UNIVERSIDADE FEDERAL DO CARIRI
CURSO DE CIÊNCIA DA COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MINESWEEPER GAME

PROFESSOR: RAMON SANTOS NEPOMUCENO

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RAISSA KAROLINY DA SILVA RODRIGUES - MAT: 2022002284
PEDRO HENRIQUE BEZERRA SIMEÃO - MAT: 2022003620

JUAZEIRO DO NORTE - CE

2023

FUNÇÃO 1: PLAY

Esta função é responsável por processar a jogada do jogador. Ela recebe as coordenadas (linha e coluna) da célula que o jogador deseja revelar e verifica se há uma bomba nessa célula ou se deve revelar as células adjacentes.

```
Unset
.include "macros.asm"

.globl play

play:
    save_context

    # Recebendo dados e definindo valores de controle
    move $s0, $a2  # $s0 recebe a posição inicial da matriz
    move $s1, $a0  # $s1 recebe o valor de i (linha)
    move $s2, $a1  # $s2 recebe o valor de j (coluna)

     li $t5, SIZE    # Carrega o tamanho da matriz (assumindo que SIZE é uma constante definida em 'macros.asm')
    li $s7, -1     # Define -1 para uso posterior
    li $t8, -2     # Define -2 para uso posterior
```

Nesta parte inicial, estamos salvando o contexto da função e inicializando alguns registradores com valores recebidos como argumentos. $s0 recebe a posição inicial da matriz, $s1 recebe o valor da linha (i), e $s2 recebe o valor da coluna (j). Os registradores $t5, $s7, e $t8 são usados para armazenar o tamanho da matriz, o valor que representa uma célula com bomba (-1), e o valor que representa uma célula não revelada (-2), respectivamente.

```
Unset
    # Acessando valor da matriz
    mul $t6, $s1, $t5  # $t6 = i * SIZE
    add $t6, $t6, $s2  # $t6 = (i * SIZE) + j
     sll $t6, $t6, 2     # $t6 = $t6 * 4 (multiplicando por 4 para acessar a posição na memória)
    add $s6, $t6, $s0  # $s6 = endereço base da matriz + (i * SIZE + j) * 4

      lw $t9, 0($s6)      # Carrega o valor da matriz no endereço calculado para $t9
```

Essas instruções calculam o endereço da célula na matriz com base nos valores de linha e coluna recebidos. Primeiro, multiplicamos o valor da linha pelo tamanho da matriz e somamos o valor da coluna. Depois, multiplicamos esse resultado por 4 (porque cada célula é de 4 bytes) e adicionamos ao endereço base da matriz. Em seguida, carregamos o valor da célula para $t9.

```
Unset
   # Condições
   beq $t9, $s7, perdeu    # Se o valor for -1, o jogador perdeu
    bne $t9, $t8, revelado  # Se o valor não for -2, já foi revelado ou é uma
bomba
```

Estas instruções verificam se o valor da célula é igual a -1, o que indica que o jogador revelou uma célula com bomba, levando à perda do jogo. Caso contrário, verifica se o valor é diferente de -2, o que indica que a célula já foi revelada ou é uma bomba. Se já foi revelada ou é uma bomba, o programa segue para a parte de revelar células vizinhas ou continuar a revelação.

```
Unset
  # Salvando dados e chamando a função 'countAdjacentBombs'
  move $a0, $s1 # Passando i para o primeiro argumento
  move $a1, $s2 # Passando j para o segundo argumento
   move $a2, $s0  # Passando a posição inicial da matriz para o
terceiro argumento

    jal countAdjacentBombs  # Chamada de função para contar
bombas adjacentes

  move $t0, $v0 # $t0 recebe o número de bombas contadas
   sw $t0, 0($s6)  # Salvando o número de bombas na posição da
matriz

    bne $t0, $zero, revelado  # Se há bombas adjacentes,
continuar revelando células vizinhas
```

Nesta parte, os registradores $a0, $a1, e $a2 são utilizados para passar os argumentos para a função countAdjacentBombs, que conta o número de bombas

adjacentes à célula atual. O valor retornado é armazenado em $t0 e salvo na matriz. Se houver bombas adjacentes, o programa continua revelando as células vizinhas.

```
Unset
    # Salvando dados e chamando a função 'revealNeighboringCells'
    move $a0, $s1  # Passando i para o primeiro argumento
    move $a1, $s2  # Passando j para o segundo argumento
      move $a2, $s0   # Passando a posição inicial da matriz para o terceiro argumento

       jal revealNeighboringCells   # Chamada de função para revelar células vizinhas

revelado:
    restore_context  # Restaurando o contexto
    li $v0, 1        # Definindo retorno como 1 (célula revelada)
    jr $ra           # Retornando

perdeu:
    restore_context  # Restaurando o contexto
    li $v0, 0        # Definindo retorno como 0 (jogador perdeu)
    jr $ra           # Retornando
```

Nesta parte, os registradores $a0, $a1, e $a2 são novamente utilizados para passar os argumentos para a função revealNeighboringCells, que revela as células vizinhas. Depois disso, o contexto é restaurado e o valor de retorno é definido como 1, indicando que a célula foi revelada com sucesso. Se o valor da célula for -1 (indicando uma bomba), o contexto é restaurado e o valor de retorno é definido como 0, indicando que o jogador perdeu o jogo.

Esta função trata da jogada do jogador, verificando se a célula revelada contém uma bomba, contando as bombas adjacentes ou revelando as células vizinhas, dependendo do caso.

FUNÇÃO 2: COUNTADJACENTBOMBS

Esta função conta o número de bombas adjacentes a uma célula específica. Ela recebe as coordenadas (linha e coluna) da célula e retorna o número de bombas adjacentes.

```
Unset
.include "macros.asm"

.globl countAdjacentBombs

countAdjacentBombs:
    save_context

    # Recebendo dados e definindo valores de controle
    move $s0, $a2  # Recebe a posição inicial da matriz
    move $s1, $a0  # Recebe o valor de i (linha)
    move $s2, $a1  # Recebe o valor de j (coluna)
    move $t0, $zero  # Inicializa o contador de bombas adjacentes como zero

    addi $s3, $s1, 1  # Incrementa i para iterar sobre as células acima
    addi $s4, $s2, 1  # Incrementa j para iterar sobre as células à direita

    addi $s1, $s1, -2  # Decrementa i para iterar sobre as células abaixo
    addi $s2, $s2, -1  # Decrementa j para iterar sobre as células à esquerda

    li $s7, -1   # Define -1 para uso posterior (valor de célula de bomba)
      li $t5, SIZE  # Carrega o tamanho da matriz (assumindo que SIZE é uma
constante definida em 'macros.asm')
```

Nesta parte, a função começa salvando o contexto. Em seguida, ela inicializa alguns registradores e variáveis. $s0, $s1, e $s2 recebem os argumentos passados para a função, que representam a posição inicial da matriz, a linha (i), e a coluna (j). $t0 é inicializado como zero para contar o número de bombas adjacentes. Os registradores $s3 e $s4 são incrementados em 1 para iterar sobre as células vizinhas acima e à direita, enquanto $s1 e $s2 são decrementados para iterar sobre as células vizinhas abaixo e à esquerda. $s7 é inicializado como -1, que representa uma célula com bomba, e $t5 recebe o tamanho da matriz.

```
Unset
    # Loop for para a contagem de i
    for_contadj_i:
        addi $s1, $s1, 1  # Incrementa i
        bgt $s1, $s3, final  # Se i for maior que s3, termina o loop
          addi $s2, $s4, -2  # Reinicializa j para a posição da célula à esquerda da
linha atual

    # Loop for para a contagem de j
```

```
    for_contadj_j:
        bgt $s2, $s4, for_contadj_i  # Se j for maior que s4, termina o loop de j e
continua com o próximo i
        blt $s1, $zero, else  # Se i for menor que zero, vá para 'else'
            bge $s1, $t5, else  # Se i for maior ou igual ao tamanho da matriz, vá
para 'else'
        blt $s2, $zero, else  # Se j for menor que zero, vá para 'else'
            bge $s2, $t5, else  # Se j for maior ou igual ao tamanho da matriz, vá
para 'else'
```

Nestes loops, a função itera sobre as células vizinhas à célula atual. O loop externo percorre as linhas (i), enquanto o loop interno percorre as colunas (j). As instruções bgt e blt são utilizadas para verificar se i e j estão dentro dos limites da matriz.

```
Unset
        mul $t6, $s1, $t5  # Calcula o índice da célula (i * SIZE)
        add $t6, $t6, $s2  # Adiciona a posição de j ao índice
            sll $t6, $t6, 2  # Multiplica o índice por 4 para acessar a posição na
memória
        add $s6, $t6, $s0  # Calcula o endereço da célula na matriz

        lw $t9, 0($s6)  # Carrega o valor da célula
        bne $t9, $s7, else  # Se não for uma bomba, vá para 'else'
        addi $t0, $t0, 1  # Incrementa o contador de bombas adjacentes
```

Nesta parte, o endereço da célula na matriz é calculado com base nos valores de i e j. Em seguida, o valor da célula é carregado e comparado com $s7, que representa uma célula com bomba. Se a célula não for uma bomba, o contador de bombas adjacentes é incrementado.

```
Unset
    else:
        addi $s2, $s2, 1  # Incrementa j
        j for_contadj_j  # Volta para o loop de j

    final:
        restore_context  # Restaura o contexto
```

```
        move $v0, $t0  # Move o número de bombas adjacentes para o retorno
da função
       jr $ra  # Retorna
```

Nesta parte, se a célula não for uma bomba, incrementamos j e voltamos para o loop interno. Quando o loop termina, o contexto é restaurado, o número de bombas adjacentes é movido para o registrador de retorno ($v0), e a função retorna.

Esta função é responsável por contar o número de bombas adjacentes à célula atual. Ela percorre as células vizinhas, verifica se são bombas, e incrementa um contador quando encontra uma. Ao final, retorna o número de bombas adjacentes encontradas.

FUNÇÃO 3: REVEALNEIGHBORINGCELLS

Esta função é chamada quando uma célula sem bombas adjacentes é revelada. Ela revela recursivamente todas as células vizinhas que também não têm bombas adjacentes.

```
Unset
.include "macros.asm"

.globl revealNeighboringCells

revealNeighboringCells:
    save_context

    # Recebendo dados e definindo valores de controle
    move $s0, $a2  # Recebe a posição inicial da matriz
    move $s1, $a0  # Recebe o valor de i (linha)
    move $s2, $a1  # Recebe o valor de j (coluna)

    addi $s3, $s1, 1  # Incrementa i para iterar sobre as células acima
    addi $s4, $s2, 1  # Incrementa j para iterar sobre as células à direita

    addi $s1, $s1, -2  # Decrementa i para iterar sobre as células abaixo
    addi $s2, $s2, -1  # Decrementa j para iterar sobre as células à esquerda

    li $s7, -2  # Define -2 para uso posterior (valor de célula não revelada)
      li $t5, SIZE  # Carrega o tamanho da matriz (assumindo que SIZE é uma
constante definida em 'macros.asm')
```

Assim como nas funções anteriores, começamos salvando o contexto e inicializando alguns registradores e variáveis. $s0, $s1, e $s2 recebem os argumentos passados para a função, representando a posição inicial da matriz, a linha (i), e a coluna (j). Variáveis $s3 e $s4 são incrementadas para iterar sobre as células vizinhas acima e à direita, enquanto $s1 e $s2 são decrementadas para iterar sobre as células vizinhas abaixo e à esquerda. $s7 é inicializado como -2, que representa uma célula não revelada, e $t5 recebe o tamanho da matriz.

```
Unset
    # Loop for para a contagem de i
    for_contadj_i:
        addi $s1, $s1, 1  # Incrementa i
        bgt $s1, $s3, final  # Se i for maior que s3, termina o loop
         addi $s2, $s4, -2  # Reinicializa j para a posição da célula à esquerda da
linha atual

    # Loop for para a contagem de j
    for_contadj_j:
         bgt $s2, $s4, for_contadj_i  # Se j for maior que s4, termina o loop de j e
continua com o próximo i
        blt $s1, $zero, else  # Se i for menor que zero, vá para 'else'
           bge $s1, $t5, else  # Se i for maior ou igual ao tamanho da matriz, vá
para 'else'
        blt $s2, $zero, else  # Se j for menor que zero, vá para 'else'
           bge $s2, $t5, else  # Se j for maior ou igual ao tamanho da matriz, vá
para 'else'
```

Estes loops percorrem as células vizinhas à célula atual, de forma semelhante à função countAdjacentBombs, garantindo que i e j permaneçam dentro dos limites da matriz.

```
Unset
        mul $t6, $s1, $t5  # Calcula o índice da célula (i * SIZE)
        add $t6, $t6, $s2  # Adiciona a posição de j ao índice
           sll $t6, $t6, 2  # Multiplica o índice por 4 para acessar a posição na
memória
        add $s6, $t6, $s0  # Calcula o endereço da célula na matriz

        lw $t9, 0($s6)  # Carrega o valor da célula
        bne $t9, $s7, else  # Se a célula já estiver revelada, vá para 'else'
```

Nesta parte, o endereço da célula na matriz é calculado com base nos valores de i e j, e o valor da célula é carregado. Se a célula já estiver revelada, a função vai para a parte else, senão continua.

```
Unset
        move $a0, $s1  # Movendo i para o primeiro argumento
        move $a1, $s2  # Movendo j para o segundo argumento
          move $a2, $s0  # Movendo a posição inicial da matriz para o terceiro
argumento

          jal countAdjacentBombs  # Chamada de função para contar bombas
adjacentes

        move $t0, $v0  # $t0 recebe o número de bombas
        sw $t0, 0($s6)  # Salvando o número de bombas na posição

        bne $t0, $zero, else  # Se há bombas adjacentes, vá para 'else'
```

Nesta parte, os argumentos são preparados para chamar a função countAdjacentBombs para contar as bombas adjacentes. O número retornado é armazenado em $t0 e salvo na posição da matriz. Se não houver bombas adjacentes, o programa continua.

```
Unset
    else:
        addi $s2, $s2, 1  # Incrementa j
        j for_contadj_j  # Volta para o loop de j

    final:
        restore_context  # Restaurando o contexto
        jr $ra  # Retorna
```

Nesta parte, se não houver bombas adjacentes, j é incrementado e o programa volta para o loop interno. Quando o loop termina, o contexto é restaurado e a função retorna.

Esta função é responsável por revelar recursivamente as células vizinhas que também não têm bombas adjacentes. Ela percorre as células vizinhas, contando

as bombas adjacentes e revelando-as conforme necessário, até que todas as células vizinhas sejam reveladas.

FUNÇÃO 4: CHECKVICTORY

Esta função verifica se o jogador venceu o jogo. Ela conta o número de células reveladas e verifica se todas as células não bomba foram reveladas. Se isso acontecer, o jogador vence.

```
Unset
.include "macros.asm"

.globl checkVictory

checkVictory:
    save_context  # Salvando o contexto
    move $s0, $a0  # Recebe o endereço base da matriz
        move $t0, $zero  # Inicializa o contador para a contagem de células reveladas
      li $t5, SIZE  # Carrega o tamanho da matriz (assumindo que SIZE é uma constante definida em 'macros.asm')
    li $t1, -1  # Inicializa o índice i como 0
```

Assim como nas funções anteriores, começamos salvando o contexto e inicializando alguns registradores e variáveis. $s0 recebe o endereço base da matriz passado como argumento. $t0 é inicializado como zero para contar o número de células reveladas. $t5 recebe o tamanho da matriz, assumindo que é uma constante definida. $t1 é inicializado como -1 para ser usado como índice i.

```
Unset
    # Loop for para a contagem de i
    for_i:
        addi $t1, $t1, 1  # Incrementa i
        bge $t1, $t5, final  # Se i for maior ou igual a SIZE, termina o loop
        li $t2, 0  # Inicializa o índice j como 0

    # Loop for para a contagem de j
    for_j:
```

```
        bge $t2, $t5, for_i  # Se j for maior ou igual a SIZE, continua com o próximo i
        mul $t6, $t1, $t5  # Calcula o índice da célula (i * SIZE)
        add $t6, $t6, $t2  # Adiciona a posição de j ao índice
        sll $t6, $t6, 2  # Multiplica o índice por 4 para acessar a posição na memória
        add $s6, $t6, $s0  # Calcula o endereço da célula na matriz
        lw $t9, 0($s6)  # Carrega o valor da célula
```

Estes loops percorrem todas as células da matriz, verificando se foram reveladas.

```
Unset
        addi $t2, $t2, 1  # Incrementa j
        blt $t9, $zero, for_j  # Se o valor da célula for menor que zero (não revelada), continua com o próximo j
        addi $t0, $t0, 1  # Se o valor da célula for maior ou igual a zero (revelada), incrementa o contador
        j for_j
```

Nesta parte, j é incrementado e o programa volta para o loop interno. Se o valor da célula for maior ou igual a zero, significa que foi revelada, então incrementamos o contador.

```
Unset
    final:
        mul $t5, $t5, $t5  # Calcula o total de células na matriz
        li $t8, BOMB_COUNT  # Carrega a quantidade de bombas (assumindo que BOMB_COUNT é uma constante definida em 'macros.asm')
        sub $t5, $t5, $t8  # Subtrai a quantidade de bombas do total de células
        beq $t0, $t5, vitoria  # Se o contador for igual ao número de células não bomba, vai para 'vitoria'
        li $v0, 0  # Define 0 (derrota)
        restore_context  # Restaura o contexto
        jr $ra  # Retorna
```

Nesta parte, calculamos o número total de células na matriz e subtraímos a quantidade de bombas. Se o contador for igual ao número de células não bomba,

o jogo está vencido. Caso contrário, o jogo continua e a função retorna 0 para indicar uma derrota.

```
Unset
    vitoria:
        li $v0, 1  # Define 1 (vitória)
        restore_context  # Restaura o contexto
        jr $ra  # Retorna
```

Se o jogador tiver vencido, $v0 é definido como 1 para indicar uma vitória. O contexto é restaurado e a função retorna.

Esta função verifica se todas as células não bomba foram reveladas, o que indica que o jogador venceu o jogo.

Isso conclui a explicação da função checkVictory e das quatro funções do jogo de campo minado em MIPS Assembly. Cada função desempenha um papel crucial no funcionamento do jogo, desde revelar células até verificar a condição de vitória.