DE LA RECHERCHE À L'INDUSTRIE

Reverse Engineering

Introduction au reverse engineering

Fabrice DESCLAUX

Direction des applications militaires

7 décembre 2015

Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

Introduction au RE

But

- Bases du reverse engineering
- Analyse statique
- Environnement inconnu (normalement) : MegaDrive

Le reverse engineering : pourquoi ?

- Interopérabilité
- Analyse de malware
- Recherche de vulnérabilité
- Protection/déprotection de code (DRM, ...)
- SimLock téléphone
- Perte de code source
- ...

Un bon avocat peut servir ...

- Différences des textes entre pays (reverse hors du pays)
- Contradiction de lois :
  - tout mettre en oeuvre pour protéger un système automatisé de données
  - pas de RE pour rechercher des vulnérabilités

Introduction au RE

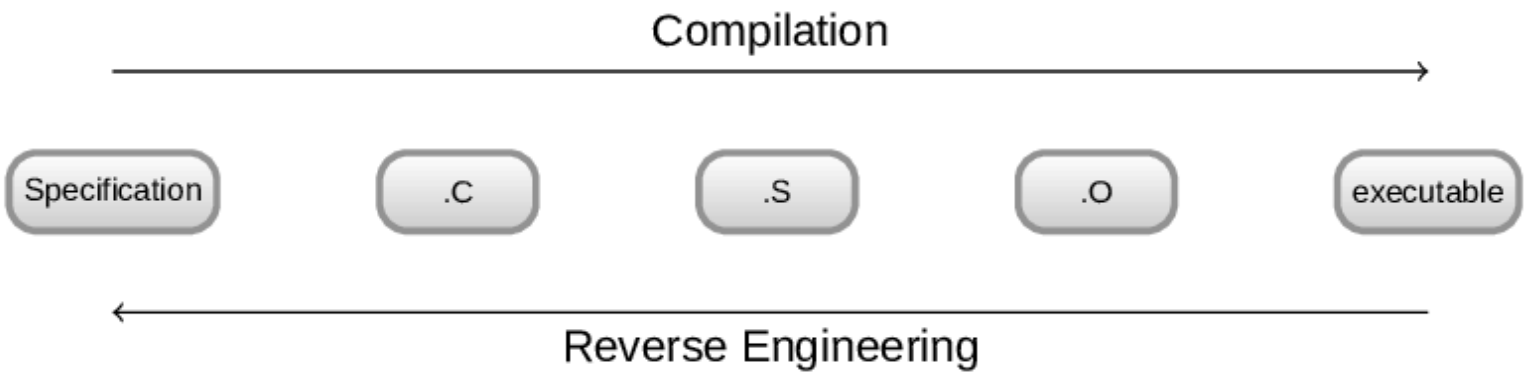
Le reverse engineering : pourquoi ?

- Interopérabilité
- Analyse de malware
- Recherche de vulnérabilité
- Protection/déprotection de code (DRM, ...)
- SimLock téléphone
- Perte de code source
- ...

Un bon avocat peut servir ...

- Différences des textes entre pays (reverse hors du pays)
- Contradiction de lois :
  - tout mettre en oeuvre pour protéger un système automatisé de données
  - pas de RE pour rechercher des vulnérabilités

Introduction au RE



Introduction au RE

Deux familles d'analyse

- analyse statique : étude du code assembleur
- analyse dynamique : observation de code pendant son exécution

Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

Informations sur la MegaDrive

Architecture

- CPU : Motorola 68000
- microprocesseur 32 bit
- bus de donnée 16 bit
- support : cartouche (ROM)

Désassembleur

Mécanisme d'un désassembleur : identique à la lecture

- connaissance d'une langue (français = x86, anglais=arm)
- unité : lettre (octet)
- instruction = plusieurs octets ; un mot = plusieurs lettres
- livre dont vous êtes le héros (le cpu plutôt)

Désassembleur

Lecture d'une phrase

```
jevaisalaplagemebaigner
```

Version lue :

```
je vais a la plage me baigner
```

Désassemblage d'un code

```
b80c00000001c889500a
```

Version lue :

```
b80c000000   mov eax, 0xC
01c8         add eax, ecx
89500a       mov [eax+0xA], edx
```

Désassembleur

Lecture d'une phrase

```
jevaisalaplagemebaigner
```

Version lue (mauvais départ) :

```
jevaisalap la geme baigner
```

Désassembleur

Coté livre

- Vous êtes un chevalier. Assis au coin d'un feu. Vous entendez un bruis derrière vous.
- Si vous allez voir, allez à la page 34, si vous restez assis, allez à la page 12
- [page 12] Un démon arrive derrière vous, vous êtes mort ...

Coté CPU

```
08: cmp eax, 666
    jz  34
12: hlt
```

Flags

Qu'est ce qu'un eflag ?

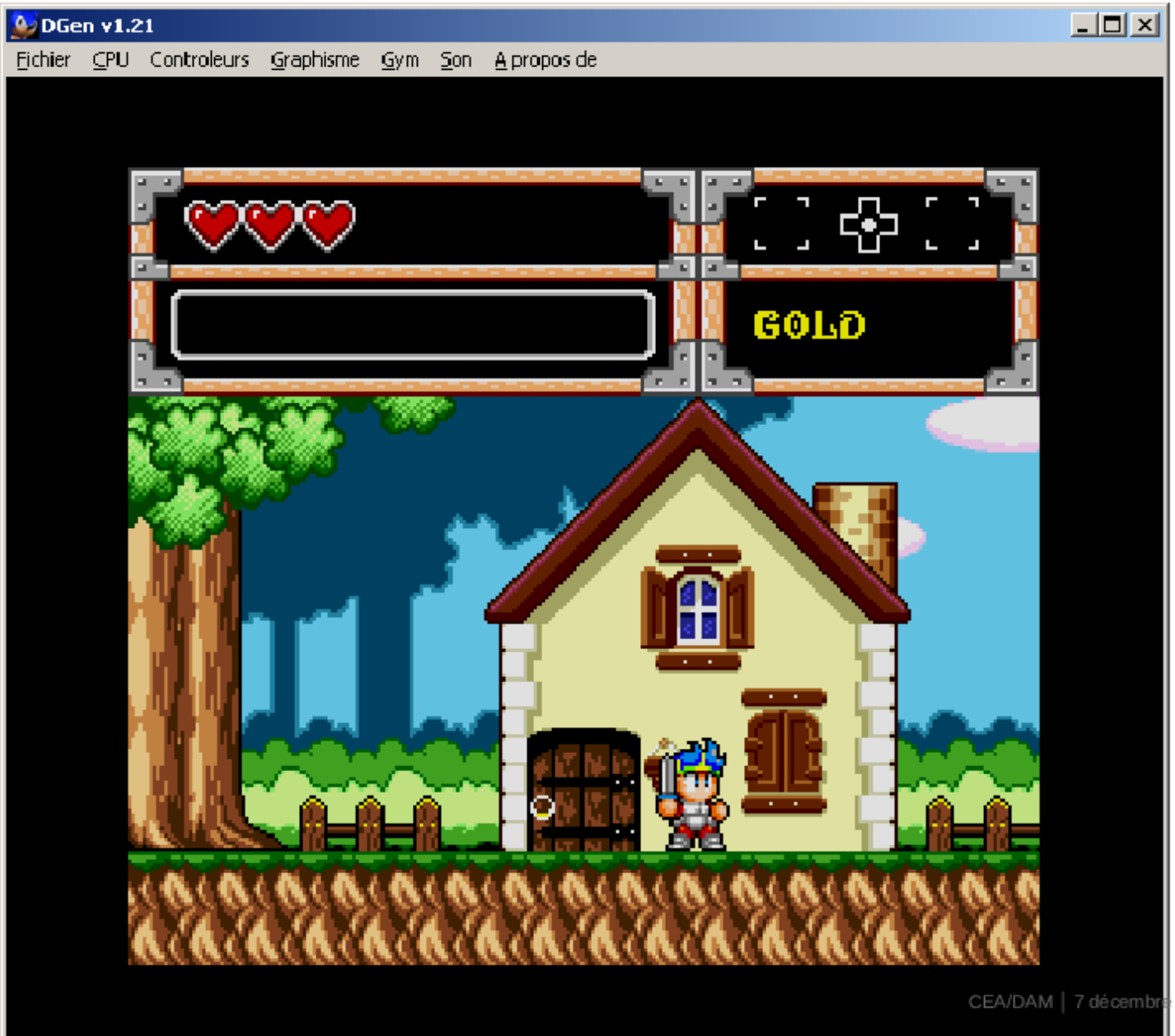
- Composition de drapeaux (éléments de 1 bit représentant un état)
- drapeux "zer0" : mis à 1 si le dernier résultat est nul
- Concrètement : un étudiant au fond de la classe qui lève le bras quand le dernier résultat est nul.

Objectif

But de l'exercice

- Bases du Reverse engineering
- Analyse statique de code
- Manipulation d'un désassembleur
- Modification d'un binaire
- $\rightarrow$ Avoir plein de conteneur de coeur

Vu du jeu


DGen v1.21
Fichier CPU Controleurs Graphisme Gym Son A propos de
GOLD

Menu initial

Premiers pas : WonderBoy 5

Nouvelle architecture

- Nous n'avons pas de débuggueur
- Analyse statique pure
- Nous pourrons lancer le jeux à travers un émulateur

En général, aide de l'environnement

- Format de fichier : PE/ELF, …ici, ROM
- OS : ici inconnu (y'a t il même un OS ?)
- Agencement mémoire : emplacement du matériel (Ex : 16 Mo mappés sur les cartes PCI, …)

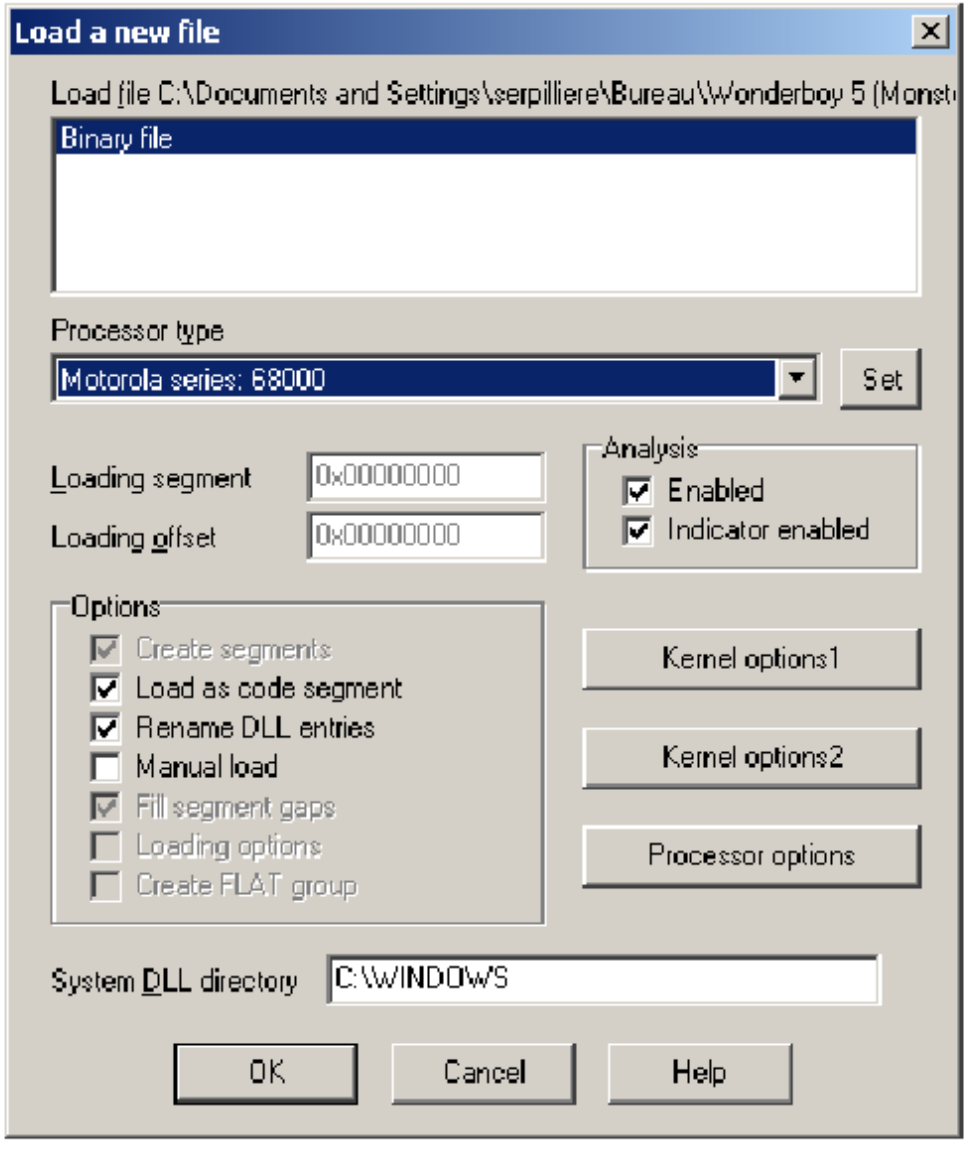
Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

Informations sur l'architecture



Informations sur l'architecture

Vue hexadécimale

- On vient juste de charger un *blob* binaire
- Nous ne connaissons pas l'emplacement du code et des données

Premier problème : trouver le code

- Programme = Algorithme + données
- ambiguïté entre octets de code et octets de données
- différence entre architecture *Harvard* et *Von Neuman*

Vue hexadécimale

```
ROM:000000E0  00 00 00 06 00 00 00 06  00 00 00 06 00 00 00 06  ............
ROM:000000F0  00 00 00 06 00 00 00 06  00 00 00 06 00 00 00 06  ............
ROM:00000100  53 45 47 41 20 4D 45 47  41 20 44 52 49 56 45 20  SEGA MEGA DRIVE
ROM:00000110  28 43 29 53 45 47 41 20  31 39 39 30 2E 4A 41 4E  (C)SEGA 1990.JAN
ROM:00000120  57 6F 6E 64 65 72 20 42  6F 79 20 56 20 4D 6F 6E  Wonder Boy V Mon
ROM:00000130  73 74 65 72 20 57 6F 72  6C 64 20 49 49 49 20 20  ster World III
ROM:00000140  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
ROM:00000150  57 4F 4E 44 45 52 20 42  4F 59 20 69 6E 20 4D 6F  WONDER BOY in Mo
ROM:00000160  6E 73 74 65 72 20 77 6F  72 6C 64 20 20 20 20 20  nster world
ROM:00000170  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
ROM:00000180  47 4D 20 47 2D 34 30 36  30 20 20 2D 30 30 9D 79  GM G-4060  -00 y
ROM:00000190  4A 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  J
ROM:000001A0  00 00 00 00 00 0B FF FF  00 FF 00 00 00 FF FF FF  .....  . ...
ROM:000001B0  52 41 E8 40 00 20 00 01  00 20 00 01 20 20 20 20  RA @. .. .
ROM:000001C0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
ROM:000001D0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
```

Vue de la ROM

```
ROM:000001FE    dc.b $20
ROM:000001FF    dc.b $20
ROM:00000200 loc_200:    ; DATA XREF: ROM:000278AEr
ROM:00000200    tst.l   ($A10008).l
ROM:00000206    bne.s   loc_20E
ROM:00000208    tst.w   ($A1000C).l
ROM:0000020E loc_20E:    ; CODE XREF: ROM:00000206j
ROM:0000020E    bne.s   loc_28C
ROM:00000210    lea     loc_28E,a5
ROM:00000214    movem.w (a5)+,d5-d7
ROM:00000218    movem.l (a5)+,a0-a4
ROM:0000021C    move.b  -$10FF(a1),d0
ROM:00000220    andi.b  #$F,d0
ROM:00000224    beq.s   loc_22E
ROM:00000226    move.l  #'SEGA',2F00(a1)
```

Récupération d'informations

Désassembleur interactif

- Transformation des labels : *0x200* en *loc_200*
- Transformation maison : *loc_200* en *le_label_voulu*

Origine/modifié

```
// Code original
bne.s   20E
// Génération automatique de label
bne.s   loc_20E
// label maison
bne.s   monlabel
```

Récupération d'informations

Références

- un code *utilise* une donnée
- un code *appelle* un autre code

Et vice-versa.

Vue hexadécimale

Trouver le code intéressant

- Nous n'avons pas d'informations sur le format des appels systèmes (s'il y en a)
- Nous n'avons pas de débuggueur
- Point d'entrée : les ressources
- Les chaînes de caractères du jeu sont sûrement dans la ROM
- $\rightarrow$ Utilisation de *strings*

Vue des chaînes de caractères

```
ROM:000082C3 00000008 C 6SELECT
ROM:000082CB 00000007 C WEAPON
ROM:000082D2 00000007 C ARMOR
ROM:000082D9 00000007 C SHIELD
ROM:000082E0 00000007 C BOOTS
ROM:000082E7 00000005 C ITEM
ROM:000082EC 00000006 C MAGIC
ROM:00008331 00000005 C  g6a
ROM:00008343 00000006 C  fa\\g
ROM:00008373 00000005 C \"g,a
ROM:000083A1 00000005 C 4Nu08
ROM:000083AE 00000006 C Nu0\ar
ROM:000083C2 00000005 C H@0<
ROM:0000845C 00000008 C g\naNaLSn
ROM:000084AA 00000006 C RGNua
ROM:000084DC 00000005 C NuAP
ROM:000084E7 00000005 C GOLD
```

Récupération d'informations

Vue assembleur

- Nous allons forcer le désassemblage du tout les octets
- Mais tous ne sont pas du code ! (images, sons, ...)
- Un travail sur l'assembleur sera nécessaire
- Les références croisées appelant/appelé seront toutefois précentes

Vue assembleur

```
ROM:000084C4 loc_84C4:           ; CODE XREF: sub_7E7C+Cj
ROM:000084C4    move.w  #209,d7
ROM:000084C8    lea     unk_84DE,a1
ROM:000084CC    moveq   #2,d3
ROM:000084CE
ROM:000084CE loc_84CE:              ; CODE XREF: loc_84D8
ROM:000084CE    bsr.w   sub_83B0
ROM:000084D2    bsr.w   sub_786A
ROM:000084D6    addq.b  #2,d7
ROM:000084D8    dbf     d3,loc_84CE
ROM:000084DC    rts
ROM:000084DC; END OF FUNCTION CHUNK FOR sub_7E7C
ROM:000084DC; ---------------------
ROM:000084DE unk_84DE :       dc.b $41; A; DATA XREF: loc_84C8
ROM:000084DF                  dc.b $50; P
ROM:000084E0; ---------------------
ROM:000084E0    ori.w   #5000,d4
ROM:000084E4    subq.w  #1,(a0)
ROM:000084E6    ori.w   #4F4C,d7
ROM:000084EA    neg.b   d0
```

Récupération d'informations

Listing assembleur

- Ici, les chaînes de caractères n'apparaissent pas (elles ont été désassemblées)
- Nous devons retravailler cela.
- Ici, il s'agit *à priori* d'un tableau de chaînes de caractères

Désassembleur

- Le désassembleur affiche les *références croisées*
- Appelant/appelé
- Idem pour les références sur les données
- Les tables de sauts
- ...

Architecture 68000

Le 68k en 1 slide

- move src, dst
- lea addr, reg $\rightarrow$ load effective address
- bsr XXX $\rightarrow$ branch sub routine
- rts $\rightarrow$ return sub routine
- DBcc Dn,<label> $\rightarrow$ decrement and branch Instruction de boucle conditionnelle CC

Vue assembleur

```
ROM:000084C4 loc_84C4:           ; CODE XREF: sub_7E7C+Cj
ROM:000084C4    move.w  #209,d7
ROM:000084C8    lea     aAP,a1
ROM:000084CC    moveq   #2,d3
ROM:000084CE
ROM:000084CE loc_84CE:              ; CODE XREF: loc_84D8
ROM:000084CE    bsr.w   sub_83B0
ROM:000084D2    bsr.w   sub_786A
ROM:000084D6    addq.b  #2,d7
ROM:000084D8    dbf     d3,loc_84CE
ROM:000084DC    rts
ROM:000084DC; END OF FUNCTION CHUNK FOR sub_7E7C
ROM:000084DC; ---------------------
ROM:000084DE aAp :            dc.b 'AP',0; DATA XREF: loc_84C8
ROM:000084E1 aDp:            dc.b 'DP',0
ROM:000084E4 aSp:            dc.b 'SP',0
ROM:000084E7 aGold:          dc.b 'GOLD',0
ROM:000084EC sub_84EC:                   ; CODE XREF: sub_85A8+12p
ROM:000084EC    lea     (FFFF959F).w,a0
ROM:000084F0    moveq   #0,d3
```

Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

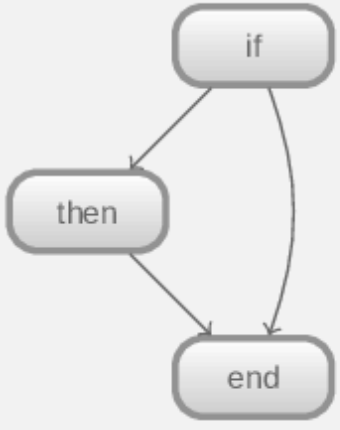
Représentation du code

Flot d'exécution

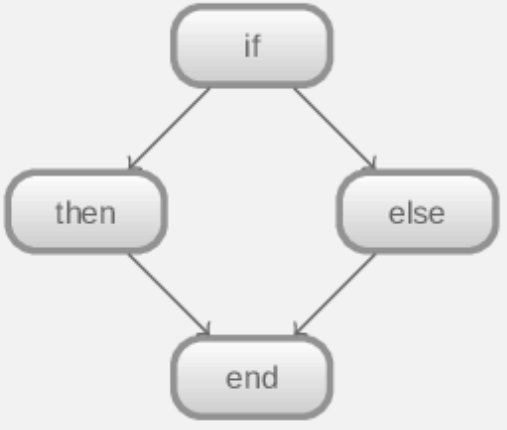
- Extraction du graphe de flot d'exécution d'une fonction
- un noeud est une suite d'instruction exécutée de façon *atomique* (pas de saut vers/depuis ces lignes)
- une arrête représente un saut/appel entre deux noeuds

Représentation du code

If … then…

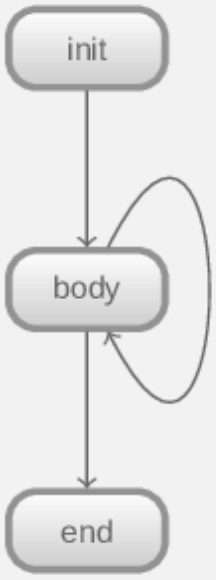


If … then… else…

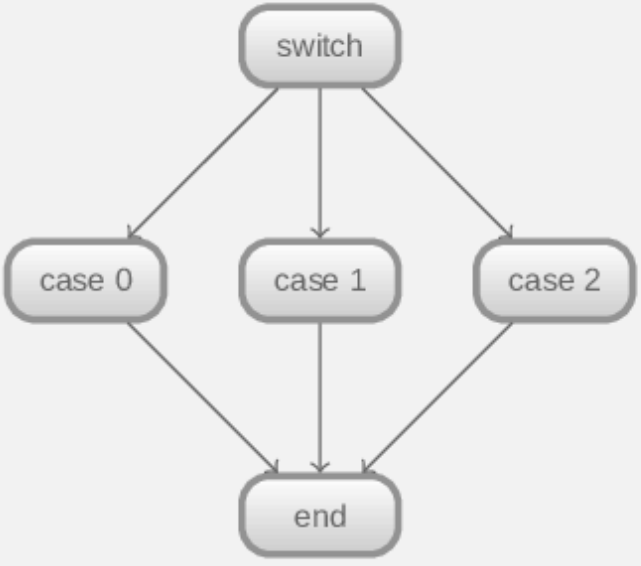


Représentation du code

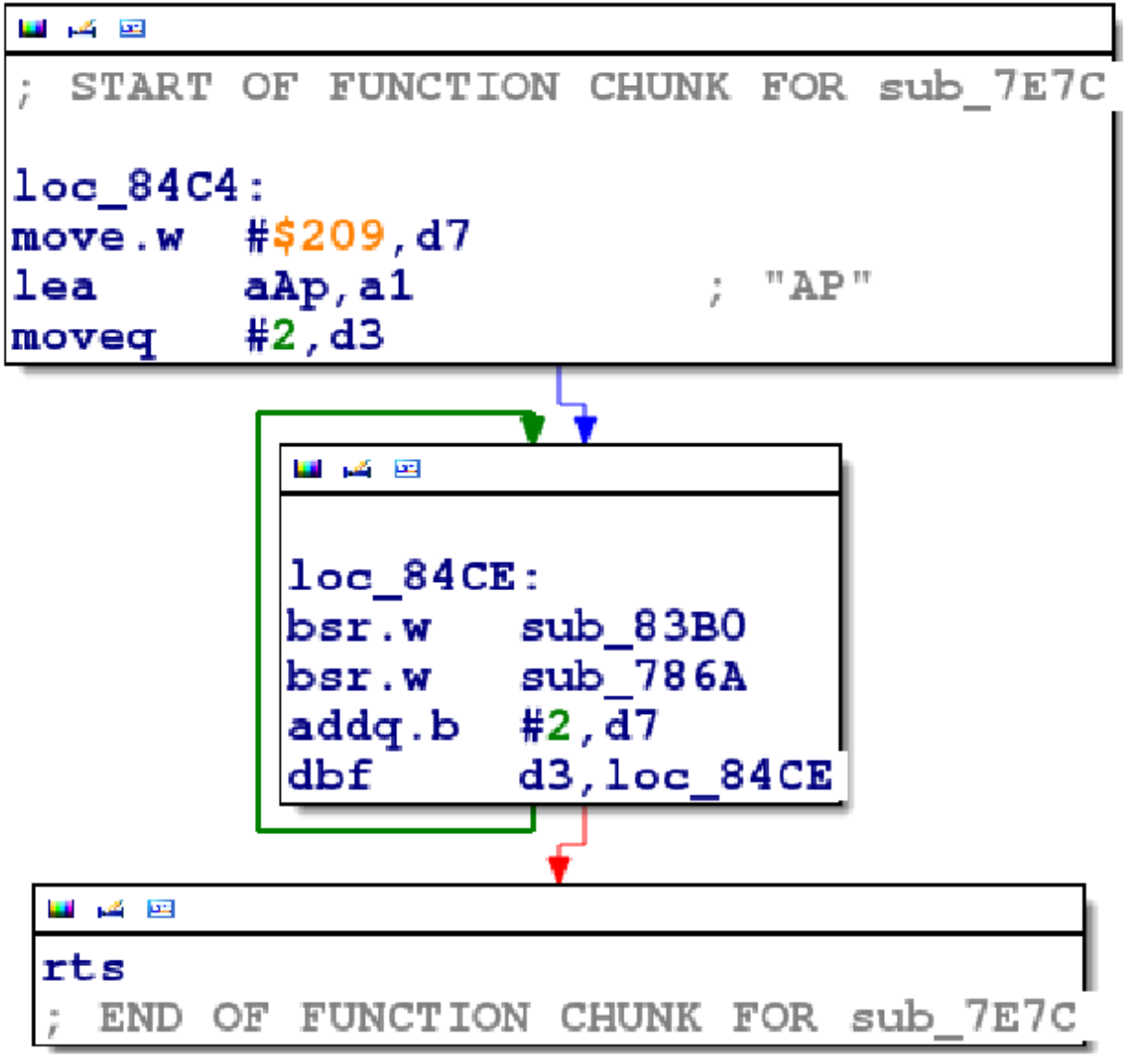
While(…) …



Switch(…) …



Menu initial



Représentation du code

Lecture de l'assembleur

- D7 = 0x209
- A1 = pointeur tableau de chaînes de caractères
- D3 = 2
  - BOUCLE :
  - 2 appels de sous fonctions
  - D7 += 2
- retour à l'appelant

→ Comment sont passés les arguments entre fonctions ?

Représentation du code

Un compilateur utilise une ABI définie

- Comment passer les arguments entre fonction (registre, pile)
- qui nettoie la pile après un appel de fonction
- quel registres doivent être conservés entre appel de fonction
- comment retourner une valeur
- ...

Représentation du code

Code

```c
int callee(int, int, int);

int caller(void)
{
        int ret;

        ret = callee(1, 2, 3);
        ret += 5;
        return ret;
}
```

```asm
caller:
        push    ebp
        mov     ebp, esp
        push    3
        push    2
        push    1
        call    callee
        add     esp, 12
        add     eax, 5
        pop     ebp
        ret
```

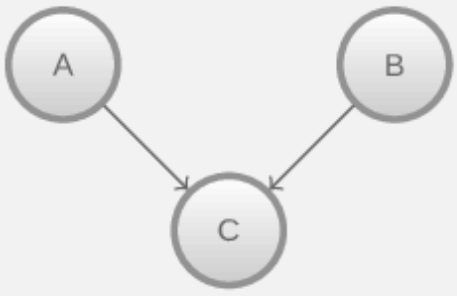
Menu du jeux

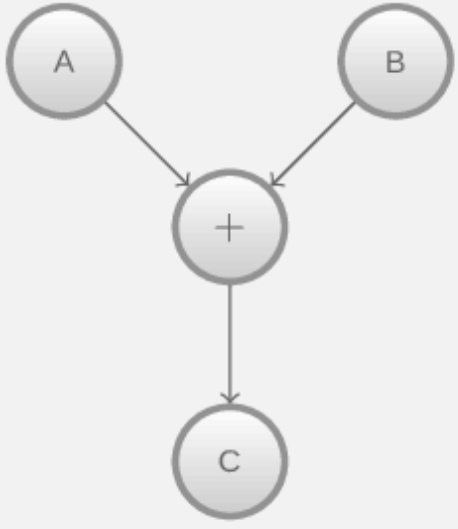
Hypothèses

- Ce code sert à afficher le menu du jeu
- AP DP SP sont les statistiques du joueur
- La boucle afficherait ces chaînes de caractères

Graphe de flot de données (plusieurs versions)

C = A+B





Première fonction

```
1   sub_83B0:
2       move.w  d7,d0
3       moveq   #0,d1
4       move.b  d0,d1
5       lsr.w   #8,d0
6   loc_83B8:
7       lsl.w   #5,d1
8       or.w    d1,d0
9       add.w   d0,d0
10      ori.w   #$5800,d0
11      swap    d0
12      move.w  #3,d0
13      move.l  d0,($C00004).l
14      rts
```

Analyse

- D7 est un argument (0x209)
- D0/D1 sont des registres temporaires
- @(0xC00004) doit être la sortie

Première fonction

```
1   sub_83B0 :
2       move.w   d7,d0
3       moveq    #0,d1
4       move.b   d0,d1
5       lsr.w    #8,d0
6   loc_83B8 :
7       lsl.w    #5,d1
8       or.w     d1,d0
9       add.w    d0,d0
10      ori.w    #$5800,d0
11      swap     d0
12      move.w   #3,d0
13      move.l   d0,($C00004).l
14      rts
```

Analyse

- D7 est un argument (0x209)
- D0/D1 sont des registres temporaires
- @(0xC00004) doit être la sortie

Représentation du code

Remontée de typage

- un registre est un *entier*
- mais un entier peut être :
  - un (vrai) entier
  - un pointeur
  - un caractères
  - ...

Typage

- les fonctions utilisées peuvent montrer le type des paramètres
- on peut alors faire de la *propagation* de types (dans les deux sens)
- réitérer
- propager aux autres fonctions soeurs.

Deuxième fonction

```
sub_786A:; CODE XREF: ROM:loc_7BD8p
    move.w  #A000,d1
    bra.s   loop
loc_7870:; CODE XREF: ROM:000077E0p
    move.w  #8000,d1
loop:
    move.b  (a1)+,d1
    beq.s   loop_end
    move.w  d1,(C00000).l
    bra.s   loop
loop_end:
    rts
```

Analyse

- A1 est un pointeur sur une chaîne de caractère
- boucle jusqu'à trouver un caractères nul
- incrémente A1
- @(0xC00000) doit être la sortie

Deuxième fonction

```
sub_786A:; CODE XREF: ROM:loc_7BD8p
    move.w  #A000,d1
    bra.s   loop
loc_7870:; CODE XREF: ROM:000077E0p
    move.w  #8000,d1
loop:
    move.b  (a1)+,d1
    beq.s   loop_end
    move.w  d1,(C00000).l
    bra.s   loop
loop_end:
    rts
```

Analyse

- A1 est un pointeur sur une chaîne de caractère
- boucle jusqu'à trouver un caractères nul
- incrémente A1
- @(0xC00000) doit être la sortie

Représentation du code

Hypothèses

- La première procédure doit préparer la position de la chaîne de caractère à afficher (SetTextXY)
- La deuxième fonction affiche la chaîne de caractère à l'écran (DisplayText)

Vérification : deux techniques

- observer les fonctions soeurs (paramètres, typage, valeur de retour)
- $\rightarrow$ Vérification : on va changer le 0x209 ce qui devrait changer la position du texte à l'écran

Analyse des références

Vérification : parent 1

```
ROM:00007F16    move.w  #E08,d7
ROM:00007F1A    bsr.w   settextXY
ROM:00007F1E    lea     aPause,a1       ; "PAUSE"
ROM:00007F22    bsr.w   display_text2
```

Analyse des références

Vérification : parent 2

```
ROM:000077D0    lea     items_positions,a0
ROM:000077D4    lea     items_tab,a1    ; "WEAPON"
ROM:000077D8    moveq   #5,d2
ROM:000077DA loop_items :
ROM:000077DA    move.w  (a0)+,d7
ROM:000077DC    bsr.w   settextXY
ROM:000077E0    bsr.w   display_text2
ROM:000077E4    dbf     d2,loop_items


ROM:000082CB items_tab :      dc.b 'WEAPON',0; DATA XREF: ROM:000077D4o
ROM:000082D2 aArmor:          dc.b 'ARMOR ',0
ROM:000082D9 aShield_0:       dc.b 'SHIELD',0
ROM:000082E0 aBoots:          dc.b 'BOOTS ',0
ROM:000082E7 aItem:           dc.b 'ITEM',0
ROM:000082EC aMagic:          dc.b 'MAGIC',0


ROM:00007D6E items_positions :dc.w $F07  ; DATA XREF: ROM:000077D0o
ROM:00007D70                  dc.w $1807
ROM:00007D72                  dc.w $F0A
ROM:00007D74                  dc.w $180A
...
```

Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

Modification de l'instruction

```
ROM:000084C4 3E3C 0209 move.w  #209,d7
```

- Changeons le 0x209 dans la ROM
- Outil : Éditeur hexadécimal

Menu initial

Modification de la position

Modification de la position (bis)

Analyse des appelants

Références de code

```
ROM:00007E7C 6100 011A    bsr.w   sub_7F98
ROM:00007E80 6100 0138    bsr.w   sub_7FBA
ROM:00007E84 6100 06B6    bsr.w   sub_853C
ROM:00007E88 6000 063A    bra.w   display_ap_dp_sp
```

Recherche des fonctions intéréssantes

- On va supprimer l'appel à une des fonctions pour voir ses effets de bords
- On remplace le code binaire du *emph* par un NOP (en espérant que la fonction n'a pas d'autre effets de bords que l'affichage)
- `ROM:00000366 4E71 nop`

Analyse des appelants

Références de code

```
ROM:00007E7C 6100 011A    bsr.w   sub_7F98
ROM:00007E80 6100 0138    bsr.w   sub_7FBA
ROM:00007E84 6100 06B6    bsr.w   sub_853C
ROM:00007E88 6000 063A    bra.w   display_ap_dp_sp
```

Recherche des fonctions intéréssantes

- On va supprimer l'appel à une des fonctions pour voir ses effets de bords
- On remplace le code binaire du *emph* par un NOP (en espérant que la fonction n'a pas d'autre effets de bords que l'affichage)
- `ROM:00000366 4E71 nop`

Fonction sub_853C *noppée*

Fonctions connues

Référence de code

```
ROM:00007E7C 6100 011A    bsr.w   sub_7F98
ROM:00007E80 6100 0138    bsr.w   sub_7FBA
ROM:00007E84 6100 06B6    bsr.w   display_hearts
ROM:00007E88 6000 063A    bra.w   display_ap_dp_sp
```

Analyse de la fonction

- Cette fonction doit récupérer le nombre de conteneur de coeur
- L'afficher à l'écran (ainsi que le coeur, …)
- Nous allons retrouver l'emplacement du nombre de coeur.

```
ROM:0000853C     display_hearts:  ; CODE XREF: sub_7E7C+8p
ROM:0000853C        bsr.w   sub_85CC
ROM:00008540        move.w  #$606,d7
ROM:00008544        bsr.w   settextXY
ROM:00008548        move.b  ($FFFF9F00).w,($FFFF95E1).w
ROM:0000854E        move.w  #$706,d7
ROM:00008552     loc_8552:        ; CODE XREF: sub_7AC0+82p
ROM:00008552        move.w  #$8017,($C00000).l
ROM:0000855A        bsr.w   settextXY
ROM:0000855E        move.w  #$8030,d2
ROM:00008562        moveq   #0,d3
ROM:00008564        move.b  ($FFFF95E1).w,d3
ROM:00008568        moveq   #0,d4
ROM:0000856A        lea     ($C00000).l,a1
ROM:00008570     loc_8570:        ; CODE XREF: display_hearts+3Cj
ROM:00008570        subi.w  #$A,d3
ROM:00008574        bcs.s   loc_857A
ROM:00008576        addq.w  #1,d4
ROM:00008578        bra.s   loc_8570
ROM:0000857A     loc_857A:        ; CODE XREF: display_hearts+38j
ROM:0000857A        addi.w  #A,d3
ROM:0000857E        tst.w   d4
ROM:00008580        beq.s   loc_858C
ROM:00008582        or.w    d2,d4
ROM:00008584        move.w  d4,(a1)
ROM:00008586        or.w    d2,d3
ROM:00008588        move.w  d3,(a1)
ROM:0000858A        rts
ROM:0000858C     loc_858C:        ; CODE XREF: display_hearts+44j
ROM:0000858C        or.w    d2,d3
ROM:0000858E        move.w  d3,(a1)
ROM:00008590        move.w  #8020,(a1)
ROM:00008594        rts
```

Menu initial

Modification de la première position (0x606)

Modification de la seconde position (0x706)

Modification du caractère (0x8017)

Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

Analyse du code

display heats :

- Affiche le logo du coeur
- Affiche la croix
- Tant que nombre_coeur $>$ 10
  - Ajoute 1 aux décimale
  - enlève 10 au nombre_coeur
- Affiche décimale
- Affiche unitées
- Affiche 0x8020 (espace)

```
ROM:0000853C display_hearts: ; CODE XREF: sub_7E7C+8p
ROM:0000853C    bsr.w   display_heart_logo                    display heart logo
ROM:00008540    move.w  #$606,d7                              Cross position
ROM:00008544    bsr.w   settextXY                             settextXY
ROM:00008548    move.b  ($FFFF9F00).w,($FFFF95E1).w           retrieve heart number
ROM:0000854E    move.w  #$706,d7                              Decimal position
ROM:00008552 loc_8552:      ; CODE XREF: sub_7AC0+82p
ROM:00008552    move.w  #$8017,($C00000).l                    Output 0x8017 character
ROM:0000855A    bsr.w   settextXY                             settextXY
ROM:0000855E    move.w  #$8030,d2                             Prepare text decimal
ROM:00008562    moveq   #0,d3
ROM:00008564    move.b  ($FFFF95E1).w,d3                      Get heart_number
ROM:00008568    moveq   #0,d4                                 2nd decimal is null
ROM:0000856A    lea     ($C00000).l,a1                        A1 receive screen address
ROM:00008570 loc_8570:      ; CODE XREF: display_hearts+3Cj
ROM:00008570    subi.w  #$A,d3                                Test heart_number <10
ROM:00008574    bcs.s   heart_number_inf_10
ROM:00008576    addq.w  #1,d4                                 Add 1 second decimal
ROM:00008578    bra.s   loc_8570                              Loop until inf 10
ROM:0000857A loc_857A:      ; CODE XREF: display_hearts+38j
ROM:0000857A    addi.w  #A,d3                                 heart_number was inf 10
ROM:0000857E    tst.w   d4                                    If Second decimal
ROM:00008580    beq.s   loc_858C
ROM:00008582    or.w    d2,d4                                 prepare second decimal
ROM:00008584    move.w  d4,(a1)                               Output second decimal
ROM:00008586    or.w    d2,d3                                 prepare first decimal
ROM:00008588    move.w  d3,(a1)                               Output first decimal
ROM:0000858A    rts
ROM:0000858C loc_858C:      ; CODE XREF: display_hearts+44j
ROM:0000858C    or.w    d2,d3                                 prepare first decimal
ROM:0000858E    move.w  d3,(a1)                               Output first decimal
ROM:00008590    move.w  #8020,(a1)                            Output Space character
ROM:00008594    rts
```

Modification du binaire

emplacement du nombre de coeur

- le nombre de coeur est à l'adresse 0xFFFF9F00
- On va modifier le code du menu pour changer le nombre de coeur
- le code va mettre le nombre de coeur à 6

Midification

- On doit assembler :
- `11FC 0006 9F00 move.b  #0x6,(FFFF9F00).w`
- Ce code va mettre le nombre de coeur à 6
- (si on va dans la fonction d'affichage du menu)

Patch


DGen v1.21
Fichier CPU Controleurs Graphisme Gym Son A propos de
GOLD
0

Sommaire

1 Introduction

2 Analyse statique
- Premier pas
- Récupération d'informations
- Représentation du code

3 Résultat
- Analyse de fonction
- Suivi du flot de données
- Bonus

Bonus

Analyse des références

```
ROM:0000786A display_text1:; CODE XREF: ROM:...
ROM:0000786A    move.w  #A000,d1
ROM:0000786E    bra.s   loc_7874
```

Référence des appelants

- La fonction d'affichage est appelée par d'autre fonctions intéressantes
- On a cette liste
- Rappel : la chaîne à afficher est dans le registre A1

Analyse des références

```
ROM:00008596 loc_8596:; CODE XREF: sub_97F0-227Aj
ROM:00008596    moveq   #6,d3
ROM:00008598    move.w  #1605,d7        ;GOLD offset
ROM:0000859C    bsr.w   settextXY       ;settextXY
ROM:000085A0    lea     aGold,a1        ; "GOLD"
ROM:000085A4    bsr.w   display_text1   ;displaytext
ROM:000085A8 loc_85A8:; CODE XREF: ROM:00001888p
ROM:000085A8    move.w  #1806,d7        ;numeric offset
ROM:000085AC    bsr.w   settextXY       ;settextXY
ROM:000085B0    moveq   #6,d5
ROM:000085B2    bsr.w   sub_8762
ROM:000085B6    move.l  (FFFF962C).w,d0;retrieve gold value
ROM:000085BA    bsr.w   sub_84EC
```

État de l'émulateur

L'émulateur peut prendre une *photo* de l'état du programme

- Il enregistre cet état dans un fichier
- On peut sauver l'état, le modifier et recharger cet état

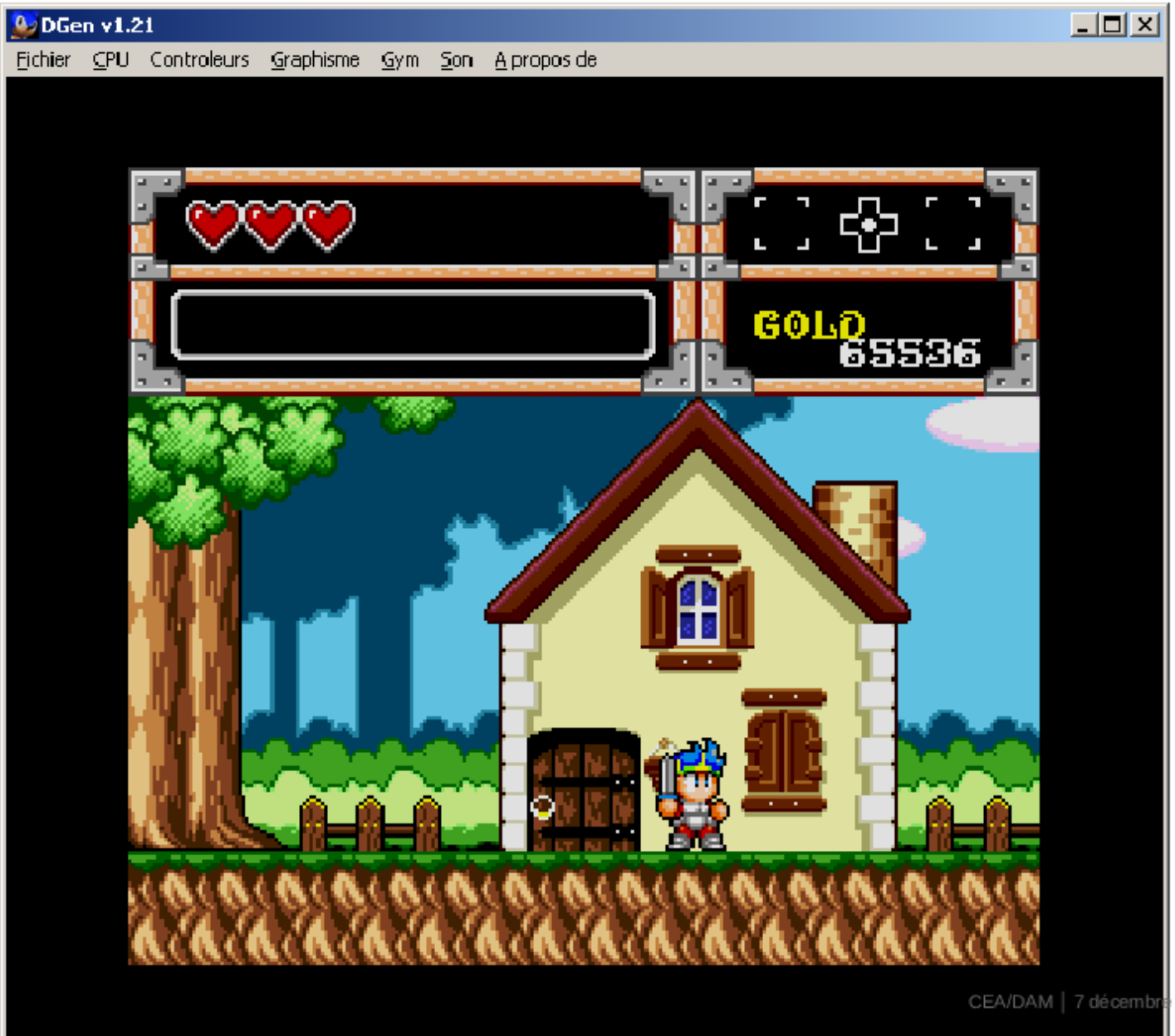
Code

```
1 fseek(hand,0x2478,SEEK_SET);
2 fget(ram,        0x10000);
3 byteswap_memory(ram,0x10000);
```

Example

- L'offset de l'argent est située à l'adresse 0x962c
- La mémoire dans l'état sauvé est à l'offset 0x2478
- L'offset de l'argent sur le le fichier d'état est à l'offset 0xbaa4

Modification de l’argent



Pour aller plus loin

Vous en voulez encore ?

- Micro corruption (MSP430, très bien nivelé)
  - `microcorruption.com/`
- RootMe, NewbieContest (recueil de crackmes en tout genre)
  - `www.root-me.org`
  - `www.newbiecontest.org`
- Challenge annuel du SSTIC (plusieurs disciplines, solutions très détaillées)
  - `communaute.sstic.org/ChallengeSSTIC201X`
- Livres, ressources en lignes, conférences, ...
  - “Practical Reverse Engineering : x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation”

Commissariat à l'énergie atomique et aux énergies alternatives CEA/DAM
Centre de Bruyères-le-Châtel | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019