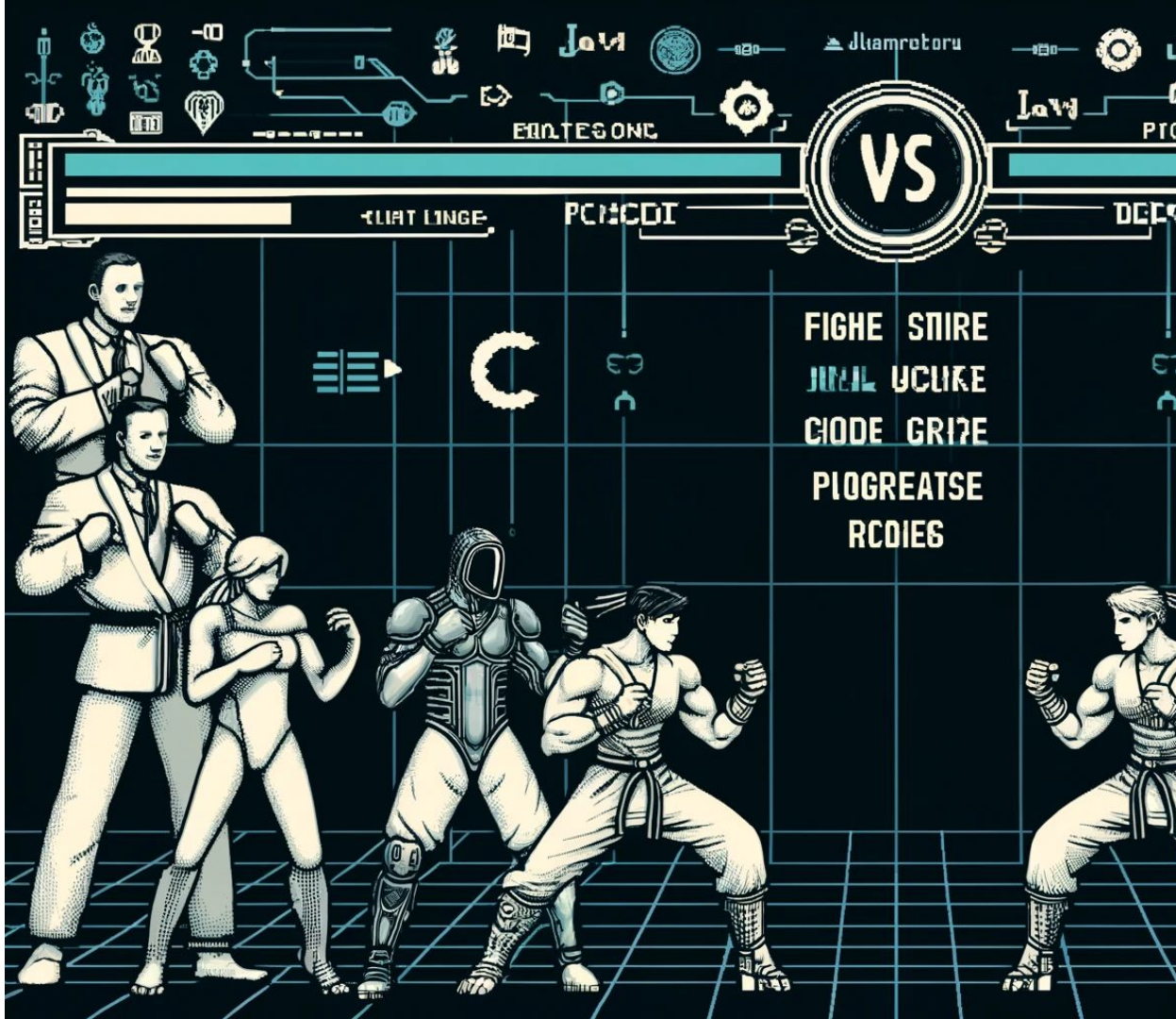
FightLang

APS - Lógica da Computação

Rafael Estefam Melhado Araújo Lima



Explicação básica sobre jogos de luta

Para entender essa linguagem de programação, é necessário entender alguns conceitos básicos sobre a maioria dos jogos de luta modernos. Neles, o objetivo do jogador é derrotar o inimigo, levando a vida dele à 0. O principal jeito de se fazer isso é atacando diretamente o oponente.

Quando um jogador usa um ataque, ele fica sem poder fazer qualquer outra ação, até que o personagem dele se recupera do ataque que ele usou.

Já o outro personagem, se bloqueia o ataque, ficará preso em uma animação de bloquear, cujo a duração depende do ataque usado, não podendo agir. O mesmo é verdade para se ele levar o ataque, sendo que normalmente, a duração da animação é maior do que quando o jogador bloqueia, deixando assim o jogador que está se defendendo vulnerável.

Explicação básica sobre jogos de luta

Muitas vezes então, a tomada de decisão depende de qual jogador se recupera primeiro. Nos jogos de luta, isso é comumente referido como “Vantagem”, ou “Plus”.

Para determinar isso, normalmente os jogadores contam manualmente quantos frames cada ataque demora para o jogador que usou se recuperar, e quantos frames o jogador que bloqueou ou tomou o ataque demorou para se recuperar.

Com tudo isso em mente, fica claro ver que os jogadores estão constantemente testando o conhecimento que o inimigo tem do próprio personagem, tentando sempre se colocar em situações vantajosas que pressionam o jogador adversário a tomar más escolhas. Muitas vezes, quase como um pedra, papel tesoura

- os jogos de luta rodam com uma framerate fixa. Por isso, jogadores de jogos de luta usam frames para contar tempo que ações no jogo demoram.
- Exemplo: O ataque 6P do Slayer ->
no Guilty Gear XX, se bloqueado, faz com que ele se recupere um frame antes do oponente, dando à ele uma oportunidade de reagir.

Motivação

Com tudo isso em mente, fica fácil de perceber que jogos de luta podem ser muito difíceis, tanto para jogadores iniciantes, quando para jogadores avançados. Para engajar nos jogos mentais acima, os jogadores precisam vencer uma árdua barreira de execução, que faz com que seja muito difícil testar soluções ótimas para situações desvantajosas.

Com isso em mente, formulei uma linguagem de programação que permitiria, à partir de uma linguagem similar ao Inglês falado, que faz uso de qualquer uma das notações para movimentos comumente usadas em jogos de luta (recomendado o uso da notação de NUMPAD), permitir a simulação de uma partida de um jogo de luta, incluindo a simulação de estados de cada jogador, a cada frame.

Estrutura no padrão EBNF

- PROGRAM = { LOCALS }
- LOCALS = {"begin", ":", BLOCK, "end" };
- BLOCK = {STATEMENT};
- STATEMENT = ("λ" | ASSIGNMENT | PRINT | WHILE | IF | WAIT | PLAYER_STATEMENT), "\n" ;
- ASSIGNMENT = (IDENTIFIER, "=", EXPRESSION) | ("attack", IDENTIFIER, "=" ,EXPRESSION, ",",EXPRESSION,",",EXPRESSION,",",EXPRESSION,",", EXPRESSION) ;
- PRINT = "PRINT", EXPRESSION ;
- WHILE = "while", ":", BOOL_EXP, "\n", "λ", { (STATEMENT), "λ" }, "end";
- IF = "if", ":", BOOL_EXP, "\n", "λ", { (STATEMENT), "λ" }, ("λ" | ("else", "\n", "λ", { (STATEMENT), "λ" })), "end" ;
- WAIT = "wait", BOOL_EXP;
- PLAYER_STATEMENT = ("PLAYER"|"ENEMY"), (("uses" BOOL_EXP "," BOOL_EXP) | ("hit with " BOOL_EXP "," BOOL_EXP "," BOOL_EXP) | (BLOCKS BOOL_EXP "," BOOL_EXP))
- BOOL_EXP = BOOL_TERM, { ("or"), BOOL_TERM } ;
- BOOL_TERM = REL_EXP, { ("and"), REL_EXP } ;
- REL_EXP = EXPRESSION, { ("==" | ">" | "<"), EXPRESSION } ;
- EXPRESSION = TERM, { ("+" | "-"), TERM } ;
- TERM = FACTOR, { ("*" | "/"), FACTOR } ;
- FACTOR = DELAY | INPUT | IDENTIFIER | PLAYERS | (("+" | "-" | "not"), FACTOR) | "(", EXPRESSION, ")" | "read", "(", ")" ;
- PLAYERS = ("ENEMY" | "PLAYER"), (("in" PLAYERSTATES);
- IDENTIFIER = LETTER, { LETTER | DIGIT | "_" } ;
- TYPE = ("delay" | "damage" | "attack");
- NUMBER = DIGIT, { DIGIT } ;
- DELAY = NUMBER, "f";
- INPUT = {VALID_INPUTS};
- VALID_INPUTS = ("a" | "b" | "c" | "d" | "u"| "d"| "l"|"r"|"LP"|"MP"|"HP"|"LK"|"MK"|"HK"|"← "| "→"| "↑"| "↓"| "P"| "K"|"S"|"HS"|DIGIT | "+"|);
- PLAYERSTATES = (HITSTUN | BLOCKSTUN | GROUNDED | IDLE);
- LETTER = ("a" | "..." | "z" | "A" | "..." | "Z") ;
- DIGIT = ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "0") ;

Características da linguagem

Na linguagem, todo código está contido dentro de sequências. Cada uma dessas sequências descreve uma possível interação entre os dois jogadores

A principal funcionalidade da linguagem é a simulação de ações possíveis em um jogo de luta, como atacar, pular, bloquear, e etc.

Os jogadores podem usar a sua notação preferida para descrever o input de cada ataque

Além disso, a linguagem também simula os estados dos jogadores.
A combinação de tudo isso é que a linguagem facilita o entendimento do estado do jogo, demonstrando também qual jogador sai em vantagem depois de cada interação.

Exemplos

```
begin : COMBO
    first_input_startup = 10f
    first_input = 412K
    enemy_hitstun = 10f
    player_recovery = 9f
    attack_damage = 10

    PLAYER uses first_input, first_input_startup
    ENEMY hit with first_input, enemy_hitstun, attack_damage

    wait player_recovery

    if: player_recovery > enemy_hitstun
        print "player is plus"
    end
end

begin: BLOCKSTRING
    first_input_startup = 8f
    first_input = 6P
    enemy_blockstun = 10f
    player_recovery = 6f
    attack_damage = 10

    PLAYER uses first_input, first_input_startup
    wait player_recovery
    ENEMY blocks first_input, enemy_blockstun

    wait player_recovery

    second_input = 623HS
    second_input_startup = 13f
    second_input_blockstun = 20f

    if: ENEMY in BLOCKSTUN and PLAYER in IDLE
        PLAYER uses second_input, second_input_startup
        ENEMY blocks second_input, second_input_blockstun
    end
end
```

```
1 begin: jump
2     jump_input = 9K
3     jump_duration = 60f
4     PLAYER uses jump_input, jump_duration
5     if: PLAYER in JUMPING
6         print "player jumped!"
7     end
8 end
```

```
begin: INFINITE_COMBO
    i = 0
    attack_input = 623HS
    attack_recovery = 10f
    hitstun = 15f
    attack_damage = 20
    while : i < 5
        PLAYER uses attack_input, attack_recovery
        ENEMY hit with attack_input, hitstun, attack_damage
        wait attack_recovery
        i = i + 1
    end
end
```