UNIVERSIDADE FEDERAL DE VIÇOSA · UFV - CAMPUS FLORESTAL
CIÊNCIA DA COMPUTAÇÃO
INTRODUÇÃO A JOGOS E GAMIFICAÇÃO

Projeto II

Alan Gabriel Martins Silva - 4663
Miguel Antônio Ribeiro e Silva - 4680
Matheus Nascimento Peixoto - 4662
Luiz César Galvão Lima - 4216

Florestal - MG
2025

1. INTRODUÇÃO

Esta documentação tem como objetivo apresentar as tarefas realizadas do Projeto II , da disciplina de Introdução a Jogos e Gamificação. Foram realizadas as seguintes atividades envolvendo o jogo Combat (1977) desenvolvido para o Atari 2600:

1. Alterações “cosméticas” em um ou mais jogos existentes, a partir dos seus códigos-fonte em assembly language.
2. Alterações de jogabilidade em um ou mais jogos existentes, a partir dos seus códigos-fonte em assembly language.
3. Criar e alterar um modo de jogo (um “videogame") em um ou mais jogos existentes, a partir dos seus códigos-fonte em assembly language.

A seguir uma breve apresentação do jogo original e como foi coletado seu código fonte seguido dos detalhes técnicos de cada tarefa.

2. COMBAT (1977)

Combat foi um dos nove jogos de lançamento do Atari 2600 e o primeiro a ser criado para o console, nele os jogadores controlam veículos que devem se enfrentar em diversas arenas. Podendo apenas ser jogado com duas pessoas, os jogadores podem escolher **três diferentes** tipos de jogo, cada um com seus veículos, sendo esses tanques, aviões e caças. Dentro de cada modo de jogo, pode-se escolher variações de arena. Além dos diferentes veículos, existem modos de jogo onde o tiro rebate nas paredes, ou até mesmo os veículos ficam invisíveis. No total o jogo possui 27 modos. [1]. O código do jogo foi coletado em [2], possui um *disassembly* original feito por Harry Dodgson, comentado por Nick Bensema (1997) e revisto por Roger Williams (2002).

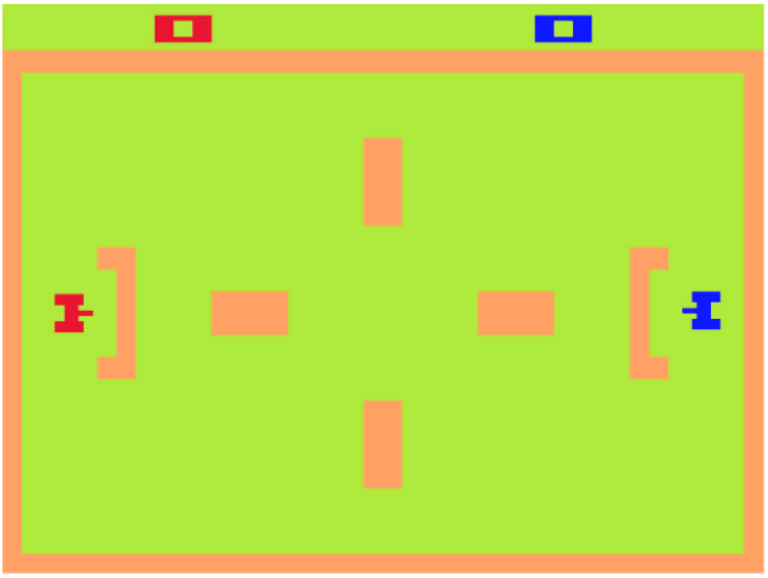

Imagem 01 - Combat (1977)

3. TAREFA I - DANICombat

Para essa tarefa de alterações cosméticas foi feito primeiramente uma alteração nas cores dos sprites dos tanques, do *playfield* e do *background*.

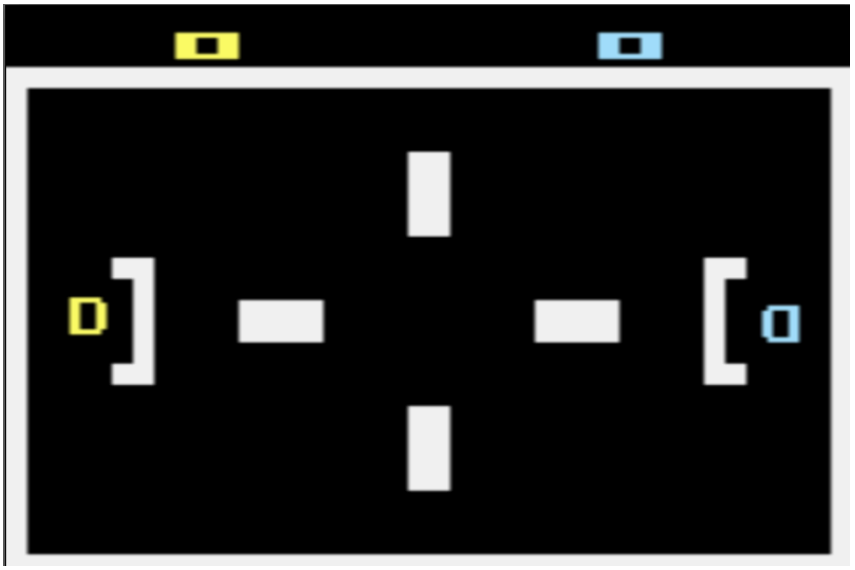

Imagem 02 - Cores diferentes

A tabela de cores a seguir, Imagem 03, mostra exatamente todas cores usadas e modificadas no jogo. A primeira linha representa as cores do modo padrão de tanques, a segunda linha os modos *tank pong*, a terceira com os jatos e a quarta com os biplanos. A sequência de cores é *background*, *playfield* (obstáculos e paredes), o *player* 2 e o *player* 1. As cores são preto, branco, amarelo e azul claro, definidas através da Imagem 04, a tabela de cores do TIA [3].

```
ColorTbl
        byte $00 ,$0f ,$AE ,$1F      ; 00 = Regular Tanks
       .byte $00 ,$0f ,$AE ,$1F      ; 01 = Tank Pong
       .byte $00 ,$0f ,$AE ,$1F      ; 10 = Jets
       .byte $0f ,$0f ,$AE ,$1F      ; 11 = Biplanes
       .byte $00 ,$0f ,$AE ,$1F      ; special B&W
```

Imagem 03 - Tabela de cores

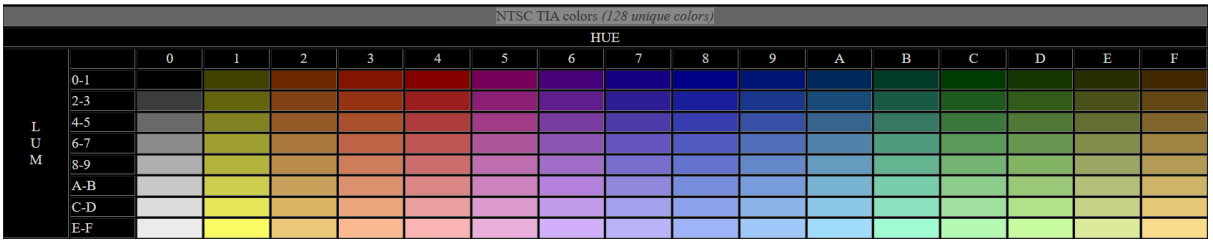

Imagem 04 - NTSC TIA colors (128 cores únicas)

Após a definição das cores, os sprites dos tanques foram atualizados. Agora, os tanques não são mais representados como tanques, mas sim como "Daniel". O conceito de "Daniel" foi criado com base nos 8 lados possíveis para o movimento do tanque, atribuindo a cada lado uma letra da palavra "DANIELMB", veja na Imagem 05:

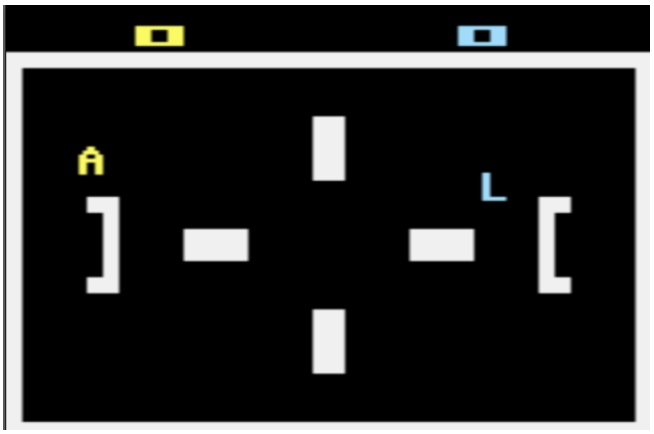

Imagem 05 - Tanque A e Tanque L

O *player* 1 virou a letra A pois está apontado para a *diagonal superior direita* e o *player* 2 virou o L pois está virado para a *diagonal superior esquerda*. Cada lado é uma letra da palavra "DANIELMB". Veja um exemplo na Imagem 06, a modificação de 3 lados. Para essa tarefa utilizamos o [4] - um editor de *sprites* 8-bit e [5] - *ChatGPT*.

```
TankShape
        ; Letra D
        .byte $7E ; | XXXXXX |
        .byte $63 ; | XX   XX|
        .byte $63 ; | XX   XX|
        .byte $63 ; | XX   XX|
        .byte $63 ; | XX   XX|
        .byte $63 ; | XX   XX|
        .byte $7E ; | XXXXXX |
        .byte $00 ; |        |

        ; Letra A
        .byte $18 ; |   XX   |
        .byte $3C ; |  XXXX  |
        .byte $66 ; | XX  XX |
        .byte $7E ; | XXXXXX |
        .byte $66 ; | XX  XX |
        .byte $66 ; | XX  XX |
        .byte $66 ; | XX  XX |
        .byte $00 ; |        |

        ; Letra N
        .byte $63 ; | XX   XX|
        .byte $73 ; | XXX  XX|
        .byte $7B ; | XXXX XX|
        .byte $6F ; | XX XXXX|
        .byte $67 ; | XX  XXX|
        .byte $63 ; | XX   XX|
        .byte $63 ; | XX   XX|
        .byte $00 ; |        |
```

Imagem 06 - Formas do tanque

4. TAREFA II - TIKTOKombat

Para as novas gerações, acostumadas à busca constante por dopamina barata e aceleração, criamos um modo pensado especialmente para elas. Nesta tarefa, o ultrapassado Combat (1977) recebe uma atualização ultramoderna em sua jogabilidade: os tanques são mais rápidos, quase não possui *bumpings*, e os obstáculos são mais amigáveis, contando com

diversos atalhos. Um verdadeiro jogo de ação para quem não suportava a jogabilidade travada do Combat (1977). Assim, nasceu o novo jogo: TIKTOKombat.

Começando pela **velocidade**, a Imagem 07 demonstra a tabela de velocidade dos *players*. Essa tabela modifica a velocidade dos tanques começando pela **Velocidade Nula**: $00, $00: Indica que o tanque está parado (sem movimento). **Velocidade Máxima**: $FF, $FF: É a velocidade mais alta possível no jogo. Em sistemas antigos como o Atari 2600, o valor $FF (255 em decimal) representa o maior valor para um *byte*, o que resulta em um deslocamento rápido. Os outros *bytes* desaceleram o tanque. Para essa etapa tivemos auxílio de [5], mas somente para a desaceleração do tanque. Agora os tanques "voam".

```
MVtable .BYTE   $00 ,$00
        .BYTE   $FF ,$FF  ; Movimento mais rápido (máximo)
        .BYTE   $F0 ,$F0  ; Velocidade alta
        .BYTE   $E8 ,$E8  ; Velocidade média-alta
        .BYTE   $E0 ,$E0  ; Velocidade média
        .BYTE   $D8 ,$D8  ; Velocidade média-baixa
        .BYTE   $D0 ,$D0  ; Velocidade baixa
```

Imagem 07 - Tabela de velocidade

A seguir a forma de *bumpings* foi modificada. Cada vez que você bate em um obstáculo, o jogador é levemente empurrado para trás, modificamos a tabela HDGTBL, para que isso não aconteça e o *bumping* seja… "diferente". A Imagem 08, responsável por controlar esses *bytes*, deixando um jogo mais dinâmico (ou acelerado). Cada valor é um deslocamento em coordenadas horizontais, medido em pixels ou unidades internas do jogo. Os valores vão de $00 (nenhum deslocamento) até $FF (máximo deslocamento possível), indicando a intensidade ou amplitude do movimento. [5] nos ajudou novamente.

```
HDGTBL  .BYTE   $F0 ,$F0 ,$E8 ,$E8    ; Deslocamentos maiores para "super bumping"
        .BYTE   $00 ,$60 ,$80 ,$90    ; Ampliação nos deslocamentos
        .BYTE   $A0 ,$B0 ,$C0 ,$D0    ; Empurrão mais forte
        .BYTE   $E0 ,$F0 ,$FF ,$FF    ; Máximo deslocamento possível
```

Imagem 08 - Tabela de *bumping*

Pra finalizar o nosso TIKTOKombat, os obstáculos de algumas fases que eram apenas um parede ou um *sprite* de algo genérico foi substituído pela carinha do *creeper*, personagem do

Minecraft [6], além da adição de atalhos nos cantos das fases, deixando mais amigável e dinâmico, os novos sprites foram feitos todos em [4] dessa vez, sem ajuda de [5].

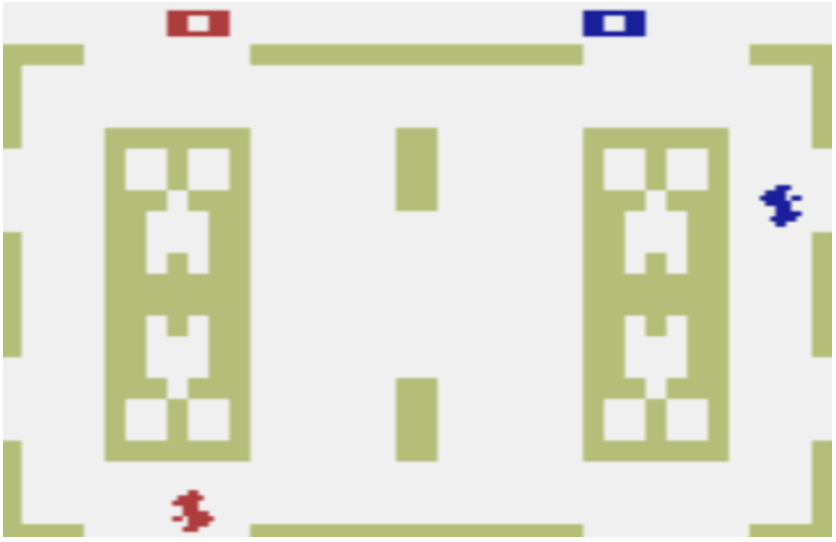

Imagem 10 - TIKTOKombat

5. TAREFA III - C∞mbat e Pega-Pega

Para essa tarefa, modificamos um modo de jogo e "criamos" um outro modo. Começando pelo modo modificado, o Combat original, possui o modo *tank pong* que basicamente é um modo em que o tiro recocheteia. Nesse modo o tiro rebate por aproximadamente 2 segundos nos diferentes obstáculos e paredes. No **C∞mbat** o tiro rebate **infinitamente**, esse modo agora se tornou um modo em que você deve fugir do tiro de seu oponente acima de qualquer custo. Imagem 11 – o bloqueio de expiração quando **MisLife** é negativo, faz com que a expiração do míssil nunca aconteça. Em resumo, pelo que entendemos, agora o míssil permanece ativo indefinidamente porque o valor de **MisLife**, que controla sua duração, é inicializado com **0x3F (63)** toda vez que é lançado, mas não é decrementado adequadamente. O código possui condições que bloqueiam a diminuição de **MisLife**, como a verificação de **CLOCK** para a redução do tempo de vida, que ocorre apenas **3/4** do tempo. Verificações de dificuldades e outras ***flags*** impedem o decremento, resultando em um míssil que nunca expira e continua em voo. Essa atividade foi a mais complicada, tivemos a ideia, mas demoramos muito pra achar onde isso era controlado no código, [5] não conseguiu ajudar muito bem e apenas através de testes modificando valores e *flags*, o modo **C∞mbat** foi feito.

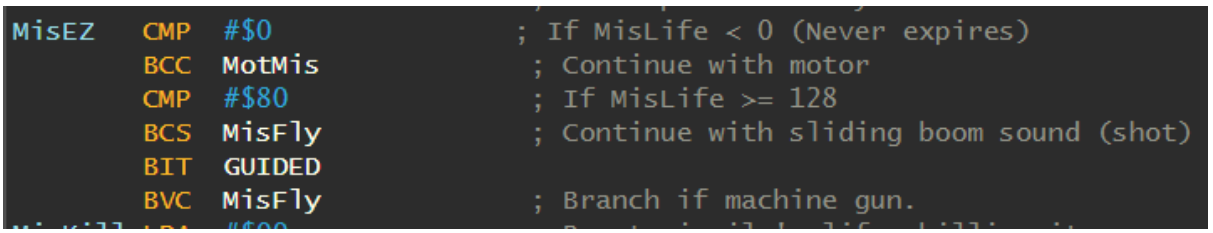

Imagem 11 - Míssil ∞

Por fim, tentamos criar um novo modo de jogo, porém estava complexo de elaborar sua lógica. Primeiramente criamos o mapeamento na tabela de modos de jogo, Imagem 12, mas não elaboramos nenhuma lógica, mas… Caso você selecione o "videogame" 28, o jogo funciona como uma espécie de pega-pega. Nesse modo, há dois jatos, e para marcar pontos, você precisa encostar a parte da frente do seu jato no jato inimigo e pressionar a barra de espaço. Isso simula o comportamento de atirar um míssil, mas sem atirar. Curiosamente, ao inserir o “BYTE $D0”, apenas isso, na tabela VARMAP, o jogo no estilo pega-pega é ativado, funcionando perfeitamente. Dessa forma, podemos considerar que “criamos” um novo modo de jogo, Imagem 13, por coincidência. Por fim, diminuímos o volume dos motores através da Imagem 09.

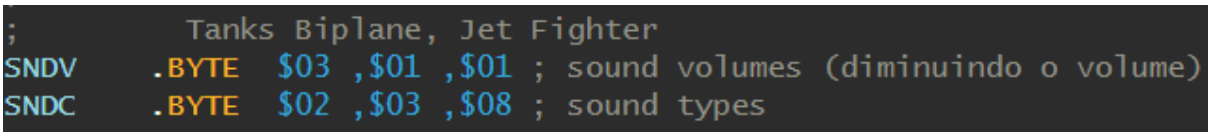

Imagem 09 - SNDV mais baixo

```
VARMAP  .BYTE  $24 ;Game 1:  0010 0100  TANK
        .BYTE  $28 ;Game 2:  0010 1000
        .BYTE  $08 ;Game 3:  0000 1000
        .BYTE  $20 ;Game 4:  0010 0000
        .BYTE  $00 ;Game 5:  0000 0000
        .BYTE  $48 ;Game 6:  0100 1000  TANK PONG
        .BYTE  $40 ;Game 7:  0100 0000
        .BYTE  $54 ;Game 8:  0101 0100
        .BYTE  $58 ;Game 9:  0101 1000
        .BYTE  $25 ;Game 10: 0010 0101  INVISIBLE TANK
        .BYTE  $29 ;Game 11: 0010 1001
        .BYTE  $49 ;Game 12: 0100 1001  INVISIBLE TANK-PONG
        .BYTE  $55 ;Game 13: 0101 0101
        .BYTE  $59 ;Game 14: 0101 1001
        .BYTE  $A8 ;Game 15: 1010 1000  BIPLANE
        .BYTE  $88 ;Game 16: 1000 1000
        .BYTE  $98 ;Game 17: 1001 1000
        .BYTE  $90 ;Game 18: 1001 0000
        .BYTE  $A1 ;Game 19: 1010 0001
        .BYTE  $83 ;Game 20: 1000 0011
        .BYTE  $E8 ;Game 21: 1110 1000  JET FIGHTER
        .BYTE  $C8 ;Game 22: 1100 1000
        .BYTE  $E0 ;Game 23: 1110 0000
        .BYTE  $C0 ;Game 24: 1100 0000
        .BYTE  $E9 ;Game 25: 1110 1001
        .BYTE  $E2 ;Game 26: 1110 0010
        .BYTE  $C1 ;Game 27: 1100 0001
        .BYTE  $D0 ; Game 28: 1101 0000 - Novo Jogo
```

Imagem 12 - Tabela de modos de jogo

Imagem 13 - “videogame” 28

6. REFERÊNCIAS

[1] CADARI, L. Combat, Atari Jogos online. Disponível em: <https://www.atari2600.com.br/Atari/Roms/016S/Combat>. Acesso em: 21 jan. 2025.

[2] JOHNIDM. GitHub - johnidm/asm-atari-2600: Sample source code games Atari 2600. Disponível em: <https://github.com/johnidm/asm-atari-2600>. Acesso em: 21 jan. 2025.

[3] TIA Color Charts. Disponível em: <https://www.qotile.net/minidig/docs/tia_color.html>. Acesso em: 21 jan. 2025.

[4] Tiny 8-Bit Sprite Editor. Disponível em: <https://www.masswerk.at/vcs-tools/TinySpriteEditor/>. Acesso em: 21 jan. 2025.

[5] OPENAI. ChatGPT. Disponível em: <https://chatgpt.com/>.

[6] TO, C. Creeper (Minecraft). Disponível em: <https://dynami-battles.fandom.com/pt-br/wiki/Creeper_(Minecraft)>. Acesso em: 21 jan. 2025.