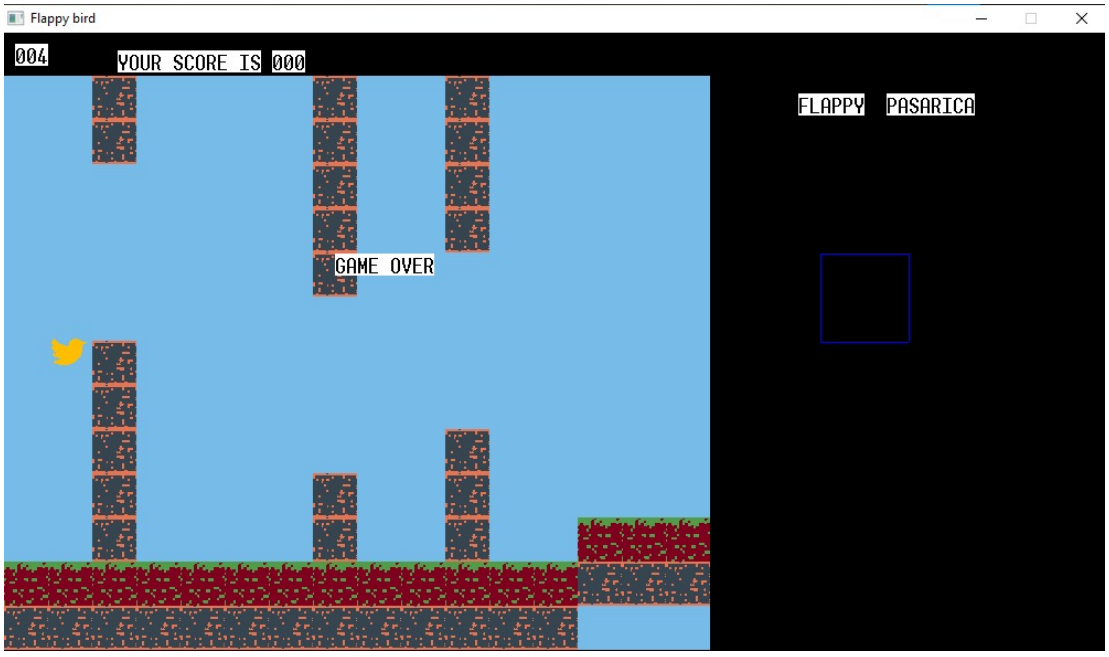
Documentatie Proiect Flappy Bird in Assembly

Muresan Davide-Andrei

20.03.2023

1 Introducere

Acest document descrie proiectul Flappy Bird" implementat in limbaj de asamblare (Assembly x86). Proiectul consta intr-un joc simplu in care jucatorul controleaza o pasare care trebuie sa evite obstacolele.



2 Structura Proiectului

Proiectul este structurat in mai multe fisiere, fiecare avand un rol specific:

- **Pasaticaproj.asm** - Fisierul principal care contine logica jocului si rutinele de desenare.

- **digits.inc** - Contine reprezentarea grafica a cifrelor pentru afisarea scorului.
- **letters.inc** - Contine reprezentarea grafica a literelor pentru afisarea mesajelor.
- **pasarica.inc** - Contine reprezentarea grafica a pasarii si a obstacolelor.
- **starimatrix.inc** - Contine matricea de stare a jocului.
- **patrat_red.inc** - Contine reprezentarea grafica a unui patrat rosu.

3 Implementare

3.1 Initializarea Jocului

Jocul incepe prin initializarea ferestrei de afisare si alocarea memoriei pentru zona de desen. Acest lucru este realizat in functia `start`:

```
start:
    ; Alocam memorie pentru zona de desenat
    mov eax, area_width
    mov ebx, area_height
    mul ebx
    shl eax, 2
    push eax
    call malloc
    add esp, 4
    mov area, eax

    ; Apelam functia de desenare a ferestrei
    push offset draw
    push area
    push area_height
    push area_width
    push offset window_title
    call BeginDrawing
    add esp, 20

    ; Terminarea programului
    push 0
    call exit
```

3.2 Desenarea Elementelor

Desenarea elementelor jocului, cum ar fi pasarica si obstacolele, este realizata prin functia `make_bird`. Aceasta functie utilizeaza datele din fisierele `pasarica.inc` si `patrat_red.inc` pentru a desena elementele pe ecran.

```
make_bird proc
    push ebp
    mov ebp, esp
    pusha

```

```
    mov eax, [ebp+arg1] ; citim simbolul de afisat
    lea esi, pasarica
    jmp draw_block

make_empty:
    mov eax, 0 ; pe pozitia 0 e gol
    lea esi, pasarica

draw_block:
    mov ebx, block_length
    mul ebx
    mov ebx, block_length
    mul ebx
    add esi, eax
    mov ecx, block_length
bucla_block_linii:
    mov edi, [ebp+arg2] ; pointer la matricea de pixeli
    mov eax, [ebp+arg4] ; pointer la coord y
    add eax, block_length
    sub eax, ecx
    mov ebx, area_width
    mul ebx
    add eax, [ebp+arg3] ; pointer la coord x
    shl eax, 2 ; inmultim cu 4, avem un DWORD per pixel
    add edi, eax
    push ecx
    mov ecx, block_length
bucla_block_coloane:
    cmp byte ptr [esi], 0 ;verifica daca e pixel fundal
    je block_pixel_fundal
    cmp byte ptr [esi], 2 ;verifica daca e pixel maro
    je block_pixel_verde
    cmp byte ptr [esi], 3 ;verifica daca e pixel albastru
    je block_pixel_maro
    cmp byte ptr [esi], 4 ;verifica daca e pixel rosu
    je block_pixel_stalp
    cmp byte ptr [esi], 5 ;verifica daca e pixel gold
    je block_pixel_pietricele
    mov dword ptr [edi], 0FFBF00h
    jmp block_pixel_next
block_pixel_fundal:
    mov dword ptr [edi], 077BCE9h
    jmp block_pixel_next
block_pixel_verde:
    mov dword ptr [edi], 0529C4Ah
    jmp block_pixel_next
block_pixel_maro:
    mov dword ptr [edi], 0800020h
    jmp block_pixel_next
block_pixel_stalp:
    mov dword ptr [edi], 036454Fh
    jmp block_pixel_next
block_pixel_pietricele:
    mov dword ptr [edi], 0E97451h
    jmp block_pixel_next
block_pixel_next:
    inc esi
```

```
    add edi, 4
    loop bucla_block_coloane
    pop ecx
    loop bucla_block_linii
    popa
    mov esp, ebp
    pop ebp
    ret
make_bird endp
```

3.3 Logica Jocului

Logica jocului este gestionata in functia `draw`, care este apelata in mod repetat pentru a actualiza starea jocului. Aceasta functie gestioneaza evenimentele de timer si click, actualizeaza pozitia pasarii si verifica coliziunile.

```
draw proc
    push ebp
    mov ebp, esp
    pusha

    mov eax, [ebp+arg1]
    cmp eax, 1
    jz evt_click
    cmp eax, 2
    jz evt_timer ; nu s-a efectuat click pe nimic

    ; Mai jos e codul care initializeaza fereastra cu pixeli albi
    mov eax, area_width
    mov ebx, area_height
    mul ebx
    shl eax, 2
    push eax
    push 0
    push area
    call memset
    add esp, 12

    jmp afisare_litere

evt_click:
    cmp game_over, 1
    je final_draw

    ; Logica pentru click
    ; ...

evt_timer:
    cmp game_over, 1
    je final_draw

    ; Logica pentru timer
    ; ...

afisare_litere:
    ; Afisarea scorului si a mesajelor
```

```
41      ; ...
42
43  final_draw:
44      popa
45      mov esp, ebp
46      pop ebp
47      ret
48  draw endp
```

4 Concluzie

In concluje am aratat cum poate fi implementat un joc simplu folosind limbajul de asamblare. Prin utilizarea unor rutine de desenare si gestionarea evenimentelor, jocul ofera o experienta interactiva si distractiva. Documentatia de mai sus ofera o privire de ansamblu asupra structurii si implementarii proiectului, cu exemple de cod pentru a ilustra functionalitatile principale.