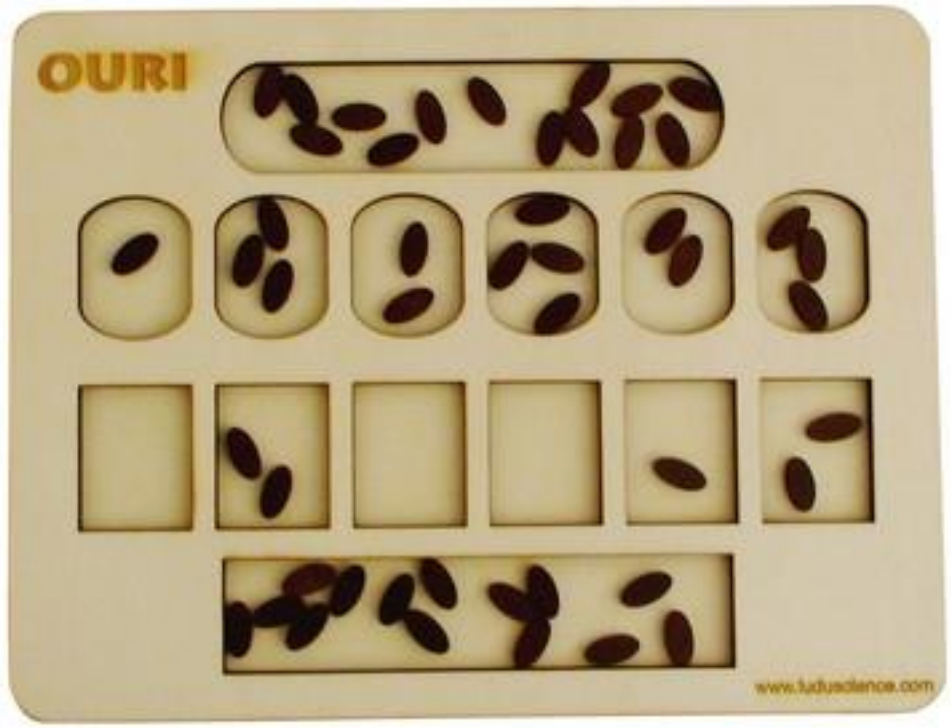
UNIVERSIDADE DE ÉVORA

CURSO DE ENGENHARIA INFORMÁTICA

Programação I



Trabalho Prático 2023/2024

Enoque Massau nº 53235

4 de janeiro de 2024

Introdução

No âmbito da cadeira de Programação I, foi pedido a criação de um jogo de tabuleiro “Ouri” também conhecido como “Mancala”, numa versão virtual onde neste relatório tem como objetivo documentar o desenvolvimento e a análise do jogo "Ouri", descrevendo seu funcionamento, lógica de jogo, implementação técnica.

Este jogo é para dois jogadores, e é jogado num tabuleiro tradicional de 6x2 onde inicialmente são colocadas 4 peças em cada casa de cada jogador no tabuleiro. O objetivo do jogo "Ouri" é acumular mais pedras no depósito do jogador do que no do oponente. Cada jogador tem um conjunto de casas (chamadas de "ouris"), durante o jogo, os jogadores alternam suas jogadas, movendo as pedras entre as casas de acordo com regras específicas, com o intuito de capturar pedras do oponente e direcioná-las para o seu próprio depósito.

Neste trbalho o jogo pode ser jogado de forma interativa, entre um jogador e jogador, ou jogdor e cpu.

Descrição dos diferentes modos de jogo

1. **Player vs Player**: Neste modo, dois jogadores humanos competem um contra o outro. O jogo começa com o jogador 1. Em cada turno, o jogador seleciona uma de suas casas que contém pedras. As pedras são então distribuídas uma por uma nas casas seguintes. Se a última pedra cair em uma casa do adversário que agora contém 2 ou 3 pedras, todas as pedras nessa casa são capturadas pelo jogador. Se todas as casas de um jogador estiverem vazias, o jogador é solicitado a repor as pedras no lado adversário. O jogo continua até que um dos jogadores tenha 25 ou mais pedras em seu depósito, nesse ponto, esse jogador é declarado vencedor.
2. **Player vs CPU**: Neste modo, um jogador humano compete contra o CPU. O jogo começa com o jogador humano. O fluxo do jogo é semelhante ao modo Player vs Player, com a diferença de que, quando é a vez do CPU, a casa é selecionada automaticamente com base em uma estratégia predefinida (por exemplo, escolhendo a casa com o maior número de pedras). O jogo continua até que o jogador humano ou o CPU tenha 25 ou mais pedras em seu depósito, nesse ponto, o vencedor é declarado.

Funções

A função Board é usada para exibir o estado atual do tabuleiro do jogo. Ela aceita dois parâmetros:

1. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) é armazenado neste array.
2. deposit [2]: Este é um array que representa os “depósitos” de cada jogador. Cada jogador tem um depósito, e o número de peças no depósito de cada jogador é armazenado neste array.

A função então imprime o estado atual do tabuleiro, incluindo o número de peças em cada casa e em cada depósito. As casas do jogador 2 são exibidas na parte superior do tabuleiro, e as casas do jogador 1 são exibidas na parte inferior.

Aqui está uma explicação mais detalhada do que cada linha de código faz:

- As linhas que começam com printf("|---|--|--|--|--|--|--|---|\n"); são usadas para imprimir a borda superior e inferior do tabuleiro.
- A linha printf ("| |%2d|%2d|%2d|%2d|%2d|%2d| |\n", player [1][5], player [1][4], player [1][3], player [1][2], player [1][1], player [1][0]); imprime o estado das casas do jogador 2.
- A linha printf ("| %2d|----------------| %2d|\n", deposit [0], deposit [1]); imprime o estado dos depósitos de cada jogador.
- A linha printf ("| |%2d|%2d|%2d|%2d|%2d|%2d| |\n", player [0][0], player [0][1], player [0][2], player [0][3], player [0][4], player [0][5]); imprime o estado das casas do jogador 1.

A função movement é usada para realizar um movimento no jogo. Ela aceita quatro parâmetros:

1. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) é armazenado neste array.
2. deposito [2]: Este é um array que representa os “depósitos” de cada jogador. Cada jogador tem um depósito, e o número de peças no depósito de cada jogador é armazenado neste array.
3. selectedHouse: Este é um inteiro que representa a casa selecionada pelo jogador para o movimento atual.
4. currentPlayer: Este é um inteiro que representa o jogador atual.

A função então realiza o seguinte:

- Inicializa algumas variáveis, incluindo currentHouse (a casa atual), stones (o número de peças na casa selecionada), e initialPlayer e initialHouse (o jogador e a casa no início do movimento).
- Entra em um loop que continua até que todas as peças da casa selecionada tenham sido distribuídas. Durante cada iteração do loop, a função move uma peça para a próxima casa e decrementa o número de peças restantes.
- Se todas as peças da casa selecionada foram distribuídas e a última peça foi colocada em uma casa do adversário que agora contém 2 ou 3 peças, então todas as peças nessa casa são capturadas e adicionadas ao depósito do jogador atual.
- Finalmente, a função verifica se todas as casas do jogador atual estão vazias. Se estiverem, o jogador é solicitado a repor as pedras no lado adversário.

findHouseWithMaxStones faz:

A função findHouseWithMaxStones é usada para encontrar a casa que tem o maior número de pedras para o jogador atual. Ela aceita dois parâmetros:

1. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) é armazenado neste array.
2. currentPlayer: Este é um inteiro que representa o jogador atual.

A função então realiza o seguinte:

- Inicializa duas variáveis, maxStones e houseIndex, para acompanhar o maior número de pedras encontrado até agora e o índice da casa correspondente.
- Percorre todas as casas do jogador atual. Para cada casa, se o número de pedras na casa for maior que maxStones, então atualiza maxStones e houseIndex para o número de pedras e o índice da casa atual, respectivamente.
- Depois de percorrer todas as casas, retorna houseIndex, que é o índice da casa com o maior número de pedras.

A função save_game_state é usada para salvar o estado atual do jogo em um arquivo. Ela aceita dois parâmetros:

1. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) é armazenado neste array.
2. deposit [2]: Este é um array que representa os “depósitos” de cada jogador. Cada jogador tem um depósito, e o número de peças no depósito de cada jogador é armazenado neste array.

A função realiza o seguinte:

- Pede ao usuário para digitar o nome do arquivo onde o estado do jogo será salvo.
- Tenta abrir o arquivo especificado para escrita. Se não conseguir abrir o arquivo, imprime uma mensagem de erro e retorna.
- Percorre os arrays player e deposit, e escreve o estado de cada casa e cada depósito no arquivo.
- Fecha o arquivo e imprime uma mensagem informando que o estado do jogo foi salvo com sucesso.
- Termina a execução do programa usando a função exit.

A função load_game_state é usada para carregar o estado do jogo a partir de um arquivo. Ela aceita três parâmetros:

1. filename: Este é um ponteiro para um caractere que representa o nome do arquivo de onde o estado do jogo será carregado.
2. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) será armazenado neste array.
3. deposit [2]: Este é um array que representa os “depósitos” de cada jogador. Cada jogador tem um depósito, e o número de peças no depósito de cada jogador será armazenado neste array.

A função realiza o seguinte:

- Tenta abrir o arquivo especificado para leitura. Se não conseguir abrir o arquivo, imprime uma mensagem de erro e retorna.
- Percorre os arrays player e deposit, e lê o estado de cada casa e cada depósito do arquivo.
- Fecha o arquivo.

A função playGame é usada para controlar o fluxo do jogo. Ela aceita dois parâmetros:

1. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) é armazenado neste array.
2. deposito [2]: Este é um array que representa os “depósitos” de cada jogador. Cada jogador tem um depósito, e o número de peças no depósito de cada jogador é armazenado neste array.

A função realiza o seguinte:

- Inicializa o jogador atual (currentPlayer) como 0 (o primeiro jogador).
- Entra em um loop infinito que representa o jogo em andamento. O jogo continua até que uma condição de vitória seja atendida.
- Em cada turno, a função exibe o tabuleiro do jogo, pede ao jogador para selecionar uma casa, aplica as regras da casa à casa selecionada e realiza o movimento.
- A função verifica se todas as casas do jogador atual estão vazias. Se estiverem, o jogador é solicitado a repor as pedras no lado adversário.
- A função verifica se algum dos jogadores alcançou a condição de vitória (ter 25 ou mais pedras em seu depósito). Se a condição de vitória for atendida, a função imprime uma mensagem indicando o vencedor e termina o jogo.
- Se nenhuma condição de vitória for atendida, a função muda a vez para o outro jogador e o loop continua.

A função playAgainstCPU é usada para controlar o fluxo do jogo quando o adversário é o CPU. Ela aceita dois parâmetros:

1. player [2][6]: Este é um array bidimensional que representa as “casas” do jogo para cada jogador. Cada jogador tem 6 casas, e o estado de cada casa (por exemplo, o número de peças em cada casa) é armazenado neste array.
2. deposito [2]: Este é um array que representa os “depósitos” de cada jogador. Cada jogador tem um depósito, e o número de peças no depósito de cada jogador é armazenado neste array.

A função realiza o seguinte:

- Inicializa o jogador atual (currentPlayer) como 0 (o primeiro jogador) e define uma flag (isPlayerOneHuman) para indicar se o primeiro jogador é humano.
- Entra em um loop infinito que representa o jogo em andamento. O jogo continua até que uma condição de vitória seja atendida.

- Em cada turno, a função exibe o tabuleiro do jogo, imprime de quem é a vez, e pede ao jogador (ou ao CPU) para selecionar uma casa. A função então aplica as regras da casa à casa selecionada e realiza o movimento.
- A função verifica se todas as casas do jogador atual estão vazias. Se estiverem, o jogador é solicitado a repor as pedras no lado adversário.
- A função verifica se algum dos jogadores alcançou a condição de vitória (ter 25 ou mais pedras em seu depósito). Se a condição de vitória for atendida, a função imprime uma mensagem indicando o vencedor e termina o jogo.
- Se nenhuma condição de vitória for atendida, a função muda a vez para o outro jogador e o loop continua.

Conclusão

Após ter sido feito um trabalho de pesquisa acerca deste jogo de tabuleiro, desde as suas regras e alguns truques recolhidos a partir de certas fontes, ter sido planeado como ia ser resolvido cada problema, defrontei-me com diferentes obstáculos, isto é, problemas em algumas implementações de certas funções, mas que foram ultrapassadas com sucesso, pois foram-nos dadas informações necessárias durante o decorrer do semestre que nos ajudou na implementação deste trabalho. Este trabalho foi importante pois ajudou-nos a perceber a importância dos conhecimentos fundamentais das linguagens de programação a prepararmo-nos para um trabalho futuro neste ramo.