UNIVERSIDADE FEDERAL DO CARIRI

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CÍCERO IGOR ALVES TORQUATO DOS SANTOS
JOÃO MARCELO LOBO MATOS

RELATÓRIO MINESWEEPER

Juazeiro do Norte

2024

INTRODUÇÃO:

O presente relatório descreve o processo de implementação do jogo Minesweeper em linguagem Assembly MIPS. O projeto foi desenvolvido com base em um código base fornecido pelo professor, organizado em diversos arquivos para facilitar a compreensão e manutenção do código. Cada funcionalidade do jogo foi encapsulada em um arquivo separado, visando uma organização eficiente e modular do projeto. Neste relatório, serão apresentados os desafios enfrentados, as estratégias adotadas para superá-los, e os resultados obtidos ao final do desenvolvimento.

DESCRIÇÃO DO JOGO:

O Minesweeper é um jogo de tabuleiro de lógica onde o objetivo principal é descobrir todas as células vazias sem revelar as minas ocultas. O tabuleiro típico do Minesweeper é uma matriz bidimensional, onde cada célula pode estar em um de dois estados: revelada ou oculta.

Alguns detalhes importantes:

- Cada célula pode conter uma mina ou estar vazia.
- O jogador pode revelar uma célula usando como entrada o número da linha e coluna.
- Se uma célula vazia for revelada, ela mostrará o número de minas adjacentes a ela.
- Se uma célula com uma mina for revelada, o jogo acaba.

O desafio do jogo está em usar a lógica para deduzir a localização das minas, com base nas dicas fornecidas pelas células reveladas. O objetivo final é revelar todas as células vazias sem clicar em uma única mina.

DESENVOLVIMENTO:

Como foi citado anteriormente, o projeto está dividido em vários arquivos para facilitar a organização. Os arquivos main, macros, play, plantBombs, initializeBoard e printBoard já tiveram seus códigos disponibilizados pelo profesor. Porém, os códigos dos arquivos play, checkVictory, countAdjacentBombs e revealNeighboringCells precisavam ser desenvolvidos pelos alunos. Vamos detalhar cada arquivo a seguir:

1. **Arquivo "main"**: Este arquivo é o ponto de entrada do programa. Ele inicializa o tabuleiro do jogo, plantando as minas em posições aleatórias e definindo o estado inicial de cada célula. Além disso, o arquivo "main" também contém a função principal do programa, que controla o fluxo de execução do jogo e chama as funções necessárias para interagir com o jogador e atualizar o estado do tabuleiro.

2. **Arquivo "macros"**: Neste arquivo, são definidas macros úteis que podem ser utilizadas em todo o projeto. Isso inclui macros para salvar e restaurar o contexto, que são dados que estavam nos registradores , e são utilizadas em quase todos os arquivos que implementam funções.

3. **Arquivo "initializeBoard"**: Este arquivo contém a função responsável por inicializar o tabuleiro do jogo. Isso inclui definir o tamanho do tabuleiro, a quantidade de minas a serem plantadas e o estado inicial de cada célula.

4. **Arquivo "plantBombs"**: Aqui, é implementada a lógica para plantar as minas em posições aleatórias no tabuleiro. Isso é feito garantindo que cada célula tenha uma chance igual de conter uma mina e que a quantidade de minas plantadas corresponda ao número especificado no início do jogo.

5. **Arquivo "printBoard"**: Este arquivo contém a função responsável por imprimir o estado atual do tabuleiro na tela. Isso inclui exibir as células reveladas, as minas são reveladas quando o jogador perde o jogo.

6. **Arquivo "play"**: Esta função representa a jogada do jogador. Se o jogador clicar em uma célula com uma bomba, o jogo termina e retorna 0 (game over).
   Se o jogador clicar em uma célula vazia, a célula é marcada como revelada e, se não houver bombas adjacentes, as células adjacentes também são reveladas utilizando a função revealAdjacentCells. Retorna 1 se o jogo continuar.

   Detalhes da implementação:

   Foram usadas as macros save_context e restore_context que salvam e restauram o contexto da execução atual, que pode incluir o estado dos registradores e outros dados relevantes.

   As instruções move $s0,$a0, move $s1,$a1, move $s2,$a2 movem os parâmetros de entrada da função play em assembly (que correspondem à board, row e column em C) para registradores salvos ($s0, $s1 e $s2) para uso posterior.

   Usamos as instruções sll, add e lw para calcular o endereço da célula específica na matriz board com base nas coordenadas row e column. Isso é feito para acessar o conteúdo dessa célula na matriz board.

   Além disso as instruções li, beq e j foram utilizadas para implementar a lógica de verificação das células da matriz board. Primeiro, verifica-se se a célula atual contém uma bomba (board[row][column] == -1). Se sim, retorna-se imediatamente indicando que o jogador perdeu o jogo (return 0). Se a célula não contém uma bomba, então verifica-se se ela é uma célula não revelada (board[row][column] == -2). Se for, a função countAdjacentBombs é chamada para contar o número de bombas adjacentes e o resultado é armazenado de volta na célula. Se o número de bombas adjacentes for zero, a função

revealAdjacentCells é chamada para revelar as células adjacentes. Se a célula não for uma bomba nem uma célula não revelada, o jogo continua.

Nesse sentido, a instrução jal é usada para chamar a função countAdjacentBombs, passando os parâmetros corretos (board, row e column). Após a chamada da função, o resultado é armazenado de volta na célula board[row][column]. A instrução sw é usada para armazenar esse valor de volta na memória.

As labels continuar_jogo e acertou_bomba representam os pontos no código onde o jogo continua (continuar_jogo) ou o jogador atinge uma bomba (acertou_bomba). Dependendo do resultado da verificação das células, o código salta para uma dessas etiquetas para continuar ou terminar o jogo.

Por fim, final_play representa o ponto final da função play em assembly, onde o contexto é restaurado e o controle é retornado ao ponto de chamada da função.

Portanto o código de play.asm funciona manipulando a matriz board e verificando as células para determinar se o jogador atingiu uma bomba, se uma célula precisa ser revelada ou se o jogo deve continuar.

7. **Arquivo "countAdjacentBombs":** Esta função conta o número de bombas adjacentes a uma célula especificada por sua posição na matriz do tabuleiro. Ela percorre todas as células adjacentes à célula especificada, verificando se estão dentro dos limites do tabuleiro e se contêm uma bomba (representada pelo valor -1). O contador é incrementado sempre que uma célula adjacente contém uma bomba. O resultado final é o número total de bombas adjacentes à célula especificada.

   Detalhes da implementação:

   No início usamos as macros save_context e restore_context que salvam e restauram o contexto da execução atual.

   As instruções move $s0,$a0, move $s1,$a1, move $s2,$a2 movem os parâmetros de entrada da função countAdjacentBombs em assembly (que correspondem a board, row e column em C) para registradores salvos ($s0, $s1 e $s2) para uso posterior.

   A instrução li é usada para carregar valores imediatos (constantes) em registradores. No caso, $s6 é inicializado com zero para representar o contador.

   begin_for_i_it_count e end_for_i_it_count são etiquetas que marcam o início e o final do loop externo que itera sobre as coordenadas i (linha) na vizinhança da célula atual.

begin_for_j_it_count e end_for_j_it_count são outras etiquetas que marcam o início e o final do loop interno que itera sobre as coordenadas j (coluna) na vizinhança da célula atual.

As instruções sll, add e lw são usadas para calcular o endereço da célula específica na matriz board com base nas coordenadas i e j e carregar o valor dessa célula.

Já as instruções blt, bge e bne são usadas para verificar se as coordenadas i e j estão dentro dos limites da matriz board e se o conteúdo da célula é uma bomba (-1). Se todas as condições forem atendidas, o contador é incrementado.

Para incrementar o addi incrementa o contador de bombas ($s6) quando uma bomba é encontrada.

Além disso, o move $v0,$s6 move o valor do contador de bombas ($s6) para o registrador de retorno da função ($v0).

A instrução jr $ra retorna o controle ao ponto de chamada da função.

Enfim, fake_else_count é uma etiqueta usada para lidar com os casos em que as coordenadas i e j estão fora dos limites da matriz board. Nesses casos, o loop interno é avançado sem incrementar o contador.

8. **Arquivo "revealAdjacentCells":** Esta função revela as células adjacentes a uma célula vazia (sem bombas adjacentes) especificada por sua posição na matriz do tabuleiro. Ela percorre todas as células adjacentes à célula especificada, verificando se estão dentro dos limites do tabuleiro e se são células vazias (representadas pelo valor -2). Se uma célula adjacente for vazia, ela é marcada como revelada e a função é chamada recursivamente para continuar a revelação das células adjacentes a essa célula.

   Detalhes da implementação:

   Novamente usamos as macros save_context e restore_context que salvam e restauram o contexto da execução atual.

   Também é necessário usar as instruções move $s0,$a0, move $s1,$a1, move $s2,$a2 que movem os parâmetros de entrada da função revealNeighboringCells em assembly (que correspondem a board, row e column em C) para registradores salvos ($s0, $s1 e $s2) para uso posterior.

   Usamos a instrução li para inicializar $s5 com um valor imediato.No caso, $s5 é inicializado com zero para representar o índice da linha i.

laco_i e final_laco_i, estas etiquetas marcam o início e o final do loop externo que itera sobre as coordenadas i (linha) na vizinhança da célula atual.

Já laco_j e final_laco_j são outras etiquetas que marcam o início e o final do loop interno que itera sobre as coordenadas j (coluna) na vizinhança da célula atual.

As instruções sll, add e lw são usadas para calcular o endereço da célula específica na matriz board com base nas coordenadas i e j e carregar o valor dessa célula.

As instruções blt, bge e bne são usadas para verificar se as coordenadas i e j estão dentro dos limites da matriz board e se o conteúdo da célula é uma célula não revelada (-2). Se todas as condições forem atendidas, a função countAdjacentBombs é chamada para contar o número de bombas adjacentes e o resultado é armazenado de volta na célula. Se o número de bombas adjacentes for zero, a função revealNeighboringCells é chamada recursivamente para revelar as células adjacentes.

Assim, usamos move $a0,$s0, move $a1, $s5, move $a2, $s7, para passar os parâmetros corretos (board, i e j) para a função countAdjacentBombs e chamá-la.

Usamos o sw para armazenar o resultado retornado pela função countAdjacentBombs de volta na célula board[i][j].

E as instruções beq e j são usadas para decidir se o loop interno deve continuar ou não, dependendo do resultado da contagem de bombas adjacentes.

A etiqueta fake_else_j é usada para lidar com os casos em que as coordenadas i e j estão fora dos limites da matriz board. Nesses casos, o loop interno é avançado sem fazer mais operações.

Nessa função também usamos a instrução jr $ra que retorna o controle ao ponto de chamada da função.

9. **Arquivo “checkVictory”:** Esta função verifica se o jogador venceu o jogo. Percorre todas as células do tabuleiro e conta o número de células que foram reveladas (ou seja, não contêm bombas). Retorna 1 se todas as células válidas do tabuleiro (ou seja, todas as células exceto as que contêm bombas) foram reveladas. Caso contrário, retorna 0.

Detalhes da implementação:

Novamente as macros save_context e restore_context foram usadas para salvar e restaurar o contexto da execução atual.

A instrução move $s0,$a0 foi utilizada para mover o parâmetro de entrada da função checkVictory em assembly (que corresponde a board em C) para um registrador salvo ($s0) para uso posterior.

Usamos a instrução li para que $t0 seja inicializado com zero para representar o contador.

As etiquetas laco_i e fim_laco_i marcam o início e o final do loop externo que itera sobre as linhas da matriz board.

E as etiquetas laco_j e fim_laco_j marcam o início e o final do loop interno que itera sobre as colunas da matriz board.

sll, add e lw foram instruções usadas para calcular o endereço da célula específica na matriz board com base nas coordenadas i e j e carregar o valor dessa célula.

Usamos a instrução blt para verificar se o conteúdo da célula é menor que zero, o que indica que a célula não foi revelada.

Já a utilização da instrução addi foi para incrementar o contador quando uma célula válida (não revelada) é encontrada.

Tivemos que usar a instrução mul para calcular o produto entre dois registradores, $t2 e $t2, que contêm o valor de SIZE.

Se aproximando do fim , a instrução bge é usada para verificar se o contador é maior ou igual ao número total de células válidas na matriz board menos o número de bombas (SIZE * SIZE - BOMB_COUNT). Se todas as células válidas foram reveladas, o jogo é vencido.

Dessa forma, li $v0,0 e li $v0,1 são instruções que definem o valor de retorno da função dependendo do resultado da verificação da vitória. Se todas as células válidas foram reveladas, retorna 0 (vitória), caso contrário, retorna 1 (jogo ainda não terminado).

Assim como nas outras funções, jr $ra retorna o controle ao ponto de chamada da função.

Em resumo, o código conta o número de células válidas não reveladas na matriz board e determina se o jogador venceu ou não o jogo.

CONCLUSÃO:

A implementação do jogo Minesweeper em linguagem Assembly MIPS representou um desafio significativo, mas também uma oportunidade valiosa para aplicar conceitos de programação de baixo nível e lógica de jogo. Ao longo do desenvolvimento, enfrentamos diversos obstáculos, desde compreender completamente a lógica do jogo até garantir a integração adequada das diversas funcionalidades em um código coeso e funcional.

A divisão do projeto em arquivos separados facilitou a organização e a manutenção do código, permitindo que cada funcionalidade fosse desenvolvida de forma modular e independente.O acompanhamento do projeto, por parte do professor, em sala, foi de extrema importância pois dúvidas em pontos fundamentais do projeto foram esclarecidas.

No final, conseguimos implementar com sucesso todas as funcionalidades essenciais do jogo, incluindo a contagem de bombas adjacentes, a função de jogada do usuário, a função de revelar as células adjacentes e a verificação da vitória.

Links para o GitHub:

Código-fonte disponível em:
https://github.com/IgorTorquatto/Arquitetura_de_Computadores/tree/main/Projeto/minesweeper