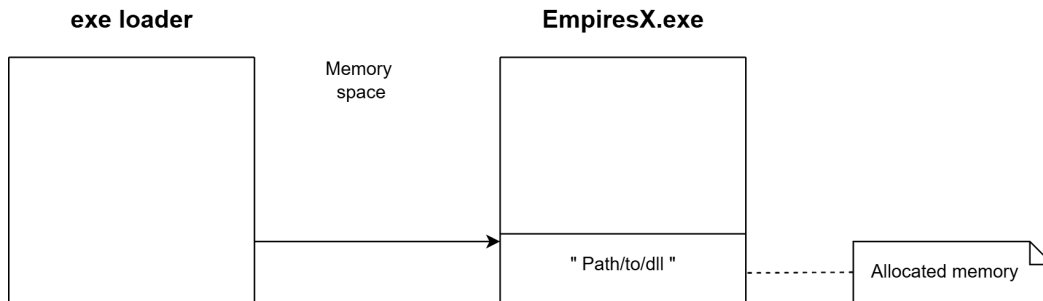


Figure 2: Something unknown

- `CreateRemoteThread(PathLibraryW("Path/to/dll"))`.
- Once the thread is created in `EmpiresX.exe`, it set to load the target dll.

### 2.1.2 Attach Process

Theory: External program use export function from dll. This external program will attach to the target process

- This approach works similarly to dll injection. The only difference is: each time it operates a function, it creates a thread in the target process.
- Once the dll injection is created and from the moment the dll is loaded, the functionality of the dll also become the functionality of the targeted process(the process that load the dll).

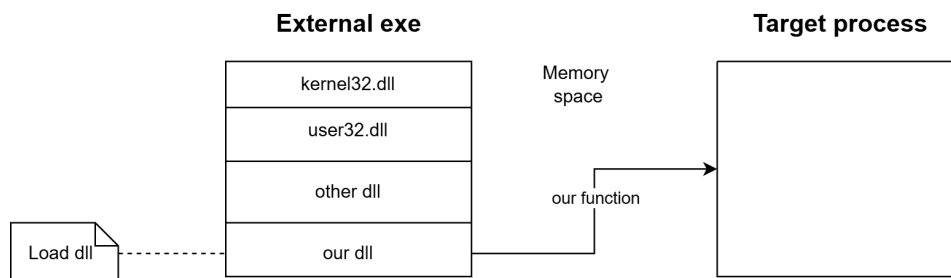


Figure 3: Something unknown

- A dll can have export function that can be loaded and used by others exe.

```

Eg: kernel32.dll ----- OpenProcess()
      |
      ----- CreateThread()
advapi.dll ----- RegOpenKeyA()
      |
      ----- RegSetValueExA()

```

- Same with our dll. Just create it with export function then use it.

## 2.2 User Interface

The user interface of the software is simple and user-friendly, and there is a reason for that. We want the user to focus and understand the underlying process like trainer, data flow, other behind-the-scenes operations. For added convenience, all functions can also be operated using hotkeys.

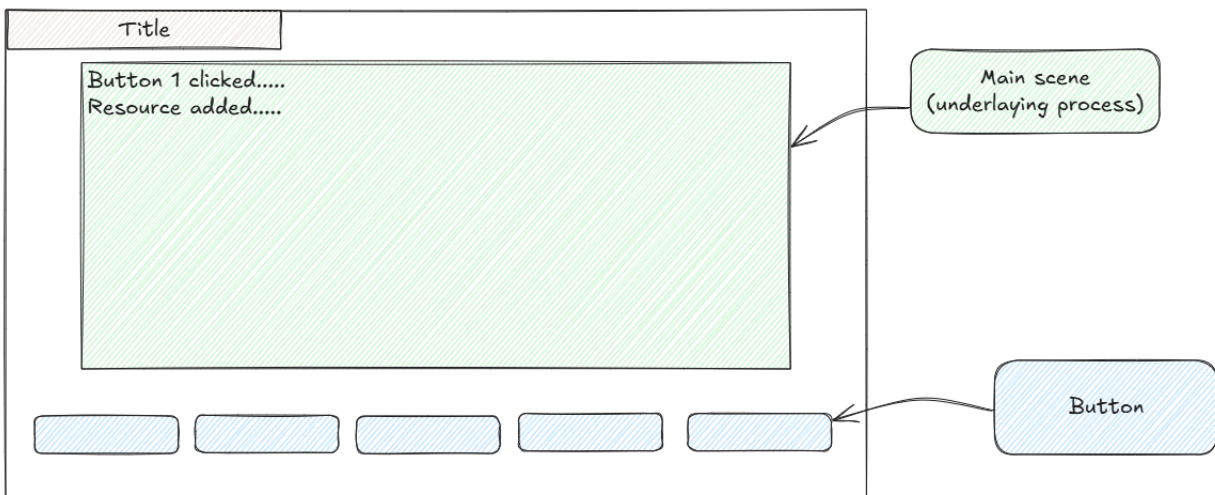


Figure 4: User-case diagram

The ui consists of 2 parts

## 2.3 Operations

Our software includes several impressive and innovative features designed to be both accessible and flexible. We aim to ensure that everyone can understand and use these features while allowing skilled users the freedom to make further improvements.



The core concept involves identifying the base address of the value you want to modify and then making the necessary adjustments to suit your needs. With the provided functions, we hope that users can expand and build upon these capabilities.

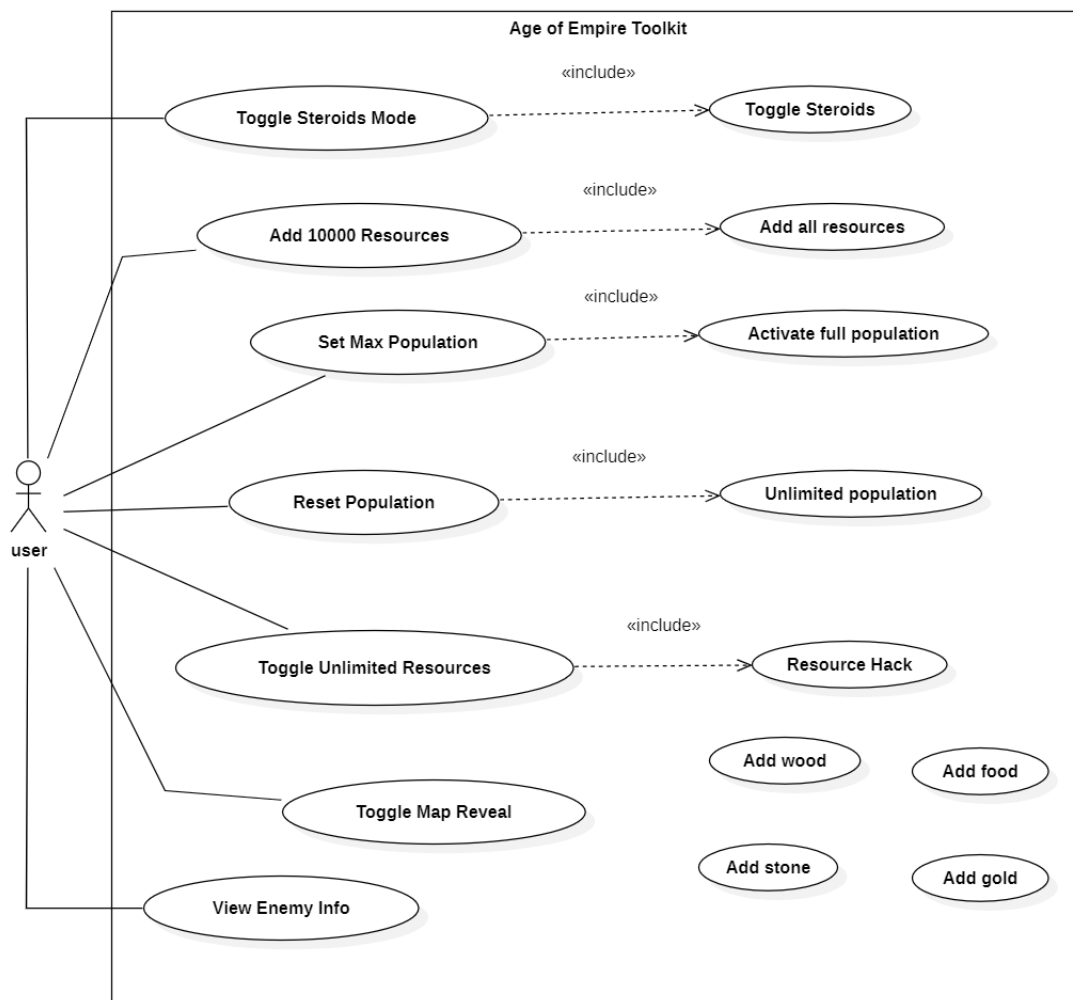


Figure 5: User-case diagram

### 2.3.1 Toggle steroids

The start of our software,

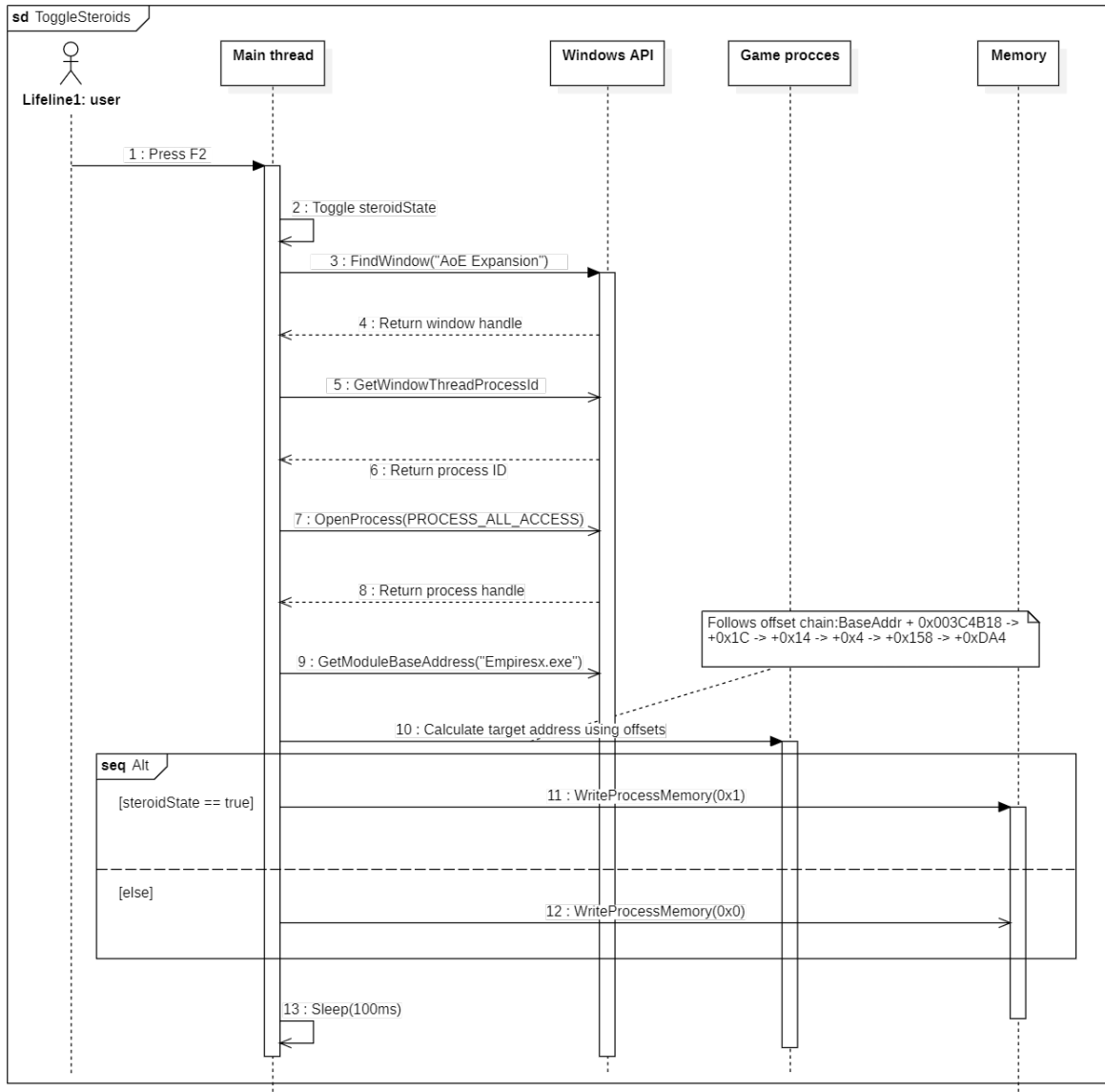


Figure 6: Something unknown

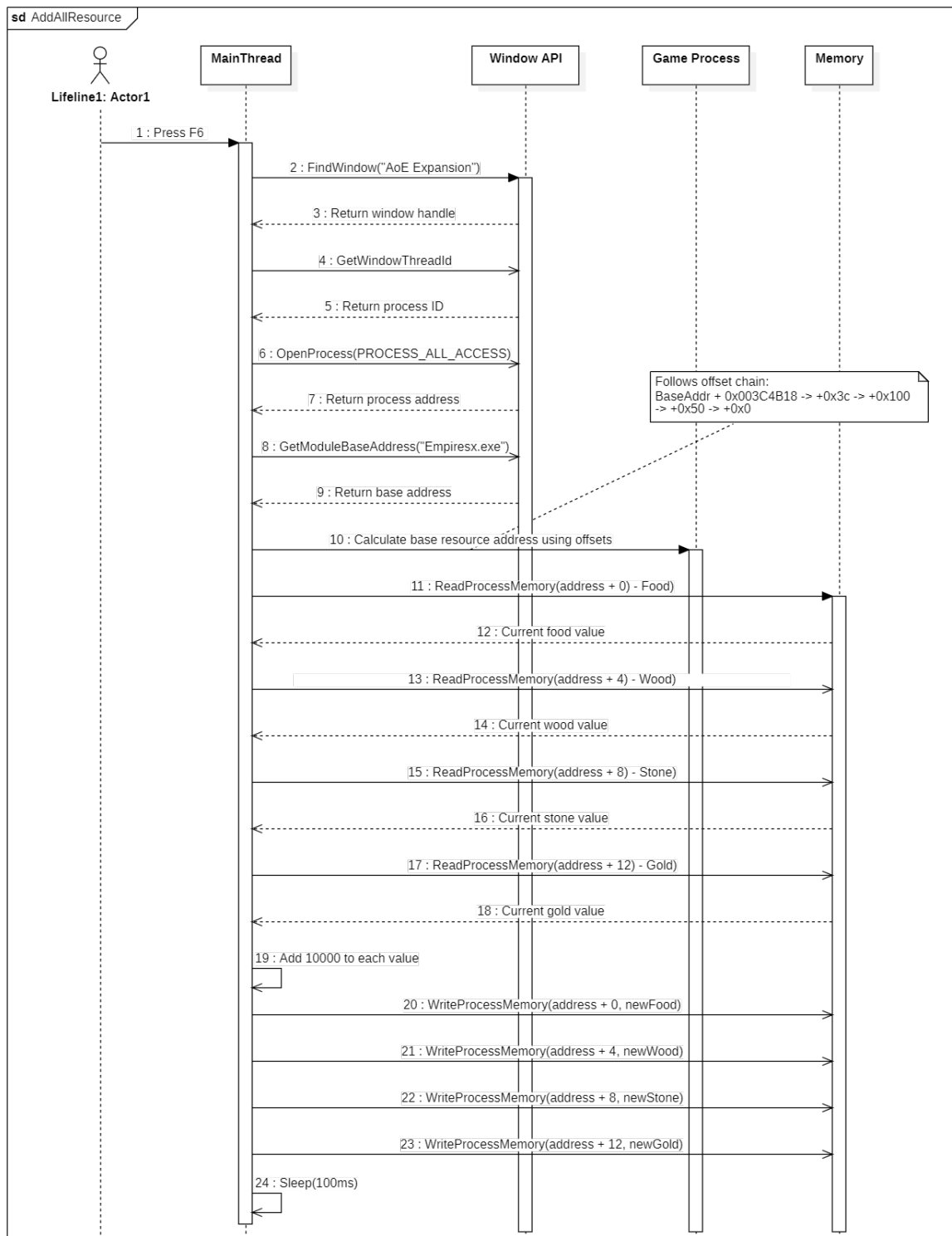


Figure 7: Something unknown

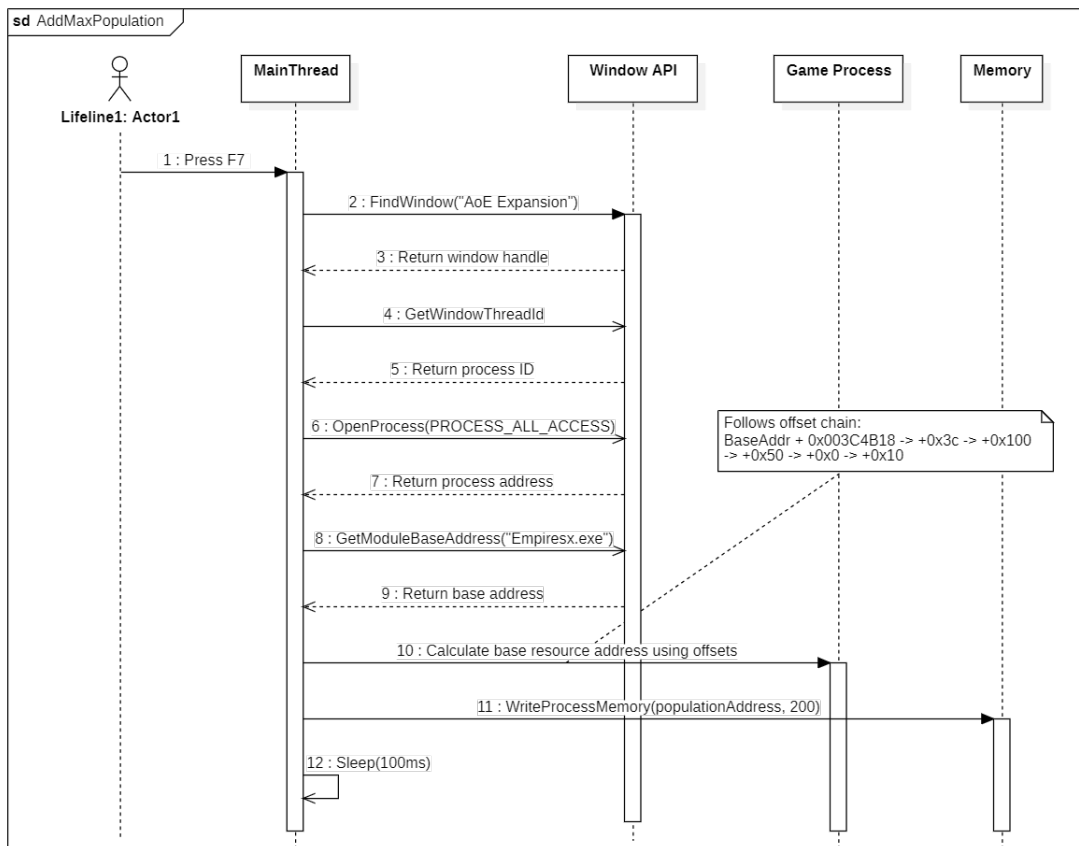


Figure 8: Something unknown

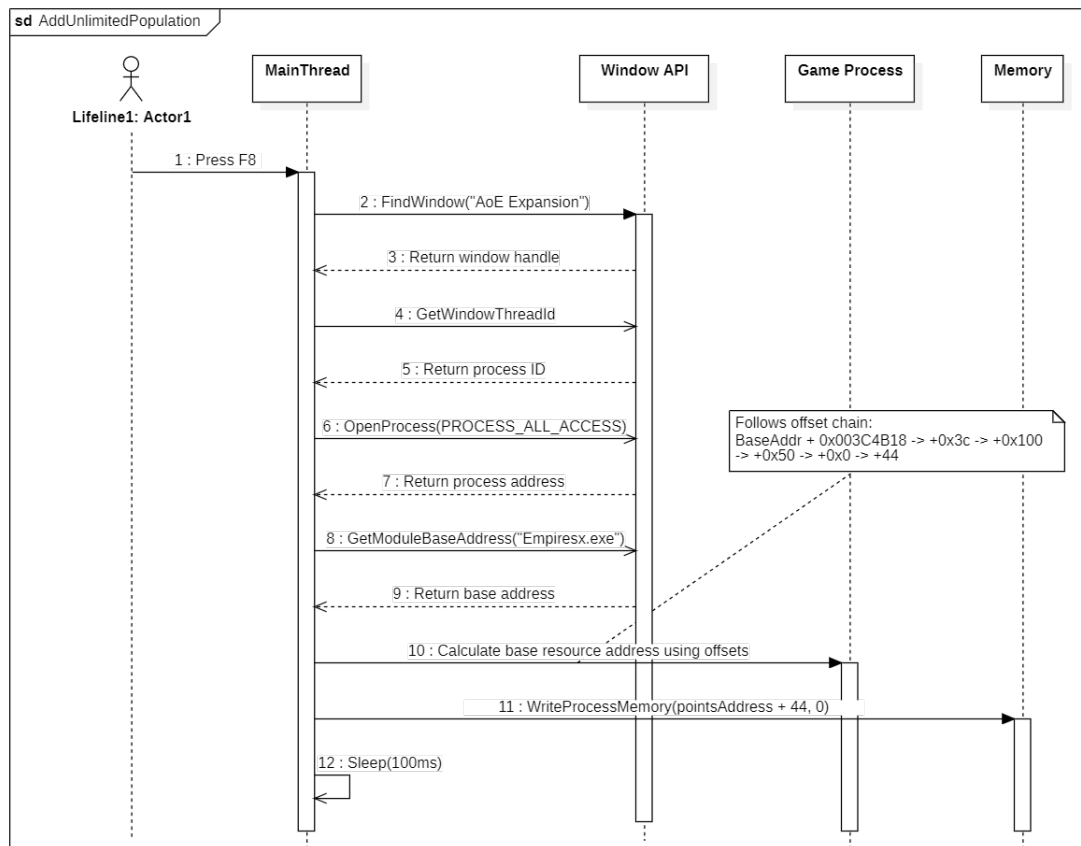


Figure 9: Something unknown

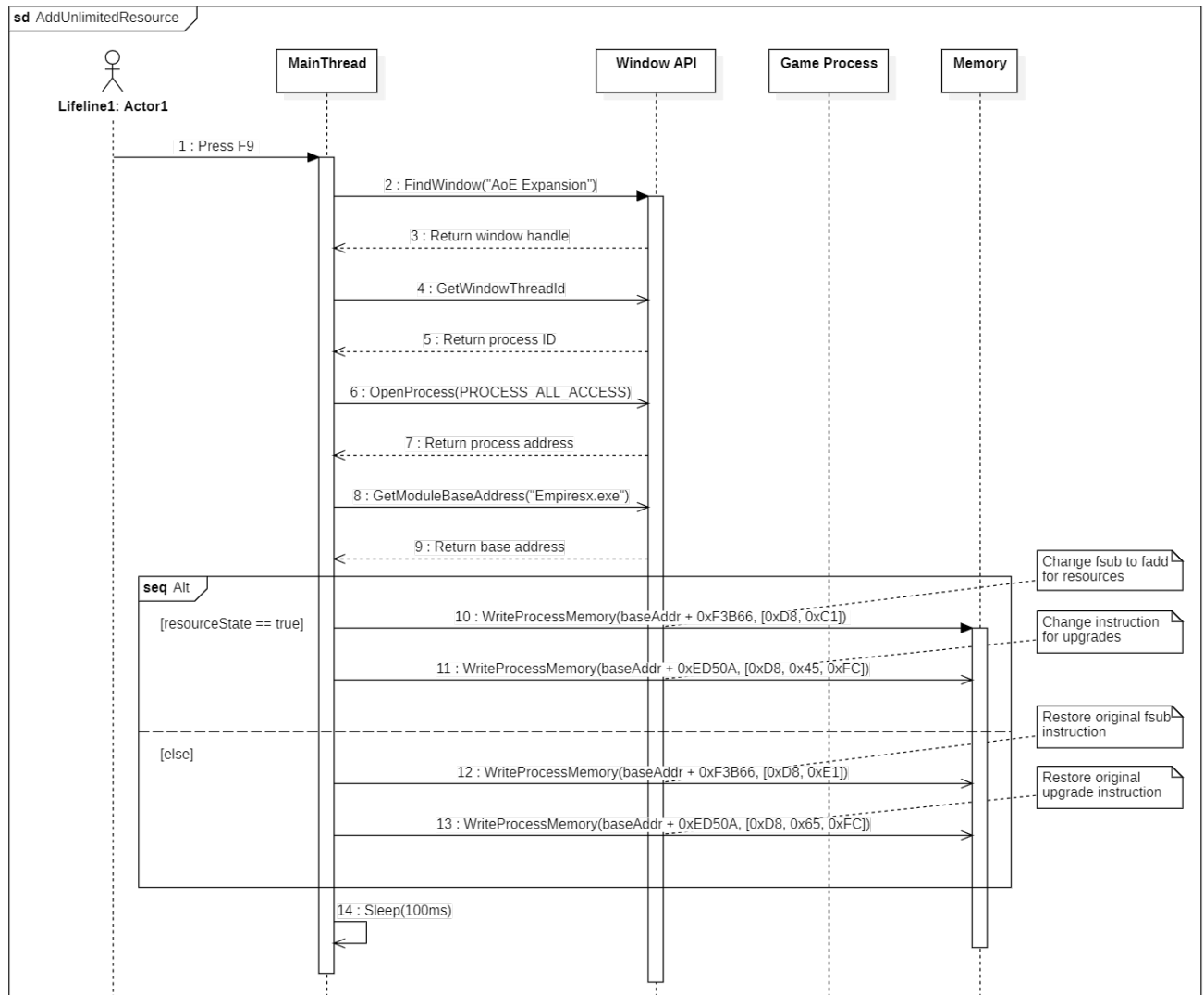


Figure 10: Something unknown

# Chapter 3

## Research

Game hacking involves understanding the underlying memory structures of a game, identifying objects of interest during runtime, and manipulating their values to observe how they affect the game's behavior. This part outlines the step-by-step process for memory scanning, retrieving multi-level pointers, and modifying memory bytes to achieve desired outcomes in a controlled environment. Finally develop a functional program to do all the above things automatically.

### 3.1 Tools

- Memory Scanning: CheatEngine
- Reverse Engineering:
  - Static Analysis: IDAPro 7.7
  - Dynamic Analysis: X32Dbg
- Development: Visual Studio

### 3.2 Memory Scanning

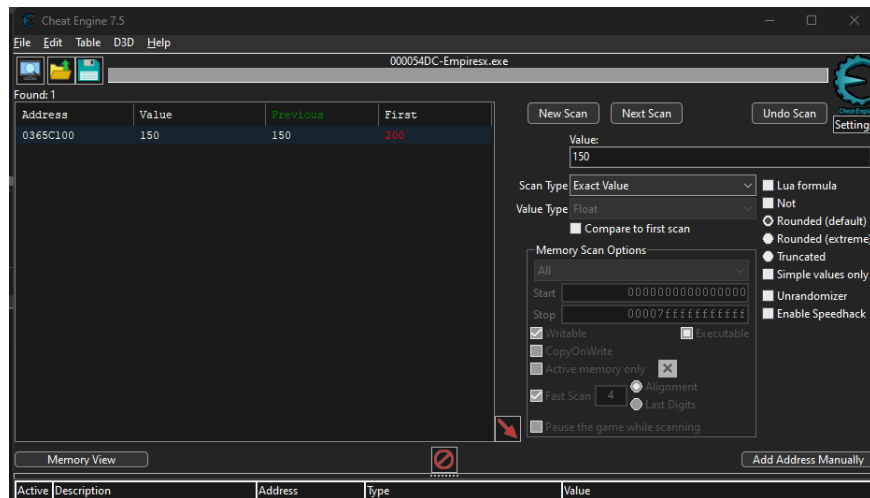
**Objectives:** To locate the memory address of in-game objects during runtime, enabling further analysis and modification.

**Theory:**

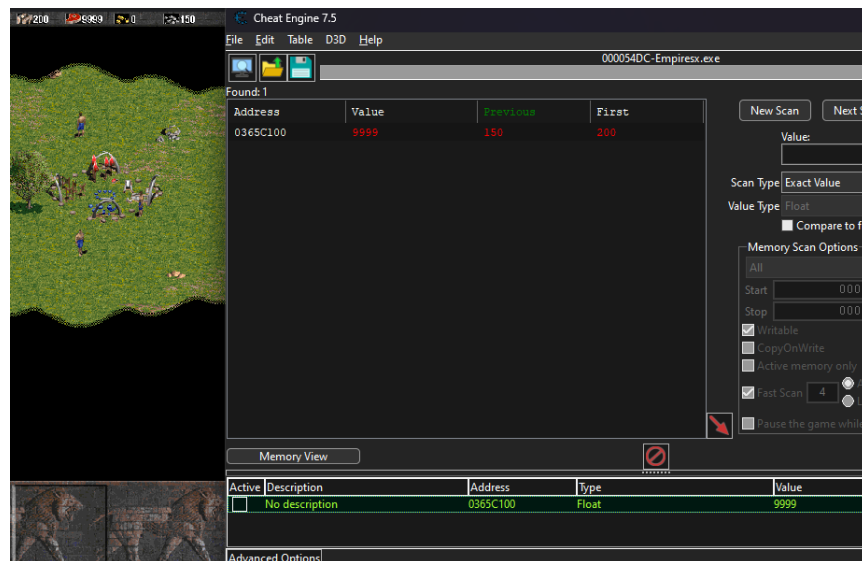
- Identify the Object of Interest: Launch the game and determine which in-game object you want to manipulate (e.g., player health, ammo count, or position).
- Attach a Memory Scanner: Use tools like Cheat Engine or x64dbg to attach to the game's process.

- Perform a Value Search:
  - Search for the initial value of the object (e.g., current health) in the memory scanner.
  - Refine the results by updating the value in-game and rescanning until a single address is identified.
- Validate the Address: Confirm the address by modifying its value and observing the effect in-game.

**Practice:** Finding for food value by first scan for floating value of any resources in the game and spent some to change values then perform second scan:



- After retrieve address of the resource at 0x0365C100, we could modify its to see if its actually the correct address:





- We could also do this in X32Dbg by search for pattern. But the process is much more complicated than using CheatEngine. But the hex view of X32Dbg might be worth it:

```

0365C100  003631C  add byte ptr ss:[esp+ebx],al
0365C103  46      inc esi
0365C104  0000    add byte ptr ds:[eax],al
0365C106  43      dec eax
0365C107  43      inc ebx
0365C108  0000    add byte ptr ds:[eax],al
0365C10A  16      push ss
0365C10B  42      inc ebx
0365C10C  0000    add byte ptr ds:[eax],al
0365C10E  0000    add byte ptr ds:[eax],al
0365C110  0000    add byte ptr ds:[eax],al
0365C112  0000    add byte ptr ds:[eax],al
0365C114  0000    add byte ptr ds:[eax],al
0365C116  0000    add byte ptr ds:[eax],al
0365C118  0000    add byte ptr ds:[eax],al
0365C11A  803F    cmp byte ptr ds:[edi],0
0365C11D  0000    add byte ptr ds:[eax],al
0365C11F  0000    add byte ptr ds:[eax],al
0365C121  0000    add byte ptr ds:[eax],al
0365C123  0000    add byte ptr ds:[eax],al

```

byte ptr ss:[esp+ebx\*1]=[02F5FDBF]=0  
bh=0

0365C100

Address	Float (32-bit)				
0365C000	0	0	0	6.06574e-036	
0365C0E0	0	7.00649e-045	1.79366e-042	2.36936e-038	
0365C100	3.60134e-043	200	3.57331e-043	0	
0365C110	9999	0	150	0	
0365C112	0	0	1	0	
0365C120	0	100	0	4	0
0365C130	0	0	0	0	0
0365C140	0	0	1	4	0
0365C150	0	0	0.2928	102	0
0365C160	103	101	10	0	
0365C170	0	0	0	0	
0365C180	200	0	0	2	
0365C190	250	4	0	0	
0365C1A0	0	0	0	0	
0365C1B0	0	1	0.25	1	
0365C1C0	0	0	0	0	
0365C1D0	0	0	0	0	
0365C1E0	0	0	0	0	
0365C1F0	6.75198e-037	6.752e-037	6.75201e-037	6.77833e-037	
0365C200	7.00649e-045	0	0	0	
0365C210	6.75203e-037	6.75214e-037	6.75226e-037	6.75237e-037	
0365C220	6.75248e-037	0	0	0	
0365C230	4.48416e-044	4.48416e-044	4.48416e-044	4.48416e-044	

- At the same memory location, we also found others resources:

- Food
- Wood
- Gold
- Stone
- Max Population
- Age
- Technology Count (Changing this will take affect on the game but the icons to update will still be there)
- Player Scores
- Map Exploration (We couldn't patch this value however we could know which asm instruction access this memory location then do more reversing to get the logic of its)

- We have found the right address. However this is a dynamic address. Which mean if the game start again, the address going to be different. We need to find static pointer that always point to this dynamic address, this way we could know the address of player's object everytime the game starts.

### 3.3 Retrieve Multi-Level Pointers

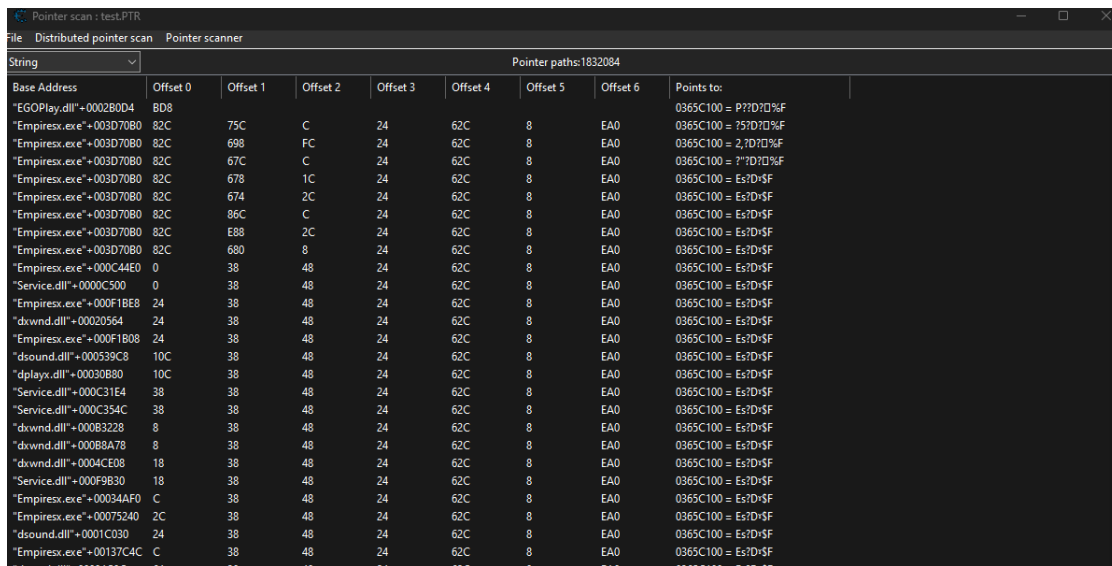
**Objectives:** To find the pointer that consistently points to the memory location of the object, even after the game restarts.

**Theory:**

- Locate the Dynamic Address: Start by identifying the dynamic address of the object as described in the memory scanning process.
- Pointer Scan: Use the memory scanner's pointer scan feature to identify potential pointers to the dynamic address.
- Test Pointer Stability: Restart the game and verify if the pointers still lead to the correct object. Eliminate invalid pointers.
- Resolve Multi-Level Pointers: If the pointer is indirect, repeat the process to find pointers to the base address. This process continues until a reliable base pointer is identified.

**Practice:** - How the multi-level Pointer works: Very simple, it dereferences a next pointer to get to the next pointer until it reach the final destination

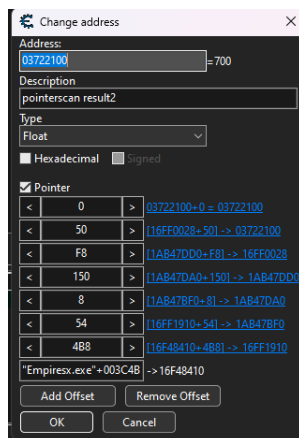
- This step is straight forward, we just need to perform a pointer scan for the address:



Base Address	Offset 0	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5	Offset 6	Points to:
"EGOPPlay.dll"+0002B0D4	BD8							0365C100 = P?D?D?F
"Empiresx.exe"+003D70B0	82C	75C	C	24	62C	8	EA0	0365C100 = 75?D?D?F
"Empiresx.exe"+003D70B0	82C	698	FC	24	62C	8	EA0	0365C100 = 2?D?D?F
"Empiresx.exe"+003D70B0	82C	67C	C	24	62C	8	EA0	0365C100 = 7?D?D?F
"Empiresx.exe"+003D70B0	82C	678	1C	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+003D70B0	82C	674	2C	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+003D70B0	82C	86C	C	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+003D70B0	82C	E88	2C	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+003D70B0	82C	680	8	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+000C44E0	0	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Service.dll"+0000C500	0	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+000F1BE8	24	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dxwnd.dll"+00020564	24	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+000F1B08	24	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dsound.dll"+000539C8	10C	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dplayx.dll"+00030B80	10C	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Service.dll"+000C31E4	38	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Service.dll"+000C354C	38	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dxwnd.dll"+000B3228	8	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dxwnd.dll"+000B8A78	8	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dxwnd.dll"+0004CE08	18	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Service.dll"+000F9B30	18	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+00034AF0	C	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+00075240	2C	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"dsound.dll"+0001C030	24	38	48	24	62C	8	EA0	0365C100 = Es?D?SF
"Empiresx.exe"+00137C4C	C	38	48	24	62C	8	EA0	0365C100 = Es?D?SF

- We could use any of theses pointers found in the table scan with base address is Empiresx.exe

- By deferencing the pointer we could get the final address of the player object:



- Deferencing each pointer to get to the final Address by using ReadProcessMemory API, iterate through each pointer to get to the final address.

### 3.4 Modify Memory Bytes

**Objectives:** To manipulate the value stored at the identified memory address and observe its effects on the game.

**Theory:**

- Identify the Address to Modify: Use the pointer or dynamic address retrieved in the previous steps.
- Change the Value: Modify the value directly in the memory scanner or write a script to do so.
- Observe the Outcome: Monitor the game to see how the changes affect the object (e.g., infinite health, increased speed, or unlimited ammo).
- Automate the Process (Optional): Use programming languages like Python or C++ to create a trainer or script that automates memory modification.

**Practice:** - After getting the final address of the player object, we could modify the value of the resources to see if it actually take affect on the game. Base on the hex view of the game, we could see that the resources are stored in the memory as float value and each resources are stored 4 offset away from each other.

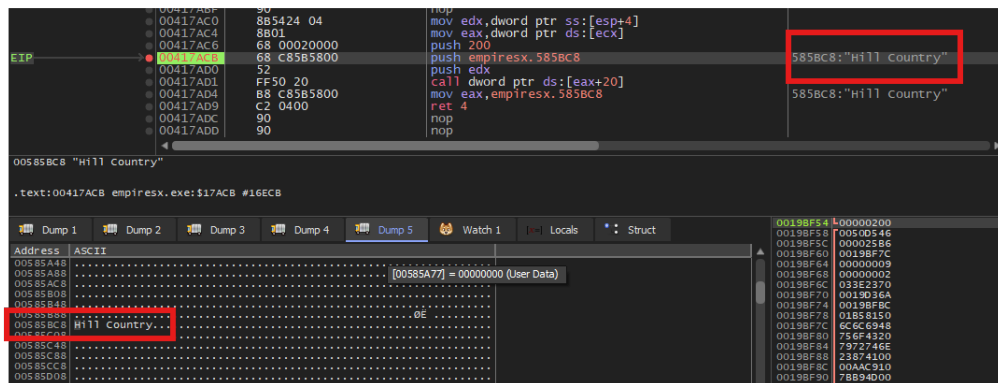
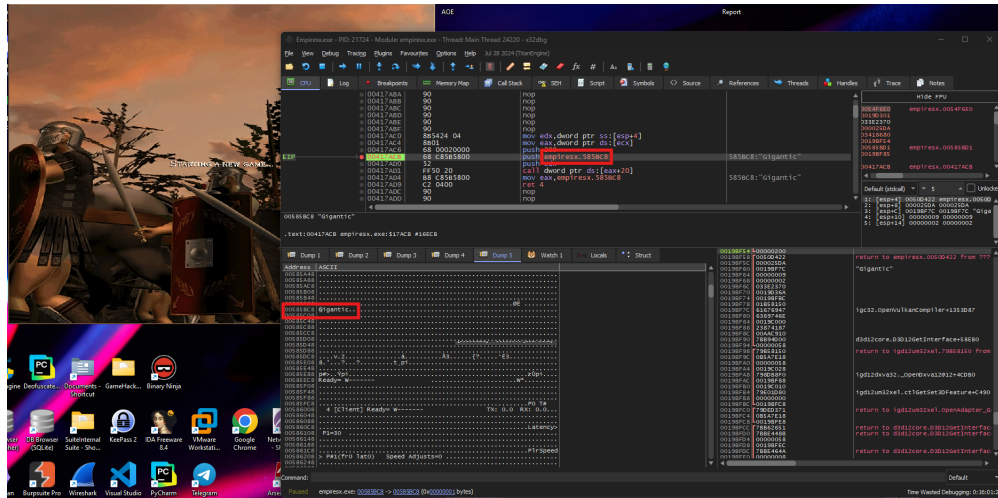
- We could use WriteProcessMemory API to write the new value to the memory location.

- Each resources are stored 4 offset away from each other. We could use the same method to modify the resources.

## 3.5 Reversing Game Logic

### 3.5.1 Game Settings Reversing

Before the game start it will load game settings. We can directly modify this setting into the game binaries



- Spectating the memory region we can see alots of others settings too:

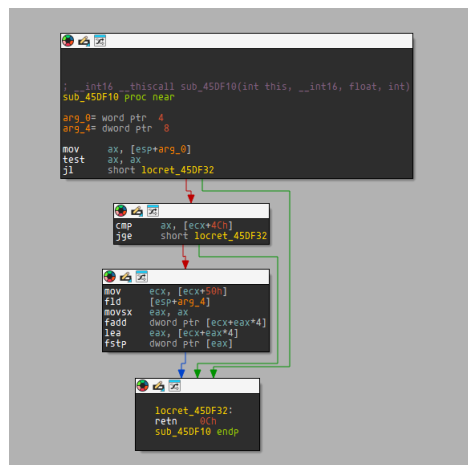
```

005858C8 Enable Cheating: %s.....
00585C08 .....
00585C48 .....
00585C88 .....
00585CC8 .....
00585D08 .....
00585D48 .....
00585D88 .....
00585DC8 ...v.2... ..â... ..A3... ..{?... ..É3...
00585E08 8.'...?...?.....t.pi.....
00585E48 .....
00585E88 p#>..ÿpi.....z0pi...
00585EC8 Ready= W----- ..W*
00585F08 .....
00585F48 .....
00585F88 .....
00586000 [00557000] = 00000000 (User Data) .....P0 T#
00586048 .....TX: 0.0 RX: 0.0...
00586088 .....
005860C8 .....Latency>
00586108 P1=30 .....
00586148 .....
00586188 .....P1rSpeed
005861C8 > P#1(fr0 lat0) Speed Adjusts=0 .....
00586208 .....
00586248 .....
00586288 .....MeOptima
00586308 l: Buf= 5 Gran= 10 Target FPS=100 My expect turn= 50ms DPMS
00586348 gs0 .....
00586388 .....
00586408 / 10 * Gran= 90ms = turn 900ms Lo=0 < Avg=0 > Hi=0 of 50 ...
00586448 .....Buf= 0
00586488 .....
00586508 : P1=1691571 P2=0 P3=0 P4=0 P5=0 P6=0 P7=
00586548 0 P8=0 .....LastComm
00586588 .....

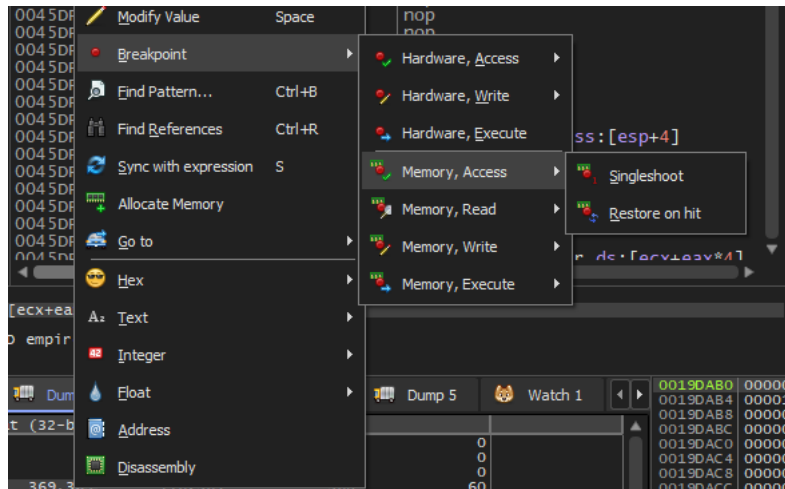
```

### 3.5.2 Game Logic Reversing

- We could also reverse the game logic to understand how the game works and how the game access the memory location. This way we could understand the game better and could make more complex cheats.



- The above function is function adding resources to the player object. We could patch this function to add more resources to the player object. To find it, set a break point at the memory location of the function then run the game and add resources to the player object. The break point will be hit and we could see the function that add resources to the player object.

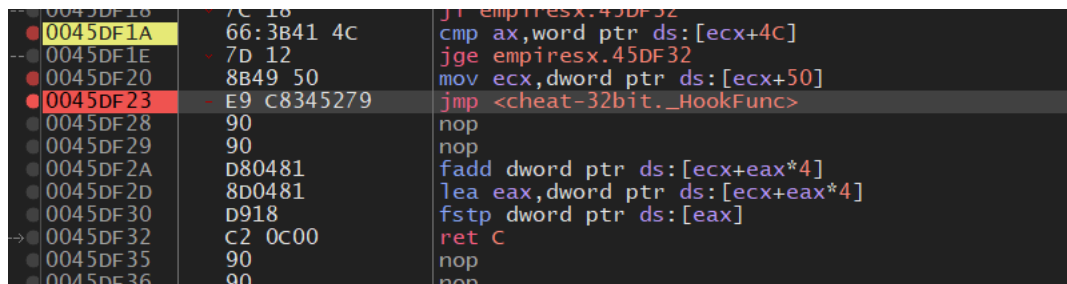


- The instruction responsible for adding resources in `fild [esp+arg_4]` to the player object is `fadd dword ptr [ecx+eax*4]`. We could patch this instruction to add more resources to the player object. Then store the object address in `[eax]` register.

Eg:

- `fadd dword ptr [ecx+eax*4]` to `fadd dword ptr [ecx+eax*4] + 10000`
- `fadd dword ptr [ecx+eax*4]` to `fmul dword ptr [ecx+eax*4]`

- The function also responsible for all the resources modification. We could patch this function to add more resources to the player object.



- Redirect instruction flow to our code and jump back after executing our code:

```

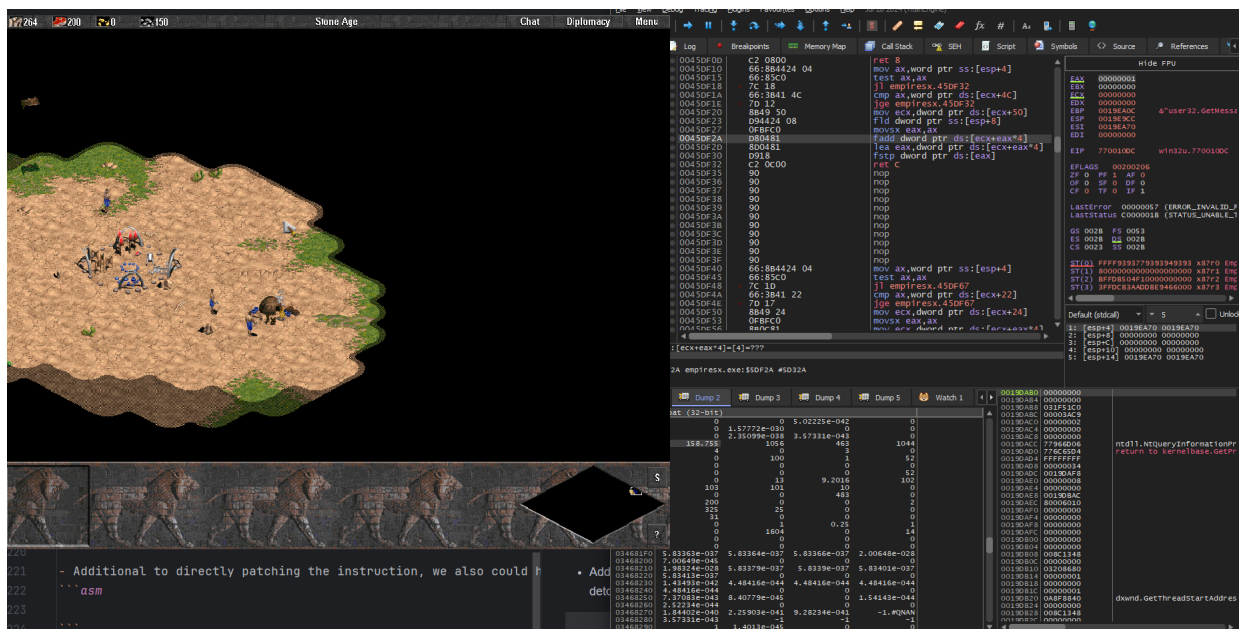
DWORD OriginalAddress = 0x0045DF2A;           // Address of the original function
__declspec(naked) void HookFunc() {
__asm {
    // Modify the floating-point value being loaded
    add dword ptr[esp + 8], 100 // Adjust the value at [esp + 8] (account for pushad)
    // Execute the original instructions
    fld dword ptr[esp + 8]      // Restore original instruction
    movsx eax, ax               // Restore original instruction

    // Jump back to the original code
    jmp OriginalAddress        // Jump back to the original code
}
}

```

799813EF	CC		int3	
799813F0	834424 08 64	add dword ptr ss:[esp+8],64		dllmain.c:213
799813F5	D94424 08	fld dword ptr ss:[esp+8]		dllmain.c:218
799813F9	0FBFC0	movsx eax,ax		dllmain.c:219
799813FC	FF25 60409879	jmp dword ptr ds:[<OriginalAddress>]		dllmain.c:222
79981402	CC	int3		
79981403	CC	int3		

- However this function is used by all player object. If we patch this function, all player object will have the same affect. This also lead we to other's player object:



-Additional to directly patching the instruction, we also could hook the specific address to perform function detouring to add more resources to the player object.

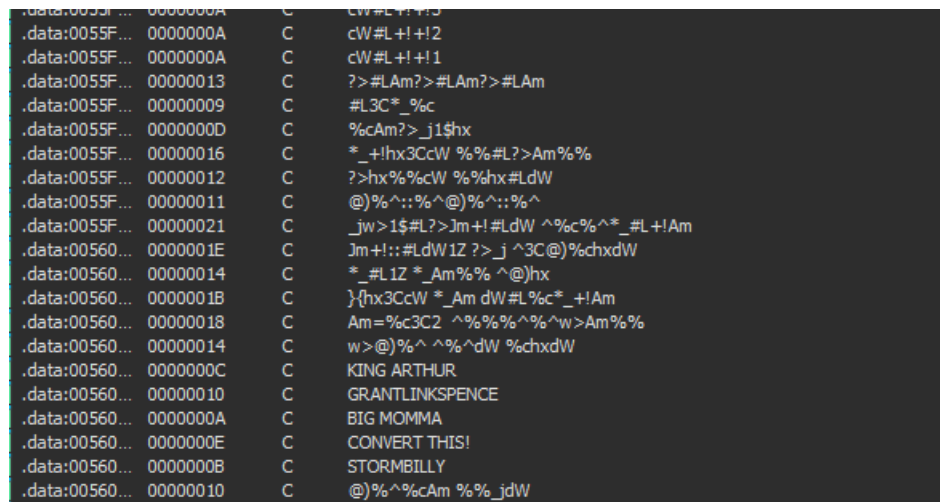
-In our case, we could hook at: `fld dword ptr ss:[esp+8]` which is 0x0045DF23 and jmp back at 0045DF2A

```
0045DF1A | 66:3B41 4C          | cmp ax,word ptr ds:[ecx+4C] |
0045DF1E | 7D 12              | jge empirex.45DF32         |
0045DF20 | 8B49 50            | mov ecx,dword ptr ds:[ecx+50] |
0045DF23 | D94424 08          | fld dword ptr ss:[esp+8]   |
0045DF27 | 0FBFC0            | movsx eax,ax               |
0045DF2A | D80481            | fadd dword ptr ds:[ecx+eax*4] |
0045DF2D | 8D0481            | lea eax,dword ptr ds:[ecx+eax*4] |
0045DF30 | D918              | fstp dword ptr ds:[eax]    |
```

## String References

- We could also find the string references in the game binaries. This could help us to understand the game better and could make more complex cheats.

- We could use IDAPro to find the string references in the game binaries.



```
.data:0055F... 0000000A C cW#L+!+!2
.data:0055F... 0000000A C cW#L+!+!1
.data:0055F... 00000013 C ?>#LAm?>#LAm?>#LAm
.data:0055F... 00000009 C #L3C*_%c
.data:0055F... 0000000D C %cAm?>_j1$hx
.data:0055F... 00000016 C *+_!hx3CcW %%#L?>Am%%
.data:0055F... 00000012 C ?>hx%%cW %%hx#LdW
.data:0055F... 00000011 C @)%%^::%^@)%%^::%^
.data:0055F... 00000021 C _jw>1$#L?>Jm+!#LdW ^%c%^*_#L+!Am
.data:00560... 0000001E C Jm+!::#LdW 1Z ?>_j ^3C@)%chxdW
.data:00560... 00000014 C *_#L 1Z *_Am%% ^@)hx
.data:00560... 0000001B C }{hx3CcW *_Am dW#L%*_+!Am
.data:00560... 00000018 C Am=%c3C2 ^%%%%^w>Am%%
.data:00560... 00000014 C w>@)%%^ ^% ^dW %chxdW
.data:00560... 0000000C C KING ARTHUR
.data:00560... 00000010 C GRANTLINKSPENCE
.data:00560... 0000000A C BIG MOMMA
.data:00560... 0000000E C CONVERT THIS!
.data:00560... 0000000B C STORMBILLY
.data:00560... 00000010 C @)%%^cAm %%_jdW
```

-If you familiar with the game, you would recognize the string references. These are game cheat codes. We could use these cheat codes to make the game easier.

-All these strings are all call from the same function. We could patch this function to enable all the cheat codes in the game.



- We have already reversed the function. We could patch this function to enable all the cheat codes in the game.
- It check user input then obfuscate it to match the strings store in the game memory.
- If the input obfuscated string match the string in the game memory, it will return an integer then pass it to second process cheat function. Otherwise it would print out as chat message.

```

char *__cdecl ObfuscateCheatCode(char *a1, char *a2, int InputLength)
{
    char *result; // eax
    char UserInput; // cl
    int v5; // esi
    char *i; // edi
    int v7; // ecx
    char *v8; // eax

    result = a2;
    UserInput = *a1;
    v5 = 0;
    for ( i = a1 + 1; UserInput; UserInput = *i++ )
    {
        if ( v5 >= InputLength )
            break;
        if ( UserInput < 'A' || UserInput > '_' )
        {
            *result++ = UserInput;
            ++v5;
        }
        else
        {
            v7 = *(_DWORD *)&aTpnlTimRenderT[4 * UserInput + 4];
            *result = BYTE1(v7);
            v8 = result + 1;
            *v8 = v7;
            result = v8 + 1;
            v5 += 2;
        }
    }
    *result = 0;
    return result;
}

```

- Base on the above obfuscate function, i have developed a function to deobfuscate all the strings in the memory:

```
> python3 .\Deofuscate.py

DEOBFUSCATE AOE2

Author: Kiendt19

[!] Press anything to display the cheat.....
[+] Deofuscating of dW/^ Jm/^1Z,....
[Inf] Result: NO FOG

=====

[+] Deofuscating of %AmTAmhx+! %chxw>....
[Inf] Result: REVEAL MAP

=====

[+] Deofuscating of 1$ ^Am%%^L?>1$....
[Inf] Result: STEROIDS

=====

[+] Deofuscating of hx_jhx%%:: ....
[Inf] Result: QUARRY

=====

[+] Deofuscating of 43%^/^?>1$ ^/^3CcW,....
[Inf] Result: WOODSTOCK

=====
```

- Now we have known the cheat logic, we could perform dynamic analysis to find the function that call this function then patch it to enable all the cheat codes in the game.
- Reversing how the cheat works: Steroids

## Chapter 4

# Results