This part will focus on how to draw player stats on the game screen. One obvious advantage to this over the previous article is that the player stats are directly in the game, thus there is no need to alt-tab out or play in windowed mode to see what the other players have. Additionally, a toggle feature will be added that allows the player to cycle through various modes to see different types of statistics on the others. The first step is to find a place where either some text, or more specifically, the players scores are drawn. The advantage to finding where the player scores are drawn is that player statistics will be drawn in a natural location and that they can correspond in color to the player. The first step is to think about how the player text is drawn on screen. There is always the player name, a colon followed by a space, and two numbers with a forward slash separating them. Searching for a format similar to this in the games referenced strings leads to what could be the right path.

```
005205B7 PUSH aoc.0066FB30    ASCII "%s %s"
00520ED7 PUSH aoc.00679D68    ASCII "%s%s %s: %d/%d"
00520F09 PUSH aoc.00679D58    ASCII "%s %s: %d/%d"
00520F2D PUSH aoc.00679D4C    ASCII "%s: %d/%d"
0052224A PUSH aoc.00679DA8    ASCII "scr_game::draw"
00522281 PUSH aoc.00679DA8    ASCII "scr_game::draw"
005222AB PUSH aoc.00679D98    ASCII "scr_game::draw2"
005222D3 PUSH aoc.00679D98    ASCII "scr_game::draw2"
0052233F PUSH aoc.00679D88    ASCII "scr_game::draw3"
005223C2 PUSH aoc.00679D88    ASCII "scr_game::draw3"
005223EA PUSH aoc.00679D88    ASCII "scr_game::draw3"
00522409 PUSH aoc.00679D88    ASCII "scr_game::draw3"
00522418 PUSH aoc.00679D78    ASCII "scr_game::draw4"
0052259E PUSH aoc.00679D78    ASCII "scr_game::draw4"
```

A string with our desired format is found, and some other interesting strings which hint to drawing. Setting a breakpoint on where the "%s: %d/%d" string is referenced shows that it does in fact relate to the players  scores. The function that uses this is called on a timer continually to update the players scores on the screen. The image below shows what values are loaded in registers on the first and second call respectively.

```
Registers (FPU)
EAX 0018D33C ASCII "qwerty"
ECX 0018D450
EDX 000000B5
EBX 10BC1050
ESP 0018D2F4
EBP 0018D3A0
ESI 029AAC37
EDI 0018D343
```

```
Registers (FPU)
EAX 0018D33C ASCII "Sejong the Great"
ECX 0018D450 ASCII "qwerty: 181/181"
EDX 000000A6
EBX 10BC1050
ESP 0018D2F4
EBP 0018D3B4
ESI 029AACC1
EDI 0018D34D
```

The general gist of how the function works is that the player whose score is to be retrieved is loaded into the EAX register. The function flow continues until the score is retrieved. The score for the next player is retrieved while the string containing the name and score of the previous player is drawn on the screen. The process then continues for the next player and restarts while the game is active. Modifying the name on a call shows that this does indeed affect how the text is drawn on the screen

This can then be taken advantage of to draw what we want. Tracing exactly where the name gets drawn leads to the call from .text:0052107D.

```
.text:0052104C                  mov     eax, [esp+350h+var_330]
.text:00521050                  mov     ecx, [esp+350h+var_340]
.text:00521054                  mov     edx, [esi+8]
.text:00521057                  push    eax             ; int
.text:00521058                  mov     eax, [esi+4]
.text:0052105B                  push    ecx             ; int
.text:0052105C                  mov     ecx, [esi]
.text:0052105E                  push    ebx             ; int
.text:0052105F                  push    edi             ; int
.text:00521060                  push    edx             ; int
.text:00521061                  mov     edx, [esp+364h+var_31C]
.text:00521065                  push    eax             ; int
.text:00521066                  mov     eax, [esp+368h+var_318]
.text:0052106A                  push    ecx             ; int
.text:0052106B                  push    edx             ; int
.text:0052106C                  mov     edx, [esp+370h+var_338]
.text:00521070                  lea     ecx, [esp+370h+Str1]
.text:00521077                  push    eax             ; int
.text:00521078                  push    ecx             ; Str1
.text:00521079                  mov     ecx, [edx]
.text:0052107B                  push    5               ; int
.text:0052107D                  call    sub_54A510
```

Immediately after this routine completes, the text is drawn on the screen. The function has 11 arguments, the second one being the player name, and third being the RGB value of the player. For the purpose of developing this portion of the hack, the other arguments are irrelevant and probably relate to the position on the screen where the text is to be drawn if I had to take a guess. The idea then is to hook .text:0054A510, grab the stats for the players name, modify the resulting string (which will be in "%s: %d/%d" format) to our custom string, and then pass this back to the original function to be drawn on the screen. The resulting code would look like

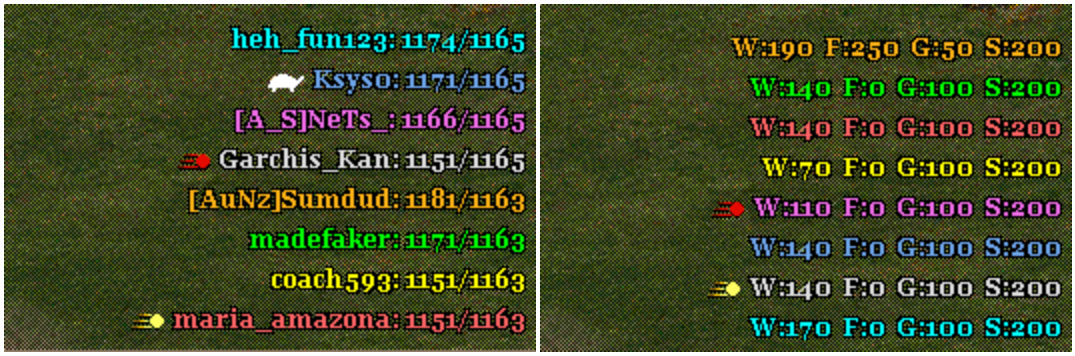
```
__declspec(naked) int score_update_hook(int always_five, char *player, int
rgb_value, int unk1, int unk2,
    int unk3, int unk4, int unk5, int unk6, int unk7, int unk8) {
    __asm pushad
    char *name; //Placeholder for address of name buffer
    __asm {
        mov ebx, dword ptr[esp+0x28]
        mov name, ebx
    }
    stats = items_find_by_name(&base_pointers, name);
    if(stats != NULL) {
        if(toggle_option == CURRENT_RES)
            _snprintf(name, SCORE_MAX_LENGTH, "W:%1.0f  F:%1.0f  G:%1.0f
S:%1.0f\0",
            stats->player_stat->wood, stats->player_stat->food, stats-
>player_stat->gold,
            stats->player_stat->stone);
        else if(toggle_option == ALL_RES)
            _snprintf(name, SCORE_MAX_LENGTH, "W:%1.0f  F:%1.0f  G:%1.0f
S:%1.0f\0",
```

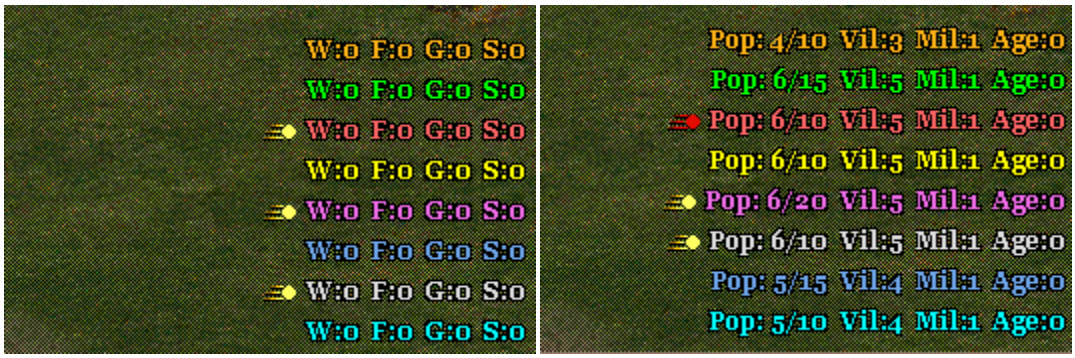
```
            stats->player_stat->total_wood_gathered, stats->player_stat-
>total_food_gathered,
            stats->player_stat->total_gold_gathered, stats->player_stat-
>total_stone_gathered);
        else if(toggle_option == POP_AGE)
            _snprintf(name, SCORE_MAX_LENGTH, "Pop: %1.0f/%1.0f  Vil:%1.0f
Mil:%1.0f  Age:%1.0f\0",
            stats->player_stat->pop_current, (stats->player_stat->pop_current
+ stats->player_stat->pop_left),
            stats->player_stat->num_villagers, stats->player_stat-
>num_military, stats->player_stat->current_age);
    }
    __asm {
        popad
        jmp score_update
    }
}
```

The actual structure of the function is abused a bit here. Since .text:0054A510 has no local variables, we can create one on the stack at [EBP-0x4], since there won't be anything to overwrite there. This dummy argument will act as our third argument at [ESP+0x28] (this function does not set up any sort of BP-based frame). Then anything we do to this dummy argument will be reflected as a change to the third parameter. Thus, the hook grabs the player name of who is to be updated, gets their stats, and checks what mode the user wants to be displayed. The modes are currently controlled through a regular enum in toggle_options.h

```
typedef enum TOGGLE_OPTIONS {
    CURRENT_RES = 1,
    ALL_RES,
    POP_AGE
} toggle_options;
```

Future plans can be to extend this system to allow the user to script their own format to be displayed with what they want. This technique still holds on multiplayer, as shown by the screenshots below.






**Multiplayer note:** The hooking technique posted below is detectable by Voobly. See the end of part 1 for suggestions on bypasses.

**Usage:** Enter a game and hit the hotkey to enable (default is F5). Use F6 to disable the hack, F7 to toggle options, and F8 to clear the stat list in case all names were not retrieved. A player is added to the list when they perform any action in game that modifies their resources. Duplicates are not stored in the list.

I'd prefer for the hack to develop through a series of articles instead of opening up a SVN server on here since that will give me motivation to continue its development.

The source for the in-game hack DLL can be found here.

A downloadable PDF of this post can be found here.