Discentes:
Gabriel Queiroz Monteiro Vieira ( ist 198432 )
Yassir Mahomed Yassin ( ist1100611)

## Sumário da estrutura do código

O código é estruturado em 4 subdivisões, sendo elas:

Main: Esta subdivisão é responsável por controlar o fluxo do programa como um todo e detectar eventos. Ao detecta-los, encaminha-os para a respectiva função de tratamento de eventos.

Event Handlers: Esta subdivisão é responsável por processar eventos, encaminhando tarefas específicas para suas respectivas funções.

Sub routines: Esta subdivisão é responsável por executar tarefas específicas, como limpar o terminal ou imprimir o campo de jogo.

Interrupt Service Routines: Esta subdivisão é responsável por tratar interrupções, alterando valores em memória que serão posteriormente detectados em Main.

## Sumário das funções implementadas

CLEAR_FIELD:
Entradas:
R1: Endereço de memória do vetor.
R2: Número de elementos do vetor.
Descrição:
A função definirá todos os elementos do vetor cujo endereço do primeiro elemento está em R1 como zero.

CLEAR_TERMINAL:
Descrição:
A função limpa o conteúdo do terminal.

PRINT_DISP7:
Entradas:
R1: Valor numérico a ser representado.
Descrição:
A função exibirá no Display de 7 segmentos o valor passado em R1.

PRINT_FIELD:
Entradas:
R1: Endereço de memória do vetor.
R2: Número de elementos do vetor.
Descrição:
A função exibirá no terminal o formato human-friendly do campo correspondente ao vetor cujo endereço inicial foi passado em R1.

PRINT_DINO:
Entradas:
R1: Altura da base do dinossauro.
Descrição:
A função exibirá no terminal um dinossauro formado por carácteres ASCII na altura passada em R1.

PRINT_TEXT:
Entradas:

R1: Endereço do primeiro elemento de uma string.
Descrição:
A função exibirá no terminal a string passada em R1.

CHECK_LOST:
Retornos:
1: Caso o jogador tenha perdido.
0: Caso o jogador não tenha perdido.

UPDATE_GAME:
Entradas:
R1: Endereço de memória do vetor.
R2: Número de elementos do vetor.
Descrição:
A função itera pelo vetor, iniciando no segundo elemento e, a cada iteração, copia o valor do elemento atual ( n ) para o endereço do elemento anterior ( n-1 ). Ao completar todas as iterações, a função invoca GEN_CACTUS e usa seu retorno para preencher o último elemento do vetor.

GEN_CACTUS:
Entradas:
R1: Altura máxima do cacto.
Retornos:
A função retorna um valor contido no intervalo [0, <R1>]. Em aproximadamente 95% dos casos, o retorno será 0. Nos outros 5%, os valores estarão homogeneamente distribuídos.

## Justificativa das principais decisões

Durante o desenvolvimento do projeto, diversas decisões relevantes foram tomadas. Dentre elas, destacam-se:

O uso de um temporizador para definir o valor inicial da semente:
A geração aleatória do campo depende de exclusivamente de uma semente. Para garantir que cada jogo seja único, um temporizador é utilizado para medir o tempo que o jogador demora para iniciar o jogo e define este tempo como a semente.

Implementação do salto:
A forma mais eficiente encontrada para implementar o salto foi guardar em memória o estado atual do jogador ( IS_JUMPING e IS_FALLING ). Desta forma, a rotina de interrupção pode ser mantida curta, precisando apenas incrementar o valor de IS_JUMPING. A cada Tick do temporizador, o programa verifica se IS_JUMPING está no valor máximo ou se o dinossauro chegou ao solo e faz as alterações necessárias a essas variáveis.

Impressão do campo de jogo:
O método escolhido para exibir o campo no terminal foi iterar pelo vetor FIELD a partir do último elemento e em direção ao primeiro. A cada elemento, verifica-se seu valor ( X ) e imprime nas X linhas acima da atual coluna do terminal o símbolo ASCII do cacto. Esta implementação minimiza o número de registros utilizado bem como o número de acessos a memória, aumentando assim a eficiência.