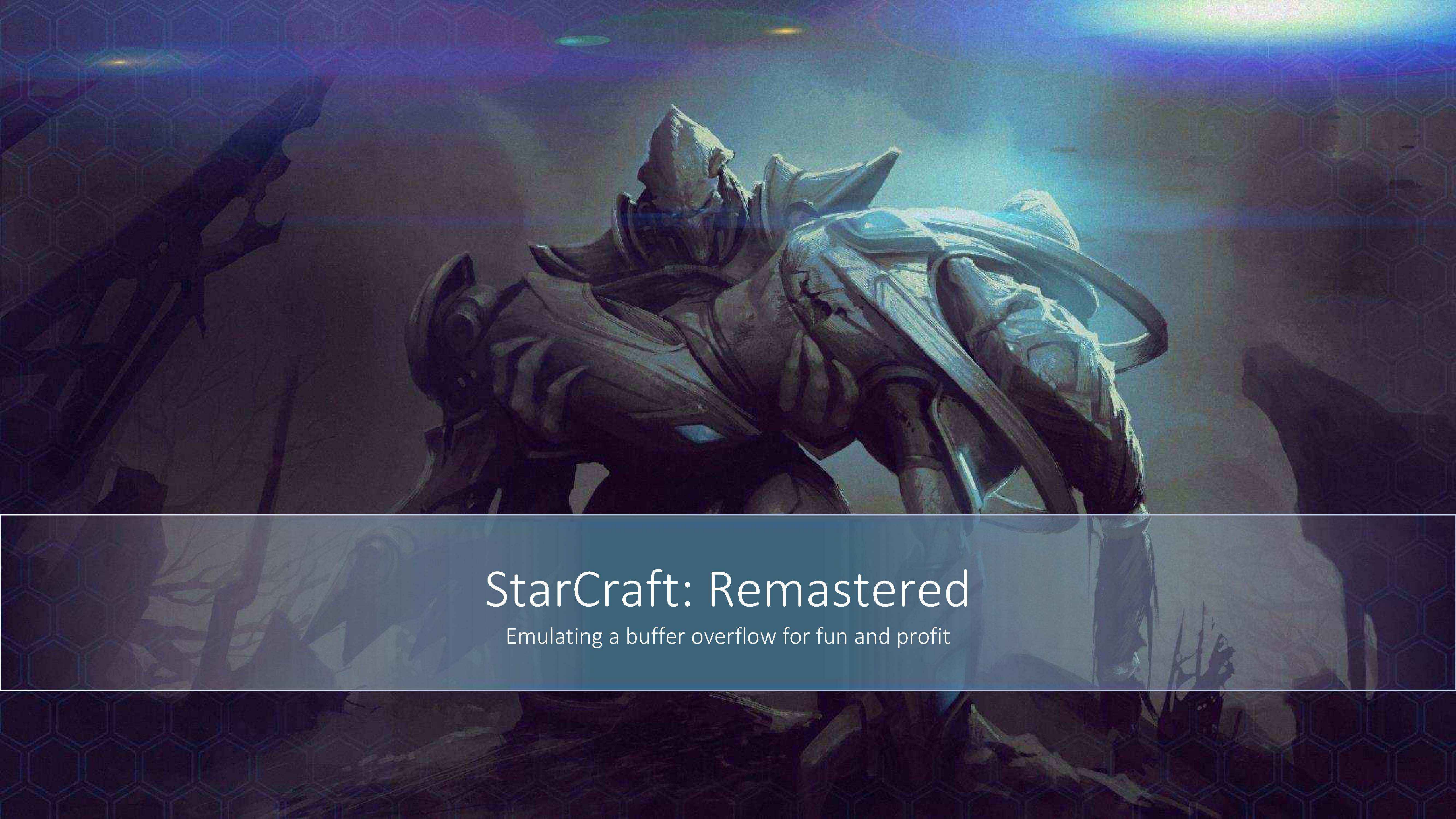


StarCraft: Remastered

Emulating a buffer overflow for fun and profit

A note before we begin

Blizzard Entertainment in no way endorses or condones reverse engineering of our properties.

The exercises herein were conducted to understand the methods used to create unlicensed behaviors.

About Me: Elias Bachaalany

- Anti-Cheat Engineer, Blizzard Entertainment
- Previously worked at Hex-Rays and Microsoft
- Technical writer:
    - Practical Reverse Engineering, Antivirus Hackers Handbook
    - Batchography
- Passionate about reverse engineering and low-level programming on MS-Windows
- Interested in debuggers, emulators, API hooking, dynamic binary instrumentation and virtualization technologies
- Contact
    - Email: [ebachaalany at blizzard.com](mailto:ebachaalany@blizzard.com)
    - Twitter: @0xeb

Comrades on the adventure

My colleagues

- Guillaume Breuil, Yi Deng, Chris Genova, Mark Chandler, James Touton, Pete Stilwell, Zak Bennett and Grant Davies

Tools

- SCMDraft2 map editor - Henrik Arlinghaus
- trgk (Trigger King) - https://github.com/phu54321/
- MPQ tools – Ladislav Zezula
- BWAPI - Adam Heinermann
- IDA Pro - Hex-Rays
- Diaphora – Joxean Koret
- EUDEnabler and the EUDDB - Farty1Billion - http://farty1billion.dyndns.org/EUDDB/

South Korean map makers and tools community

- Kongze1004 – Random Tower Defense map author
- Sksljh2091 – Mario Exodus map author
- Jacksell12, Deation, Sato

Community Sites

- TeamLiquid, StarEdit Network, Naver.com

Backstory /1

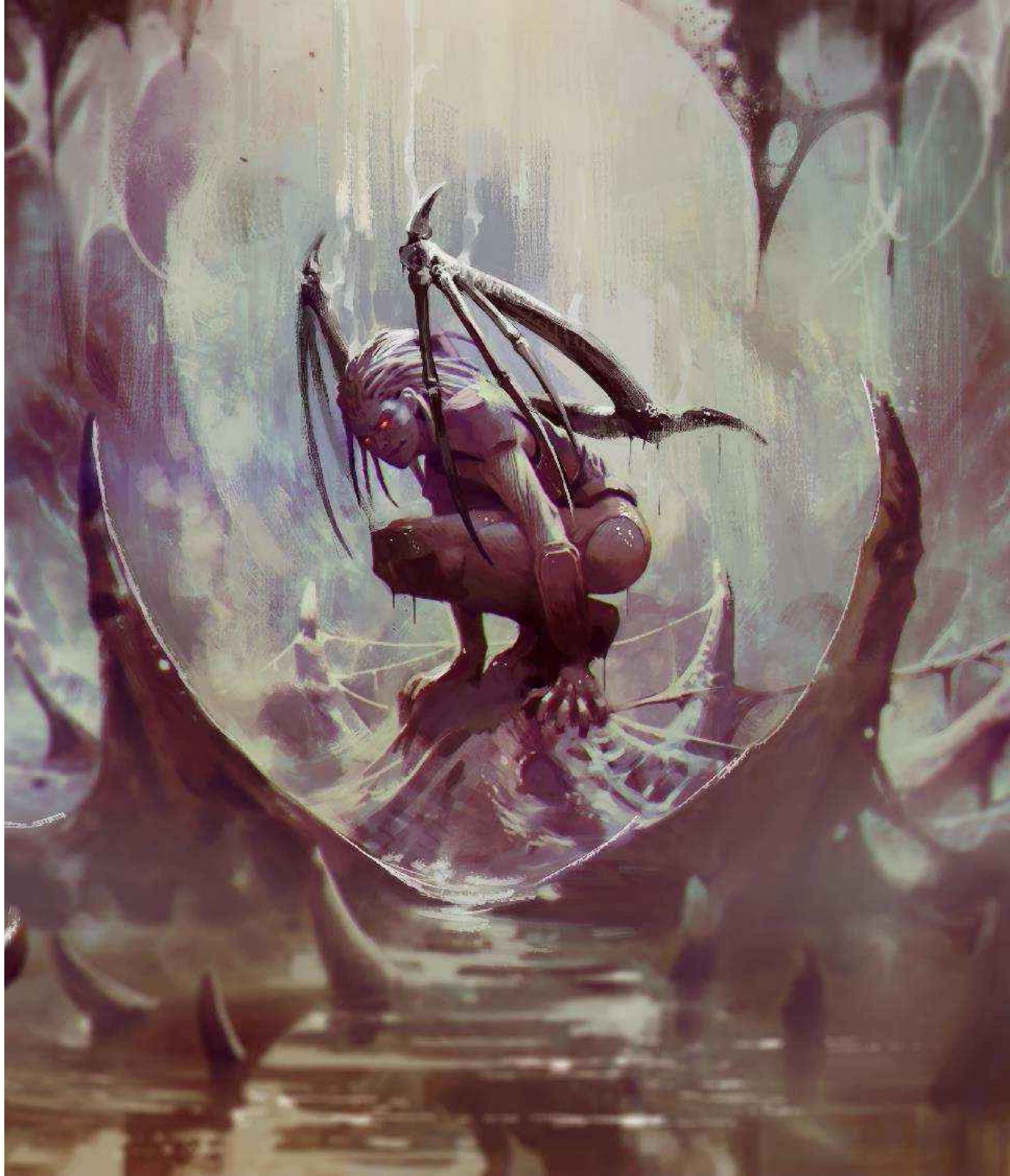
- StarCraft is a science fiction RTS (real-time strategy)
- Released for PC and Mac on March 31, 1998
- StarCraft: Brood War - Expansion pack released on November 30, 1998
- Significant patches to this talk:
    - 1.16.1 - 01/21/2009 – Last patch for 8 years
    - 1.18.0 - 04/18/2017 – First modern patch
    - 1.20.0 – 08/14/2017 – StarCraft: Remastered
    - 1.21.0 – 12/07/2017 – EUD *reintroduced via emulation*

Backstory /2

- StarCraft had various buffer overflow bugs, but one was related to a particular trigger condition and action:
  - The Extended Unit Death trigger
  - Or simply: EUD

- Blizzard did not update StarCraft between 2009 and early 2017
  - The community re-enabled the bug with custom launchers and tools

- Patch 1.17 was slated for release but was held back because it would break mods, tools, and launchers:
  - wMode
  - wLauncher, ChaosLauncher
  - BWAPI – Plugin to write AI bots that play StarCraft

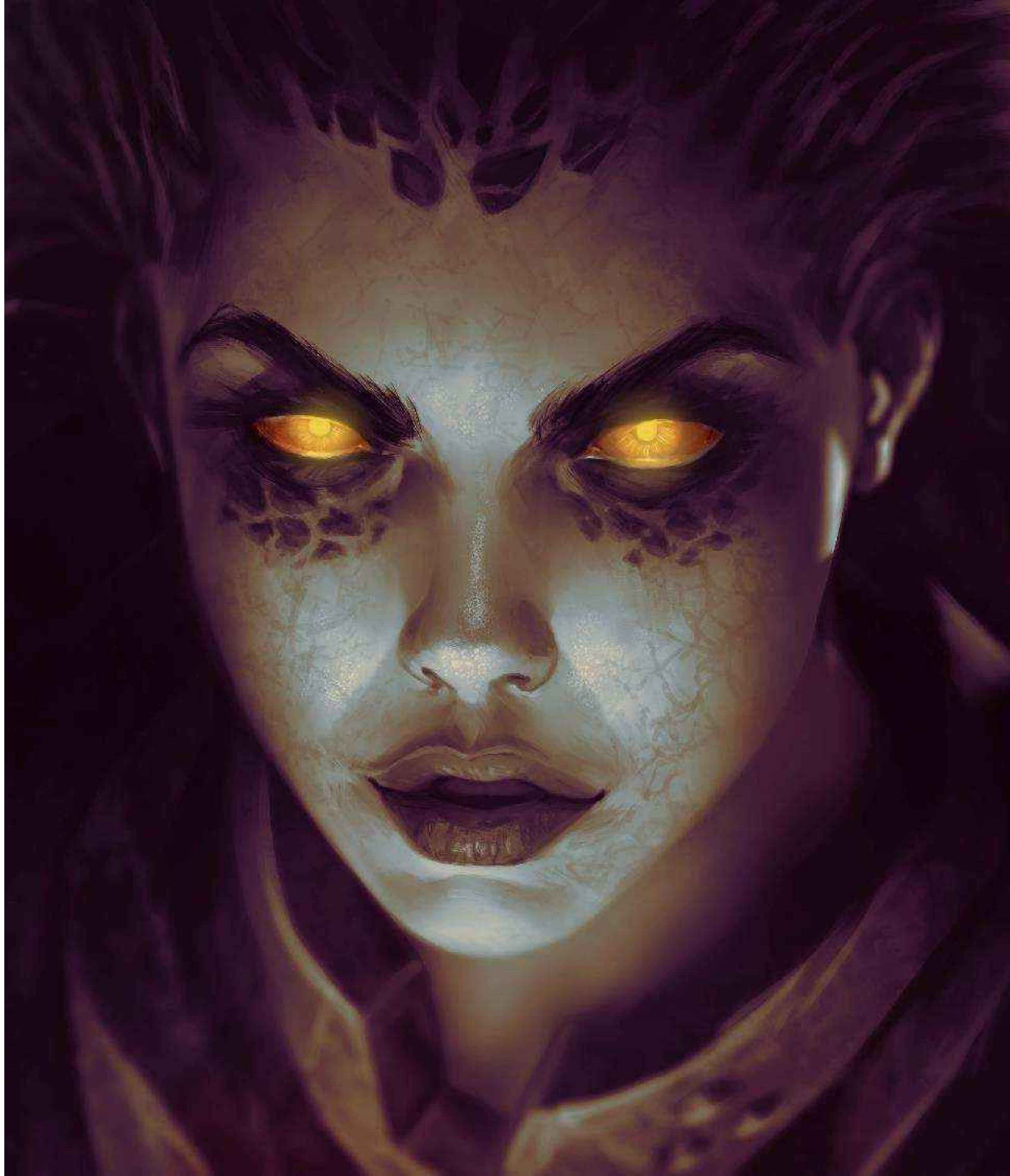


Backstory /3

- StarCraft maps based on EUD triggers thrived among the South Korean map makers community

- The EUD triggers:
  - Are encoded in the map file
  - Allowed arbitrary memory read and write:
    - The majority of the public EUD maps in circulation have hardcoded addresses compatible with StarCraft 1.16.1 on Windows
    - ➢ I am not aware of any EUD maps for the MacOS version of the game

- The EUD exploit allowed modders to author maps that modify the game radically:
  - Random Tower Defense
  - Mario Exodus Map
  - Etc.



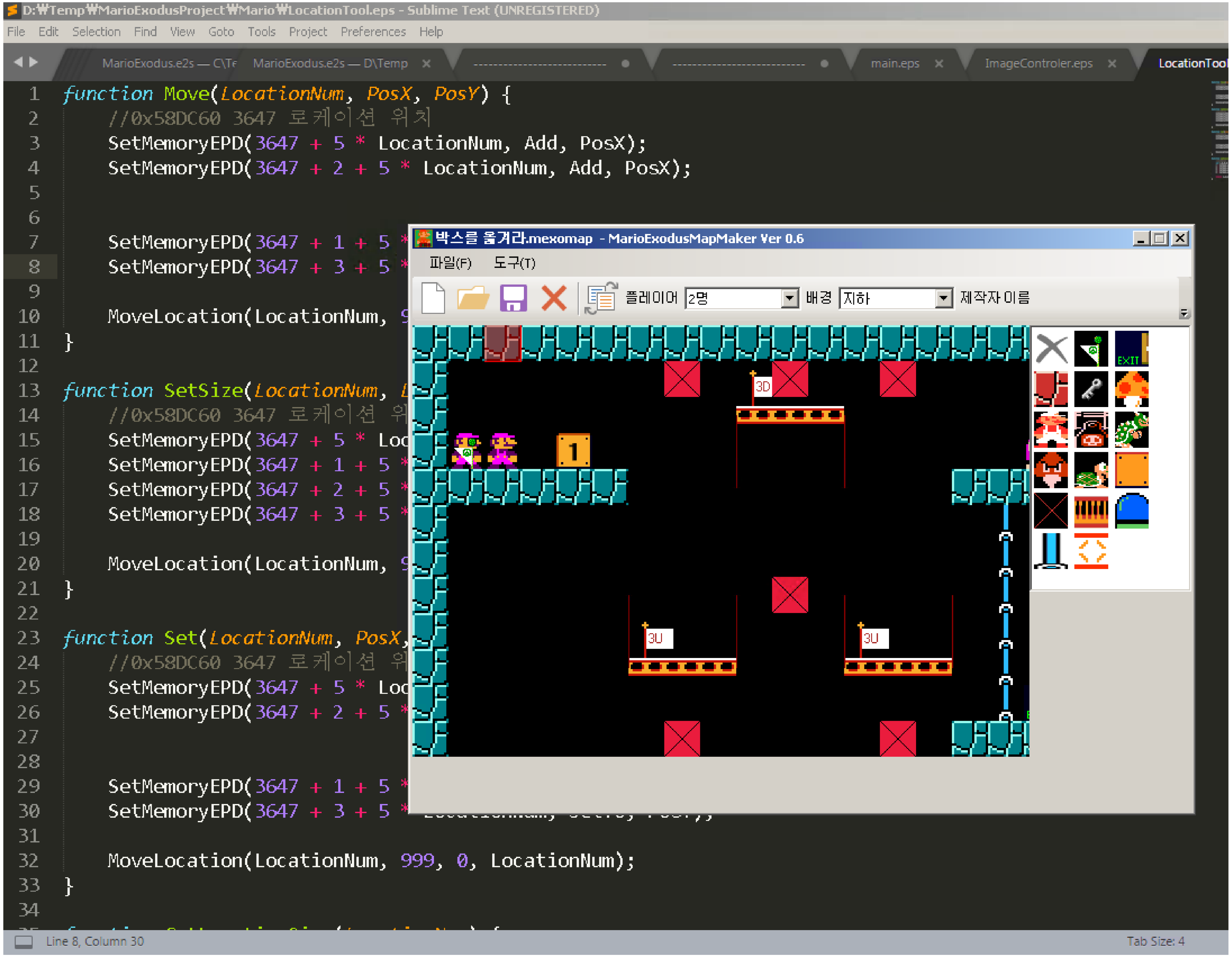
Random Tower Defense – EUD map

Bouncing Ball EUD map (SC 1.16.1)

Bouncing Ball EUD map (SC:R w/ emulation)

스타크래프트 유즈맵 [EUD] Mario EXODUS V0.4 (StarCraft Use map)

- The Mario Exodus map author created a level editor!
- The map was developed using trgk's epScript language and compiler



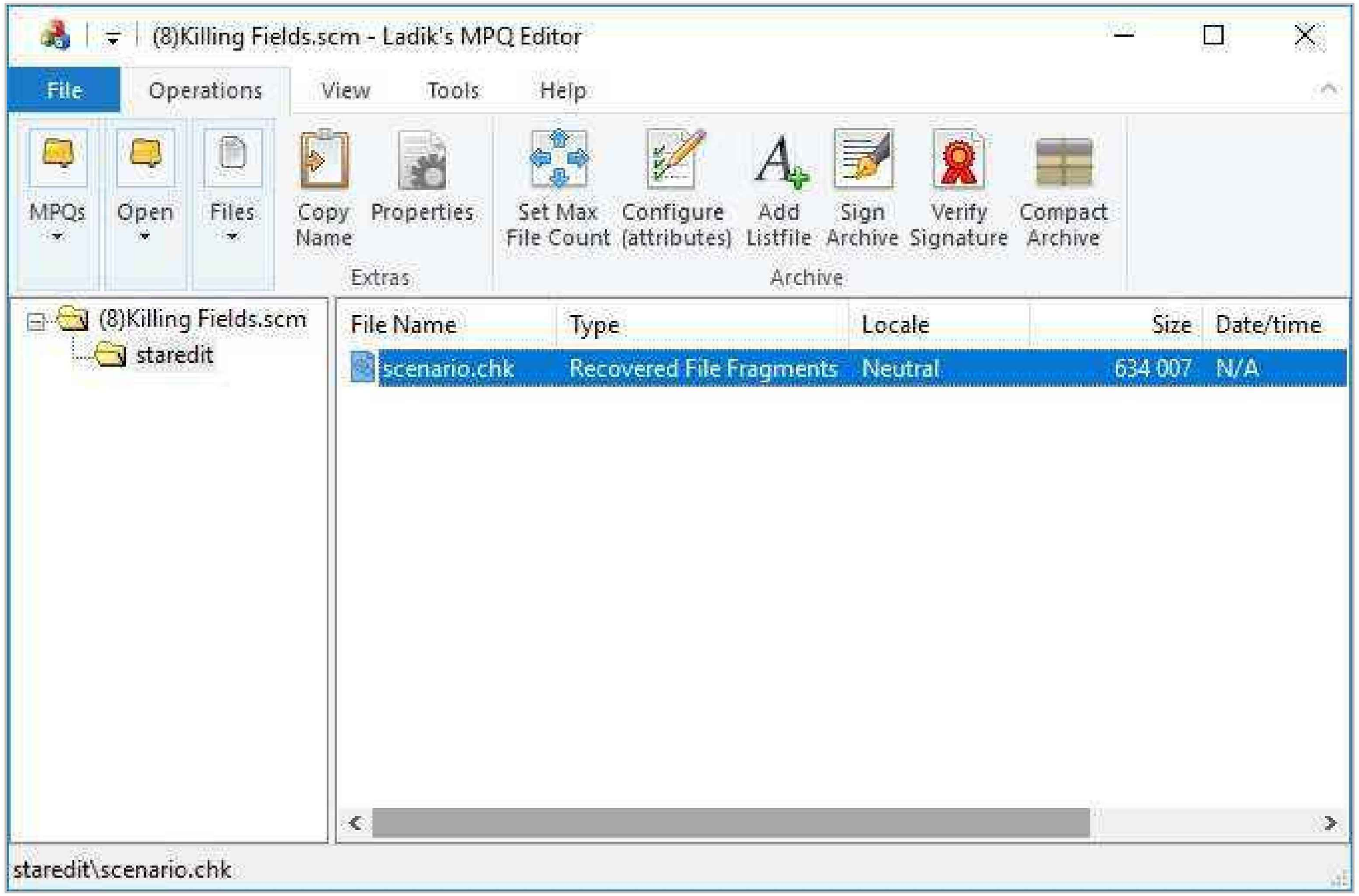
StarCraft map file format

- They are just MPQ archives
    - The MPQ format has been extensively reverse engineered and documented by the community

- They contain various files:
    - They contain custom WAV audio used by the map
    - staredit/scenario.chk ← The actual map chunk file
        - This file contains the triggers chunk
        - It contains strings table chunk
        - It contains a chunk describing buildings and units
    - Etc.

Map file in MPQ Editor

- Ladik's MPQ editor can be used to view or modify the contents of an MPQ map file
  http://zezula.net/en/mpq/download.html



Note the chunk file: "staredit/scenario.chk"

Scenario chunk file /1

- Made of one or more chunks:

```
typedef struct {
    union  {
        BYTE FourCC[4];
        DWORD ckID;
    };
    DWORD ckSize;
} CK_HDR;
```

- Chunk header is followed by the chunk body
- The game parses each chunk based on its ID:

```
static const CHUNKTABLE Bwar100Pass2_MapSettings[] = {
    {{'S','T','R',' '}, maphdr_STR,          true},
    {{'M','T','X','M'}, maphdr_MTXM,         true},
    {{'T','H','G','2'}, maphdr_THG2,         true},
    {{'M','A','S','K'}, maphdr_MASK,         true},
    {{'U','N','I','x'}, maphdr_UNIS_Bwar,    true},  // Brood War handler
    {{'U','P','G','x'}, maphdr_UPGS_Bwar,    true},  // Brood War handler
    {{'T','E','C','x'}, maphdr_TECS_Bwar,    true},  // Brood War handler
    {{'P','U','N','I'}, maphdr_PUNI,         true},
    {{'P','U','P','x'}, maphdr_PUPG_Bwar,    true},  // Brood War handler
    {{'P','T','E','x'}, maphdr_PTEC_Bwar,    true},  // Brood War handler
    {{'U','N','I','T'}, maphdr_UNIT,         true},
    {{'U','P','R','P'}, maphdr_UPRP,         true},
    {{'M','R','G','N'}, maphdr_MRGN_Ext,     true},
    {{'T','R','I','G'}, maphdr_TRIG,         true},
    {{'C','O','L','R'}, maphdr_COLR,         true},  // new chunk
};
```

Scenario chunk file /2

- Some chunks might have their own sub-headers

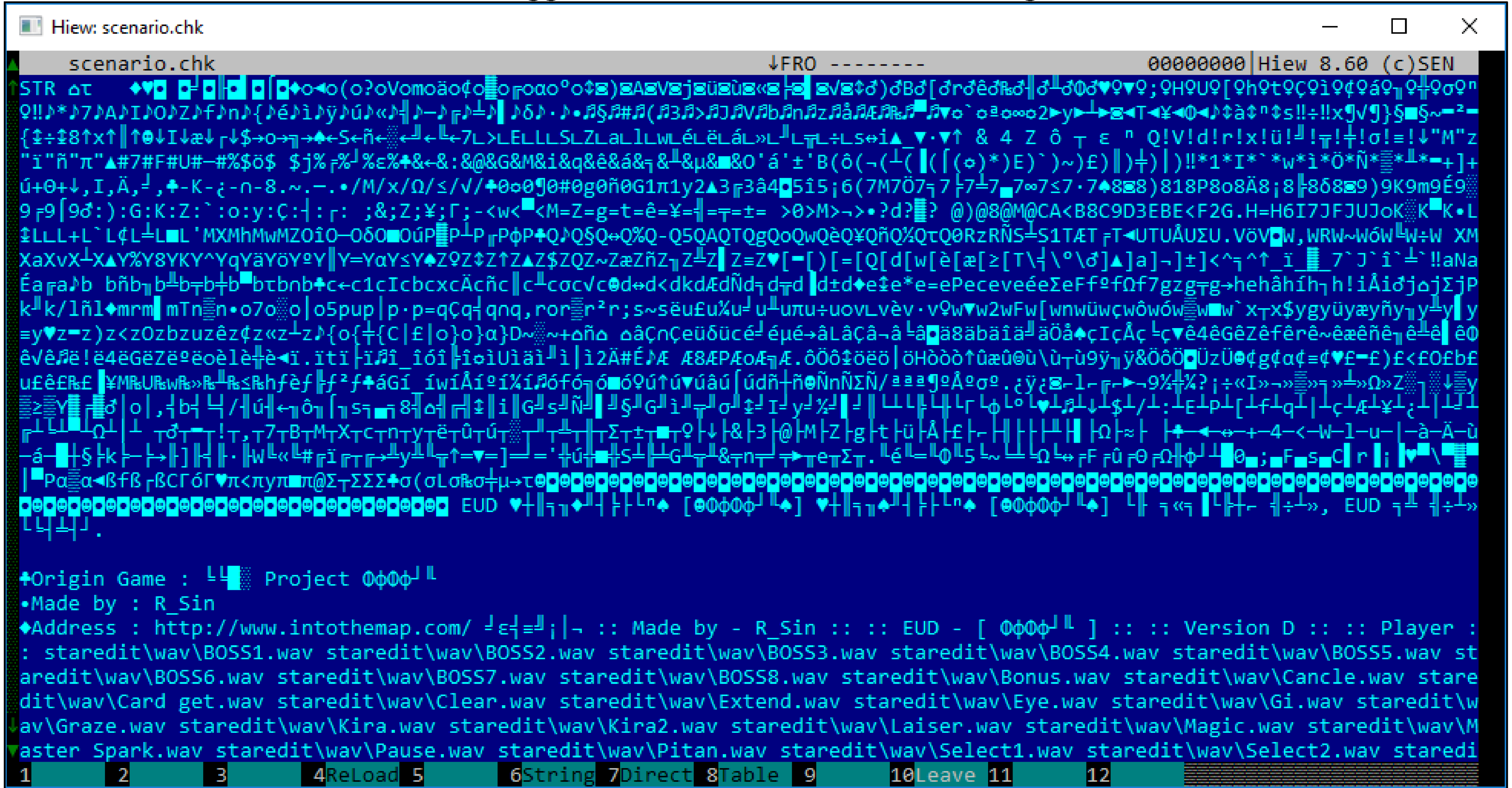
- The strings chunk is such an example:

```
typedef struct TStrTbl {
    UWORD wStrCount;         // number of strings in table
    UWORD wStrOffsets[1];    // variable number of str pointers
    // variable length string data follows pointers
} TStrTbl, * TPStrTblPriv;
```

```
//**************************************
// STR - MAP STRING TABLE
//**************************************
static BOOL CALLBACK maphdr_STR(HCHUNK hChunk, DWORD dwSize, LPARAM data) {
    gpMapStrs = (TPStrTbl)ALLOC(dwSize);
    if (!gpMapStrs)
        return false;
    gdwMapStrSize = dwSize;

    if (!ReadChunk(hChunk, gpMapStrs))
        return false;

    return true;
}
```

Scenario chunk file /3

- The strings chunk can be used to hide data not used by the game directly
  - When CK_HDR.ckSize > (sizeof(the complete TStrTbl header) + ∑strlen(of all strings in the table))
- The modders hide additional triggers in the cave area of the string chunk



Scenario chunk file /4

- This screenshot shows the last string in the strings table
- That’s not the chunk’s end though, it is just the string table’s end
- The remaining bytes are additional triggers inserted by the EUD trigger compiler

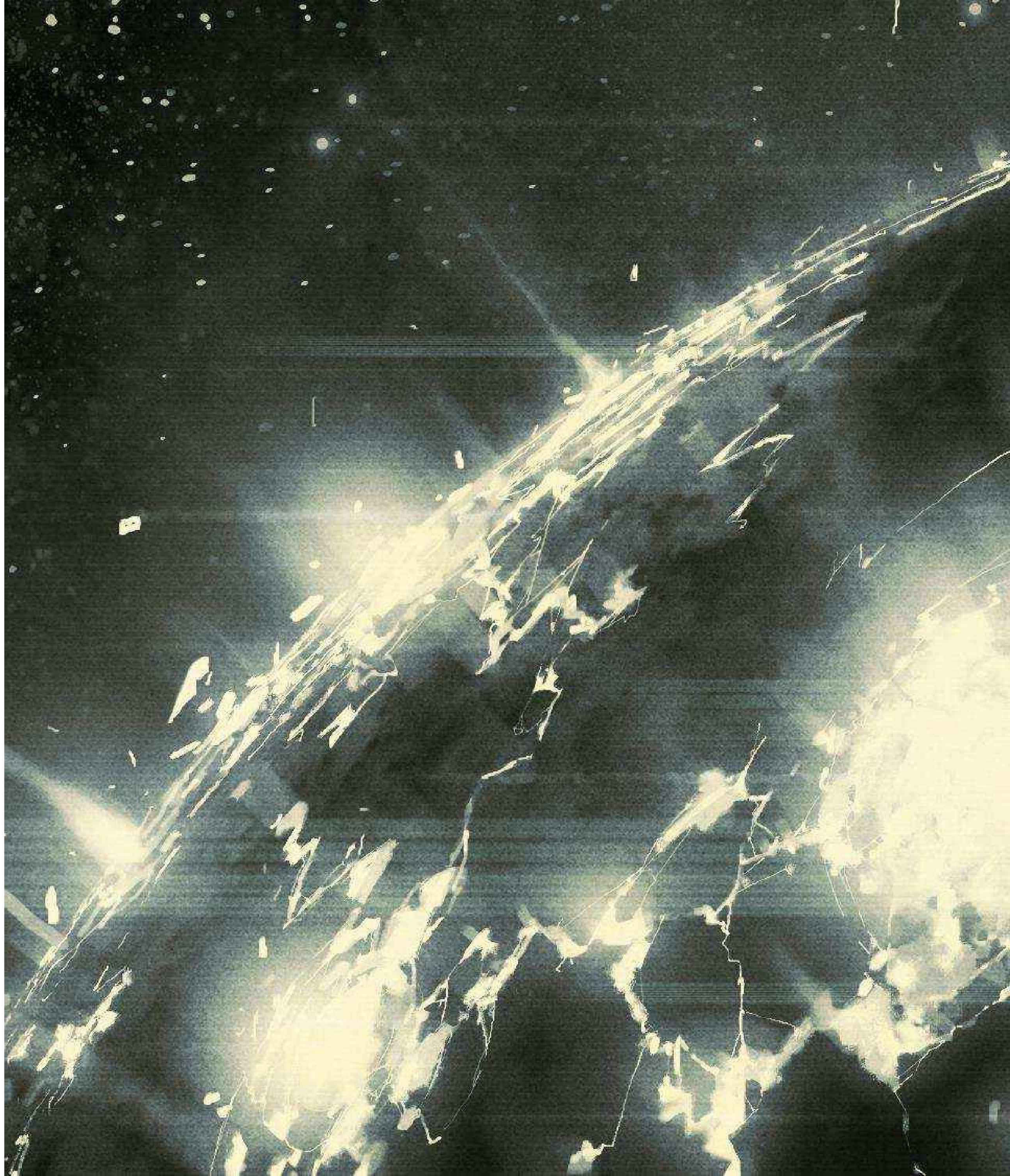
https://github.com/phu54321/

What are triggers? /1

- They are a set of conditions and actions that get evaluated during the game loop

- There are trigger conditions that tell you when:
  - A certain time period has elapsed (timers)
  - Player resources reached a certain amount
  - A map location has been reached
  - Etc.

- When all the trigger conditions are fulfilled, then you can do actions such as:
  - Play WAV file
  - Display a message
  - Create, kill, move a unit, etc.
  - Change unit owner and health points
  - Give player resources
  - Etc.



What are triggers? /2

- Triggers are stored inside the map chunk file
- The triggers chunk is simply an array of _trigger structs
- Each trigger has an array of the CONDITION and ACTION structures

- The dwPlayer and wType fields are user controlled
  - ➢ They are used to read/write out-of-bounds inside an array

- The bOpCode field dictates the trigger condition and action type

```
typedef struct _condition {
    .
    .
    DWORD   dwPlayer;         // stores a player slot or a _player_codes value
    ULONG   lQuantity;        // quantity of whatever -- units, kills, resources, time
    .
    UTYPE   wType;            // stores a unit type or a _unit_codes value
    .
    UBYTE   bQualifier;       // specifies the comparison operation (less than, greater than, etc.)
    UBYTE   bOpCode;          // stores a _trigger_codes value
    .
    .
    .
} CONDITION, *PCONDITION;

typedef struct _action {
    .
    .
    .
    DWORD   dwPlayer;         // a player slot or mission briefing portrait slot
    ULONG   lParm;            // stores a switch #, timer modification/value
    .
    UTYPE   wType;            // stores a unit code, res code, score code
    .
    UBYTE   bOpCode;          // stores a _action_codes value
    UBYTE   bQualifier;       // specifies the operation modifier
    .
    .
} ACTION, *PACTION;

//**************************************
// trigger struct
//**************************************
typedef struct _trigger {
    CONDITION   tConditions[MAX_CONDITIONS];
    ACTION      tActions[MAX_ACTIONS];

    ULONG       lFlags;
    UBYTE       ubPlayer[NUM_PLAYER_CODES];
    UBYTE       bCurrAction;
} TRIGGER, *PTRIGGER;

NODEDECL(TRIGGERNODE) {
    TRIGGER     t;
} *PTRIGGERNODE;
```

What are triggers? /3

- The bOpCode field is used to select which condition or action to execute:

```
ConditionFcn sgConditionFcns[NUM_CONDITION_CODES] = {
    cond_always,

    cond_timer,
    cond_control,
    .
    .
    .
    cond_opponents,
    cond_deaths,

    cond_least_control,
    cond_least_control_atloc,
    cond_least_kills,
    cond_lowest_score,
    cond_least_resource,

    cond_score,

    cond_always,
    cond_never,
};
```

```
ActionFcn sgActionFcns[NUM_ACTION_CODES] = {
    act_none,
    act_victory,
    act_defeat,
    .
    .
    .
    act_doodad,
    act_invincible,
    act_create_unit,

    act_set_deaths,

    .
    .
    .
    act_set_unit_res,
    act_set_hangar,

    act_stop_timer,
    act_start_timer,

    act_draw,
    act_alliance,

    act_disable_escape,
    act_enable_escape,
};
```

What are triggers? /4

- Each trigger condition is evaluated, then the actions are performed if all conditions succeed:

```
static void trigger_parse(LISTPTR(TRIGGERNODE) pTrigList) {
    // parse the conditions for each trigger
    ITERATELISTPTR(TRIGGERNODE, pTrigList, pTrigger) {

        // has this trigger already executed?
        if (pTrigger->t.lFlags & TF_COMPLETE) continue;

        sgpCurrTrigger = pTrigger;
        if (! (pTrigger->t.lFlags & TF_EXECUTING)) {
            // if the trigger isn't already executing, parse the conditions
(1)         if (! trigger_cond_parse(pTrigger))
                continue;                           // conditions were not met
            pTrigger->t.bCurrAction = 0;              // start with action 0
        }

(2)     // execute the trigger's actions
        trigger_execute(pTrigger);
    }
}
```

What are triggers? /5

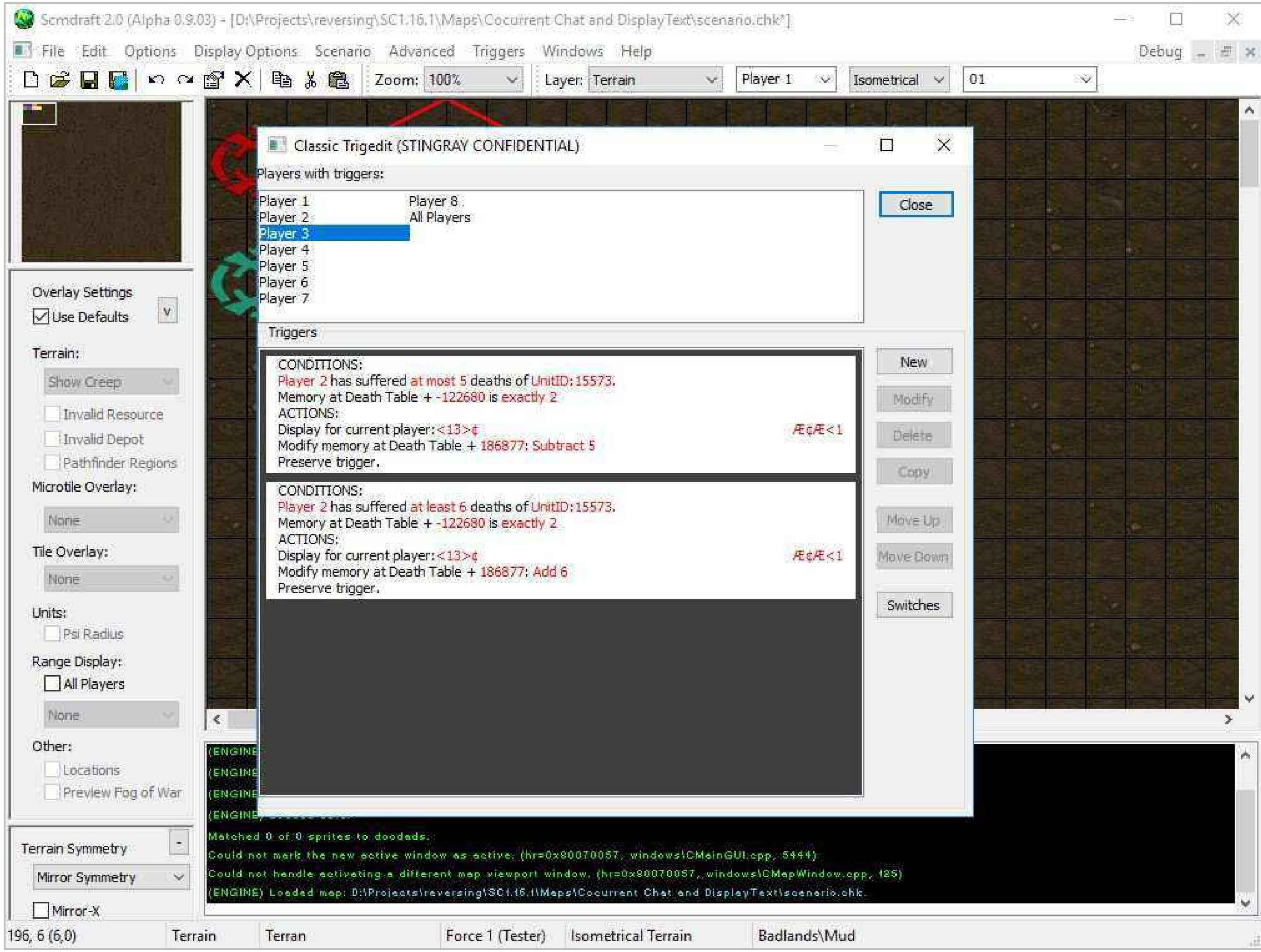
```
static int trigger_cond_parse(PTRIGGERNODE pTrigger) {
    PCONDITION pCond;

    for (int i = 0; i < MAX_CONDITIONS; i++) {
        pCond = &pTrigger->t.tConditions[i];

        // the map editor can be used to disable a condition
        if (pCond->bFlags & CF_DISABLED)
            continue;

        // no condition indicates the end of the list
        if (pCond->bOpCode == COND_NONE)
            break;

        // call the function associated with the current condition
        app_assert(pCond->bOpCode < NUM_CONDITION_CODES);
        if (sgConditionFcns[pCond->bOpCode](pCond))
            continue;

        // one condition failed -- no need to check the rest
        return FALSE;
    }
    // all conditions were met
    return TRUE;
} « end trigger_cond_parse »
```

```
static void trigger_execute(PTRIGGERNODE pTrigger) {

    // execute trigger actions until an action doesn't complete
    int result = 1;
    while (result && (pTrigger->t.bCurrAction < MAX_ACTIONS)) {
        pAct = &pTrigger->t.tActions[pTrigger->t.bCurrAction];

        .
        .
        .
        // no action indicates the end of the list
        if (pAct->bOpCode == ACT_NONE) {
            pTrigger->t.bCurrAction = MAX_ACTIONS;
            break;
        }

        // call the action function
        app_assert(pAct->bOpCode < NUM_ACTION_CODES);
        result = sgActionFcns[pAct->bOpCode](pAct);
        if (result) {
            pTrigger->t.bCurrAction++;
        }
    }
} « end trigger_execute »
```

What are triggers? /6

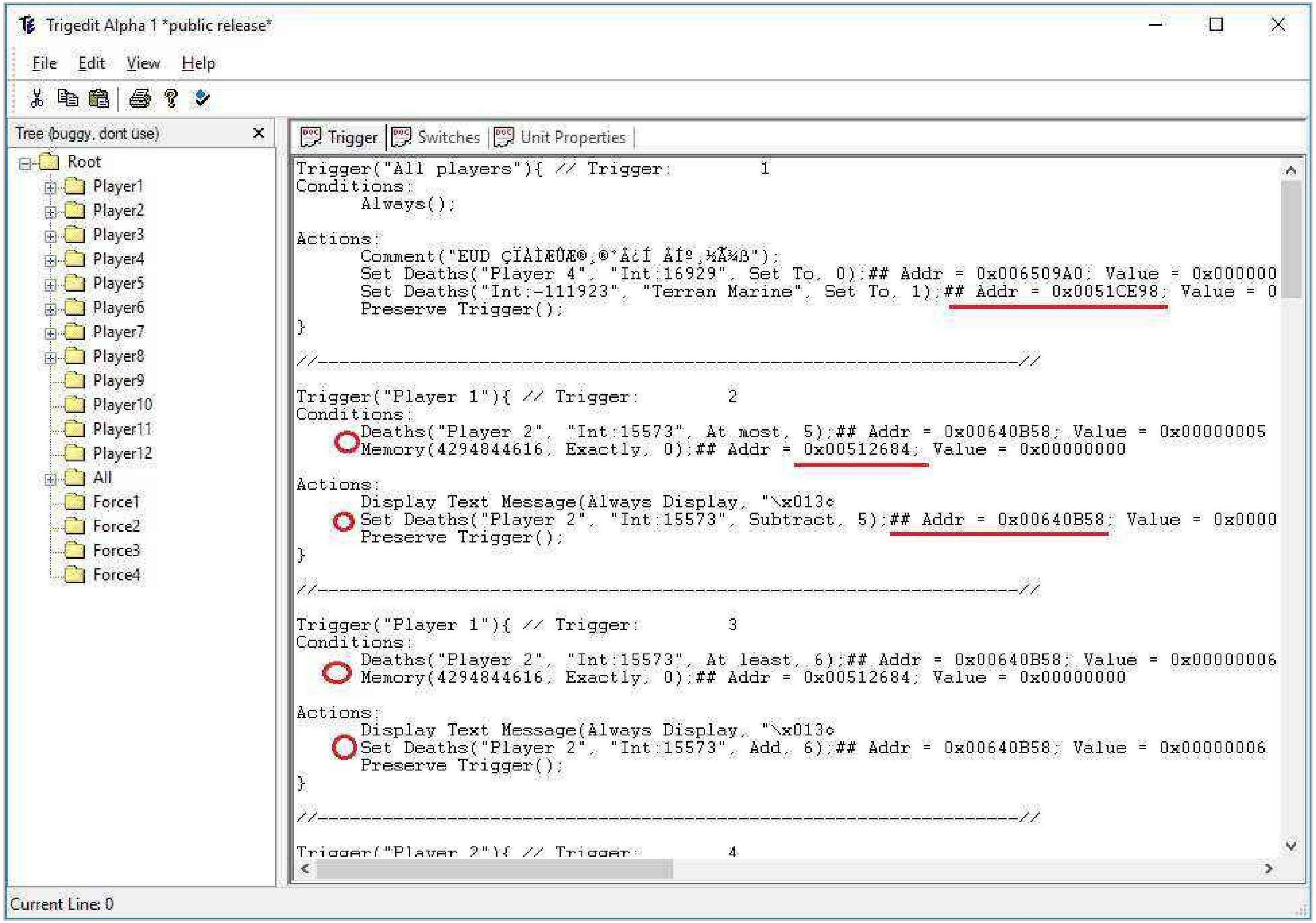
- Classic (visual) trigger editor (SCMDraft 2.0 – by Henrik Arlinghaus)

- Note the large values:
  - UnitID
  - Death table index
  - Etc.



What are triggers? /7

- Text trigger editor
- A private build of SCMDraft shows the EUD overflow addresses



The buffer overflow /1

- The buffer overflow bug in question is found in the “Extended Unit Death” trigger code:
  - The death_count() trigger <u>condition</u>
    - → Read anywhere primitive

  - The set/add/sub_death_count() trigger <u>action</u>
    - → Write anywhere primitive

- Triggers are read as-is from the chunk file and stored in a doubly-linked list:

```
static BOOL CALLBACK maphdr_TRIG(HCHUNK hChunk, DWORD dwSize, LPARAM data) {
    if (dwSize % sizeof(TRIGGER)) return false;
    PTRIGGER pTrigBuf = (PTRIGGER)ALLOCTEMP(dwSize);
    if (!ReadChunk(hChunk, pTrigBuf)) {
        FREE(pTrigBuf);
        return false;
    }

    PTRIGGER pTrigger = pTrigBuf;
    int nTriggers = dwSize / sizeof(TRIGGER);

    for (int n = 0; n < nTriggers; n++, pTrigger++) {
        if(!AddTrigger(pTrigger))
            break;
    }

    FREE(pTrigBuf);
    return true;
}
```

The buffer overflow /2

- A death condition with out-of-bounds unit type (wType) or player number (dwPlayer) causes the read anywhere primitive

```
ULONG death_count(DWORD dwPlayer, UWORD wType, ULONG) {
    app_assert(dwPlayer < NUM_PLAYER_CODES);
    switch (dwPlayer) {
        case PLYR_THIS:
            dwPlayer = TriggerPlayer();
        break;

        case PLYR_NAVA:
            .
            .
        case PLYR_UNUSED1:
        case PLYR_UNUSED2:
        case PLYR_UNUSED3:
        case PLYR_UNUSED4:
        return 0;

        default:
            // dwPlayer is not a special code -- advance to next switch
        break;
    }

    switch (wType) {
        case UNITS_ALL:
        return s.glGameCounts[COU_LOST_MEN][dwPlayer] + s.glGameCounts[COU_LOST_BLDGS][dwPlayer];

        case UNITS_MEN:
        return s.glGameCounts[COU_LOST_MEN][dwPlayer];

        case UNITS_BLDGS:
        return s.glGameCounts[COU_LOST_BLDGS][dwPlayer];

        case UNITS_FACTORIES:
        return s.glGameCounts[COU_LOST_FACTORIES][dwPlayer];

        default:
        return s.glUnitCounts[COU_UNI_DEATH][wType][dwPlayer];
    }
} « end death_count »
```

```
00460446 loc_460446:                             ; CODE XREF: death_count+21↑j
00460446                 lea     eax, [eax+eax*2]
00460449                 lea     ecx, [ecx+eax*4]
0046044C                 mov     eax, (g_s.glUnitCounts+8040h)[ecx*4]
00460453
00460453 locret_460453:
00460453                 retn    4
```

The buffer overflow /3

- A set death action causes a <u>write anywhere</u> and provic the following primitives:
  - [mem] += lQuantity
  - [mem] -= lQuantity
  - [mem] = lQuantity

```c
static int act_set_deaths(PACTION pAct) {
    app_assert(pAct->bQualifier < NUM_QUALIFIER_CODES);
    switch (pAct->bQualifier) {
        case QUAL_SET:
            set_deaths(pAct->dwPlayer, pAct->wType, pAct->lParm);
        break;

        case QUAL_ADD_TO:
            add_deaths(pAct->dwPlayer, pAct->wType, pAct->lParm);
        break;

        case QUAL_SUB_FROM:
            sub_deaths(pAct->dwPlayer, pAct->wType, pAct->lParm);
        break;
    }
    return 1;
}
```

```
004C5EBD lea     ecx, [ecx+ecx*2]
004C5EC0 lea     edx, [eax+ecx*4]
004C5EC3 mov     eax, [ebp+lQuantity]
004C5EC6 mov     (g_s.glUnitCounts+8040h)[edx*4], eax
```

```c
static ULONG set_deaths(DWORD dwPlayer, UTYPE wType, ULONG lQuantity) {
    app_assert(dwPlayer < NUM_PLAYER_CODES);
    switch (dwPlayer) {
        case PLYR_NAVA:     return nava_enum(dwPlayer, wType, lQuantity, set_deaths);
        case PLYR_FOES:     return foes_enum(dwPlayer, wType, lQuantity, set_deaths);
        case PLYR_ALLIES:   return allies_enum(dwPlayer, wType, lQuantity, set_deaths);
        case PLYR_NEUTRALS: return neutrals_enum(dwPlayer, wType, lQuantity, set_deaths);
        case PLYR_ALL:      return all_enum(dwPlayer, wType, lQuantity, set_deaths);
        case PLYR_GROUP_A:
        case PLYR_GROUP_B:
        case PLYR_GROUP_C:
        case PLYR_GROUP_D:  return group_enum(dwPlayer, wType, lQuantity, set_deaths);
        case PLYR_UNUSED1:
        case PLYR_UNUSED2:
        case PLYR_UNUSED3:
        case PLYR_UNUSED4:  return 0;

        case PLYR_THIS:
            dwPlayer = TriggerPlayer();
        break;

        default:
            // dwPlayer is not a special code -- advance to next switch
        break;
    } « end switch dwPlayer »

    // dwPlayer is valid -- give resource to specified player
    if (dwPlayer >= NET_MAX_NODES)
        return 0;

    switch (wType) {
        .
        .
        case UNITS_BLDGS:
            s.glGameCounts[COU_LOST_BLDGS][dwPlayer] = lQuantity;
        break;

        case UNITS_FACTORIES:
            s.glGameCounts[COU_LOST_FACTORIES][dwPlayer] = lQuantity;
        break;

        default:
            s.glUnitCounts[COU_UNI_DEATH][wType][dwPlayer] = lQuantity;
        break;
    }
    return 0;
} « end set_deaths »
```

The buffer overflow /4

- An example of EUD triggers found inside an EUD map:

```
//-----------------------------------------------------------------//

Trigger("All players"){ // Trigger:      107
Conditions:
    Deaths("Player 11", "Terran Marine", At least, 134217728);## Addr = 0x0058A38C; Value = 0x08000000

Actions:
    Set Deaths("Player 11", "Terran Marine", Subtract, 134217728);## Addr = 0x0058A38C; Value = 0x08000000
    Set Deaths("Current Player", "Terran Marine", Add, 134217728);## Addr = 0x0058A398; Value = 0x08000000
}

//-----------------------------------------------------------------//

Trigger("All players"){ // Trigger:      108
Conditions:
    Deaths("Player 11", "Terran Marine", At least, 67108864);## Addr = 0x0058A38C; Value = 0x04000000

Actions:
    Set Deaths("Player 11", "Terran Marine", Subtract, 67108864);## Addr = 0x0058A38C; Value = 0x04000000
    Set Deaths("Current Player", "Terran Marine", Add, 67108864);## Addr = 0x0058A398; Value = 0x04000000
}

//-----------------------------------------------------------------//

Trigger("All players"){ // Trigger:      109
Conditions:
    Deaths("Player 11", "Terran Marine", At least, 33554432);## Addr = 0x0058A38C; Value = 0x02000000

Actions:
    Set Deaths("Player 11", "Terran Marine", Subtract, 33554432);## Addr = 0x0058A38C; Value = 0x02000000
    Set Deaths("Current Player", "Terran Marine", Add, 33554432);## Addr = 0x0058A398; Value = 0x02000000
}
```

EUD map emulation – Problem statement

- Given a StarCraft map that contains malformed input that triggers a read/write anywhere:
  - Is there is a way to emulate the buffer overflow in a newer game version where:
    - The buffer overflow bug is fixed
    - Some addresses no longer exist in the new game version
    - Some addresses refer to new/different data structure format

  ?

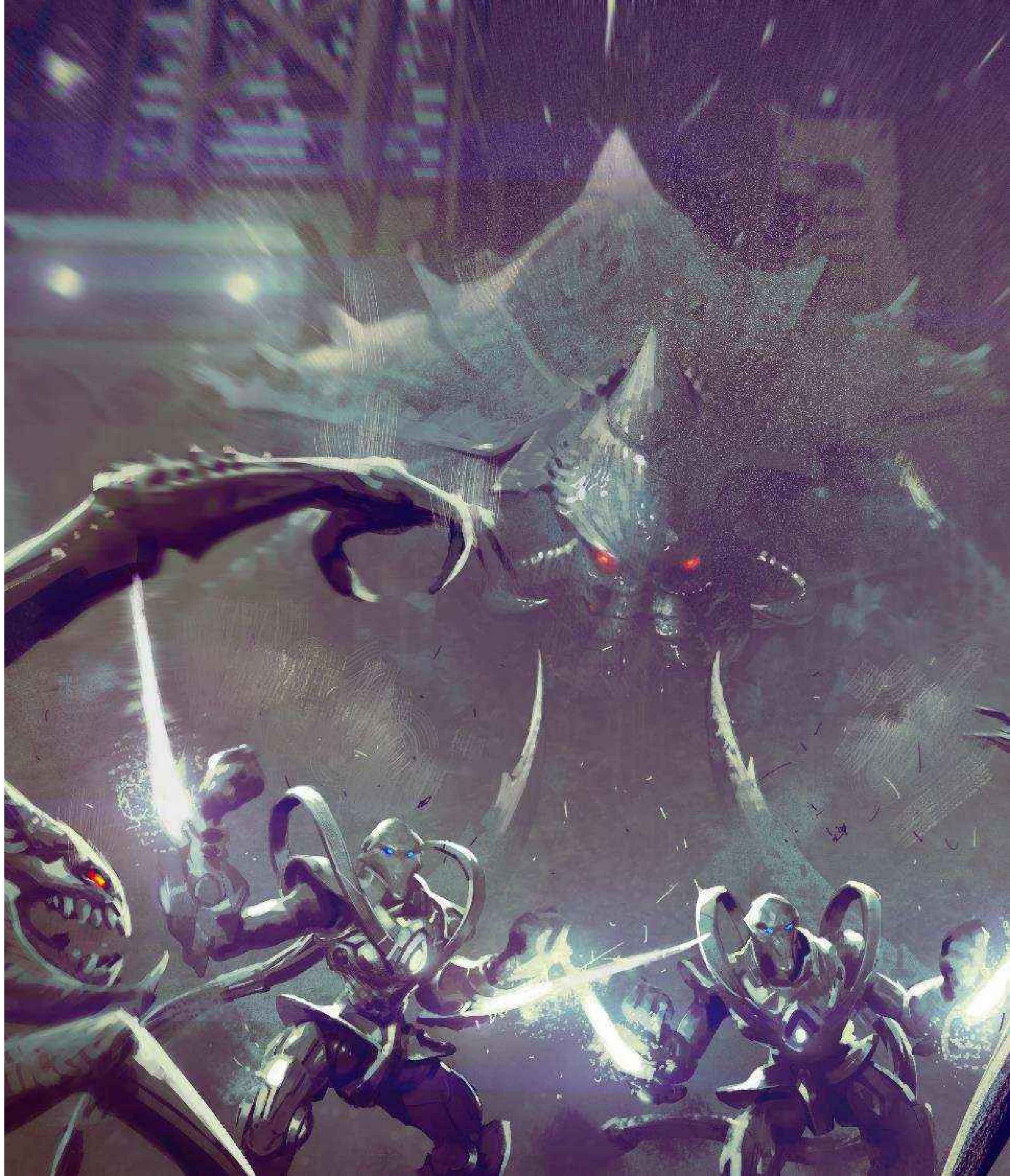
  - Can the emulator work on different architectures and operating systems?

Three steps solution

1. Identify
   - Identify / trace all the addresses used by an EUD map
   - Build a table of the addresses and identify what they represent in the game source code

2. Intercept
   - Intercept all out-of-bounds access
   - Redirect access using a translation table
     - Old address → New address

3. Emulate
   1. Missing memory addresses should be handled by code
   2. Dangerous memory changes should be filtered / changed accordingly (pointers, function callbacks, etc.)

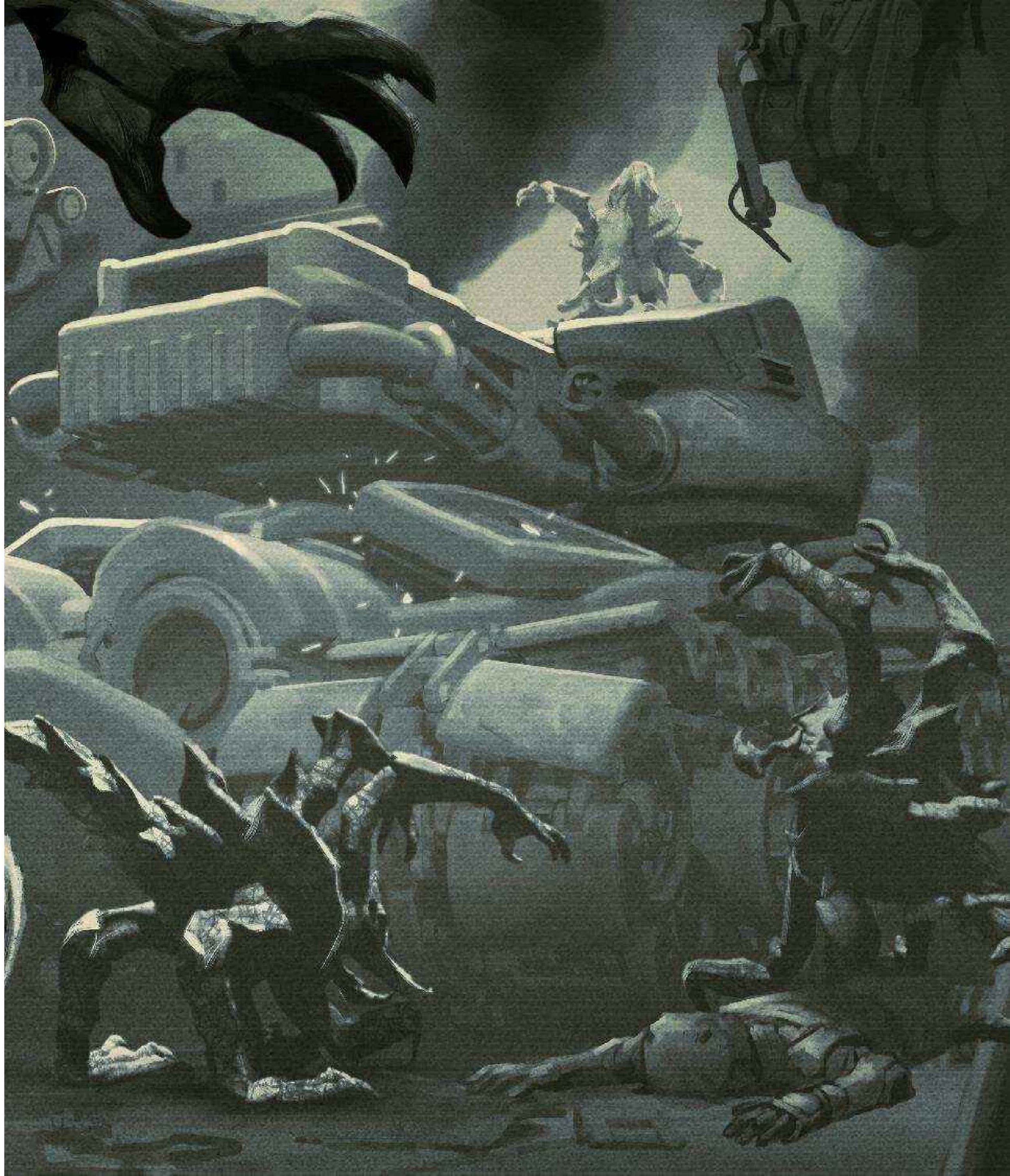


Implementation challenges

1. Identify
   - Unfortunately, we did not have private or public symbols for StarCraft 1.16.1. I had to start reversing the game executable from scratch
   - How can I tell what addresses the maps are accessing?
   - What is the goal/intent behind a memory access?

2. Intercept
   1. No problems here. Luckily, we can funnel all the out-of-bounds read/writes to the emulation layer

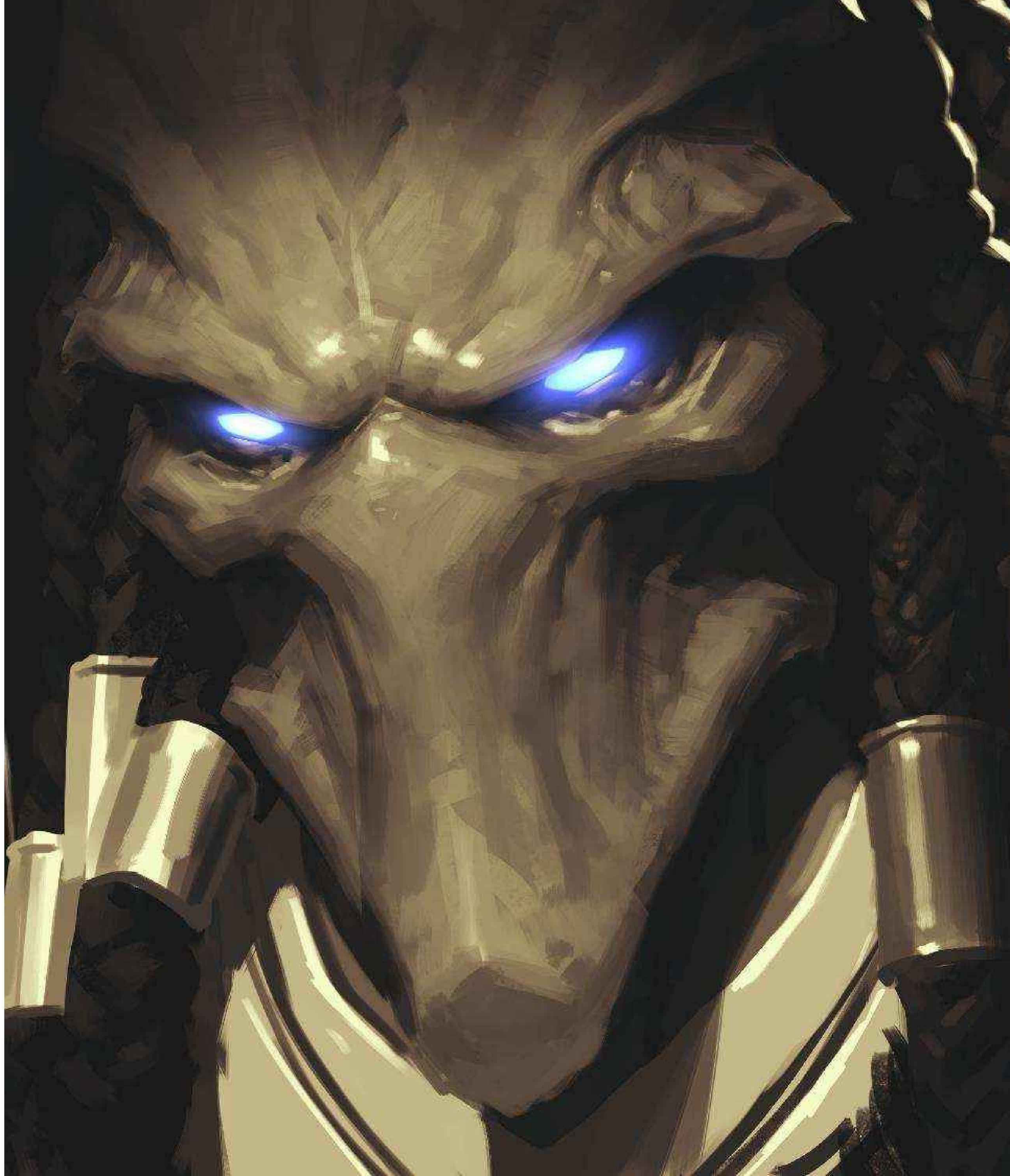
3. Emulate
   1. Handle basic memory access emulation
   2. Emulate addresses that are no longer present
   3. Emulate incompatible structure types



Identify – Reversing the game /1

1. Reverse engineering efforts were impeded by the lack of debugging symbols:
   - Reverse engineered the game client from scratch
   - Used the closest source code snapshot for 1.16.1
   - Found the right compiler (VS 2003) and the approximate optimization switches
     - ➢ Now I have debugging symbols for a binary that is very close to the public build

2. I used binary diffing plugins for IDA Pro
   1. PatchDiff2 - Tenable Network Security, Inc
   2. Diaphora - http://diaphora.re/



Identify – Reversing the game /2

- Binary diffing was limited:
  - Mismatched functions between the diffed binaries
  - Global variables were not identified
  - Optimized code and inlined functions made diffing harder

- Resorted to manual reverse engineering to bridge the limitations from BinDiffing

- Used scripting to automate the reversing task
  - Lots of IDAPython scripting was involved

Source code vs Disassembly view

```
// returns TRUE if trigger is completed
//**************************************
static void trigger_execute(PTRIGGERNODE pTrigger
    PACTION pAct;

    pTrigger->t.lFlags |= TF_EXECUTING;

    // if this is a victory trigger, there may be a dela
    if (pTrigger->t.lFlags & TF_VICTORY)
        sgbVictoryCodes[sgdwTriggerPlayer] = CODE_DELAYE

    // execute trigger actions until an action doesn't c
    int result = 1;
    while (result && (pTrigger->t.bCurrAction < MAX_ACTI
        pAct = &pTrigger->t.tActions[pTrigger->t.bCurrAc

        // the map editor can be used to disable actions
        if (pAct->bFlags & AF_DISABLED) {
            // skip this action
            pTrigger->t.bCurrAction++;
            continue;
        }

        // no action indicates the end of the list
        if (pAct->bOpCode == ACT_NONE) {
            pTrigger->t.bCurrAction = MAX_ACTIONS;
            break;
        }

        // call the action function
        app_assert(pAct->bOpCode < NUM_ACTION_CODES);
        result = sgActionFcns[pAct->bOpCode](pAct);
        if (result) {
            pTrigger->t.bCurrAction++;
        }
    } « end while result&&(pTrigger->t.... »

    // are we done with this trigger?
    if (pTrigger->t.bCurrAction < MAX_ACTIONS)
        return;

    // if this trigger paused the game, but never unpaus
    if (pTrigger->t.lFlags & TF_PAUSED)
        sgActionFcns[ACT_UNPAUSE_GAME](NULL);

    // reset the trigger appropriately
    if (pTrigger->t.lFlags & TF_KEEP_TRIGGER) {
        pTrigger->t.bCurrAction = 0;
//  fgp.patch 1.05 begin
        pTrigger->t.lFlags &= ~(TF_EXECUTING | TF_ABORTE
//  fgp.patch 1.05 end
    }
    else {
        pTrigger->t.lFlags |= TF_COMPLETE;
    }
} « end trigger_execute »
```

```
Symbol Name
Symbol
tree_desc
tree_desc
tree_desc
tree_desc
tree_desc
TREPLAYI
TREPULSE
TResNod
TResNod
TResNod
TResNod
TRIGGER
_trigger
_trigger::t
_trigger::l
_trigger::t
_trigger::t
_trigger::u
TriggerPl
TriggerPl
trigger_al
trigger_al
trigger_cc
trigger_ex
trigger_fr
trigger_fr
TRIGGER_
trigger_pa
TrimGam
TrimGam
TrimGam
TrimGam
TRUE
TRUE
TRUE
TRY
TryAbort(
TryAbort(
TRYFREE
TryNextO
TryNextO
```

IDA View-A | Recent scripts

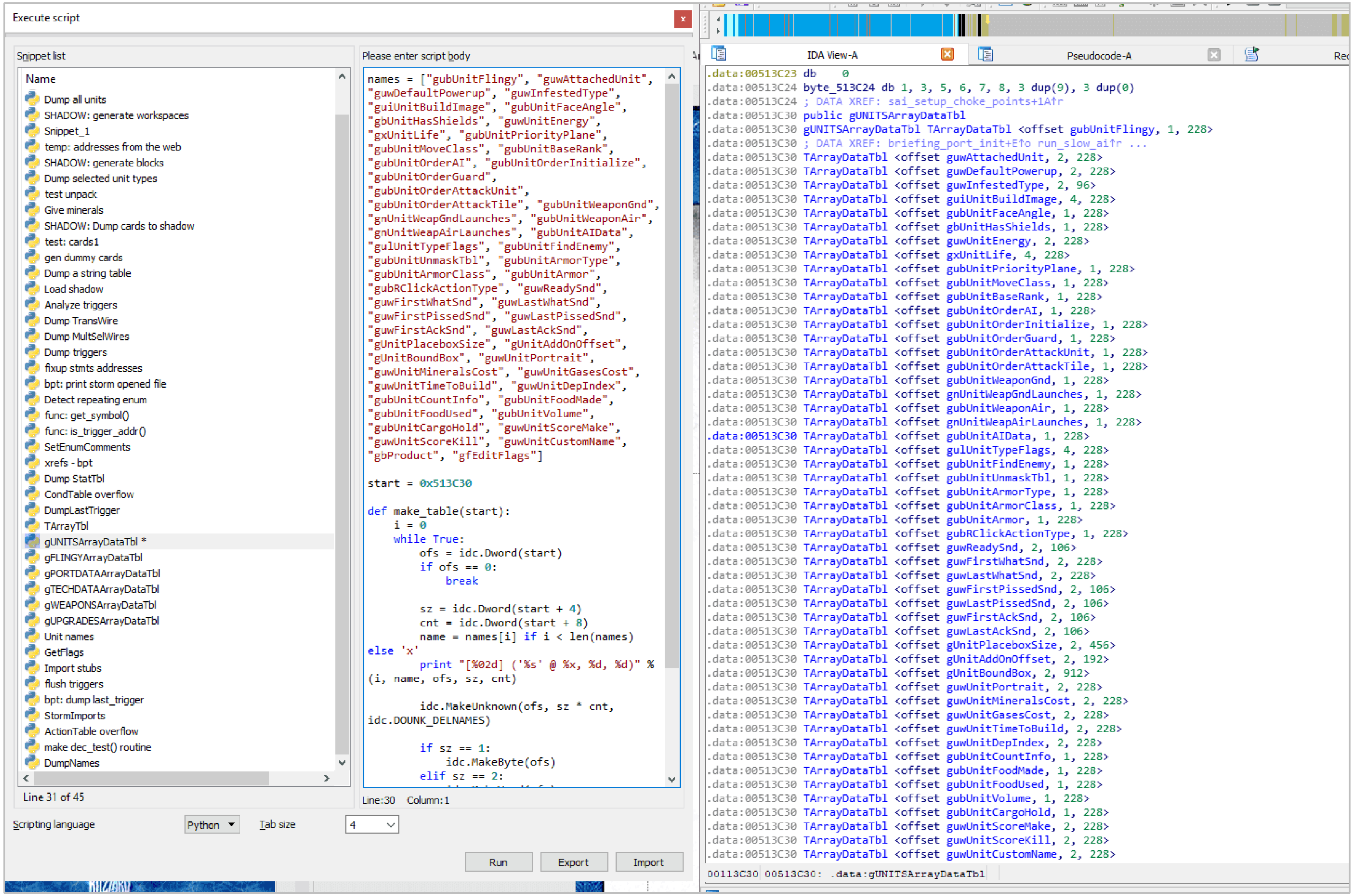
```
.text:00489130 ; void __usercall trigger_execute(z_TRIGGERNODE *pTrigger@<esi>)
.text:00489130 public trigger_execute
.text:00489130 trigger_execute proc near
.text:00489130 ; CODE XREF: trigger_parse:loc_489488↓p
.text:00489130 mov     edx, [esi+z_TRIGGERNODE.t.lFlags]
.text:00489136 or      edx, TF_EXECUTING
.text:00489139 mov     eax, edx
.text:0048913B
.text:0048913B test    al, TF_VICTORY
.text:0048913D mov     [esi+z_TRIGGERNODE.t.lFlags], edx
.text:00489143
.text:00489143 jz      short loc_489151
.text:00489143
.text:00489145 mov     eax, sgdwTriggerPlayer
.text:0048914A
.text:0048914A mov     sgbVictoryCodes[eax], CODE_DELAYED_VICTORY
.text:0048914A
.text:00489151
.text:00489151 loc_489151:; CODE XREF: trigger_execute+13↑j
.text:00489151 mov     eax, 1
.text:00489156 jmp     short @@loop_start
.text:00489156
.text:00489156 ; ---------------------------------------------------------------------------
.text:00489158 align 10h
.text:00489160
.text:00489160 @@loop_start:; CODE XREF: trigger_execute+26↑j
.text:00489160 ; trigger_execute+79↓j
.text:00489160 mov     dl, [esi+z_TRIGGERNODE.t.bCurrAction]
.text:00489166 cmp     dl, MAX_ACTIONS
.text:00489169 jnb     short @@ret_z
.text:00489169
.text:0048916B movzx   ecx, dl
.text:0048916E shl     ecx, 5
.text:00489171 test    [ecx+esi+z_TRIGGERNODE.t.tActions.bFlags], AF_DISABLED
.text:00489179 lea     ecx, [ecx+esi+z_TRIGGERNODE.t.tActions]; act
.text:00489180 jz      short loc_48918C
.text:00489180
.text:00489182 inc     dl
.text:00489184 mov     [esi+z_TRIGGERNODE.t.bCurrAction], dl
.text:0048918A jmp     short loc_4891A7
.text:0048918A
.text:0048918C ; ---------------------------------------------------------------------------
.text:0048918C
.text:0048918C loc_48918C:; CODE XREF: trigger_execute+50↑j
.text:0048918C mov     al, [ecx+_action.bOpCode]
.text:0048918F test    al, al
.text:00489191 jz      short @@no_opcode
```

Source code vs Hex-Rays pseudo-code

```
static void trigger_execute(PTRIGGERNODE pTrigger) {
    PACTION pAct;

    pTrigger->t.lFlags |= TF_EXECUTING;

    // if this is a victory trigger, there may be a delayed victory whil
    if (pTrigger->t.lFlags & TF_VICTORY)
        sgbVictoryCodes[sgdwTriggerPlayer] = CODE_DELAYED_VICTORY;

    // execute trigger actions until an action doesn't complete
    int result = 1;
    while (result && (pTrigger->t.bCurrAction < MAX_ACTIONS)) {
        pAct = &pTrigger->t.tActions[pTrigger->t.bCurrAction];

        // the map editor can be used to disable actions
        if (pAct->bFlags & AF_DISABLED) {
            // skip this action
            pTrigger->t.bCurrAction++;
            continue;
        }

        // no action indicates the end of the list
        if (pAct->bOpCode == ACT_NONE) {
            pTrigger->t.bCurrAction = MAX_ACTIONS;
            break;
        }

        // call the action function
        app_assert(pAct->bOpCode < NUM_ACTION_CODES);
        result = sgActionFcns[pAct->bOpCode](pAct);
        if (result) {
            pTrigger->t.bCurrAction++;
        }
    } « end while result&&(pTrigger->t.... »

    // are we done with this trigger?
    if (pTrigger->t.bCurrAction < MAX_ACTIONS)
        return;

    // if this trigger paused the game, but never unpaused, we want to u
    if (pTrigger->t.lFlags & TF_PAUSED)
        sgActionFcns[ACT_UNPAUSE_GAME](NULL);

    // reset the trigger appropriately
    if (pTrigger->t.lFlags & TF_KEEP_TRIGGER) {
        pTrigger->t.bCurrAction = 0;
//  fgp.patch 1.05 begin
        pTrigger->t.lFlags &= ~(TF_EXECUTING | TF_ABORTED | TF_NO_ABORT)
//  fgp.patch 1.05 end
    }
    else {
        pTrigger->t.lFlags |= TF_COMPLETE;
    }
} « end trigger_execute »
```

```
void __usercall trigger_execute(z_TRIGGERNODE *pTrigger@<esi>)
{
  unsigned __int32 trig_flag; // eax
  int ok; // eax
  unsigned __int8 bCurAction; // dl
  int iAction; // ecx
  bool b_disabled; // zf
  _action *act; // ecx
  unsigned __int8 bOpCode; // al
  unsigned int fl; // eax

  trig_flag = pTrigger->t.lFlags | TF_EXECUTING;
  pTrigger->t.lFlags = trig_flag;
  if ( trig_flag & TF_VICTORY )
    sgbVictoryCodes[sgdwTriggerPlayer] = CODE_DELAYED_VICTORY;
  //
  ok = 1;
  do
  {
    bCurAction = pTrigger->t.bCurrAction;
    if ( bCurAction >= (unsigned __int8)MAX_ACTIONS )
      break;
    //
    iAction = bCurAction;
    b_disabled = (pTrigger->t.tActions[iAction].bFlags & AF_DISABLED) == offsetof(z_TRIGGERNODE, m_link);
    act = &pTrigger->t.tActions[iAction];
    if ( b_disabled )
    {
      bOpCode = act->bOpCode;
      if ( !bOpCode )
      {
        pTrigger->t.bCurrAction = MAX_ACTIONS;
        break;
      }
      ok = sgActionFcns[bOpCode]();
      if ( !ok )
        break;
      ++pTrigger->t.bCurrAction;
    }
    else
    {
      pTrigger->t.bCurrAction = bCurAction + 1;
    }
  }
  while ( ok );
  if ( pTrigger->t.bCurrAction >= (unsigned __int8)MAX_ACTIONS )
```

Automating data structure recovery



Identify – Statically identify all addresses /1

- StarCraft Remastered collects game telemetry (including map information, etc.)

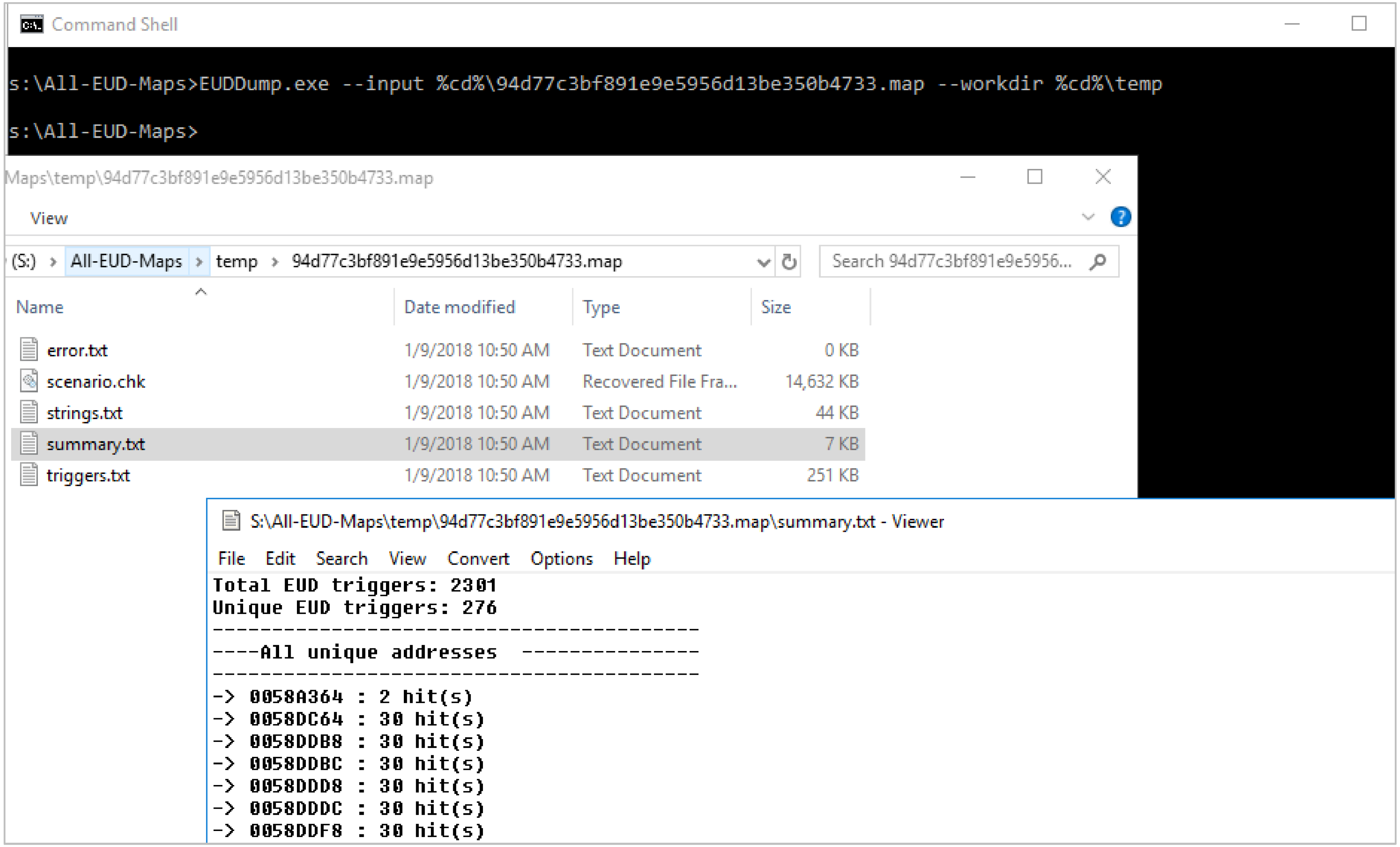
- As of October 2017, we had around ~603,773 total unique maps played
  - Of which 17,916 were EUD maps (i.e. contained out of bounds indices)

- After I managed to reverse engineer enough of the game, I wrote a tool to process all the maps, identify EUD maps and dump the out-of-bounds EUD addresses

Command Shell

```
s:\All-EUD-Maps>EUDDump.exe
Usage:
    --use-cache                 Do not parse again, just read the cache file.
    --input map_file            Specify the input map file
    --workdir                   Specify the work directory where caches and report files will be generated
    --hidden-triggers [hexofs]  Specify the hidden trigger offset within the strings table

s:\All-EUD-Maps>
```

Identify – Statically identify all addresses /2



Identify – Statically identify all addresses /3

- After aggregating the unique EUD addresses across all of the 17k EUD maps, I ended up with around ~800 variables used by popular EUD maps

- I wrote an IDAPython script to emit a table for all the unique addresses, their names and sizes

```
{'addr': 0x006610B0, 'size': 0x00000390, 'ida_name': "guiUnitBuildImage", 'flags': 0},
{'addr': 0x006616E0, 'size': 0x000000E4, 'ida_name': "gubUnitWeaponAir", 'flags': 0},
{'addr': 0x006617C8, 'size': 0x00000720, 'ida_name': "gUnitBoundBox", 'flags': 0},
{'addr': 0x00662180, 'size': 0x000000E4, 'ida_name': "gubUnitArmorClass", 'flags': 0},
{'addr': 0x00662268, 'size': 0x000000E4, 'ida_name': "gubUnitOrderInitialize", 'flags': 0},
{'addr': 0x00662350, 'size': 0x00000390, 'ida_name': "gxUnitLife", 'flags': 0},
{'addr': 0x00662860, 'size': 0x00000390, 'ida_name': "gUnitPlaceboxSize", 'flags': 0},
{'addr': 0x00662098, 'size': 0x000000E4, 'ida_name': "gubRClickActionType", 'flags': 0},
{'addr': 0x00661518, 'size': 0x000001C8, 'ida_name': "gfEditFlags", 'flags': 0},
{'addr': 0x006647B0, 'size': 0x000000E4, 'ida_name': 'gbUnitHasShields', 'flags': 0},
{'addr': 0x0065FC18, 'size': 0x000000E4, 'ida_name': 'gnUnitWeapAirLaunches', 'flags': 0},
{'addr': 0x0065FD00, 'size': 0x000001C8, 'ida_name': 'guwUnitGasesCost', 'flags': 0},
{'addr': 0x0065FFB0, 'size': 0x000001C8, 'ida_name': 'guwFirstWhatSnd', 'flags': 0},
{'addr': 0x00660178, 'size': 0x000000E4, 'ida_name': 'gubUnitAIData', 'flags': 0},
{'addr': 0x00660260, 'size': 0x000001C8, 'ida_name': 'guwUnitCustomName', 'flags': 0},
{'addr': 0x00660428, 'size': 0x000001C8, 'ida_name': 'guwUnitTimeToBuild', 'flags': 0},
{'addr': 0x006606D8, 'size': 0x000000E4, 'ida_name': 'gbProduct', 'flags': 0},
{'addr': 0x006607C0, 'size': 0x000001C8, 'ida_name': 'guwAttachedUnit', 'flags': 0},
{'addr': 0x00660988, 'size': 0x000000E4, 'ida_name': 'gubUnitCargoHold', 'flags': 0},
{'addr': 0x00660A70, 'size': 0x000001C8, 'ida_name': 'guwUnitDepIndex', 'flags': 0},
{'addr': 0x00660C38, 'size': 0x000001C8, 'ida_name': 'guwDefaultPowerup', 'flags': 0},
{'addr': 0x00660E00, 'size': 0x000001C8, 'ida_name': 'guwUnitEnergy', 'flags': 0},
{'addr': 0x00660FC8, 'size': 0x000000E4, 'ida_name': 'gubUnitMoveClass', 'flags': 0},
{'addr': 0x00661440, 'size': 0x000000D4, 'ida_name': 'guwLastAckSnd', 'flags': 0},
{'addr': 0x00661EE8, 'size': 0x000000D4, 'ida_name': 'guwLastPissedSnd', 'flags': 0},
{'addr': 0x00661FC0, 'size': 0x000000D4, 'ida_name': 'guwReadySnd', 'flags': 0},
{'addr': 0x006626E0, 'size': 0x00000180, 'ida_name': 'gUnitAddOnOffset', 'flags': 0},
{'addr': 0x00662BF0, 'size': 0x000001C8, 'ida_name': 'guwLastWhatSnd', 'flags': 0},
{'addr': 0x00663408, 'size': 0x000001C8, 'ida_name': 'guwUnitScoreMake', 'flags': 0},
{'addr': 0x006635D0, 'size': 0x000000E4, 'ida_name': 'gubUnitArmorType', 'flags': 0},
{'addr': 0x00663888, 'size': 0x000001C8, 'ida_name': 'guwUnitMineralsCost', 'flags': 0},
{'addr': 0x00663B38, 'size': 0x000000D4, 'ida_name': 'guwFirstPissedSnd', 'flags': 0},
{'addr': 0x00663C10, 'size': 0x000000D4, 'ida_name': 'guwFirstAckSnd', 'flags': 0},
{'addr': 0x00663DD0, 'size': 0x000000E4, 'ida_name': 'gubUnitBaseRank', 'flags': 0},
{'addr': 0x00663EB8, 'size': 0x000001C8, 'ida_name': 'guwUnitScoreKill', 'flags': 0},
{'addr': 0x00664410, 'size': 0x000000E4, 'ida_name': 'gubUnitVolume', 'flags': 0},
{'addr': 0x006645E0, 'size': 0x000000E4, 'ida_name': 'gnUnitWeapGndLaunches', 'flags': 0},
{'addr': 0x006646C8, 'size': 0x000000E4, 'ida_name': 'gubUnitFoodMade', 'flags': 0},
{'addr': 0x00664980, 'size': 0x000000C0, 'ida_name': 'guwInfestedType', 'flags': 0},
```

Identify – Statically identify all addresses /4

- Static address discovery was not enough:
    - Some EUD maps were dereferencing pointers and reaching into the heap
    - Some structures are complicated and linked to other structures (linked lists, TCtrl*, TDialog*, etc.)

- Need more tools:
    - I realized the need for a dynamic EUD address tracer
    - I also needed a way to single step / debug triggers

- I developed an EUDTracer, a DLL that hooks the game and instruments all the relevant trigger handling code

Identify – Dynamic tracer /1

- The instrumented game binary calls into the tracer DLL upon each read/write

```
.text:004C5EBC 90                    nop
.text:004C5EBD 8D 0C 49              lea     ecx, [ecx+ecx*2]
.text:004C5EC0 8D 14 88              lea     edx, [eax+ecx*4]
.text:004C5EC3 8B 45 08              mov     eax, [ebp+lQuantity]
.text:004C5EC6 89 04 95 64 A3 58 00  mov     g_s.glUnitCounts+8040h[edx*4], eax
.text:004C5EC6
.text:004C5ECD
.text:004C5ECD                       loc_4C5ECD:                  ; CODE XREF: set_death
.text:004C5ECD                                                    ; set_deaths+EB↑j ...
.text:004C5ECD E8 FE 61 22 00        call    hook_set_deaths
```

```
.text:00460446
.text:00460446                       loc_460446:                  ; CODE XREF: death_cou
.text:00460446 8D 04 40              lea     eax, [eax+eax*2]     ; jumptable 004603A7 d
.text:00460449 8D 0C 81              lea     ecx, [ecx+eax*4]     ; bpt_cond_deaths:addr
.text:0046044C 8B 04 8D 64 A3 58 00  mov     eax, [ecx*4+58A364h] ; cond_deaths: symbols
.text:00460453
.text:00460453                       loc_460453:
.text:00460453 E8 F8 BC 28 00        call    trace_death_count
```

```
//-----------------------------------------------------------------
static void install_tracer_hooks()
{
    HOOK_PTR(EHI_COND_PARSE)        = (DWORD)trigger_cond_parse;
    HOOK_PTR(EHI_TRIG_EXEC)         = (DWORD)trigger_execute;
    HOOK_PTR(EHI_SET_DEATHS)        = (DWORD)eud_act_set_deaths;
    HOOK_PTR(EHI_SUB_DEATHS)        = (DWORD)eud_act_sub_deaths;
    HOOK_PTR(EHI_ADD_DEATHS)        = (DWORD)eud_act_add_deaths;
    HOOK_PTR(EHI_COND_DEATH_COUNT)  = (DWORD)eud_cond_deaths;
    HOOK_PTR(EHI_TRIGGERS_LOOP)     = (DWORD)eud_triggers_loop;
}
```

Identify – Dynamic tracer /2

- The Python table containing EUD addresses is passed to a source code generator to emit C code and tables
- The tracer uses that table to account for memory access

```
eud_itemdef_t eud_items[EUD_ITEM_COUNT] =
{
    DEF_EUD_ITEM(0x0068C1F0, 0x00000004, sgpStatDataDlg),
    DEF_EUD_ITEM(0x0068C234, 0x00000004, sgpStatResDlg),
    DEF_EUD_ITEM(0x0059CB5C, 0x00000004, sgpMinimapDlg),
    DEF_EUD_ITEM(0x0068C140, 0x00000004, sgpTextBoxDlg),
    DEF_EUD_ITEM(0x0068C148, 0x00000004, sgpStatCmdDlg),
    DEF_EUD_ITEM(0x0068C224, 0x00000004, sgpStatMiscDlg),
    DEF_EUD_ITEM(0x00666570, 0x00000205, gubSpriteCanBeHit),
    DEF_EUD_ITEM(0x00628458, 0x00000004, gfpMtxSet),
    DEF_EUD_ITEM(0x006645E0, 0x000000E4, gnUnitWeapGndLaunches),
    DEF_EUD_ITEM(0x0065FC18, 0x000000E4, gnUnitWeapAirLaunches),
    DEF_EUD_ITEM(0x006CEFF8, 0x000004B0, gbInvalMap),
    DEF_EUD_ITEM(0x006D1260, 0x00000004, gpSquareMap),
    DEF_EUD_ITEM(0x006CA240, 0x000000D1, gubFlingyMinBank),
    DEF_EUD_ITEM(0x006D5EC8, 0x00000004, gpMtxInfo),
    DEF_EUD_ITEM(0x00628444, 0x00000004, gfpCellBuf),
    DEF_EUD_ITEM(0x00628494, 0x00000004, gfpCellMap),
    DEF_EUD_ITEM(0x004FF900, 0x0000000C, szLastReplayDesc),
    DEF_EUD_ITEM(0x0041E0D0, 0x00000004, addr_0041E0D0),
    DEF_EUD_ITEM(0x00655C58, 0x000001B8, gszFidgetSmk),
    DEF_EUD_ITEM(0x00655E80, 0x000001B8, gszTalkSmk),
```

Identify – Dynamic tracer /3

- When the game loads an EUD map, the tracer DLL intercepts all out-of-bounds access
- Any unknown address triggers a breakpoint for further analysis and identification
- After I identify an unknown address, I add it to the Python table which is used to update the tracer's EUD items table

```
void __stdcall eud_act_set_deaths(
    uint32_t val,
    uint32_t idx)
{
    inc_perf_counter(CT_SET_DEATH);

    DWORD addr = MAKE_EUD_ADDR_IDX(idx);
    auto eud = eud_get_item(addr, val, 1);

    if (eud == nullptr)
    {
        DBG_OUT("<Addr %08x: set_deaths; val: %08X>\n", addr, val);

        BPT_ONCE;
    }
}
```

```
void __stdcall eud_cond_deaths(
    uint32_t val,
    uint32_t idx)
{
    inc_perf_counter(CT_COND_DEATHS);

    DWORD addr = MAKE_EUD_ADDR_IDX(idx);

    auto eud = eud_get_item(addr, val, -2);

    if (eud == nullptr)
    {
        DBG_OUT("<Addr %08x: cond_deaths; val: %08X>\n", addr, val);
        BPT_ONCE;
    }
}
```

Identify – Dynamic tracer /4

- The tracer's main role is to guarantee that all the addresses referred to from the EUD map are accounted for

```
void eud_init_dynamic_items()
{
    // Get Storm base
    g_storm_base = (uint32_t)GetModuleHandleA("storm.dll");

    DBG_OUT("Initializing dynamic EUD items....\n");

    init_stringmap();
    init_stattxt();
    init_repulse_map();
    init_triggers_list();

    init_mpq_freeze(EUD_ITEM_DBG_MPQ_FREEZE);
    init_storm(EUD_ITEM_DBG_STORM_FLAGS);
    init_scripts();
    init_groups();
    init_graphics();
    init_overlaytrans();
}
```

```
void init_triggers_list()
{
    std::set<TRIGGERNODE *> visited;
    TSList_TRIGGERNODE *pTriggerList = sc_p_sgTriggers;
    for (int iPlayer = 0; iPlayer < NET_MAX_NODES; ++iPlayer, ++pTriggerList)
    {
        DBG_OUT("trigger list %d: %0x\n", iPlayer, pTriggerList);

        TRIGGERNODE *next = pTriggerList->m_terminator.m_next;
        for (int iTrig = 0;; ++iTrig)
        {
            uint32_t trig_addr = uint32_t(next);

            // Terminal?
            if ((trig_addr & 0x1) != 0)
                break;

            // Detect circular
            if (visited.find(next) != std::end(visited))
            {
                DBG_OUT("found circular dependency for %p in %d.%d\n", next, iPlayer, iTrig);
                break;
            }

            // Format the trigger name
            char trig_str[90];
            _snprintf_s(trig_str, _countof(trig_str), "trig%02d_%05d", iPlayer, iTrig);

            auto trig = eud_insert_item(trig_addr, trig_str, sizeof(TRIGGERNODE));
            trig->flags |= EIF_SRC_TRIGGERS | EIF_IS_DYNAMIC | EIF_DYNAMIC_NAME;

            visited.insert(next);

            next = next->m_link.m_next;
        }
    }
}
```

Identify – More debugging tools

- Having a way to record all accessed EUD addresses was not enough to understand the intent behind the access
- I had no real way to debug an EUD map:
  - I needed a way to nicely represent an EUD address
  - I needed to single step after each trigger
  - I needed a way to convert a series of read/write primitives to pseudo-code

Identify – EUD address to symbolic name /1

- If I wanted to trace triggers, I needed to have a way to convert an address to a nice variable representation
- So what is the symbolic representation of:
  - 0x5187E8 + (0xC * 3) + 4?
  - ➢ gCards[3].pBtns

```
00000000 TCard     struc ; (sizeof=0xC, align=0x4, mappedto_225)
00000000                                   ; XREF: .data:gCards/r
00000000 wBtnCount dd ?                    ; base 10
00000004 pBtns     dd ?                    ; offset
00000008 wSecondaryCard dd ?
0000000C TCard     ends
```

```
.data:005187E8              ; struct TCard gCards[250]
.data:005187E8 06 00 00 00+gCards    TCard <6, offset sgTMarineCard, 0FFFFh>; 0
.data:005187E8 D8 77 51 00+                              ; DATA XREF: statcmd_set_action_btns+1D↑o
.data:005187E8 FF FF 00 00+                              ; CUnit__SetOwner+91↑r ...
.data:005187E8 09 00 00 00+          TCard <9, offset sgTGhostCard, 0FFFFh>; 1
.data:005187E8 B8 7A 51 00+          TCard <6, offset sgTVultureCard, 0FFFFh>; 2
.data:005187E8 FF FF 00 00+          TCard <5, offset sgCombatCard, 0FFFFh>; 3
.data:005187E8 06 00 00 00+          TCard <0, 0, 0FFFFh>; 4
.data:005187E8 10 7C 51 00+          TCard <7, offset sgTTankCard, 0FFFFh>; 5
.data:005187E8 FF FF 00 00+          TCard <0, 0, 0FFFFh>; 6
.data:005187E8 05 00 00 00+          TCard <9, offset sgTSCVCard, 0FFFFh>; 7
.data:005187E8 D0 5D 51 00+          TCard <7, offset sgTWraithCard, 0FFFFh>; 8
.data:005187E8 FF FF 00 00+          TCard <8, offset sgTVesselCard, 0FFFFh>; 9
.data:005187E8 00 00 00 00+          TCard <6, offset sgTMarineCard, 0FFFFh>; 10
.data:005187E8 00 00 00 00+          TCard <7, offset sgTransportCard, 0FFFFh>; 11
.data:005187E8 FF FF 00 00+          TCard <6, offset sgTCruiserCard, 0FFFFh>; 12
.data:005187E8 07 00 00 00+          TCard <0, 0, 0FFFFh>; 13
.data:005187E8 88 7C 51 00+          TCard <0, 0, 0FFFFh>; 14
.data:005187E8 FF FF 00 00+          TCard <5, offset sgCombatCard, 0FFFFh>; 15
.data:005187E8 00 00 00 00+          TCard <8, offset sgTKerriganCard, 1>; 16
```

Identify – EUD address to symbolic name /2

- With the help of the Hex-Rays decompiler and other metadata, I wrote the function "R" to resolve an address into a nice symbolic name
  - ➢ If the array's indices are based on enums, then "R" will properly show the enum name instead of a numeric index

Please enter script body

```
print R(0x0058C348)[1]
print R(0x0058C378)[1]
print R(0x0058C3A8)[1]
print R(0x0058C3D8)[1]
print R(0x0058C408)[1]
print R(0x0058C474)[1]
print R(0x0058C478)[1]
print R(0x0058C498)[1]
print R(0x0058C4C8)[1]
print R(0x0058C4F8)[1]
print R(0x0058C528)[1]
print R(0x0058C558)[1]
print R(0x0058C56C)[1]
print R(0x0058CE28)[1]
print R(0x0058D2B0)[1]
print R(0x0058D2B4)[1]
print R(0x0058D2B8)[1]
print R(0x0058D314)[1]
print R(0x0058D338)[1]
print R(0x0058D340)[1]
print R(0x0058D368)[1]
print R(0x0058D36C)[1]
print R(0x0058D370)[1]
print R(0x0058D394)[1]
print R(0x0058D39C)[1]
print R(0x0058D3CC)[1]
print R(0x0058D3F8)[1]
print R(0x0058DC60)[1]
```

Output window

```
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_TRIBUNAL][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_ROBOTICS_BAY][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_SHIELD_BATTERY][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_KHAY_FORMATION][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_TEMPLE][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_MINE_V1][PLYR_FIVE]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_MINE_V1][PLYR_SIX]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_MINE_V2][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_MINE_V3][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_CAVE][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_CAVEIN][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_CANTINA][PLYR_TWO]
&g_s.glUnitCounts[COU_UNI_DEATH][UNI_CANTINA][PLYR_SEVEN]
(unsigned int *)&g_s.gubTechAllowed[0][4]
(unsigned int *)g_s.gubUpgradeLevels
(unsigned int *)&g_s.gubUpgradeLevels[0][4]
(unsigned int *)&g_s.gubUpgradeLevels[0][8]
(unsigned int *)&g_s.gubUpgradeLevels[2][8]
(unsigned int *)&g_s.gubUpgradeLevels[2][44]
(unsigned int *)&g_s.gubUpgradeLevels[3][6]
(unsigned int *)g_s.gubUpgradeLevels[4]
(unsigned int *)&g_s.gubUpgradeLevels[4][4]
(unsigned int *)&g_s.gubUpgradeLevels[4][8]
(unsigned int *)&g_s.gubUpgradeLevels[4][44]
(unsigned int *)&g_s.gubUpgradeLevels[5][6]
(unsigned int *)&g_s.gubUpgradeLevels[6][8]
(unsigned int *)&g_s.gubUpgradeLevels[7][6]
(unsigned int *)g_s.gExtMapRegions[0]
```

Python

Identify – Static pseudocode generator /1

- SCMDraft trigger editor textually represents the trigger script:

```
Trigger(" ¦¡ Team 1 ¦¡"," ¦¡ Team 2 ¦¡"){ // Trigger:     1022
Conditions:
    Deaths("Current Player", "Right Wall Flame Trap", Exactly, 1);## Addr = 0x0058CB88; Value = 0x00000001
    Deaths("Current Player", "Right Upper Level Door", Exactly, 2);## Addr = 0x0058CA38; Value = 0x00000002
    Deaths("Current Player", "Mineral Field (Type 1)", Exactly, 11);## Addr = 0x0058C498; Value = 0x0000000B
    Bring("Current Player", "Terran Science Vessel", "Invalid Location", At least, 1);
    Bring("Current Player", "Terran Science Vessel", "Invalid String", At least, 1);

Actions:
    Set Deaths("Current Player", "Floor Missile Trap", Set To, 0);## Addr = 0x0058C9A8; Value = 0x00000000
    Set Deaths("Current Player", "Right Wall Flame Trap", Set To, 100);## Addr = 0x0058CB88; Value = 0x00000064
    Set Deaths("Current Player", "Right Wall Missile Trap", Subtract, 25);## Addr = 0x0058CB58; Value = 0x00000019
    Set Deaths("Current Player", "Terran Valkyrie", Add, 50);## Addr = 0x0058AE78; Value = 0x00000032
    Set Deaths("Current Player", "Right Upper Level Door", Set To, 0);## Addr = 0x0058CA38; Value = 0x00000000
    Set Deaths("Current Player", "Protoss Shield Battery", Set To, 144);## Addr = 0x0058C3D8; Value = 0x00000090
    Move Location("Current Player", "Terran Science Vessel", "Invalid Location", "Invalid Location");
    Remove Unit("Current Player", "Terran Science Vessel");
    Set Deaths("Player 7", "Protoss Arbiter", Set To, 12);## Addr = 0x0058B0CC; Value = 0x0000000C
    Set Deaths("Player 12", "Int:18768", Add, 20905984);## Addr = 0x00666290; Value = 0x013F0000
    Set Deaths("Player 4", "Int:27270", Add, 1473);## Addr = 0x006C9C90; Value = 0x000005C1
    Set Deaths("Player 2", "Int:27284", Add, 1287);## Addr = 0x006C9F28; Value = 0x00000507
    Set Deaths("Player 11", "Int:27278", Add, 40);## Addr = 0x006C9E2C; Value = 0x00000028
    Create Unit with Properties("Current Player", "Yggdrasill (Overlord)", 1, "Invalid String", 3);
    Set Deaths("Player 12", "Int:18768", Subtract, 20905984);## Addr = 0x00666290; Value = 0x013F0000
    Set Deaths("Player 4", "Int:27270", Subtract, 1473);## Addr = 0x006C9C90; Value = 0x000005C1
    Set Deaths("Player 2", "Int:27284", Subtract, 1287);## Addr = 0x006C9F28; Value = 0x00000507
    Set Deaths("Player 11", "Int:27278", Subtract, 40);## Addr = 0x006C9E2C; Value = 0x00000028
    Set Deaths("Current Player", "Right Wall Flame Trap", Set To, 0);## Addr = 0x0058CB88; Value = 0x00000000
    Play WAV("sound\\Bullet\\zdeAtt00.wav", 0);
    Comment("±â%ú%ÄÀü");
    Preserve Trigger();
} « end Trigger »
```

Identify – Static pseudocode generator /2

- I wrote a converter from the triggers text to C pseudo-code
  (convert triggers to an AST and then emit as C pseudo-code)

```python
def trig2cpp(fn, dbg_output_file=False):
    try:
        # Check if IDA is running
        R

        # Convert all addresses to names
        out_fn = fn + '.tmp'
        trig_to_trigaddr(fn, out_fn)

        # Switch input file
        t = fn + '.cpp'
        fn = out_fn
        out_fn = t
    except:
        # No address conversion
        out_fn = fn + '.cpp'

    f = open(fn, 'r')
    lines = f.readlines()
    f.close()

    # Create a parser
    p = triglang.parser_t(lines, False)

    # Parse input file
    p.parse(dbg_output_file)

    f = open(out_fn, 'w')
    for trigger in p.triggers:
        f.write(str(trigger) + "\n")

    f.close()

    # Expose the parsed triggers
    global ptriggers
    ptriggers = p.triggers
```

```python
class trigger_t(object):
    def __init__(self, expr = '', id = 0, addr=0):
        self.expr = expr

        self.addr = addr
        """Trigger node address value"""

        self.id = id
        """Trigger serial number"""

        self.conditions = []
        self.actions = []
        self.in_conditions = False
        self.in_actions = False

        self.raw_body = None
        """Raw trigger body"""

        self.obj_body = None
        """Trigger body as a Python object"""
```

```python
class stmt_t(object):
    def __init__(self, func,
                 stmt='',
                 generic=True, addr=0,
                 sym=None, compare=None,
                 value= 0, var=None,
                 is_cond=False,
                 eud_idx=None,
                 parent=None):

        self.func = func
        """Statement function name"""
        self.stmt = stmt
        """Raw statement"""
        self.addr = addr
        """Target or source address"""
        self.eud_idx = eud_idx
        """EUD index as captured by tracer"""
        self.sym = sym
        """Symbol at address"""
        self.compare = compare
        """Comparator or operator"""
        self.value = value
        """Target or source value"""
        self.generic = generic
        """Generic statement"""
        self.var = var
        """Variable name from statement"""
        self.is_cond = is_cond
        """Is this an action or condition"""

        self.parent = parent
        """Parent. Usually the trigger"""


    def __str__(self):
        """Render statement to string"""

        sym = self.sym if self.sym is not None else self.var

        stmt = self.stmt
        if (sym is not None) and (self.value is not None) and (self.compare is not None):
            if sym.startswith('&'):
                sym = '_' + sym[1:]

            # Treat value as number or string
            try:
                stmt = "%s %s 0x%08X" % (sym, self.compare, self.value)
            except:
                stmt = "%s %s %s" % (sym, self.compare, self.value)

            if self.is_cond:
                stmt = "(%s)" % (stmt.replace(';', ''))
            else:
                stmt = "%s; // %08x " % (stmt, self.addr)


        return stmt
```

Identify – Static pseudocode generator /3

- Trigger text converted to C pseudo-code (trig2cpp()):

```
// " ¦¡ Team 1 ¦¡"," ¦¡ Team 2 ¦¡"
void trigger_1022()
{
  if (((_ g_s.glUnitCounts[COU_UNI_DEATH][UNI_STARTLOC][PLYR_TWO] == 0x00000001)) &&
       ((_ g_s.glUnitCounts[COU_UNI_DEATH][UNI_INSTALL_SPIKED_DOOR1][PLYR_TWO] == 0x00000002)) &&
       ((_ g_s.glUnitCounts[COU_UNI_DEATH][UNI_MINE_V2][PLYR_TWO] == 0x0000000B)) &&
       (Bring("Current Player", "Terran Science Vessel", "Invalid Location", At least, 0x00000001)) &&
       (Bring("Current Player", "Terran Science Vessel", "Invalid String", At least, 0x00000001)))
  {
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_INSTALL_HATCH][PLYR_TWO] = 0x00000000; // 0058c9a8
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_STARTLOC][PLYR_TWO] = 0x00000064; // 0058cb88
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_INSTALL_WALL_FLAMERF][PLYR_TWO] -= 0x00000019; // 0058cb58
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_Z_COCOON][PLYR_TWO] += 0x00000032; // 0058ae78
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_INSTALL_SPIKED_DOOR1][PLYR_TWO] = 0x00000000; // 0058ca38
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_KHAY_FORMATION][PLYR_TWO] = 0x00000090; // 0058c3d8
    Move Location("Current Player", "Terran Science Vessel", "Invalid Location", "Invalid Location");
    Remove Unit("Current Player", "Terran Science Vessel");
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_P_ARBITER][PLYR_SEVEN] = 0x0000000C; // 0058b0cc
    _ guwSpriteImage[SPR_Z_MUTALID_DEATH] += 0x013F0000; // 00666290
    _ gxFlingyAccel[FLI_Z_OVERLORD] += 0x000005C1; // 006c9c90
    _ gxFlingyMaxVel[FLI_Z_OVERLORD] += 0x00000507; // 006c9f28
    _ gubFlingyMaxTurn[FLI_Z_OVERLORD] += 0x00000028; // 006c9e2c
    Create Unit with Properties("Current Player", "Yggdrasill (Overlord)", 1, "Invalid String", 0x00000003);
    _ guwSpriteImage[SPR_Z_MUTALID_DEATH] -= 0x013F0000; // 00666290
    _ gxFlingyAccel[FLI_Z_OVERLORD] -= 0x000005C1; // 006c9c90
    _ gxFlingyMaxVel[FLI_Z_OVERLORD] -= 0x00000507; // 006c9f28
    _ gubFlingyMaxTurn[FLI_Z_OVERLORD] -= 0x00000028; // 006c9e2c
    _ g_s.glUnitCounts[COU_UNI_DEATH][UNI_STARTLOC][PLYR_TWO] = 0x00000000; // 0058cb88
    Play WAV("sound\\Bullet\\zdeAtt00.wav", 0x00000000);
    Comment("±â%ú%ÄÀü");
    Preserve Trigger();
  } « end if ((_g_s.glUnitCounts[C... »
} « end trigger_1022 »
```

Identify – Dynamic pseudocode generator /1

- With IDA's conditional breakpoints and the Appcall feature, I wrote a dynamic pseudocode generator:
  - It helps debug the map trigger logic during runtime
  - It helps in the discovery and understanding of dynamic triggers (generated by the EUD compiler from *trgk*)

- Conditional breakpoints are set at strategic entrypoints (pre, in and post trigger execution)

Abs	0x42CB6C	bpt_cond_deaths(stage=0)	Break	cond_deaths: start	EUD trace
Abs	0x42CB71	bpt_cond_deaths(stage=2)	Break	bpt_cond_deaths:flush	EUD trace
Abs	0x46044C	bpt_cond_deaths(stage=1)	Break	cond_deaths: symbols	EUD trace
Abs	0x4891E8	bpt_trigger_parse(stage=2)	Break	flush trig	EUD trace
Abs	0x4891F2	bpt_trigger_parse(stage=2)	Break	flush trig	EUD trace
Abs	0x489478	bpt_trigger_parse(stage=0)	Break	trigger: start one	EUD trace
Abs	0x4C5BEC	bpt_act_set_deaths(stage=1, reg= 'ecx')	Break		EUD trace
Abs	0x4C5BFA	bpt_act_set_deaths(stage=1, reg= 'ecx')	Break	loc_4C5BFA	EUD trace
Abs	0x4C5D75	bpt_act_set_deaths(stage=1, reg='eax')	Break		EUD trace
Abs	0x4C5EC6	bpt_act_set_deaths(stage=1, reg='edx')	Break		EUD trace
Abs	0x4C6CC0	bpt_act_set_deaths(stage=0)	Break	act_set_deaths: capture	EUD trace
Abs	0x4C6CDF	bpt_act_set_deaths(stage=2)	Break	act_set_deaths: flush (sub)	EUD trace
Abs	0x4C6CF5	bpt_act_set_deaths(stage=2)	Break	act_set_deaths: flush (add)	EUD trace
Abs	0x4C6D0B	bpt_act_set_deaths(stage=2)	Break	bpt_act_set_deaths:flush	EUD trace

Identify – Dynamic pseudocode generator /2

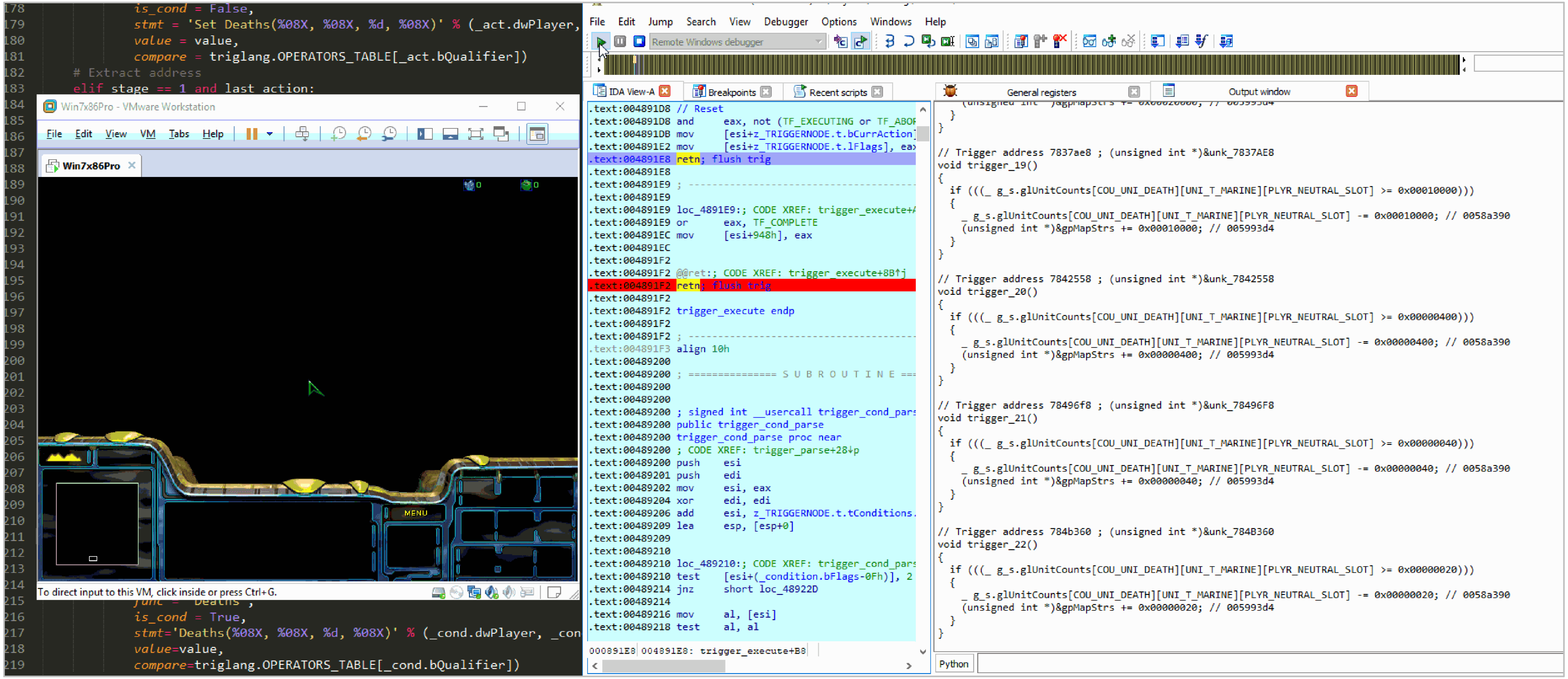
- Conditional breakpoints dynamically build the AST on access

```python
def bpt_trigger_parse(stage=0, reg=None):
    """Called to handle a trigger lifetime"""
    global last_trig
    bpt_ret = val_resume_bpt

    #
    # Capture
    if stage == 0:
        if not reg:
            reg = 'esi'

        trig_addr = getattr(cpu, reg)
        last_trig = triglang.trigger_t(
            id=len(triggers),
            addr=trig_addr)

        last_trig.expr = "Trigger address %x ; %s" % (trig_addr, get_symbol(trig_addr))

    #
    # Flush
    elif stage == 2 and last_trig:
        # Only remember non empty triggers
        if not last_trig.empty():
            triggers.append(last_trig)

            if single_step_triggers > 0:
                print triggers[-1]
                if single_step_triggers == 1:
                    bpt_ret = 1


        last_trig = None

    return bpt_ret
```

```python
def bpt_act_set_deaths(stage=0, reg=None):
    """Called to handle the set death action lifetime"""
    global last_action

    # Capture
    if stage == 0 and last_trig:
        if not reg:
            reg = 'Ecx'

        ok, _act = tp_action.retrieve(getattr(cpu, reg))
        if not ok:
            print("Failed to deserialize condition!")
            # Suspend
            return 1

        value = _act.lParm & 0xffffffff

        last_action = triglang.stmt_t(
            func = 'Set Deaths',
            is_cond = False,
            stmt = 'Set Deaths(%08X, %08X, %d, %08X)' % (_act.dwPlayer, _act.wType, _act.bQualifier, value),
            value = value,
            compare = triglang.OPERATORS_TABLE[_act.bQualifier])
    # Extract address
    elif stage == 1 and last_action:
        last_action.eud_idx = getattr(cpu, reg)
        last_action.addr    = E(last_action.eud_idx)
        last_action.sym     = get_symbol(last_action.addr)
    # Flush
    elif stage == 2 and last_action:
        last_trig.add_stmt(last_action)
        last_action = None

    # Always resume
    return val_resume_bpt
```

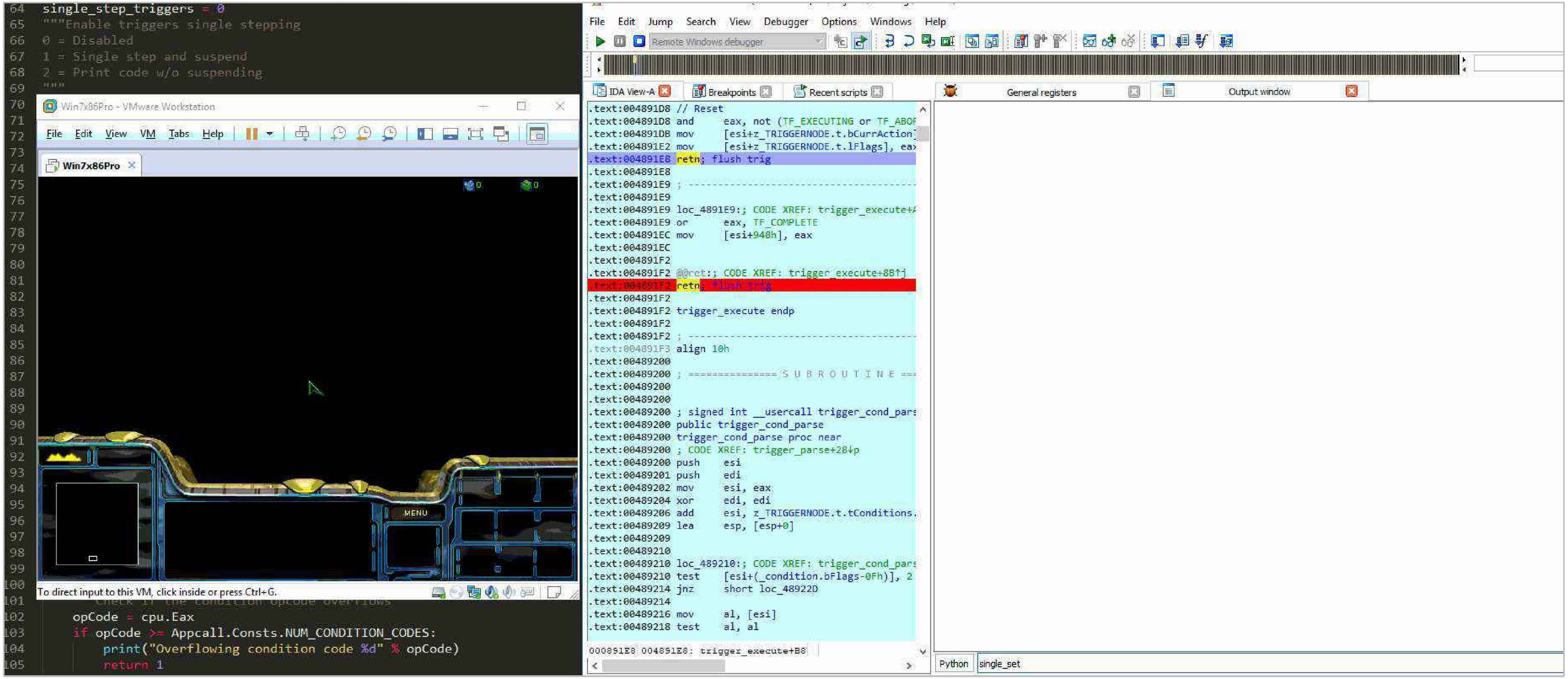
Demo – Dynamic pseudocode generator /1

- The debug script has a ‘Single step’ switch to break after each trigger
- Pseudocode is emitted on the fly



Demo – Dynamic pseudocode generator /2

- The “Single step” switch can be configured to print the pseudocode on the fly as the map triggers executes without suspending the game



Intercept /1

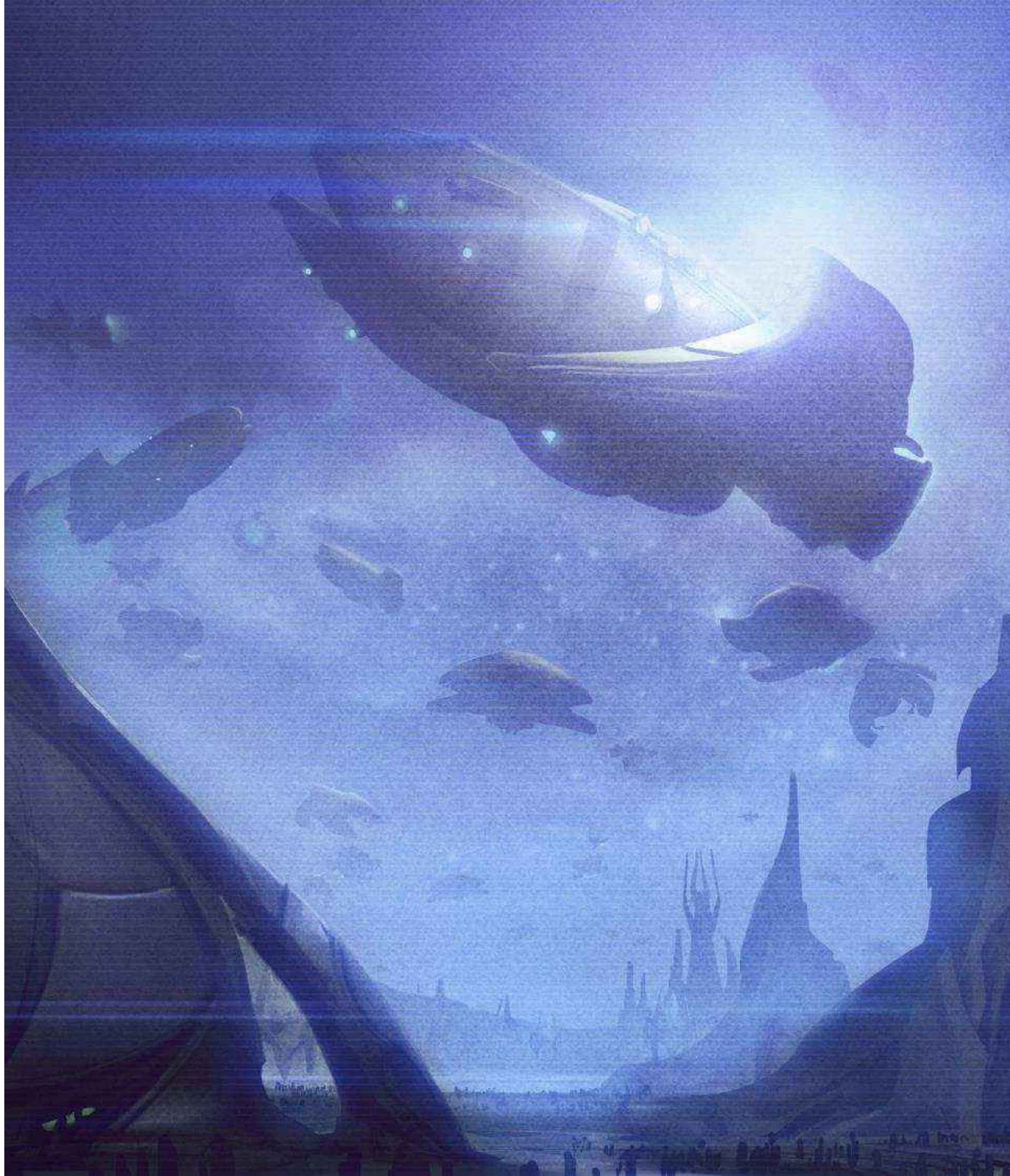
In the first step (identify):

1. We built all the required static and dynamic tracers

2. We created the EUD table with all known addresses and their symbolic names

3. We have enough tools to identify any address and trace where it came from

Now we need to intercept the out-of-bounds access in the new code base



Intercept /2

Read primitives interception

```
switch (wType)
{
    case UNITS_ALL:
        return bOverflow ? 0 : s->glGameCounts[COU_LOST_MEN][dwPlayer] + s->glGameCounts[COU

    case UNITS_MEN:
        return bOverflow ? 0 : s->glGameCounts[COU_LOST_MEN][dwPlayer];

    case UNITS_BLDGS:
        return bOverflow ? 0 : s->glGameCounts[COU_LOST_BLDGS][dwPlayer];

    case UNITS_FACTORIES:
        return bOverflow ? 0 : s->glGameCounts[COU_LOST_FACTORIES][dwPlayer];

    default:
        if (b_eud_is_eud_map)
        {
            auto pCond = (PCONDITION)lParam;
            return eud_cond_deaths(
                dwPlayer,
                wType,
                pCond);
        }
        app_assert(wType < NUM_UNITS);
        return bOverflow ? 0 : s->glUnitCounts[COU_UNI_DEATH][wType][dwPlayer];
} « end switch wType »
} « end death_count »
```

Write primitives interception

```
        default:
            if (b_eud_is_eud_map)
            {
                eud_set_deaths(dwPlayer, wType, lQuantity, 0);
                return 0;
            }
            if (bOverflow || wType >= NUM_UNITS)
                return 0;

            s->glUnitCounts[COU_UNI_DEATH][wType][dwPlayer] = lQuantity;
        break;
    } « end switch wType »
    return 0;
} « end set_deaths »
```

Intercept /3

- From the emulator's perspective, all EUD map logic boils down to two actions:

1. Read anywhere → value = read_vmem(eud_addr)
2. Write anywhere → write_vmem(eud_addr, value)

```
//--------------------------------------------------------------------------
// Base of the EUD overflow in SC 1.6.1
#define EUD_OVERFLOW_BASE \
    (0x582324 + (3 /*COU_UNI_DEATH*/ * 228 /*NUM_UNITS*/ * 12 /*MAX_PLAYER_SLOTS*/ * 4 /*sizeof(DWORD)*/))

// Helper macro to return a SC 1.6.1 address from dwPlayer and wType
#define EUD_MAKE_ADDR(dwPlayer, wType) \
    (EUD_OVERFLOW_BASE + ((dwPlayer) + 12 * (uint16_t)(wType)) * 4)
```

```
uint32_t eud_cond_deaths(
    uint32_t dwPlayer,
    unsigned short wType,
    void *pcond)
{
    GET_EUD_ADDR;

    /*
    return s.glUnitCounts[COU_UNI_DEATH][wType][dwPlayer];
    */
    eud_value_type value;
    bool ok = eud_emu->read_vmem(addr, value);

    if (!ok)
    {
        eud_fail(addr);
        return 0;
    }
    return value;
} « end eud_cond_deaths »
```

```
bool eud_set_deaths(
    uint32_t dwPlayer,
    unsigned short wType,
    uint32_t lQuantity,
    int q)
{
    /*
    s.glUnitCounts[COU_UNI_DEATH][wType][dwPlayer] = lQuantity;
    */
    if (!eud_emu->write_vmem(addr, lQuantity, q))
    {
        eud_fail(addr);
        return false;
    }
    return true;
}
```

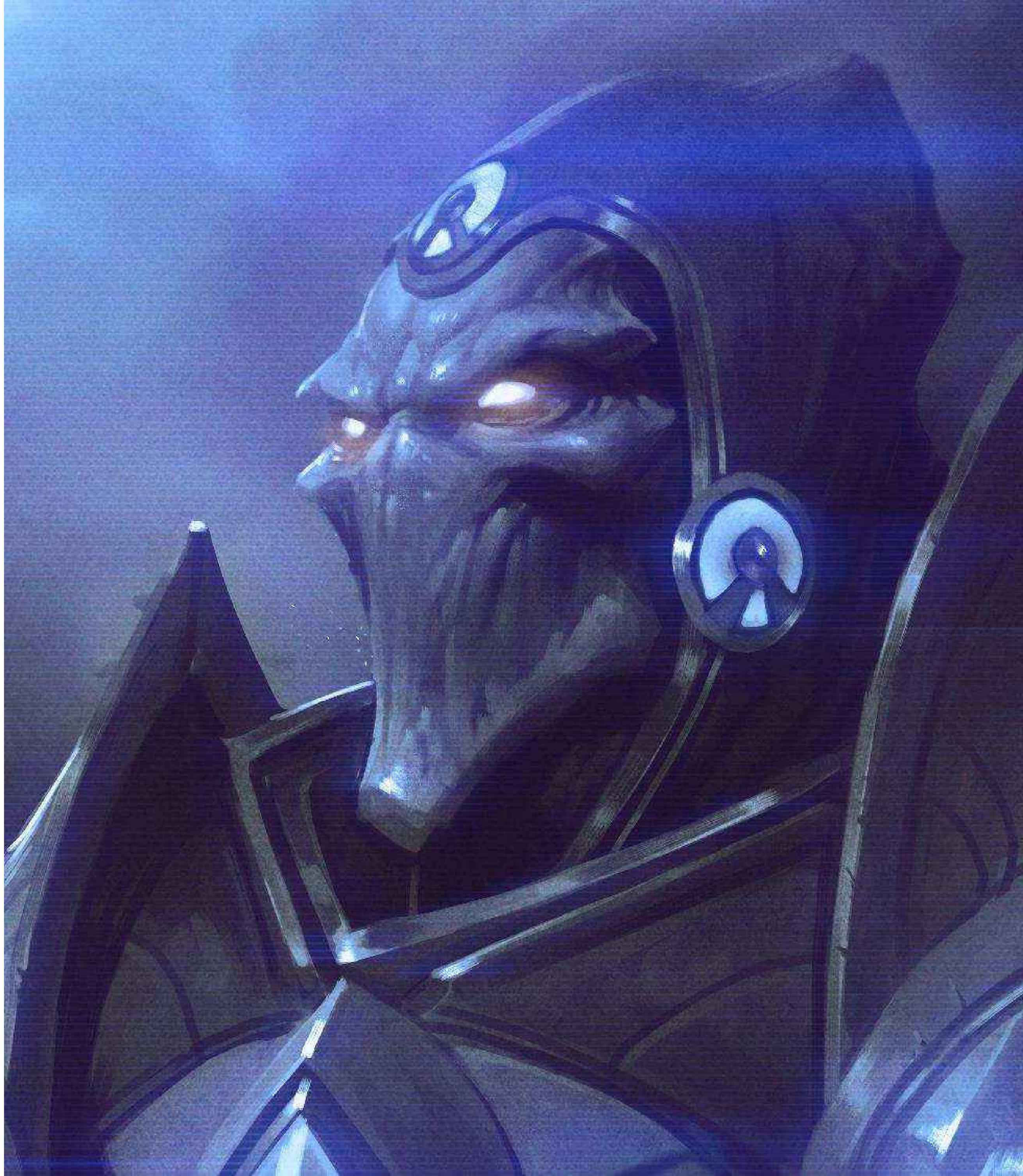
Emulate

<u>In basic scenarios, the emulation is very simple:</u>

1. Compute the full virtual address (EUD address) from the *dwPlayer* and *wType* indices

2. From the EUD address, find the equivalent new address (backing data) in the current game version

3. Compute the offset and read or write from/to the new address



Emulate – Variables mapping /1

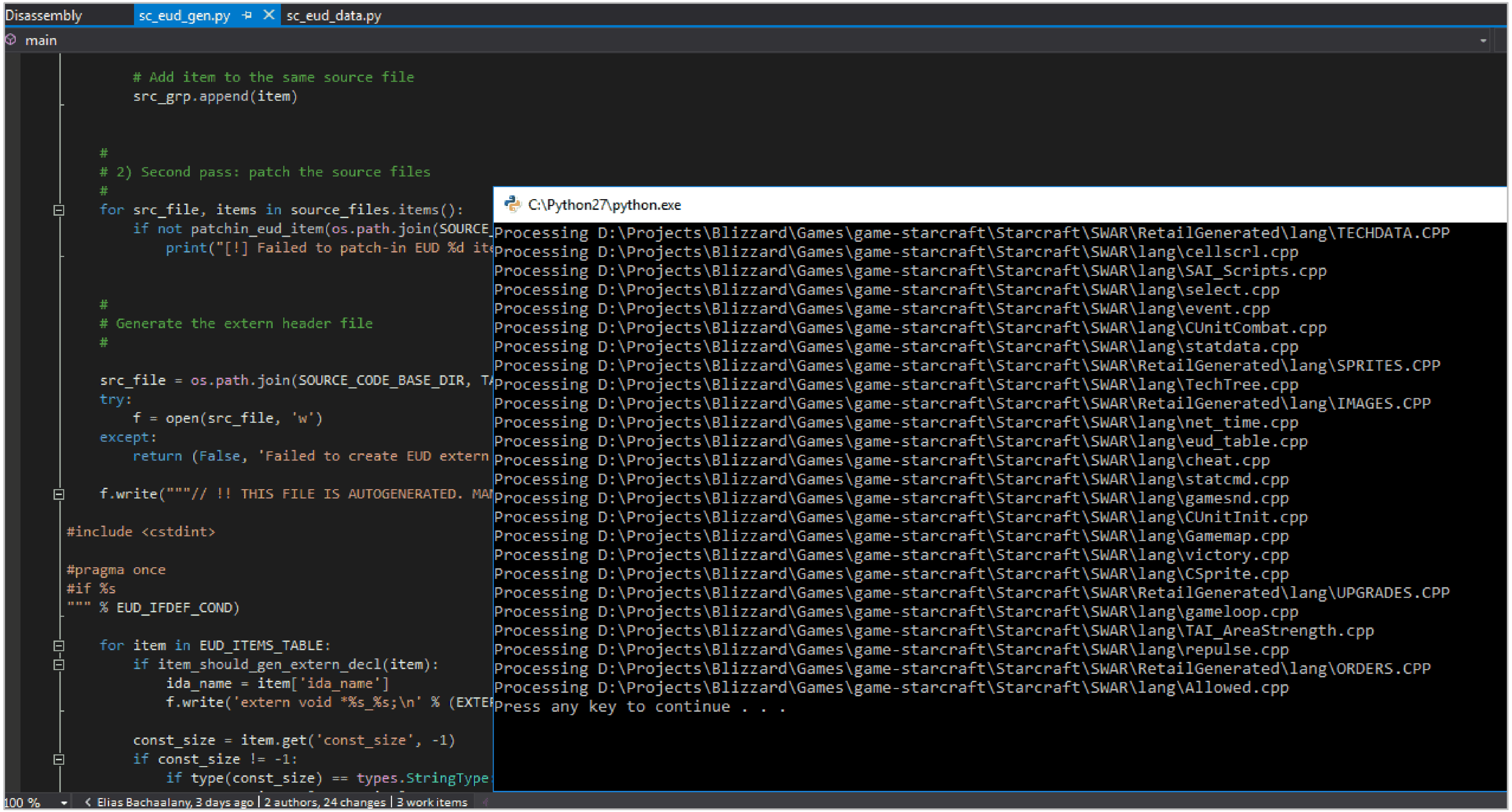
- Let's extend the previous Python table and attach the source file name were each variable is located

- The table defines: virtual address, item size, source file name, emulation flags, and backing variable name

```
# statport.cpp
{'src_file': r'SWAR\lang\statport.cpp',
    'addr': 0x0068AC74, 'size': 0x00000001, 'ida_name': 'sgbStatPortUpdate', 'flags': 0},


# Flingy
{'src_file': r'SWAR\RetailGenerated\lang\FLINGY.CPP', 'group': 'Flingy',
    'addr': 0x006C9858, 'size': 0x000000D1, 'ida_name': "gubFlingyMoveType", 'flags': 0},
   {'addr': 0x006C9930, 'size': 0x00000344, 'ida_name': "gxFlingySlow", 'flags': 0},
   {'addr': 0x006C9C78, 'size': 0x000001A2, 'ida_name': "gxFlingyAccel", 'flags': 0},
   {'addr': 0x006C9E20, 'size': 0x000000D1, 'ida_name': "gubFlingyMaxTurn", 'flags': 0},
   {'addr': 0x006C9EF8, 'size': 0x00000344, 'ida_name': "gxFlingyMaxVel", 'flags': 0},
   {'addr': 0x006CA318, 'size': 0x000001A2, 'ida_name': "guwFlingySprite", 'flags': 0},
   {'addr': 0x006CA240, 'size': 0x000000D1, 'ida_name': 'gubFlingyMinBank', 'flags': 0},

# Glues
{'src_file': r'SWAR\lang\glues.cpp',
    'addr': 0x0050E064, 'size': 0x00000004, 'ida_name': 'sgnPrevPalId', 'flags': 0},


# Repulse
{'src_file': r'SWAR\lang\repulse.cpp', 'group': 'Repulse',
    'addr': 0x006D5CD8, 'size': 0x00000004, 'ida_name': "sgpRpMap", 'flags': 'EIF_SRC_REPULSE_PTR | EIF_IS_PVOID',
    'const_size': 'REPULSE_MAP_SIZE', 'gen_opt': GEN_NO_SASSERT | GEN_FORCE_EXTERN},

# Net_data
{'src_file': r'SWAR\lang\net_data.cpp',
    'addr': 0x0057EEE0, 'size': 0x000001B0, 'ida_name': "gPlayerData",        'flags': 'EIF_SRC_PLAYER_DATA'},
   {'addr': 0x00512678, 'size': 0x00000004, 'ida_name': 'g_ActiveNationID',    'flags': 'EIF_SRC_NATION_ID'},
   {'addr': 0x00512684, 'size': 0x00000004, 'ida_name': 'g_LocalNationID',     'flags': 'EIF_SRC_NATION_ID'},
   {'addr': 0x0051267C, 'size': 0x00000004, 'ida_name': 'g_ActiveHumanID',     'flags': 'EIF_SRC_NATION_ID'},
   {'addr': 0x0057F0B4, 'size': 0x00000001, 'ida_name': 'gbMultiPlayerMode',   'flags': 'EIF_READ_ONLY'},
   {'addr': 0x0057F090, 'size': 0x00000004, 'ida_name': 'gdwDefTurnsInTransit','flags': 'EIF_READ_ONLY'},
```

Emulate – Variables mapping /2

Running the EUD table generation script patches the source code and exports all referenced variables:



Emulate – Variables mapping /3

Exported variables example:

```
//**************************************
// data tables
//**************************************
    UWORD guwFlingySprite[NUM_FLINGIES];
    ULONG gxFlingyMaxVel[NUM_FLINGIES];
    UWORD gxFlingyAccel[NUM_FLINGIES];
    ULONG gxFlingySlow[NUM_FLINGIES];
    UBYTE gubFlingyMaxTurn[NUM_FLINGIES];
    UBYTE gubFlingyMinBank[NUM_FLINGIES];
    UBYTE gubFlingyMoveType[NUM_FLINGIES];

/// EUD EXTERNS - AUTOGENERATE BEGIN ///
static_assert(sizeof(gubFlingyMoveType) == 0xd1, "EUD size mismatch for gubFlingyMoveType");
void *eud_ptr_gubFlingyMoveType = reinterpret_cast<void*>(&gubFlingyMoveType);
static_assert(sizeof(gxFlingySlow) == 0x344, "EUD size mismatch for gxFlingySlow");
void *eud_ptr_gxFlingySlow = reinterpret_cast<void*>(&gxFlingySlow);
static_assert(sizeof(gxFlingyAccel) == 0x1a2, "EUD size mismatch for gxFlingyAccel");
void *eud_ptr_gxFlingyAccel = reinterpret_cast<void*>(&gxFlingyAccel);
static_assert(sizeof(gubFlingyMaxTurn) == 0xd1, "EUD size mismatch for gubFlingyMaxTurn");
void *eud_ptr_gubFlingyMaxTurn = reinterpret_cast<void*>(&gubFlingyMaxTurn);
static_assert(sizeof(gxFlingyMaxVel) == 0x344, "EUD size mismatch for gxFlingyMaxVel");
void *eud_ptr_gxFlingyMaxVel = reinterpret_cast<void*>(&gxFlingyMaxVel);
static_assert(sizeof(guwFlingySprite) == 0x1a2, "EUD size mismatch for guwFlingySprite");
void *eud_ptr_guwFlingySprite = reinterpret_cast<void*>(&guwFlingySprite);
static_assert(sizeof(gubFlingyMinBank) == 0xd1, "EUD size mismatch for gubFlingyMinBank");
void *eud_ptr_gubFlingyMinBank = reinterpret_cast<void*>(&gubFlingyMinBank);
```

Emulate – Variables mapping /4

No need to make static variables global:

- The generator has an option that lets you pick a name for the exported variable

```
// EUD table.py
# CImage
{'src_file': r'SWAR\lang\CImage.cpp', 'group': 'CImage',
    {'addr': 0x0057EB68, 'size': 0x00000004, 'ida_name': 'images_sgpFreeHead', 'name': "sgpFreeHead", 'flags': 'EIF_READ_ONLY'},
    {'addr': 0x0057EB70, 'size': 0x00000004, 'ida_name': 'images_sgpFreeTail', 'name': "sgpFreeTail", 'flags': 'EIF_READ_ONLY'},


// CImage.cpp

static CLists *sgpFreeHead;
static CLists *sgpFreeTail;

/// EUD EXTERNS - AUTOGENERATE BEGIN ///
#if EUD_ENABLED
static_assert(sizeof(sgpFreeHead) == 0x4, "EUD size mismatch for sgpFreeHead");
void *eud_ptr_images_sgpFreeHead = reinterpret_cast<void*>(&sgpFreeHead);
static_assert(sizeof(sgpFreeTail) == 0x4, "EUD size mismatch for sgpFreeTail");
void *eud_ptr_images_sgpFreeTail = reinterpret_cast<void*>(&sgpFreeTail);
#endif // (EUD_ENABLED)
/// EUD EXTERNS - AUTOGENERATE END ///
```

Emulate – The EUD table /1

- The “eud_table.cpp” is autogenerated from the Python table. It refers to all the exported variables from various source code files

- It is used to populate the emulator’s virtual memory layout

- Items also have associated flags that instruct the emulator which EUD adapter handles which address

- Note: the “g_nothing” variables are alignment bytes in SC 1.16.1. The map makers use that space for storing variables

- A “nullptr” backing data almost always indicates that the variable is to be handled purely by an adapter code

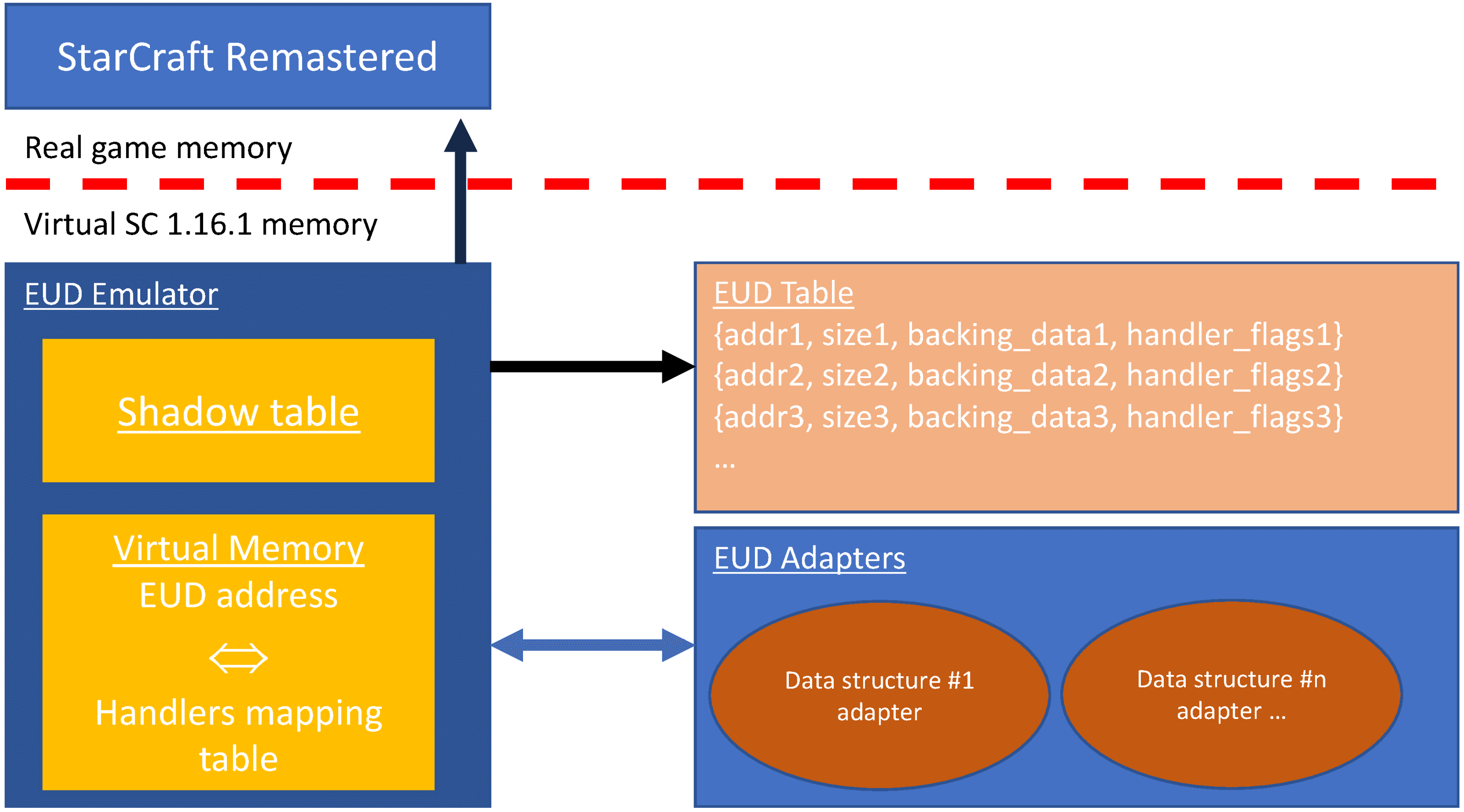
```
eud_itemdef_t __static_eud_items[__STATIC_EUD_ITEMS_COUNT] =
{
    DEF_EUD_ITEM(0x0068C14C, 0x00000002, eud_ptr_sgCard, 0x00000000),
    DEF_EUD_ITEM(0x00513B68, 0x00000031, eud_ptr_sgnScrollRates, 0x00000000),
    DEF_EUD_ITEM(0x0068C144, 0x00000001, eud_ptr_gbInMsgMode, 0x00000000),
    DEF_EUD_ITEM(0x0068C10C, 0x00000032, eud_ptr_sgszToPlayerPrompt, 0x00000000),
    DEF_EUD_ITEM(0x005967F8, 0x0000008D, eud_ptr_gGameHeader, 0x00000000),
    DEF_EUD_ITEM(0x00597208, 0x00000034, nullptr, EIF_SRC_STAT_UNITS), // gpStatUnits
    DEF_EUD_ITEM(0x0068AC74, 0x00000001, eud_ptr_sgbStatPortUpdate, 0x00000000),
    DEF_EUD_ITEM(0x006C9858, 0x000000D1, eud_ptr_gubFlingyMoveType, 0x00000000),
    DEF_EUD_ITEM(0x006C9930, 0x00000344, eud_ptr_gxFlingySlow, 0x00000000),
    DEF_EUD_ITEM(0x006D5CD8, 0x00000004, eud_ptr_sgpRpMap, EIF_SRC_REPULSE_PTR | EIF_IS_PVO
    DEF_EUD_ITEM(0x006562A0, 0x00000058, eud_ptr_guwTechStr, 0x00000000),
    DEF_EUD_ITEM(0x00656350, 0x0000002C, eud_ptr_gubTechAlwaysAllowed, 0x00000000),
    DEF_EUD_ITEM(0x0062843C, 0x00000004, eud_ptr_sgpFreeTail_CUnit, EIF_SRC_CUNIT_PTR),
    DEF_EUD_ITEM(0x00628438, 0x00000004, eud_ptr_sgpFreeHead_CUnit, EIF_SRC_CUNIT_PTR),
    DEF_EUD_ITEM(0x00662BF0, 0x000001C8, eud_ptr_guwLastWhatSnd, 0x00000000),
    DEF_EUD_ITEM(0x00515B68, 0x00000008, eud_ptr_MapHops, 0x00000000),
    DEF_EUD_ITEM(0x00640B60, 0x00000B12, eud_ptr_sgszMsgTbl, EIF_SRC_MSG_TBL),
    DEF_EUD_ITEM(0x00640B58, 0x00000001, eud_ptr_sgbNextMsg, 0x00000000),
    DEF_EUD_ITEM(0x0051A280, 0x0000000C, nullptr, EIF_SRC_TRIGGERS_LIST), // sgTriggers0
    DEF_EUD_ITEM(0x0051A28C, 0x0000000C, nullptr, EIF_SRC_TRIGGERS_LIST), // sgTriggers1
    DEF_EUD_ITEM(0x00664894, 0x00000004, &g_nothing_19, 0x00000000),
    DEF_EUD_ITEM(0x0066497C, 0x00000004, &g_nothing_20, 0x00000000),
    DEF_EUD_ITEM(0x006646C4, 0x00000004, &g_nothing_21, 0x00000000),
```

Emulate – The EUD table /2

- The “eud_extern.h” is autogenerated from the Python table
- It exposes all the known EUD variables
  - Very handy for accessing static variables from anywhere in the code when needed

```
// !! THIS FILE IS AUTOGENERATED. MANUAL MODIFICATION WIL

#include <cstdint>

#pragma once
extern void *eud_ptr_sgCard;
extern void *eud_ptr_sgnScrollRates;
extern void *eud_ptr_gbInMsgMode;
extern void *eud_ptr_sgszToPlayerPrompt;
extern void *eud_ptr_gGameHeader;
extern void *eud_ptr_gpIconsGrp;
extern void *eud_ptr_sgbSelectionUpdate;
extern void *eud_ptr_sgbStatPortUpdate;
extern void *eud_ptr_gubFlingyMoveType;
extern void *eud_ptr_gxFlingySlow;
extern void *eud_ptr_gxFlingyAccel;
extern void *eud_ptr_gubFlingyMaxTurn;
extern void *eud_ptr_gxFlingyMaxVel;
extern void *eud_ptr_guwFlingySprite;
extern void *eud_ptr_gubFlingyMinBank;
extern void *eud_ptr_sgnPrevPalId;
extern void *eud_ptr_sgpRpMap;
extern uint32_t eud_export_REPULSE_MAP_SIZE;
extern void *eud_ptr_g_ActiveNationID;
extern void *eud_ptr_g_LocalNationID;
```

Emulator architecture /1



Due to the nature of the overflow, the following restrictions apply:

- An EUD address is always 4 bytes aligned
- An EUD value is a 32bits integer

Emulator architecture /2

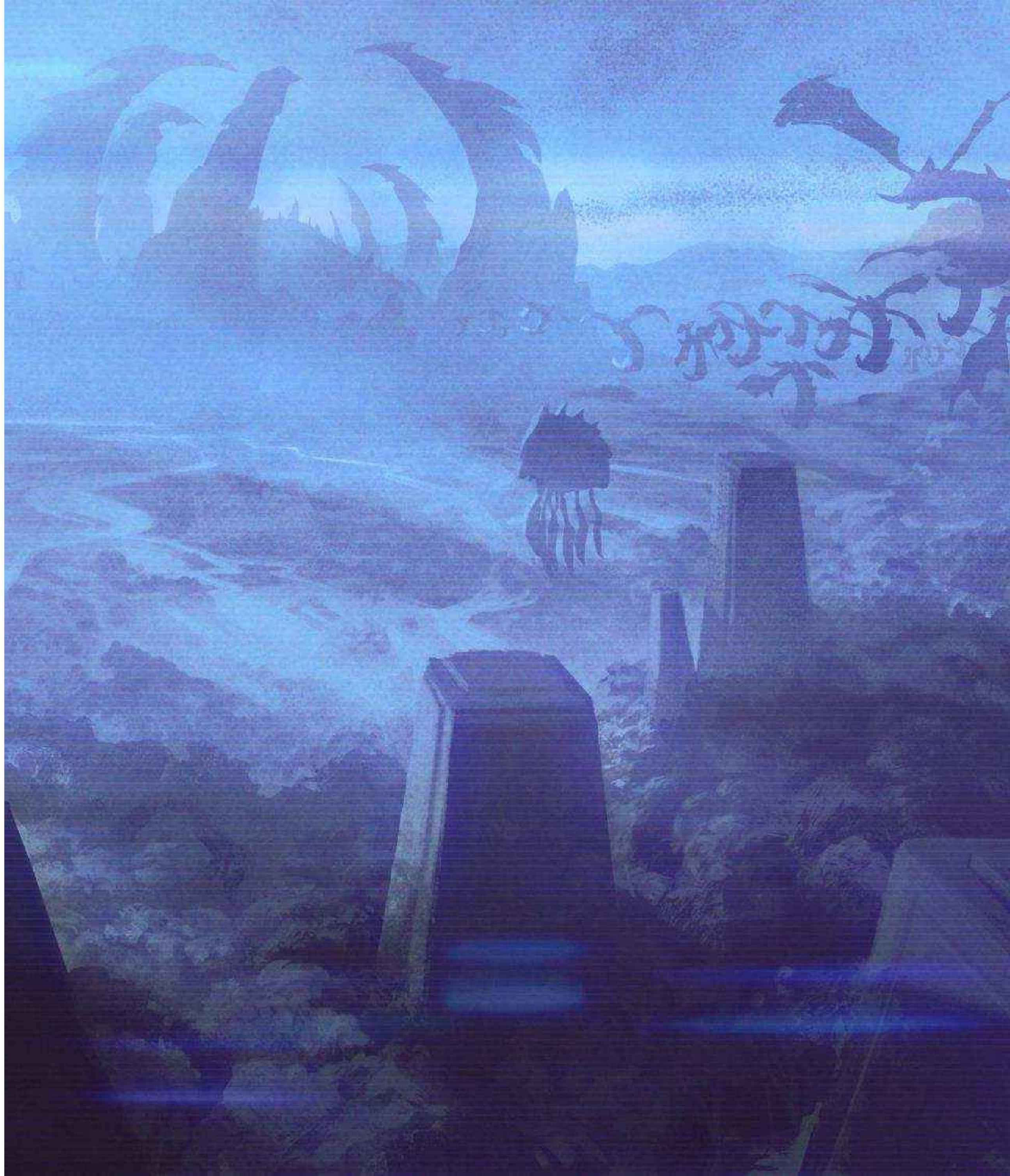
<u>Shadow table</u>

- It contains the needed memory contents from the SC 1.16.1 binary

<u>Virtual memory</u>

- It uses the address-to-handlers lookup table
- It maps an EUD address range to an EUD table entry → EUD handler/adapter

- The table entry for an EUD item describes:
  - The backing data (the new variable address, if present)
  - The flags which tell the emulator which EUD adapter (handler) to use for emulation



Emulator architecture /3

A specialized EUD adapter is needed when:

- Handling non-standard data types
- When dealing with EUD addresses that no longer map to anything in the new game client

The following 5 virtual methods are exposed

- read_vmem() → Return a 32bits value
- write_vmem() → Write a 32bits value
- backup() → Item specific backup code
- restore() → Item specific restore code
- deferred_write() → Invoked after all the triggers have executed. Gives a chance to batch process writes

EUD adapters – Basic /1

The basic EUD adapter (*eud_vmemitem_t* class) handles basic data types:

1. The emulator computes the full EUD address
2. Finds the new variable’s base address and converts the EUD address to an offset
3. The appropriate adapter is then called with the desired offset to read/write from/to

```
bool eud_emu_t::read_vmem(
    eud_addr_type addr,
    eud_value_type &value)
{
    eud_vmemitem_t *vitem = find_range(vmem, addr);
    if (vitem == nullptr)
    {
        set_not_supported(false);
        return false;
    }

    return vitem->read_vmem(
                this,
                addr - vitem->addr,
                value);
}
```

```
bool eud_vmemitem_t::read_vmem(
    eud_emu_t *emu,
    void *backing_data,
    eud_addr_type offs,
    eud_value_type &value)
{
    value = *(eud_value_type *)((char *)backing_data + offs);
    return true;
}
```

This simple translation approach works nicely for basic types

1.16.1

```
.data:006CA316 db    ? ;
.data:006CA317 db    ? ;
.data:006CA318 ; __int16 guwFlingySprite[209]
.data:006CA318 guwFlingySprite dw 0D1h dup(?)
.data:006CA318 ; DATA XREF: CUnit__DisplayLandi
.data:006CA318 ; sub_468280+14D↑r ...
```

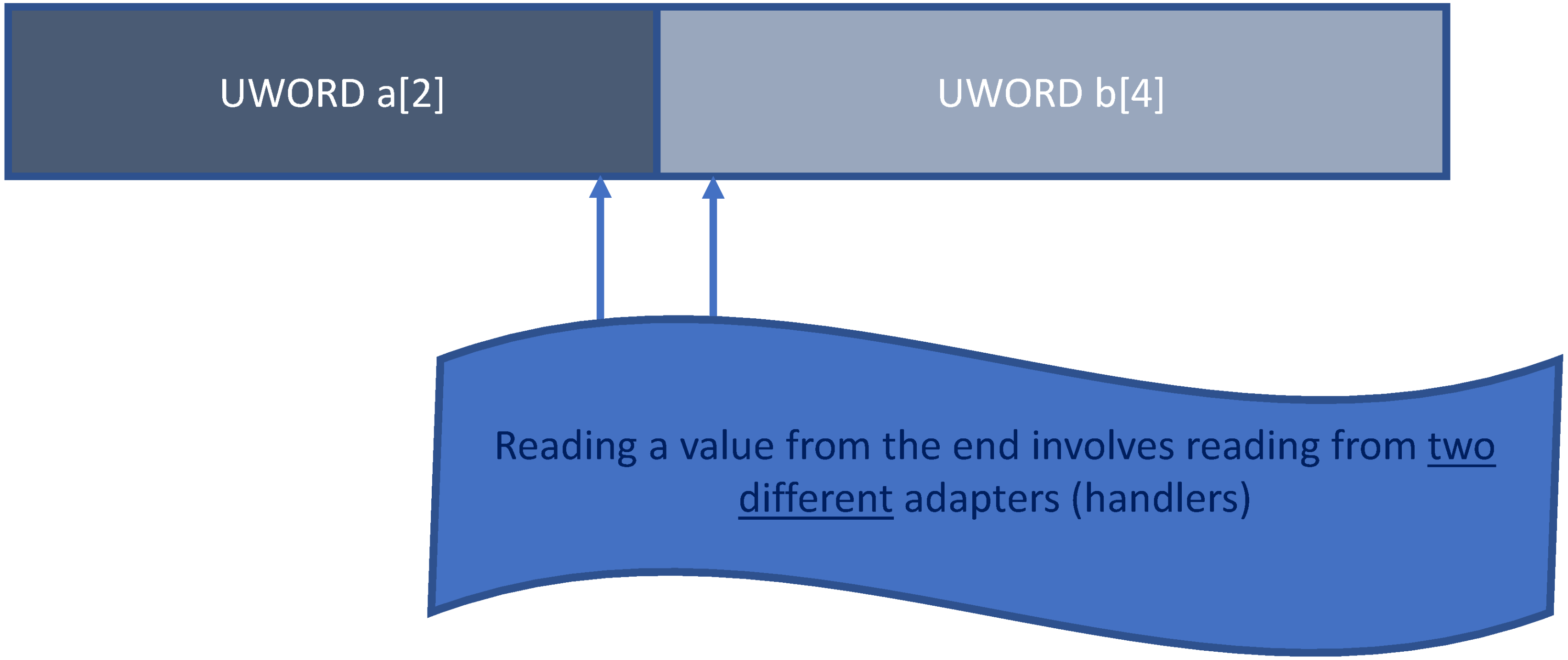
New builds

```
.data:00AC831B                 align 10h
.data:00AC8320 ; unsigned __int16 guwFlingySprite[209]
.data:00AC8320 unsigned short * guwFlingySprite db 0D1h dup(0)
.data:00AC8320                                  ; DATA XREF: AllocFlingy(
.data:00AC8320                                  ; CFlingy::Init(ushort,sh
.data:00AC83F1                 db    0
```

EUD adapters – Basic /2

The basic (pass-thru) adapter is good for most cases:

- Byte, Word, Dword
- The emulator can cross boundaries between two items
- Basic types arrays are also supported



Wait a minute, we need one more primitive!

- We covered two primitives:

   1. *mem asg_op = const
      - asg_op → += , = , -=

   2. if (*mem cmp_op const) { actions … }
      - cmp_op → ==, >=, <=

- How do we get the following primitive?
   - *mem1 asg_op *mem2

Using binary search!

The *a = *b primitive

- Trigger condition:
  1. Probes the value of *src_var*
- Trigger action:
  1. Increments the value of *dst_var*
  2. Decrement the value of *src_var*
  3. *src_var*'s value eventually reaches zero
  4. Backup changes into *var_copy*

The same primitive is repeated to copy *var_copy* back to *dst_var*

This primitive is expensive and generates lots of triggers

```
void trigger_0()
{
    var_copy = 0;
    dst_var = 0;
}
void trigger_1()
{
  if ((src_var >= 0x80000000))
  {
    src_var -= 0x80000000;
    dst_var += 0x80000000;
    var_copy += 0x80000000;
  }
}
void trigger_2()
{
  if ((src_var >= 0x40000000))
  {
    src_var -= 0x40000000;
    dst_var += 0x40000000;
    var_copy += 0x40000000;
  }
}
void trigger_3()
{
  if ((src_var >= 0x20000000))
  {
    src_var -= 0x20000000;
    dst_var += 0x20000000;
    var_copy += 0x20000000;
  }
}
void trigger_...()
{
  if ((src_var >= 0x00000100))
  {
    src_var -= 0x00000100;
    dst_var += 0x00000100;
    var_copy += 0x00000100;
  }
}
void trigger_...()
{
  if ((src_var >= 0x00000004))
  {
    src_var -= 0x00000004;
    dst_var += 0x00000004;
    var_copy += 0x00000004;
  }
}
void trigger_...()
{
  if ((src_var >= 0x00000002))
  {
    src_var -= 0x00000002;
    dst_var += 0x00000002;
    var_copy += 0x00000002;
  }
}
void trigger_...()
{
  if ((src_var >= 0x00000001))
  {
    src_var -= 0x00000001;
    dst_var += 0x00000001;
    var_copy += 0x00000001;
  }
} « end trigger_0 »
```

EUD adapters – Pointers /1

- Pointers are 32bits in SC 1.16.1
- Obviously, we cannot just use the pass-thru basic emulation
  - Pointers have to be translated from EUD virtual addresses to real addresses
- The primitive “*ptr1 = *ptr2” invoked from the EUD triggers will spoil the pointer value until the binary search is over
  - What to do with incomplete pointer values?

EUD adapters – Pointers /2

- Changes to a physical pointer value should not take effect unless the virtual pointer value passes a “pointer validity check function”
  → Does the virtual pointer have a proper real pointer equivalent?

- Rely on the shadow pointer value when working with incomplete virtual pointer values for future reads / writes:

Real memory	EUD virtual memory
void *game_ptr;	uint32_t game_ptr;
uint32_t game_ptr_shadow; bool game_ptr_dirty;	

EUD adapters – Pointers /3

- The eud_cobject_ptr_adapter_t is constructed with backing data pointing to a reference to a real pointer that we want to expose to the EUD emulator

```cpp
//---------------------------------------------------------------------------
template <class T>
class eud_cobject_ptr_adapter_t: public eud_vmemitem_t
{
protected:
    eud_addr_type shadow_ptr;

public:
    T *&ptr() const { return *(T **)backing_data; }

    // Convert an physical pointer to an EUD pointer
    eud_value_type physical_to_eudaddr(eud_emu_t *emu)
    {
        // Read the live value...
        auto obj = (T *)ptr();
        // ...and translate it to a virtual address
        return obj == nullptr ? 0 : emu->get_cobject_vptr(obj);
    }

    virtual bool read_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type &value) override
    {
        // When not dirty, read the live value
        if (!is_dirty())
            shadow_ptr = physical_to_eudaddr(emu);

        value = shadow_ptr;
        return true;
    }
```

```cpp
    virtual bool write_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type value,
        int q) override
    {
        if (!is_dirty())
        {
            shadow_ptr = get_vptr(emu);
            set_dirty();
        }

        // Update the shadow value
        set_pval(&shadow_ptr, value, q);

        // Allow nullptr assignment
        if (shadow_ptr == 0)
        {
            ptr() = nullptr;
        }
        else
        {
            // Update the real pointer only if it gets translated
            // from an EUD addr to a physical pointer
            T *unit;
            emu->get_cobject_ptr(shadow_ptr, true, &unit);
            if (unit != nullptr)
            {
                ptr() = unit;
                clear_dirty();
            }
        }
        return true;
    } « end write_vmem »
} « end eud_cobject_ptr_adapter_t » ;
```

EUD adapters – Function pointers /1

- What about EUD logic that does function pointer arithmetic?

```
// "Player 8"
void trigger_497()
{
  if ((Always();))
  {
    _ sgTEngineeringCard[5].pfBtnAction += 0x000000C0; // 005184ec
  }
}
```

```
00000000 TButton struc ; (sizeof=0x14, align=0x4, mappedto_224)
00000000 ; XREF: .data:sgReplayPlayCard/r .data:sgReplayPauseCard/r ...
00000000 wLocation dw ?
00000002 wPortrait dw ?
00000004 pfCanDisplay dd ?; offset
00000008 pfBtnAction dd ?; offset
0000000C bCanDisplayParm db ?
0000000D db ? ; undefined
0000000E BtnActionParm dw ?
00000010 wStrIndex dw ?; XREF: statcmd_set_ctrl+6D/r; base 10
00000012 wReqStr dw ?
00000014 TButton ends
```

```
.data:00518480 public sgTEngineeringCard
.data:00518480 ; TButton sgTEngineeringCard[7]
.data:00518480 sgTEngineeringCard dw 1; [0].wLocation
.data:00518480 ; DATA XREF: .data:gCards↓o
.data:00518480 dw 0E4h; [0].wPortrait
.data:00518480 dd offset bf_is_ship; [0].pfCanDisplay
.data:00518480 dd offset order_move; [0].pfBtnAction
.data:00518480 db 0; [0].bCanDisplayParm
.data:00518480 db 0; 0
.data:00518480 dw 0; [0].BtnActionParm
.data:00518480 dw 664; [0].wStrIndex
.data:00518480 dw 0; [0].wReqStr
.data:00518480 dw 1; [1].wLocation
...
.data:00518480 dw 0; [4].wReqStr
.data:00518480 dw 9; [5].wLocation
.data:00518480 dw 11Ah; [5].wPortrait
.data:00518480 dd offset bf_liftoff_ok; [5].pfCanDisplay
.data:00518480 dd offset order_bldg_liftoff; [5].pfBtnAction
.data:00518480 db 0; [5].bCanDisplayParm
.data:00518480 db 0; 5
.data:00518480 dw 0; [5].BtnActionParm
.data:00518480 dw 671; [5].wStrIndex
.data:00518480 dw 0; [5].wReqStr
```

```
.text:004232F0 ; =============== S U B R O U T I N E =======================================
.text:004232F0
.text:004232F0 ; Attributes: bp-based frame
.text:004232F0
.text:004232F0 public order_cancel_upgrade
.text:004232F0 order_cancel_upgrade proc near
.text:004232F0 ; CODE XREF: statdata_slot_cmd+
.text:004232F0 ; DATA XREF: .data:sgZLairCard↓
.text:004232F0
.text:004232F0 Cmd= byte ptr -1
.text:004232F0
.text:004232F0 push    ebp
.text:004232F1 mov     ebp, esp
.text:004232F3 push    ecx
.text:004232F4 mov     edx, 1; dwBytes
.text:004232F9 lea     ecx, [ebp+Cmd]; lpCmd
.text:004232FC mov     [ebp+Cmd], 33h
.text:00423300 call    netmgr_queue_cmd
.text:00423300
.text:00423305 mov     esp, ebp
.text:00423307 pop     ebp
.text:00423308 retn
```

Evaluate expression

Expression: order_cancel_upgrade-0xc0

Hex: 42 3230 (order_bldg_liftoff)

Decimal: 4 338 224

Octal: 20 431 060

Binary: 0000 0000 0100 0010 0011 0010 0011 0000

Character: '02B.'

OK | Cancel | Help

EUD adapters – Function pointers /2

- Pointer arithmetic make sense only in the EUD virtual memory addressing space

- For the real pointer addressing we have to translate to proper pointers and account for function prototype compatibility

- <u>Basic implementation idea:</u>
  1. vaddr += voffs
  2. paddr = find_real_fptr(vaddr, function_prototype_id)
  3. if (paddr != nullptr) → struct.pFn = paddr;

- In the emulator, such cases are handled with the *eud_struct_with_ptr_adapter_t*

<u>Virtual function pointers and their prototypes table</u>

```
# TButtons function pointers: pfCanDisplay and pfBtnAction
{'src_file': r'SWAR\lang\statbtn.cpp', 'group': 'Card/Buttons function pointers',
     'addr': 0x004282d0, 'size': 0x00000004, 'ida_name': 'bf_always', 'name': 'bf_always', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
    {'addr': 0x00424440, 'size': 0x00000004, 'ida_name': 'order_move', 'name': 'order_move', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
    {'addr': 0x004233f0, 'size': 0x00000004, 'ida_name': 'order_stop', 'name': 'order_stop', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
    {'addr': 0x00428f30, 'size': 0x00000004, 'ida_name': 'bf_can_attack', 'name': 'bf_can_attack', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
    {'addr': 0x00424380, 'size': 0x00000004, 'ida_name': 'order_attack', 'name': 'order_attack', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
    {'addr': 0x00424140, 'size': 0x00000004, 'ida_name': 'order_patrol', 'name': 'order_patrol', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
    {'addr': 0x00423370, 'size': 0x00000004, 'ida_name': 'order_hold_pos', 'name': 'order_hold_pos', 'flags': 'EIF_FUNC_PTR | EIF_SRC_TBUTTON'},
```

EUD adapters – Incompatible structures /1

- Various data structures have changed between SC 1.16.1 and SC:R
- Pass-thru adapters are not helpful in this case

```
struct eud_CUnit
{
  int unit_id;                  /* 0x00 */
  char unit_name[80];           /* 0x04 */
  eud_CUnit *linked_unit;       /* 0x54 */
  eud_CImage *linked_Sprite;    /* 0x58 */
}
```

```
struct CUnit
{
  char unit_name[80];           // 0x00
  int field1;                   // 0x50
  int field2;                   // 0x54
  int unit_id;                  // 0x58
  eud_CUnit *linked_unit;       // 0x5C
  eud_CImage *linked_Sprite;    // 0x60
}
```

- A specialized adapter is needed to convert between both structures:
  - <u>Read operation:</u> translates from physical structure to virtual structure
  - <u>Write operation:</u> translates from virtual structure to physical structure

EUD adapters – Incompatible structures /2

```
//---------------------------------------------------------------
bool eud_csprite_adapter_t::read_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type &value)
{
    switch (offs)
    {
        // 0x000
        case offsetof(eud_CSprite, ptr_CSprite_pPrevNode):
        {
            EUD_FIELD_READ_VPTR(
                CSprite,
                pPrevNode,
                csprite()->prop_CLists_PrevNode(),
                emu->sprites->get_addr);
            break;
        }
        // 0x004
        case offsetof(eud_CSprite, ptr_CSprite_pNextNode):
        {
            EUD_FIELD_READ_VPTR(
                CSprite,
                pNextNode,
                csprite()->prop_CLists_NextNode(),
                emu->sprites->get_addr);
            break;
        }
        // 0x008
        case offsetof(eud_CSprite, uwType):
        {
            uwType_ubCreator_union_t u;
            u.ubSelectedNdx = csprite()->prop_ubSelectedNdx();

            EUD_FIELD_READ_PARTIAL_VAL(
                uwType,
                csprite()->prop_uwType(),
                u.uwType);

            EUD_FIELD_READ_PARTIAL_VAL(
                ubCreator,
                csprite()->prop_ubCreator(),
                u.ubCreator);

            value = u.val;
            break;
        }
        default:
            return false;
    } « end switch offs »
    return true;
} « end read_vmem »
```

```
bool eud_csprite_adapter_t::write_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type value,
        int g)
{
    switch (offs)
    {
        // 0x000
        case offsetof(eud_CSprite, ptr_CSprite_pPrevNode):
        {
            EUD_FIELD_UPDATE_VPTR(
                CSprite,
                pPrevNode,
                csprite()->prop_CLists_PrevNode(),
                emu->sprites->get_addr,
                emu->sprites->get_ptr);
            break;
        }
        // 0x004
        case offsetof(eud_CSprite, ptr_CSprite_pNextNode):
        {
            EUD_FIELD_UPDATE_VPTR(
                CSprite,
                pNextNode,
                csprite()->prop_CLists_NextNode(),
                emu->sprites->get_addr,
                emu->sprites->get_ptr);
            break;
        }
        // 0x008
        case offsetof(eud_CSprite, uwType):
        {
            EUD_FIELD_UPDATE_BATCH(
                uwType_ubCreator_union_t,
                csprite()->prop_ubSelectedNdx() = u.ubSelectedNdx);

            EUD_FIELD_UPDATE_PARTIAL_VAL(
                ubCreator,
                u.ubCreator,
                csprite()->prop_ubCreator(),
                check_owner_bounds);

            EUD_FIELD_UPDATE_PARTIAL_VAL(
                uwType,
                u.uwType,
                csprite()->prop_uwType(),
                check_utype_bounds);

            break;
        }
        default:
            return false;
    } « end switch offs »
    return true;
} « end write_vmem »
```

EUD adapters – Linked lists

- In SC 1.16.1
  - Triggers were stored in a Storm linked list data structure
  - Storm is a library that provides containers and platform independent functionality

- In SC:R
  - Triggers are stored as blz::list<_trigger>
  - ‘blz’ is the equivalent of STL’s std namespace

- Other structures in the old game also use Storm lists while the new game uses different containers

EUD adapters – Triggers /1

Because triggers are hard to program, the South Korean hacker (nicknamed Trigger King / trgk) wrote a trigger compiler:

1. You write proper logic in a JavaScript/Python like language called **epScript**

2. The **epScript** gets compiled into a bunch of triggers and is then injected into the appropriate map chunks

3. Map containing triggers compiled with **epScript** can be identified using the bootstrap code that links regular triggers into the dynamic triggers (inside the strings table)

EUD adapters – Triggers /2

- epScript is a very powerful language:
  - The Mario Exodus EUD map was written in that language

- Its compiler hides additional triggers in the cave area of the strings chunk:
  - ➢ Making it hard to reverse-engineer compiled triggers
  - ➢ One needs to write a triggers decompiler to recover the logic

- Compiled triggers are self-modifying and very optimized:
  - ➢ Loops, function calls and other control flow related functionality are implement using self-modifying triggers that change the trigger node links (next and prev links)

EUD adapters – Triggers /3

- EUD maps locate the pointer to the string table (gpMapStr) and adds a constant offset pointing to the additional dynamic triggers inside the string table (see slide 17)

- EUD maps then patch the *m_prevlink* and *m_next* links as needed to introduce as many triggers as needed
    - Inserting new triggers dynamically was never supported in StarCraft. Only the EUD emulator allows such activity.

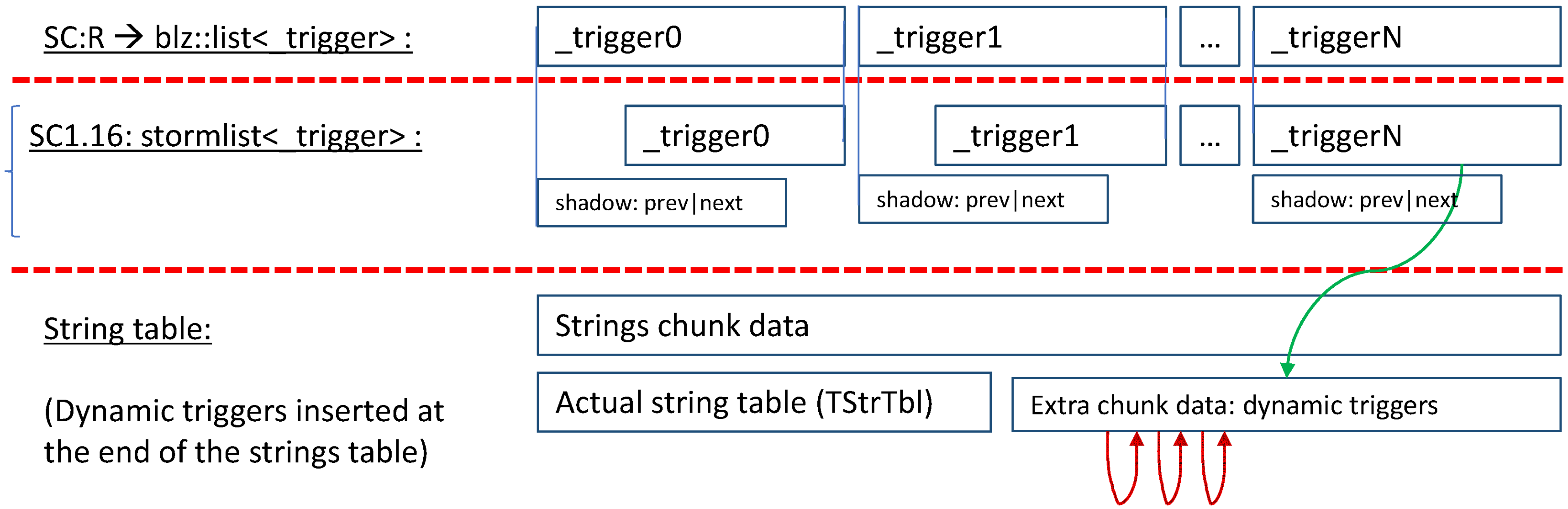
- Compiled/dynamic triggers are the basis of complex and elaborate EUD maps
    - Therefore, supporting dynamic triggers was the first thing added to the EUD emulator

```
struct TSLink_TRIGGERNODE
{
  TSLink_TRIGGERNODE *m_prevlink;
  TRIGGERNODE *m_next;
};

struct _trigger
{
  _condition tConditions[16];
  _action tActions[64];
  unsigned int lFlags;
  char ubPlayer[27];
  char bCurrAction;
};

struct TRIGGERNODE
{
  TSLink_TRIGGERNODE m_link;
  _trigger t;
};
```

EUD adapters – Triggers /4

- From the emulator's perspective, there are two kinds of triggers:
  - Initial triggers originating from the triggers chunk
  - Dynamic triggers linked to the triggers list by patching their node links

- When StarCraft needs to execute triggers after each game loop:
  - The emulator knows how to serve both static triggers and dynamic EUD triggers
  - The emulator does not replicate the backing data (the trigger node data) whenever possible



EUD adapters – Triggers /5

The Storm node EUD adapter hosts the node links as shadow variables

```
//------------------------------------------------------------------------------
template <class T>
class eud_storm_node_adapter_t: public eud_vmemitem_t
{
    template <class TT>
    friend class eud_storm_list_adapter_t;
private:
    eud_STORM_TSLink shadow_link;

    // Returns nullptr if offset is outside the shadow link bounds, the caller
    // then knows how to read the node data in that case
    eud_value_type *get_pval(eud_addr_type offs)
    {
        if (offs == offsetof(eud_STORM_TSLink, m_prevlink))
            return &shadow_link.m_prevlink;
        else if (offs == offsetof(eud_STORM_TSLink, m_next))
            return &shadow_link.m_next;
        else
            return nullptr;
    }

public:
    enum {
        NODE_SIZE = sizeof(eud_STORM_TSLink) + sizeof(T)
    };

    static eud_storm_node_adapter_t *create(
        eud_emu_t *emu,
        T *data)
    {
        auto vitem = new eud_storm_node_adapter_t();
        vitem->addr = emu->reserve_addr(NODE_SIZE);
        vitem->flags = EIF_DYNAMIC | EIF_IS_STORM_LIST_NODE;
        vitem->size = NODE_SIZE;
        vitem->backing_data = data;

        emu->set_item(vitem);
        return vitem;
    }
```

```
virtual bool write_vmem(
    eud_emu_t *emu,
    eud_addr_type offs,
    eud_value_type value,
    int q = 0) override
{
    // Accessing node link structure?
    eud_addr_type *pval = get_pval(offs);
    if (pval != nullptr)
    {
        // Update node structure
        set_pval(pval, value, q);
        return true;
    }

    return eud_vmemitem_t::write_vmem(
                emu,
                offs - sizeof(shadow_link),
                value,
                q);
} « end write_vmem »

virtual bool read_vmem(
    eud_emu_t *emu,
    eud_addr_type offs,
    eud_value_type &value) override
{
    eud_addr_type *pval = get_pval(offs);
    if (pval != nullptr)
    {
        value = *pval;
        return true;
    }

    return eud_vmemitem_t::read_vmem(
                emu,
                offs - sizeof(shadow_link),
                value);
}
```

EUD adapters – Triggers /6

- The Storm list adapter implements an STL compatible iterator
- From the iterator's perspective, any node pointers outside the list has their node links and data in the virtual memory

```
iterator &operator++()
{
    eud_addr_type next = m_cur + offsetof(eud_STORM_TSLink, m_next);

    // Return a terminal iterator if it is not possible to read the next link
    // or the link is negative (terminal by nature)
    if (!container->emu->read_vmem(next, m_cur) || finished())
        *this = container->end();

    return *this;
}

iterator operator++(int)
{
    iterator tmp = *this;
    ++*this;
    return tmp;
}

reference operator*()
{
    // Find the item hosting that address
    eud_vmemitem_t *vitem = container->emu->find_item(m_cur);

    if (vitem == nullptr)
    {
        EUD_ASSERT(("Failed to dereference storm list iterator!", false));
        static auto empty_node = value_type();
        return empty_node;
    }

    // Is that an adapted trigger node?
    eud_addr_type offs;
    if ((vitem->flags & EIF_IS_STORM_LIST_NODE) != 0)
    {
        // The backing data is used as-is
        offs = 0;
    }
    // Is that another host type?
    else
    {
        // This trigger node exists in arbitrary memory,
        // Use the backing data and the current node as offset
        offs = (m_cur - vitem->addr + sizeof(eud_STORM_TSLink));
    }

    return *pointer((char *)vitem->backing_data + offs);
} « end operator* »

pointer operator->()
{
    return &**this;
}
```

EUD adapters – Partial buffers

- Partial buffers adapters are used whenever the virtual item size is greater than the physical item size:

- The adapter serves the mapped data when the access offset is within the mapped range
- It will serve zeros w/o failing when the unmapped area is accessed

EUD adapters – Deferred writes /1

1. Certain adapters resort to using deferred writes as means to speed-up the emulation

2. The EUD map writes in chunks of 4 bytes at a time
   - We don't want to re-construct real game data while the EUD map is still writing the changes

3. Instead, a write handler simply passes-thru the writes to a temporary buffer and marks the adapter as dirty
   - (Reads from dirty offsets are served from the temporary buffer for consistency)

4. After all triggers are executed in that game loop, the emulator invokes all the dirty adapters' deferred write callbacks

5. Inside the deferred write callback, the temporary buffer is then used to reconstruct the real structures used by the game. The adapter dirty flag is then cleared.

EUD adapters – Deferred writes /2

Deferred write example adapter:

1. The status text adapter lets the EUD maps write to a temporary buffer

2. Afterwards, the adapter re-constructs the proper status text structures that are compatible with the new game (SC:R) code

```
class eud_stattxt_adapter_t: public eud_vmemitem_t
{
    size_t orig_tbl_size;
    UWORD orig_str_count;
    blz::string prev_hotkey_profile;

    // This function updates the encoding format of stat text
    bool fix_hotkeys(void *buf, uint32_t tbl_size);
    virtual bool write_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type value,
        int q) override
    {
        emu->set_dirty_and_defer_write(this);
        return eud_vmemitem_t::write_vmem(emu, offs, value, q);
    }

    virtual bool deferred_write_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type value,
        int q) override
    {
        if (!is_dirty())
            return true;
        else
            clear_dirty();

        // Fix the hotkeys from 1.16.1 EUD map to work with SCR
        if (!fix_hotkeys(backing_data, size))
            return false;

        // Convert the EUD patched stat_txt to a TStrTbl (for UTF-8 conversion) using the extended size (and not the original size)
        TPStrTbl str_tbl = str_load_table_from_memory(backing_data, size);
        if (str_tbl == nullptr)
            return false;

        str_allow_kr_cp_conv(str_tbl, true);

        extern StringTable gpStatStrs;

        // Let's patch existing strings and add new ones (if EUD string count is bigger than original string table contents)
        for (UWORD istr = 0, c = MIN(orig_str_count, str_tbl->priv->wStrCount);
             istr < c;
             ++istr)
        {
            // Update the string table from the TStrTblPriv (and convert to UTF-8)
            auto str = str_get_string_kor_eud(str_tbl, istr + 1);
            eud_dbgprint("%04d - %s\n", istr, str);
            gpStatStrs.UpdateString(istr, str);
        }

        str_unload_table_from_memory(str_tbl);

        // Select the no hotkeys profile for this map
        prev_hotkey_profile = CHotkeyManager::Get().SetNoHotkeysProfile();

        return true;
    } « end deferred_write_vmem »
} « end eud_stattxt_adapter_t » ;
```

EUD adapters – Bounded array elements /1

- Various game data variables are integer arrays
- Sometimes, the elements in the array must have bounded values
    - Naturally, the pass-thru (basic) adapter is not suitable (because no validation takes place)
- The bounded array adapter also leverage a shadow array table for all the elements that have incomplete / invalid values
- Only after the written values are valid (within the specified bounds) then changes are reflected into the backing data

EUD adapters – Bounded array elements /2

- The Unit Flingy array's values have an upper bound of 209

```
else if (src_id == EIF_SRC_UNIT_FLINGY)
{
    vitem = new eud_number_array_adapter_t<sizeof(UBYTE), EUD_NUM_FLINGIES>(
        this,
        item);
}
```

```
template <int WIDTH, uint32_t MAX_VAL>
class eud_number_array_adapter_t: public eud_vmemitem_t
{
protected:
    struct shadow_vals_t
    {
        uint32_t val;
        bool     dirty;
        shadow_vals_t(): dirty(false), val(0) { }
    };

    blz::vector<shadow_vals_t> shadow_vals;

public:
    eud_number_array_adapter_t(
        eud_emu_t *emu,
        const eud_itemdef_t *item): eud_vmemitem_t(emu, item)
    {
        flags |= EIF_BACKUP_DATA;

        size_t arr_size = item->size / sizeof(eud_value_type);
        if ((item->size % sizeof(eud_value_type)) != 0)
            ++arr_size;

        // Create a parallel shadow table
        shadow_vals.resize(arr_size);
    }

    virtual bool read_vmem(
        eud_emu_t *emu,
        eud_addr_type offs,
        eud_value_type &value)
    {
        size_t idx = offs / sizeof(eud_value_type);
        EUD_ASSERT(("Reading out of bounds!", idx < shadow_vals.size()));

        // Not dirty? Just read the live value
        if (!shadow_vals[idx].dirty)
            return eud_vmemitem_t::read_vmem(emu, offs, value);

        // Read the shadow value when dirty
        value = shadow_vals[idx].val;
        return true;
    }
```

```
virtual bool write_vmem(
    eud_emu_t *emu,
    eud_addr_type offs,
    eud_value_type value,
    int q = 0)
{
    size_t idx = offs / sizeof(eud_value_type);
    EUD_ASSERT(("Writing out of bounds!", idx < shadow_vals.size()));

    // Still not dirty at the first write, read the live value first
    if (!shadow_vals[idx].dirty)
    {
        // Read the live value directly into the shadow value
        if (!eud_vmemitem_t::read_vmem(emu, offs, shadow_vals[idx].val))
            return false;

        // Mark as dirty
        shadow_vals[idx].dirty = true;
    }

    // Compute the final value
    set_pval(&shadow_vals[idx].val, value, q);

    // On overflow, just update the shadow value
    uint32_t new_val = shadow_vals[idx].val;
    ...
     else if (WIDTH == 1)
    {
        if (     (new_val        & 0xff)  >= MAX_VAL
            || ((new_val >> 8)  & 0xff)  >= MAX_VAL
            || ((new_val >> 16) & 0xff)  >= MAX_VAL
            || ((new_val >> 24) & 0xff)  >= MAX_VAL)
        {
            return true;
        }
    }

    // Finally, we have a full complete value. Clear dirty and update real backing data
    shadow_vals[idx].dirty = false;
    return eud_vmemitem_t::write_vmem(emu, offs, new_val, q = 0);
} « end write_vmem »
```

EUD adapters – Full adapters list /1

Throughout the creation of the EUD emulator, various adapters were devised whenever a new problem is encountered:

- eud_adapter_cards
    - Supports total customization of units command cards
- eud_adapter_csprites and eud_adapter_cunit
    - Allows controlled modifications into the CSprite and CUnit structures
- eud_adapter_group
    - Allows bitmap shuffling inside certain game animation frames
- eud_adapter_keytable
    - Allows EUD maps to intercept key presses (‘a’, ‘s’, ‘w’, ‘d’, key up and key down for example)

EUD adapters – Full adapters list /2

- eud_adapter_mpq
  - Allows support for protected maps.
  - Refer to MPQ frozen maps: https://github.com/phu54321/euddraft/tree/master/freeze
- eud_adapter_msgtbl
  - Read access into the in-game chat messages ("Chatting War" EUD maps)
- eud_adapter_partial_buffer
  - Various non-emulated or no longer existent variables are handled with this adapter
- eud_adapter_playerdata
  - Lets EUD maps read player information (name, race, color, etc.)

EUD adapters – Full adapters list /3

- eud_adapter_pointers
  - All pointer related adaption code
  - Supports partial pointers (backed by shadow values)
- eud_adapter_stattxt
  - Unit status text and hotkeys manipulation
- eud_adapter_stormlist
  - Allows high-level emulation of Storm lists
- eud_adapter_structwithptr
  - Used to emulate structures that contain a mix of basic types (pass-thru) and pointers (incomplete pointers + virtual <-> physical conversion)
- eud_adapter_triggers
  - Supports dynamic triggers emulation

Questions?