EASY DISK

Alunos:

Renato Vinicius Gomes Campos

Paulo Roberto Ribeiro Morais

Professor:

Bruno Otavio Piedade Prado

Disciplina:

Interface Hardware-Software

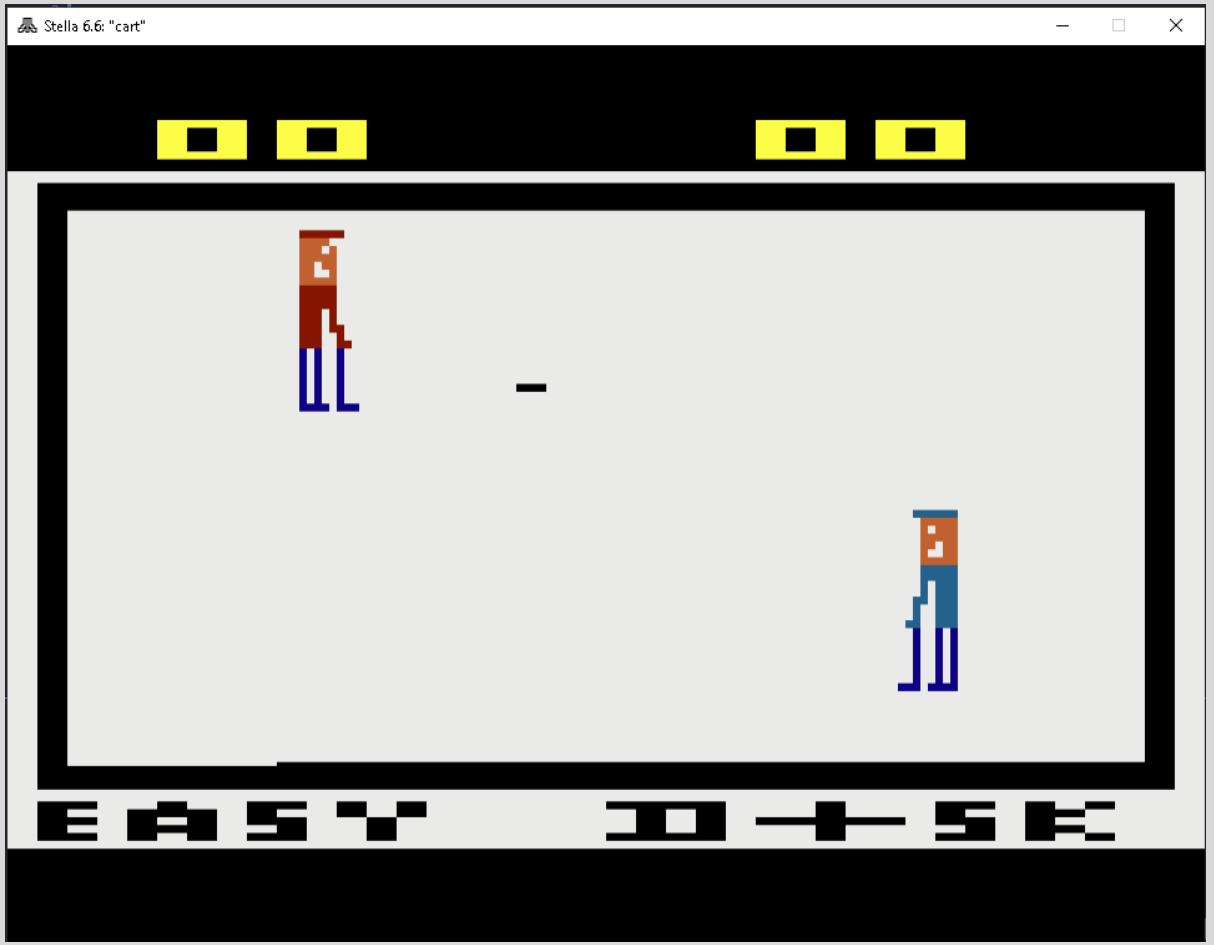
Elementos Atari

Playfield

Forma a arena, o placar e o nome

Players

Jogadores. Têm registradores especiais



Divisão da Memória

TIA:

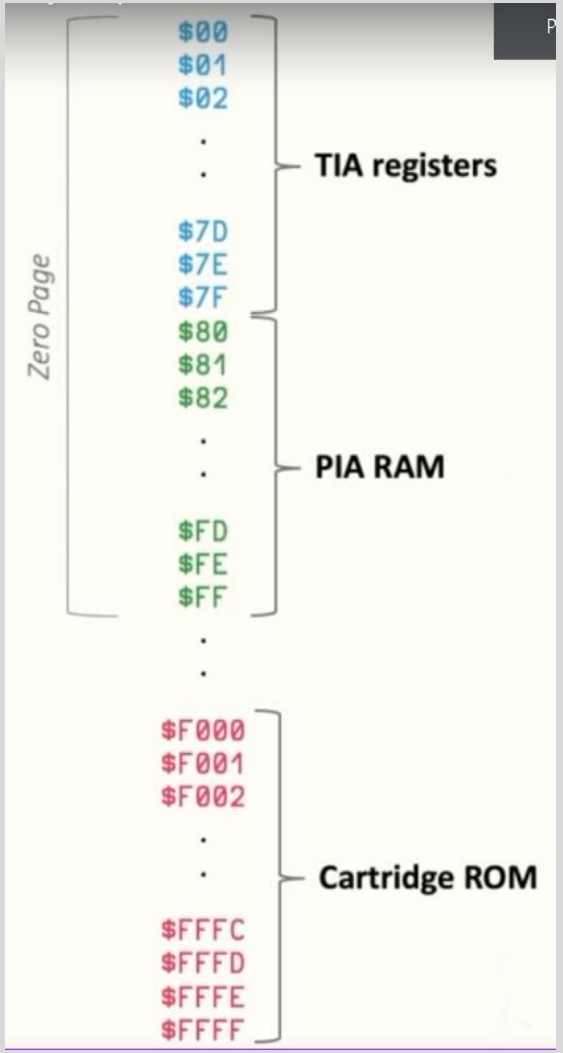
Cuida do vídeo e do som

PIA:

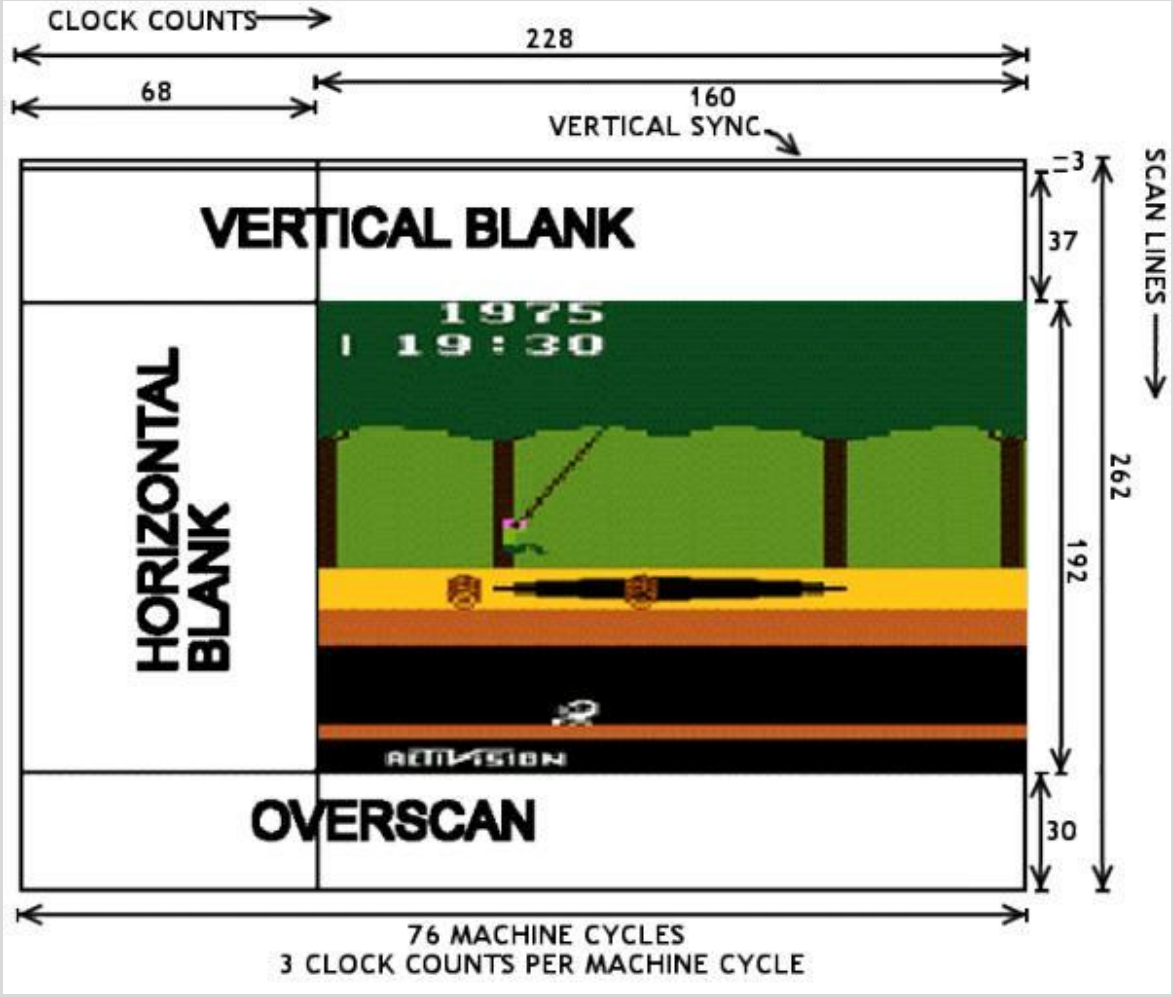
RAM e controles (input)

ROM:

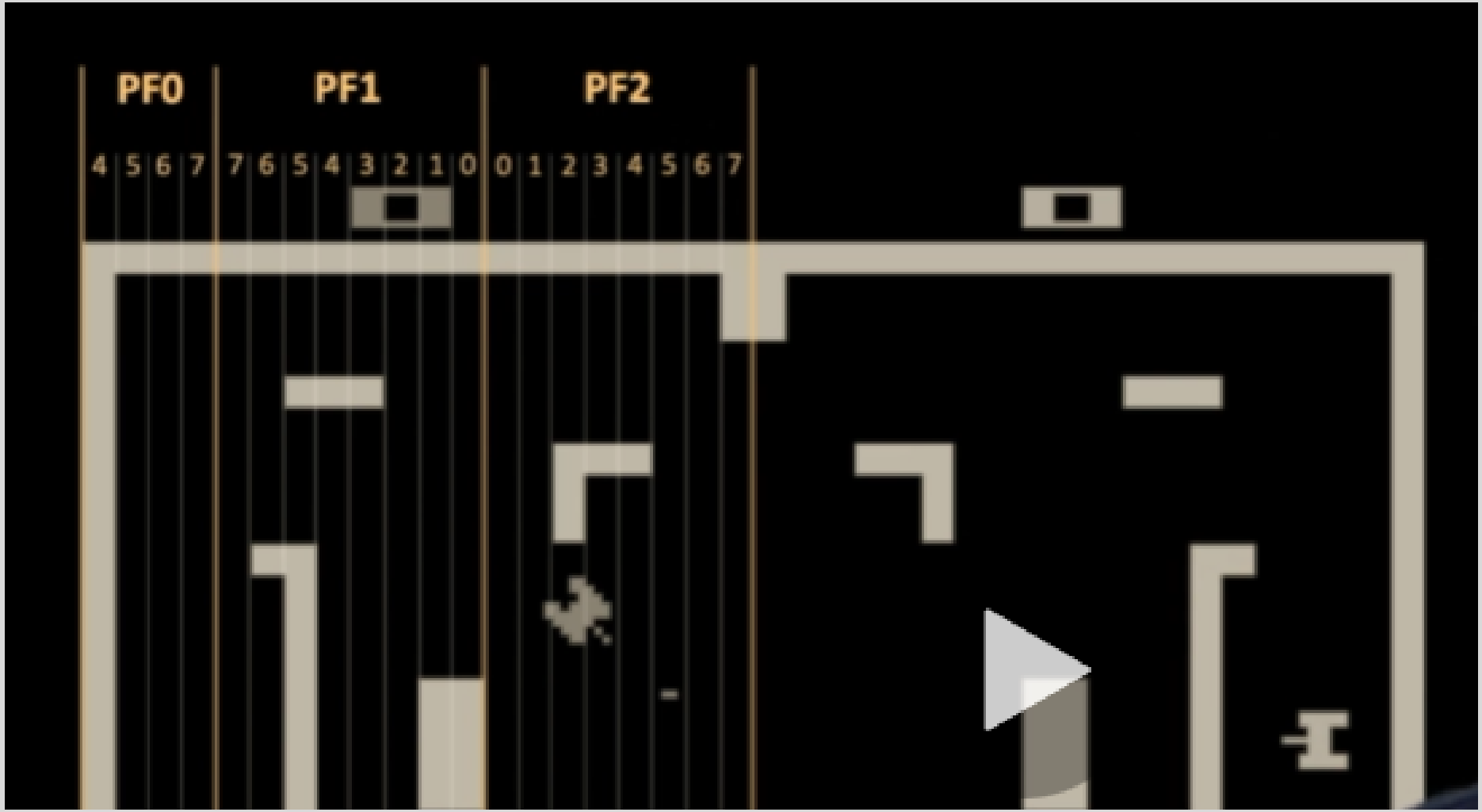
O jogo propriamente dito



Scanlines


CLOCK COUNTS
228
68
160
VERTICAL SYNC
VERTICAL BLANK
HORIZONTAL BLANK
OVERSCAN
3
37
192
30
262
SCAN LINES
76 MACHINE CYCLES
3 CLOCK COUNTS PER MACHINE CYCLE

Playfield



Playfield - Placar

Placar:

Para fazer o placar não basta Apenas setar os valores de PF0, PF1 e PF2 para uma determinada linha, mas sim, mudar o seus valores antes do feixe de elétrons chegar ao placar da esquerda

Playfield - Nome

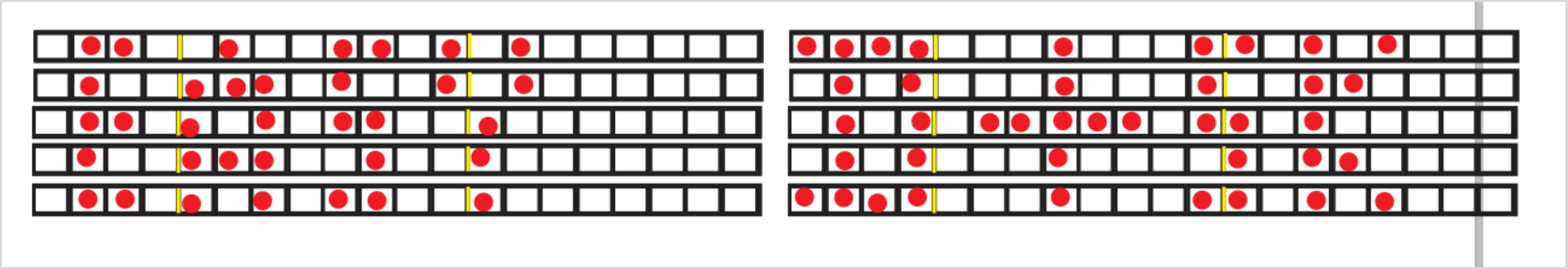
Nome do Jogo:

O mesmo foi feito para o nome do jogo, afinal, os bits setados na esquerda têm que ser diferente dos bits da direita. Ou seja, contar clocks de cada instrução.

Playfield - Nome

Criação:

Para facilitar setar os bits corretamente, foi criado um "molde" no Corel Draw



PF0 PF1 PF2 PF0 PF1 PF2

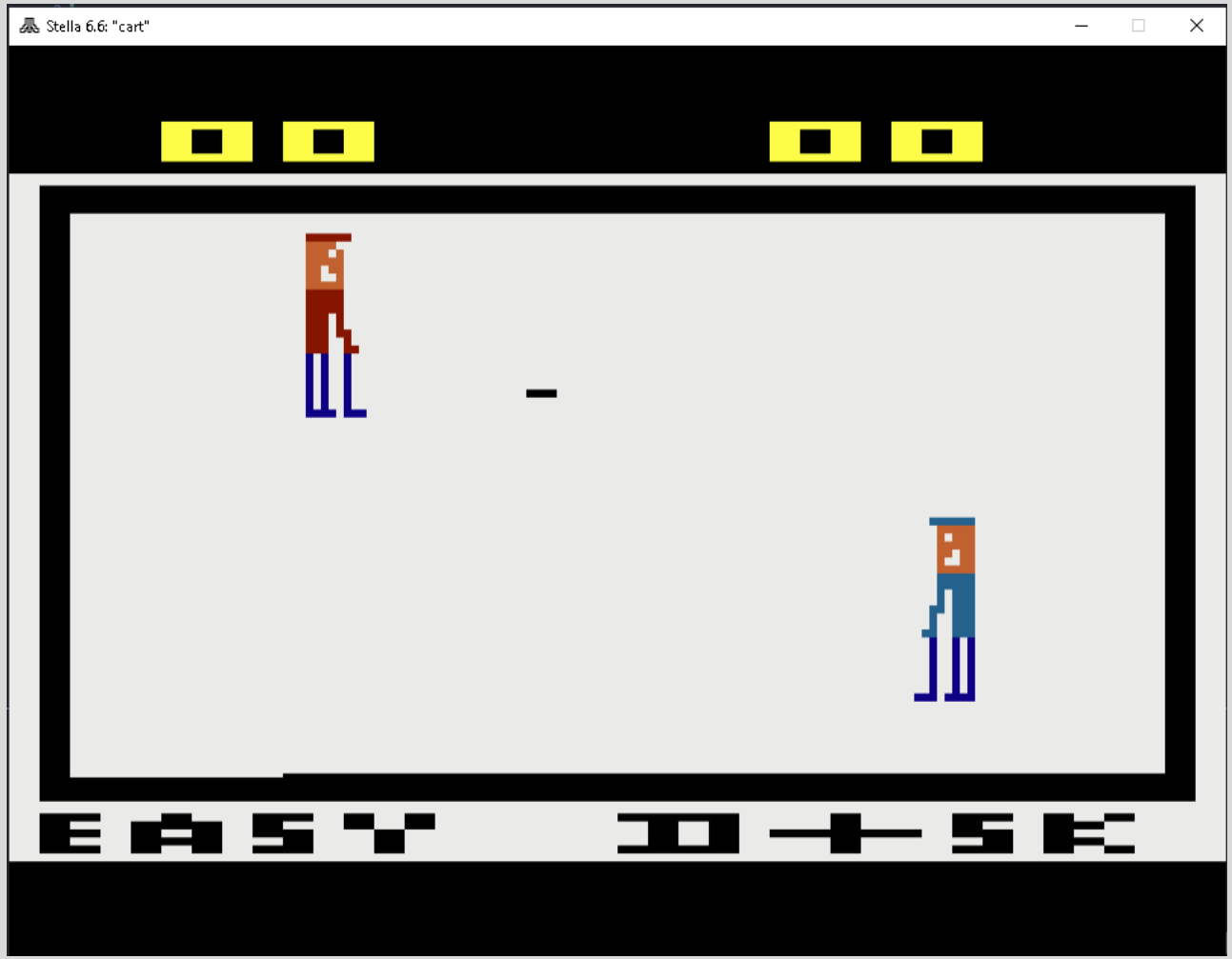
ATARI - Fácil

```
lda #$40
Sta COLUBK

lda #$40
Sta COLUPF
```

```
lda #$40
Sta COLUP1

lda #%00010000
bit SWCHA
```

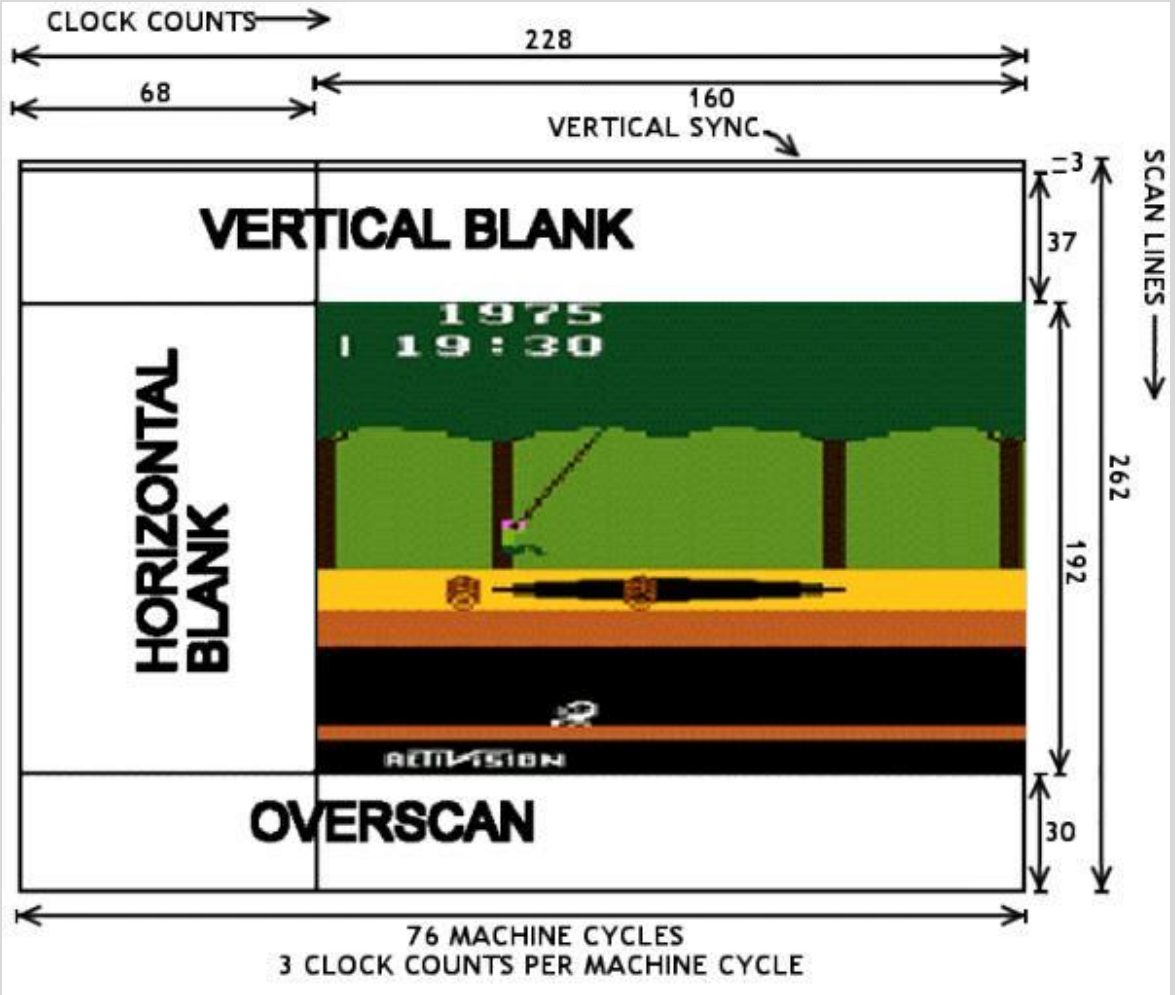
```
lda #%01000000
bit CXP0FB

lda #0
sta AUDV0
```

Posicionar Verticalmente



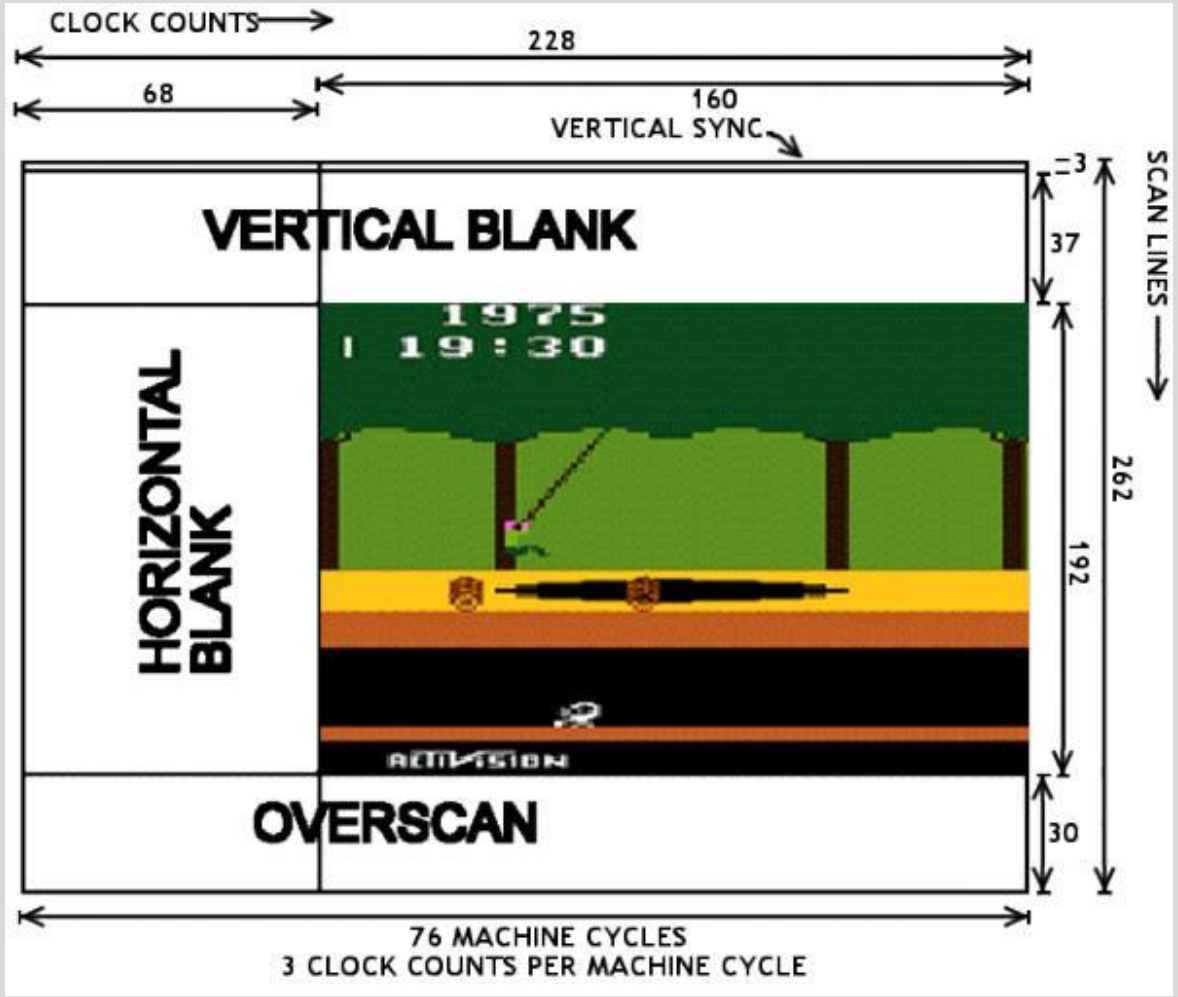
Posicionar Horizontalmente

```
    ldx #5          ; X = 5

    sta WSYNC       ; wait for scanline to start

Loop:
    dex             ; X--
    bne Loop        ; loop 5 times

    sta RESP0       ; fix player0 horizontal position
```



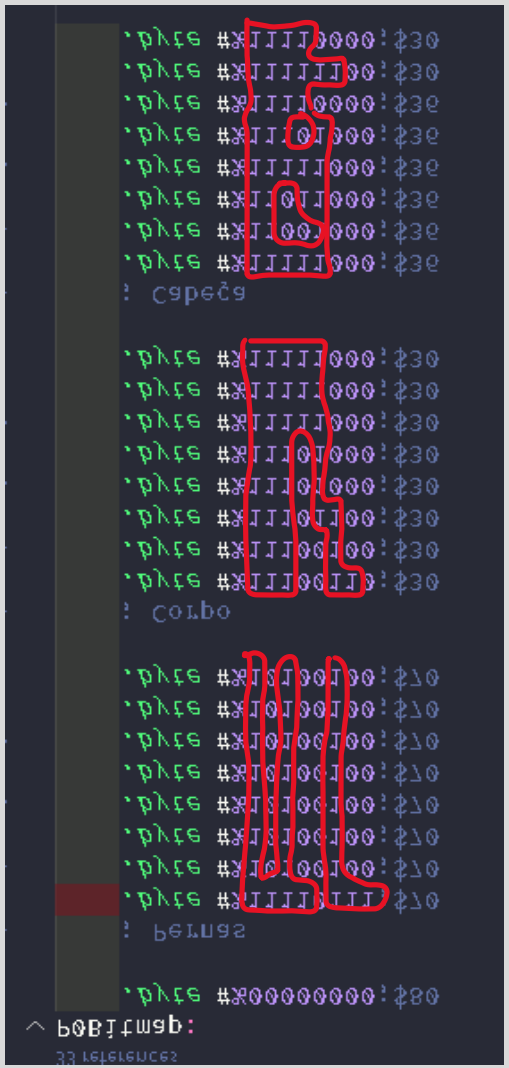
Posicionar Horizontalmente

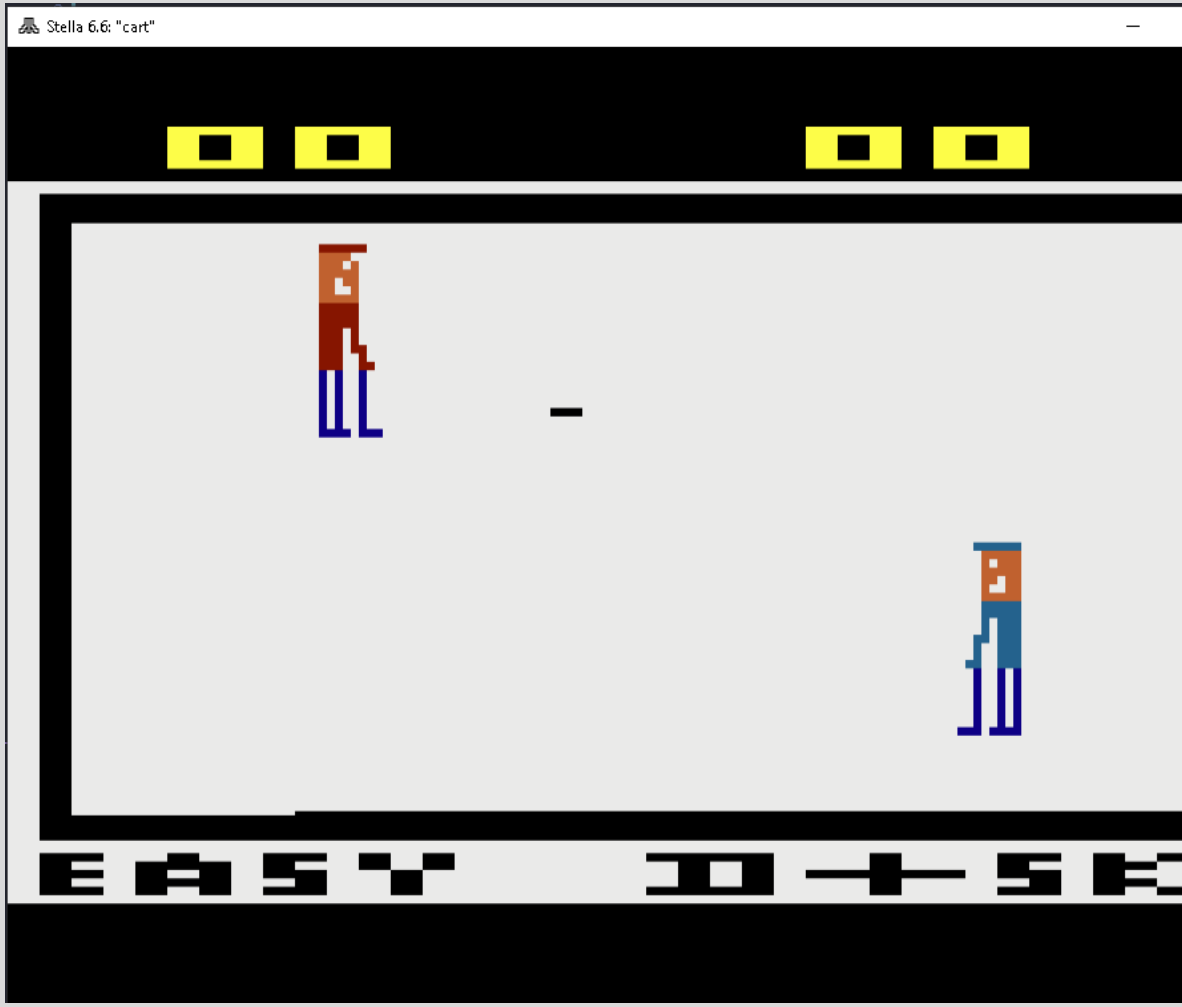
```
SetObjectXPos subroutine
    sta WSYNC
    sec
.Div15Loop
    sbc #15
    bcs .Div15Loop
    eor #7
    asl
    asl
    asl
    asl
    sta HMP0,Y
    sta RESP0,Y
    rts
```



Lógica - Sprite

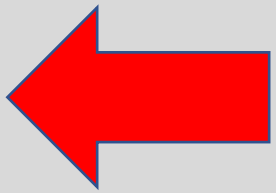
```
33 references
P0Bitmap:
    .byte #%00000000;$80

    ; Pernas
    .byte #%11110111;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70

    ; Corpo
    .byte #%11100110;$30
    .byte #%11100100;$30
    .byte #%11101100;$30
    .byte #%11101000;$30
    .byte #%11101000;$30
    .byte #%11111000;$30
    .byte #%11111000;$30
    .byte #%11111000;$30

    ; Cabeça
    .byte #%11111000;$36
    .byte #%11001000;$36
    .byte #%11011000;$36
    .byte #%11111000;$36
    .byte #%11101000;$36
    .byte #%11110000;$36
    .byte #%11111100;$30
    .byte #%11110000;$30
```





Lógica - Sprite

```
.AreWeInsideP0Sprite:
    txa
    sec
    sbc P0YPos
    cmp PLAYER0_HEIGHT
    bcc .DesenhaSpriteP0
    lda #0
.DesenhaSpriteP0:
    clc
    adc CharAnimOffset0
    tay
    lda P0Bitmap,Y
    sta WSYNC
    sta GRP0
    lda P0Color,Y
    sta COLUP0
```

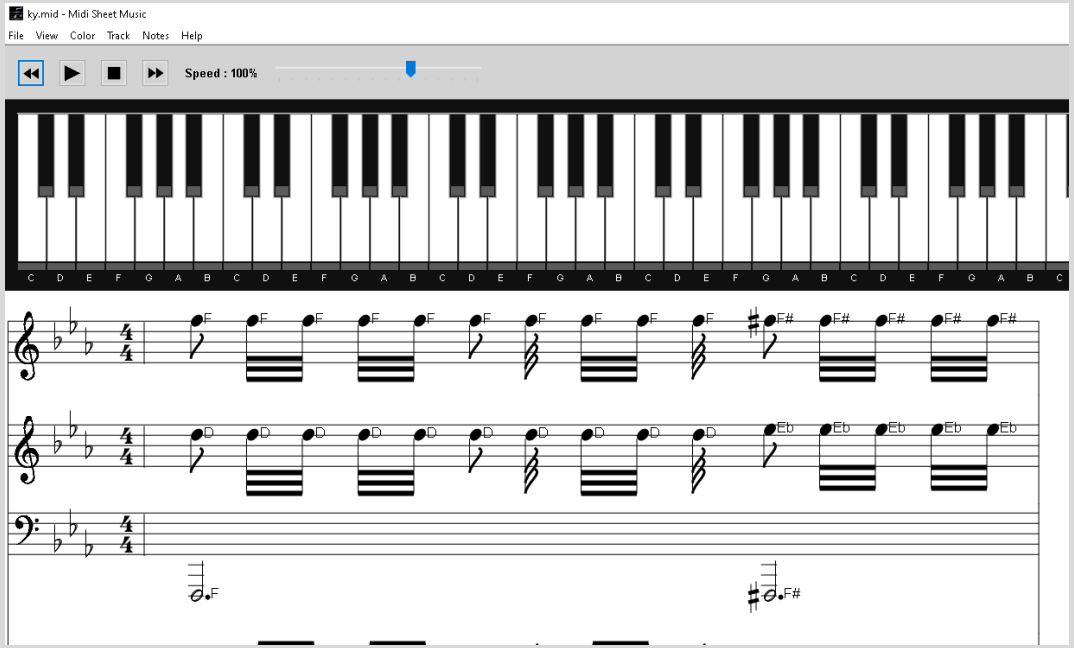


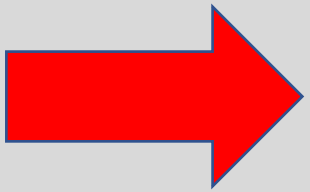
```
33 references
P0Bitmap:
    .byte #%00000000;$80

    ; Pernas
    .byte #%11110111;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70
    .byte #%10100100;$70

    ; Corpo
    .byte #%11100110;$30
    .byte #%11100100;$30
    .byte #%11101100;$30
    .byte #%11101000;$30
    .byte #%11101000;$30
    .byte #%11111000;$30
    .byte #%11111000;$30
    .byte #%11111000;$30

    ; Cabeça
    .byte #%11111000;$36
    .byte #%11001000;$36
    .byte #%11011000;$36
    .byte #%11111000;$36
    .byte #%11101000;$36
    .byte #%11110000;$36
    .byte #%11111100;$30
    .byte #%11110000;$30
```

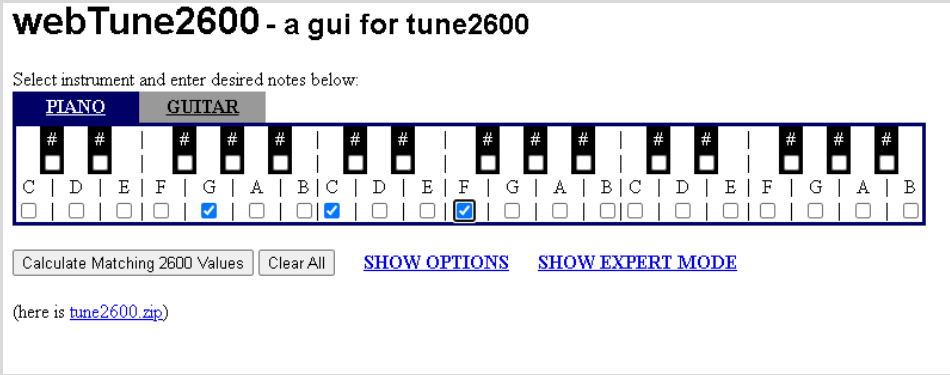
```
33 references
P0Color:
    .byte #$00;

    ; Cor das pernas
    .byte #$70;
    .byte #$70;
    .byte #$70;
    .byte #$70;
    .byte #$70;
    .byte #$70;
    .byte #$70;
    .byte #$70;

    ; Cor do corpo
    .byte #$30;
    .byte #$30;
    .byte #$30;
    .byte #$30;
    .byte #$30;
    .byte #$30;
    .byte #$30;
    .byte #$30;

    ; Cor da cabeça
    .byte #$36;
    .byte #$36;
    .byte #$36;
    .byte #$36;
    .byte #$36;
    .byte #$36;
    .byte #$30;
    .byte #$30;
```

Som

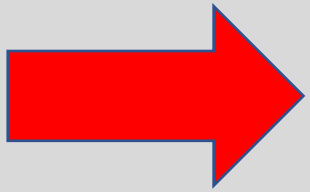
```
GenerateSoundGol subroutine
    lda #3
    sta AUDV0                   ; Volume
    lda #13
    sta AUDC0                   ; Tipo do som
    lda #4
    sta AUDF0                   ; Frenquencia
    rts
```







Som



```
SFX_F
    .byte#00
    .byte#00
    .byte#00
    .byte#23
    .byte#23
    .byte#23
```

Problema:

Arquivo muito pequeno. Som quase não sai

Som

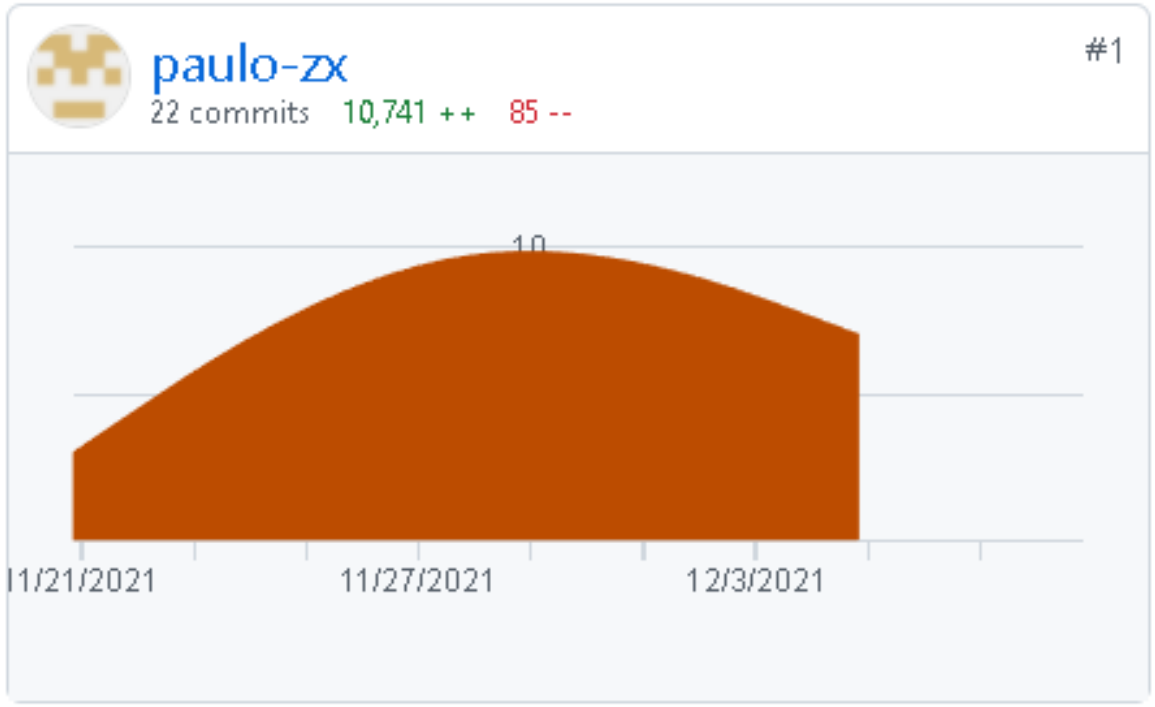
Solução:

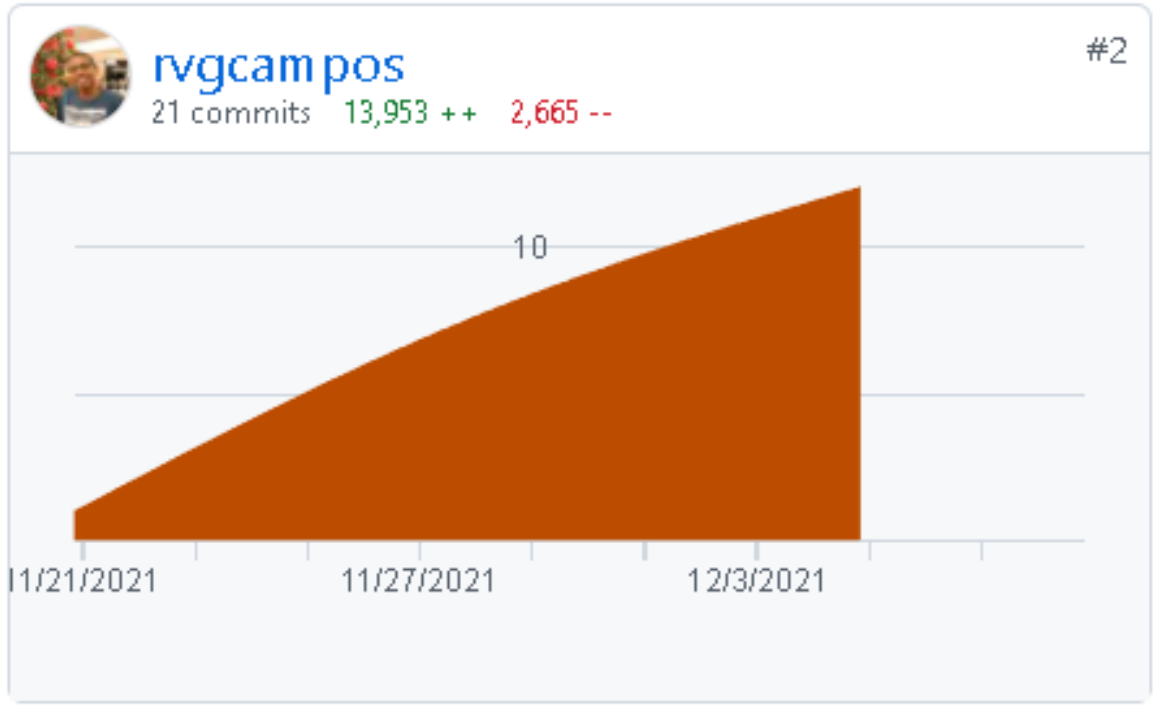
Fazer um programa em python pra ler um arquivo .midi, obter as notas musicas, e fazer o mapeamento dessas notas para os seus respectivos números compatíveis com o ATARI. Gerando por fim um arquivo para ser incluído no código do jogo.

Som

```python
import music21
parsed = music21.converter.parse('doom.mid')
lista = []

for p in parsed.pitches:
    lista.append(p.name)

mapa = {
    'A': '6',
    'B': '31',
    'C': '29',
    'D': '26',
    'E': '23',
    'F': '22',
    'G': '3'
}

atari = []
for l in lista:
    for k, v in mapa.items():
        if k == l[0]:
            for _ in range(12):
                atari.append(v)


arq = open('sfx_f.txt', 'w')
arq.write('SFX_F\n')
contador = 0
for a in atari:
    arq.write(f'    .byte #{a}\n')
    contador = contador+1
```

Contribuição





OBRIGADO