Implementação do rummikub utilizando a linguagem de programação C.

Edson Andrade da Costa Filho	N°USP: 10407051	p0
Gabriell Tavares Luna	N°USP: 10716400	p1
Vinícius Rodrigues Geraldo	N°USP: 10438122	p2
Felipe Caldana Ricioli	N°USP: 10716654	p3
Eike Sato	N°USP: 10716588	p4
Enzo Sato	N°USP: 10716571	p5
Isabela Oliveira Costa	N°USP: 10747972	p6

1. Introdução

O trabalho consiste na implementação, em linguagem C, do clássico jogo de tabuleiro, Rummikub. Na sua versão original pode ser jogado tanto com peças de cores diferentes, quanto com cartas. No caso do jogo programado, foram usados números hexadecimais (1 a D), 4 naipes diferentes (!, @, # e $) e dois coringas (**).
Todas as regras foram implementadas ao projeto, permitindo ações como a alteração de sequências já existentes na mesa e também verificações de sequências inválidas.

1.1 Implementação

As cartas são do tipo *pilha*, que guarda seu valor e naipe. Elas são alocados dinamicamente em um vetor de 106 posições, o que permite a passagem por referência. Para se referir a tal carta, basta saber qual sua posição no vetor.
Os jogadores também são alocados dinamicamente, de acordo com a quantidade. Eles são do tipo *player*, que guarda a *mao* (um ponteiro de inteiros que deve armazenar as posições das cartas que estão na mão do jogador), *manga* (quantidade de cartas na mão) e *turno* (turno do jogador).
A mesa é uma variável do tipo *tabela*, alocada dinamicamente, que guarda *jogada* (ponteiro do tipo *rhcp* que deve armazenar as posições das cartas que estão na mesa), *quantidadejogada* (quantidade de cartas na mesa) *, trucos* (quantidade de combinações já realizadas na mesa).
Para realizar as jogadas, foi implementado o esquema de *Área de transferência*, que consiste em combinar cartas na mão e na mesa para a área de transferência e assim realizar uma jogada. As cartas que estão na área de transferência ainda não foram retiradas da mão ou mesa, assim permitindo que o jogador desfaça sua jogada. Uma jogada

só é válida se as cartas presentes nessa área satisfaçam uma combinação.

2. Descrição do Projeto

O projeto foi desenvolvido utilizando a IDE Code::Blocks 16.02 para o sistema operacional Linux x64. O código foi compilado em Linux x64 utilizando o GCC 5.4.0 e em Windows x64 utilizando a versão 5.1.0 do GCC, sem problemas de compatibilidade. Os headers do projeto são “stdio.h”, ”stdlib.h”, “malloc.h”, “string.h”, “time.h”, e o código fonte é “main.c”.

3. Tutorial

3.1 Compilando com o GCC:

Abra o terminal, ou Prompt de Comando, acesse a pasta onde está o código fonte “main.c”. Para a compilação, execute: “gcc main.c -o rummikub”.

3.2 Executando:

• **Linux:** Depois de compilado, execute o comando: “./rummikub”.
• **Windows:** Depois de compilado, execute o arquivo “rummikub.exe” que foi criado na pasta onde está o programa.

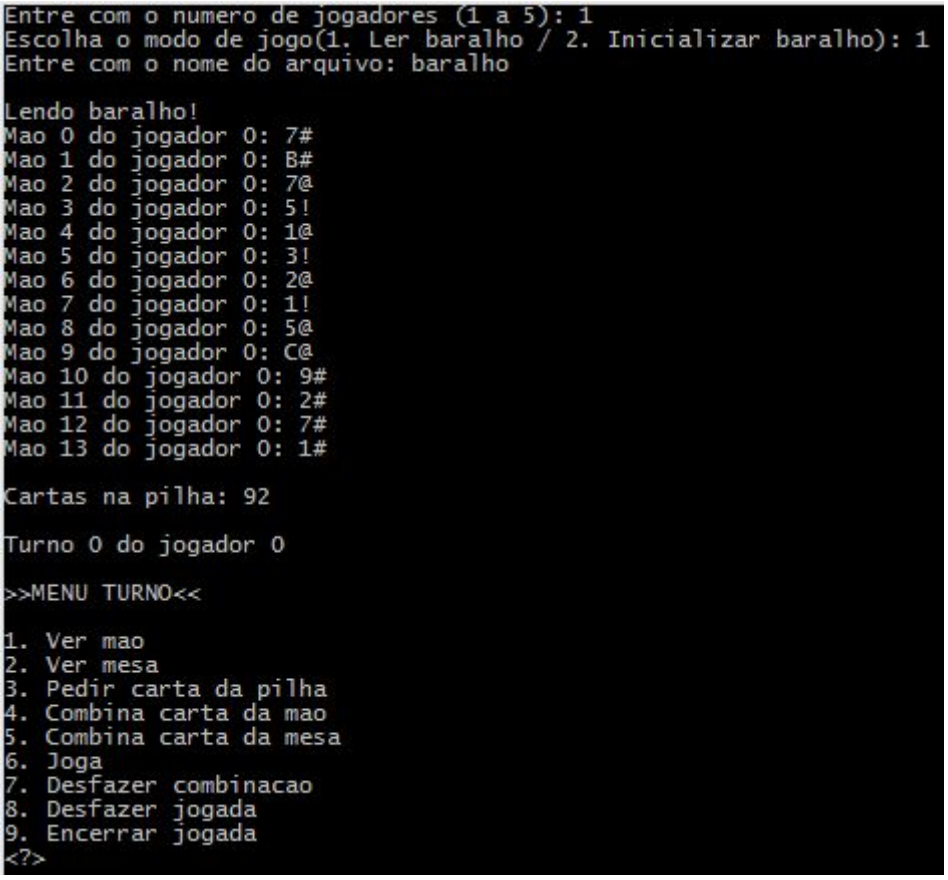
3.3 Jogando:

Na interface do jogo, o usuário deverá escolher a quantidade de jogadores que irão jogar, que varia de 1 a 5 jogadores.

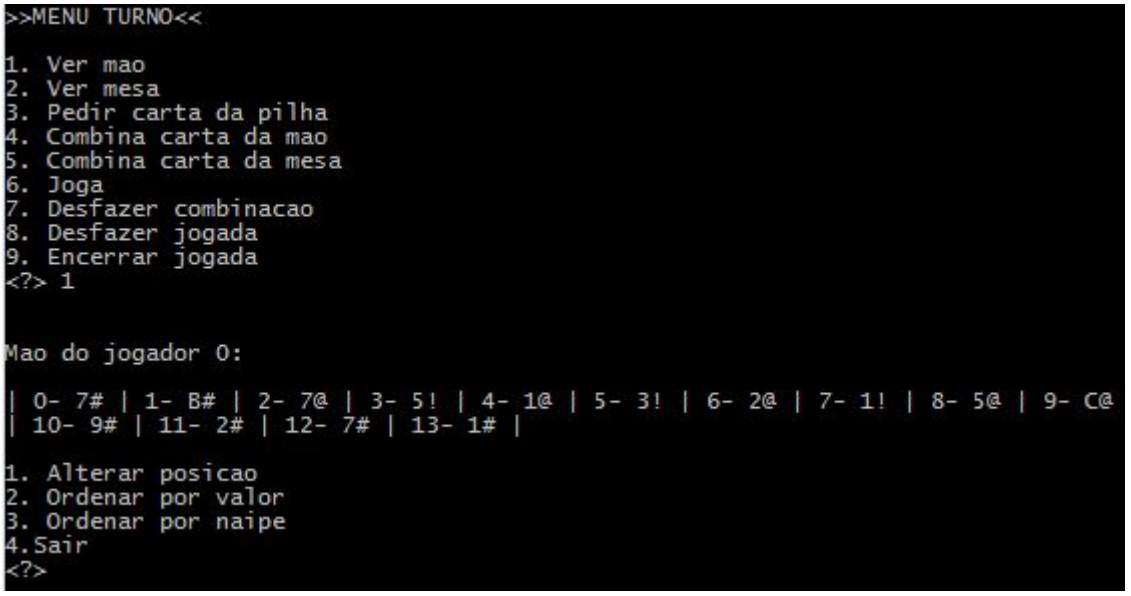
Para escolher o tipo do baralho, há duas opções: controlado ou aleatório.

- Controlado: O programa lê o baralho de um arquivo do tipo “.txt” que deverá estar na mesma pasta que o rummikub. Nesse modo o usuário entra com o nome do arquivo desejado e o programa concatena o “.txt”. O arquivo deve possuir a formatação que foi proposta, para evitar bugs.
- Aleatório: O programa inicializa e embaralha as cartas randomicamente.

Em seguida, o computador distribui as cartas alternadamente para os jogadores. Terminado o modo “dealer”, que exibe essa distribuição alternada, inicia o turno do jogador 0, assim seguindo a ordem dos jogadores pela sua posição.

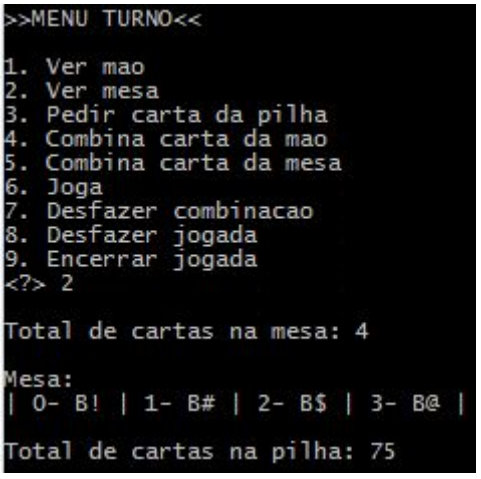


1. **Ver mão**:



Permite visualizar a mão do jogador, com a opção de alterar a posição das cartas e realizar uma ordenação por valor ou naipe.
Usa a função *void* visualizaMao(*pilha* *, *player* *, *int* , *int* *).

2. Ver mesa:



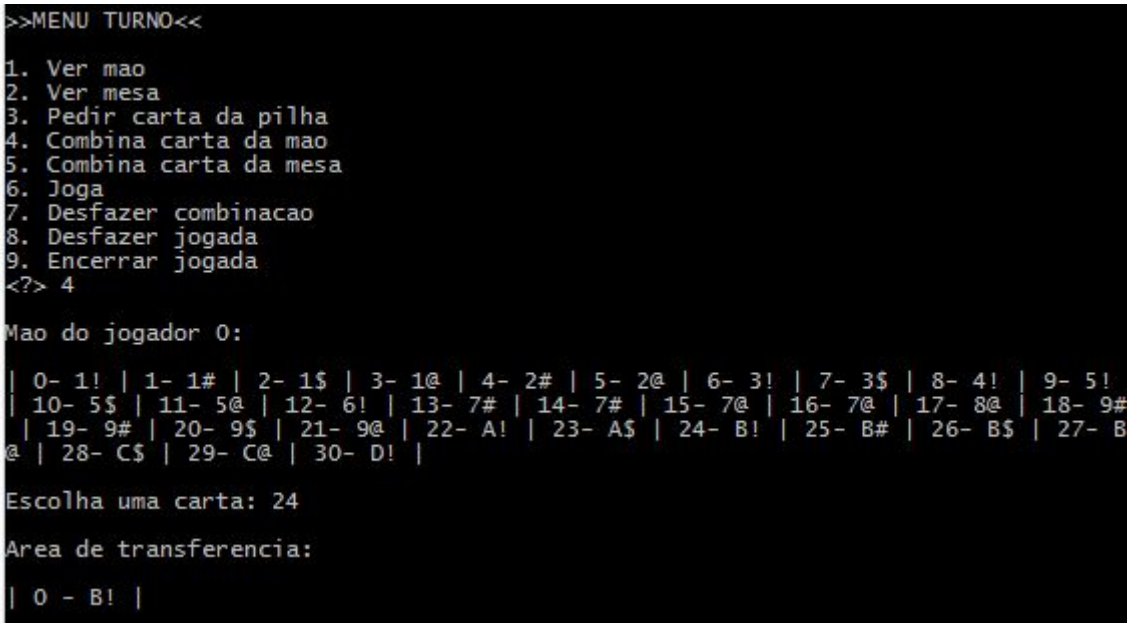
Permite visualizar cartas e sequências que estão na mesa. No caso da figura, nenhuma jogada tinha sido feita.
Usa a função *void* visualizaMesa(*pilha* * ,*tabela* * , *int*).

3. Pedir carta da pilha:

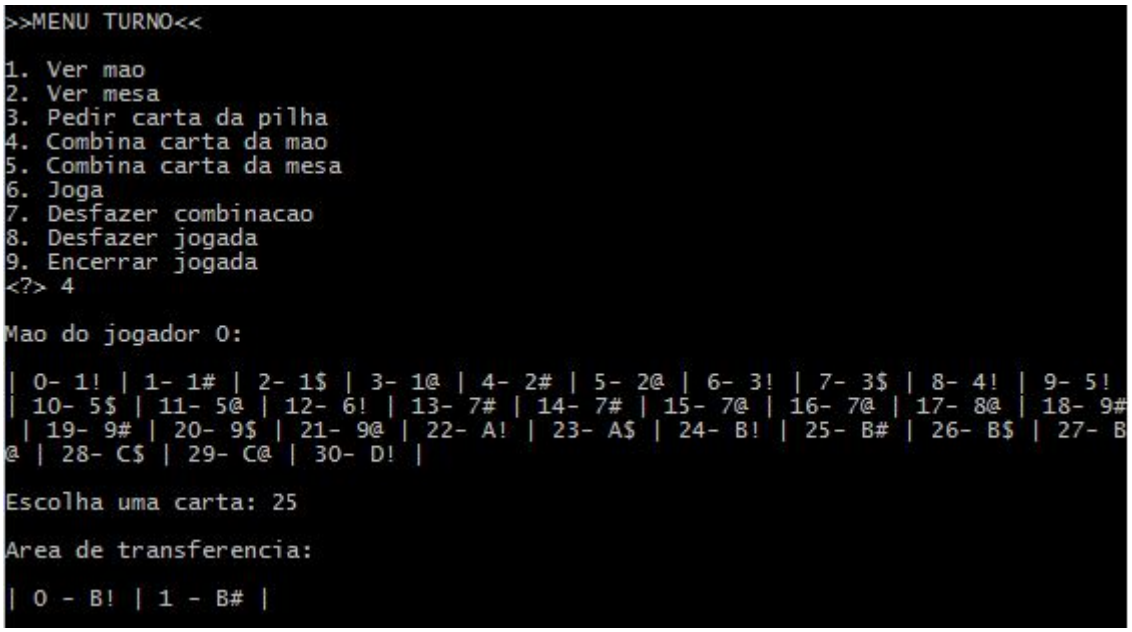
Se o fim de turno for válido, ou seja, a mesa possui apenas sequências válidas, o jogador recebe uma carta do baralho e encerra o turno.
Usa as funções *int* pedeCarta(*pilha* *, *player* *, *int* , *int*) e *int* validaMesa(*tabela* *, *pilha* *, *int*).

4. Combinar carta da mão:

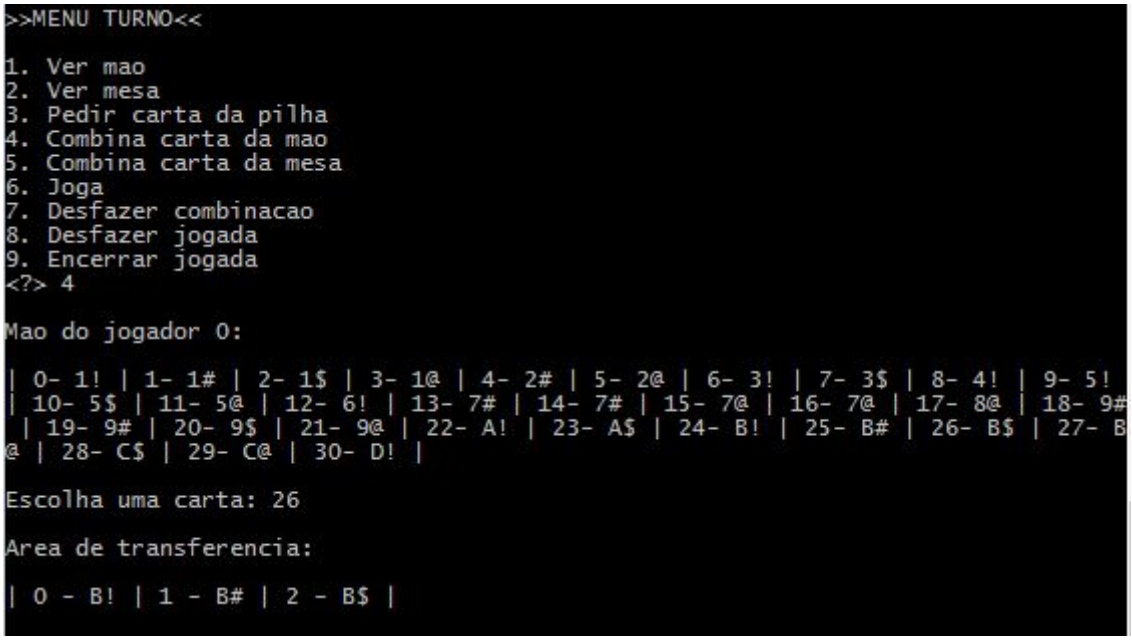
Permite que o jogador combine uma carta da mão para realizar a jogada. Essa carta só pode ser usada uma vez por jogada e ela é copiada para uma área de transferência, permanecendo ainda na mão do jogador. O jogador deve se referir a carta pelo seu índice.
Na primeira jogada de cada jogador, deve ser feita a partir da combinação de 3 ou mais cartas, cujo a soma resulte em um valor maior ou igual a 30, usando apenas as cartas da mão, caso não consiga formar tal sequência, ele deverá comprar e esperar a próxima jogada.



(escolha da primeira carta “ B! ”)



(escolha da segunda carta “ B# “)

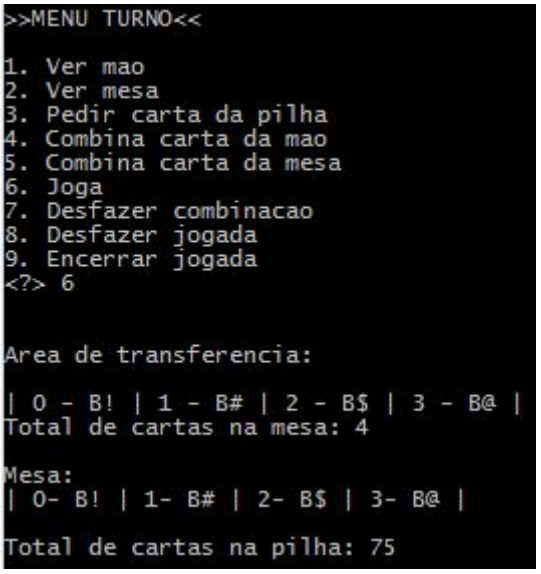


(escolha da terceira carta “ B$ “, sequência de 3 cartas(B!-B#-B$)

5. Combinar carta da mesa:

Permite que o jogador combine uma carta da mesa para realizar a jogada. Essa carta só pode ser usada uma vez por jogada e ela é copiada para uma área de transferência, permanecendo ainda na mesa. O jogador deve se referir a carta pelo seu índice.

6. Joga:



Se a jogada for válida, as cartas presentes na área de transferência devem ser transferidas para a mesa. As cartas são jogadas com o novo valor do grupo, que as identifica como uma combinação.
Usa a função *int* validaJogada(*pilha* * , *rhcp* * , *int* , *int*).

7. Desfazer combinação:

Limpa a área de transferência e habilita que o jogador refaça sua combinação.

8.Desfazer jogada:

Desfaz tudo que foi feito pelo jogador, retornando assim para as condições iniciais do turno. Para tal, é realizado no início do turno um backup das condições da mão e da mesa.

9.Encerrar jogada:

Se a mesa está válida e o jogador realizou pelo menos uma combinação, ele pode encerrar seu turno.

4.Outras informações:

4.1 Realizar combinação:

Para realizar uma combinação o jogador deve combinar as cartas da mão ou mesa na ordem correta. A área de transferência utiliza desse esquema para garantir jogadas válidas e facilitar o jogo.

4.2 Ordenação:

Foi usado o algoritmo selection sort para realizar as ordenações.

4.3 Embaralhar as cartas:

Foi usado o algoritmo de embaralhamento de Fisher-Yates.

4.4 Fim do jogo:

Ganha o jogo aquele que primeiro conseguir descartar todas as suas cartas.

Se a pilha de cartas acabar, ganha o jogador que possuir menos pontos, que são obtidos a partir do valor das cartas que restaram na mão do jogador. Nesse caso, o coringa vale 20 pontos. No caso de 2 ou mais jogadores com o mesmo número de pontos, o programa declara empate.