UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO

Aleixo Damas Neto - 10310975
Francisco Reis Nogueira - 11954374
João Augusto Fernandes Barbosa - 11953348

Relatório 5 - Qwirkle

São Carlos/SP
2020

1. Integrantes do grupo na ordem

k = 0; Francisco Reis Nogueira

k = 1; João Augusto Fernandes Barbosa

k = 2; Aleixo Damas Neto

2. Descrição e funcionalidade do programa

O programa foi subdividido em vários arquivos que se comunicam entre si, porém é possível dividir em 3 partes:

2.1 Começo e Entrada da ação do jogador.

2.1.1 Main

A função main do programa está no arquivo "*main.c*" e é nela que ocorre o principal do programa. Na função main, há as declarações das variáveis que são utilizadas no programa em um todo, ou seja, são variáveis que serão passadas para as outras funções. Uma observação a ser feita é a declaração de duas variáveis especiais, "*array_pecas*" e "*garbage_array*", essas duas variáveis auxiliam na distribuição de peças além de manter o controle de quantas peças estão no jogo, mecanismo que será explicado abaixo.

Após isso, a função main faz duas coisas essenciais: pedir o modo de jogo e pedir quantos e quem são os jogadores, o modo de jogo é salvo na variável "*game_mode*" que é utilizada para o modo cheat do jogo. Para pedir o nome dos jogadores, a função "*PainelJogadores()*" é chamada para inputar a quantidade de jogadores e o nome dos jogadores, uma observação a ser feita nessa função é que nela também é zerado alguns valores para todos os jogadores, como pontuação e uma flag "*choice*" que auxilia nas regras.

Depois de ter escolhido o modo de jogo e quem são os jogadores, o programa entrega para os jogadores as suas peças do jogo, para isso há uma função chamada "*NewPieces*" que realiza isso somente quando o jogo inicia. A distribuição de peças será explicada abaixo. Assim, a matriz é declarada e o loop principal do programa inicia, ele continua rodando até que sejam jogadas 108 peças, quando isso ocorrer o loop para, o jogo termina e as memórias são desalocadas com "*free*".

2.1.2 Loop principal

O loop do main imprime certas coisas toda vez que uma ação acontece, seja passar a vez, jogar ou trocar peças. Primeiro ele imprime o tabuleiro no estado atual dele, logo após ele chama a função "*PainelPrincipale*" que realiza a impressão de certos dados na tela, como: de quem é a vez,

como funciona o esquema das peças, quais são os comandos e quais são as peças que o jogador atual possui. Após isso a função "*PainelPontuacao*" imprime na tela os valores das pontuações de cada jogador.

A variável "*OqueDaVez*" mostra para o loop qual foi a última ação do jogador e qual deve ser a ação do loop, essa variável muda de valor sempre que uma nova ação é feita, porém, há dois casos especiais: "*OqueDaVez*" valendo 0 ou valendo 7. Se após a ação do jogador a variável tiver valor 0 significa que o jogador realizou alguma ação incorreta ou infringiu as regras do jogo e assim, antes de realizar uma nova ação o loop imprime que a última ação foi inválida. Se a variável tiver valor 7, simboliza que o jogador não possui a peça que ele deseja **jogar** ou **trocar**, mesmo com o modo cheat ligado o jogador não pode **trocar** peças que ele não possui, mas pode realizar jogadas com quaisquer peças.

Dessa forma, o código vai para função "*jogada*" e assim pode realizar sua ação conforme será explicado posteriormente. Quando o jogador volta da função "jogada" ele pode ter realizado algumas ações: ter jogado, ter passado a vez, ter trocado alguma peça, ou ter dado erro (quando o "*OqueDaVez*" fica com seus valores especiais). Se ele jogou, o "OqueDaVez" tem valor 1 e se ele passou, tem valor 2. Nisso ele entra em um switch case para os casos 1 e 2 em relação ao valor da variável "*OqueDaVez*".

O caso 1 simboliza que o jogador realizou uma ação de jogada válida e agora ele chama a função "*PecaPlacer*" para colocar a peça no tabuleiro. Logo após isso, a função "*LastMoveNeedsRealloc*" é chamada para verificar se, de acordo com a última posição jogada, precisa realocar o tabuleiro. A função "*LastMoveNeedsRealloc*" recebe os valores de linha e coluna da última jogada. Assim ele verifica se o tabuleiro precisa de realocação, ou seja, se a última jogada foi em uma das bordas do tabuleiro. Se for um lugar que precisa de realocação é chamada a função "*ReallocFunction*" e será passado à função os parâmetros de realocação, por exemplo: se for jogado no limite da coluna direita, porém a linha está dentro do limite, será passado os valores de [0 , 1]. Se fosse aumentar de linha e coluna os parâmetros seriam [1 , 1]. E por fim o contador de jogadas gerais "*contador_geral*" aumenta de valor, além de também incrementar às vezes que o jogador atual jogou.

O caso 2 simboliza que o jogador deseja passar a vez, assim é zerado a quantidade de vezes que o jogador atual jogou e sua escolha "*choice*" de acordo com as regras do jogo. Depois disso, é verificado em uma relação de "*if , else*" a vez dos jogadores, se a vez for igual ao valor da quantidade de jogadores menos um (começa no 0), a jogada irá retornar para o primeiro jogador, o

jogador zero. Se a vez (“*vez_do_jogador*”) for menor que a quantidade de jogadores menos um, é incrementado 1 à variável.

2.1.2 Jogadas e ações

As jogadas, e ações dos jogadores, são contidas no arquivo “*jogada.c*”. Na função “*jogada*” ele recebe vários parâmetros, pois todos são utilizados em alguma parte do código tornando-os necessários. É requisitado a entrada da ação do jogador, podendo ser entre JOGAR, TROCAR ou PASSAR. Após isso, ele verifica a primeira palavra da ação e redireciona para as respectivas funções, utilizando “*strcmp*”.

Uma observação a ser feita é a função “*SymbolConverter*”. Como o programa utiliza da variáveis inteiras para simbolizar o valor de cada peça, ex: (11 = ◆), essa função serve para converter o modelo de entrada para esse valor inteiro. Há dois switch cases, um referente a primeira parte da string peça e o outro referente a segunda parte da string peça. Por exemplo, a peça “*b4*” é um quadrado roxo, cujo valor inteiro é 24, assim a função verifica a primeira letra (b) e a segunda letra (4). Para a primeira letra é atribuído o valor da peça de 20, para a segunda letra é **somado** 4, tornando seu valor correto e por fim a função retorna esse valor inteiro.

Para a realização da troca, o jogador deve entrar o comando “***trocar peça***” assim o programa chama a função “*BoraTrocar*” que verifica a validade da peça na função “*SymbolConverter*”. Se a peça for válida ele chama a função “*TrocaPeca*” que será melhor explicado na parte de distrubuição. No entanto, ele verifica se o jogador possui a peça que será trocada e depois ele atribui uma nova peça ao jogador, porém antes de atribuir essa nova peça ele verifica se há mais de 3 peças iguais já jogadas ou com outro jogador.

Para passar a vez o jogador simplesmente deve escrever “**passar**” e a próxima jogada irá para o próximo jogador, conforme explicado no caso 2 do loop do main.

Para realizar a jogada o jogador deve escrever o comando “**jogar peça linha coluna**” e assim começa uma série de verificações para ver se a jogada está válida. Primeiro o programa verifica se a peça jogada é válida segundo a função “*SymbolConverter*”. Se a peça não for valida o programa já retorna como 0 e é classificado como ação inválida segundo o loop do main. Caso a peça seja válida, o programa chama a função “*WasMoveValid*”. Essa função verifica a linha e coluna que foi jogado, ela verifica se jogou dentro do tabuleiro, ou seja, não foi um valor não existente, e ela verifica se a jogada foi em um espaço “vazio” do tabuleiro. Por fim, o código realiza a última verificação somente se o modo cheat não estiver funcionando, pois, se o modo cheat estiver ligado, é possível jogar qualquer peça, de acordo com as regras. Se o modo cheat estiver desligado, o jogador só

pode jogar as peças que ele possui. Se todas essas verificações validarem a jogada, a função retorna valor 1 e assim **se não for a primeira jogada do jogo** o programa chama as verificações de regras que são explicadas em tópicos abaixo. Dessa forma, se as regras forem validadas o programa retornará ao loop do main, com o valor de retorno igual a 1 que fará com que entre no caso 1 do switch case.

2.1.3 Distribuição

A função de distribuição está dentro do arquivo “distribuicao.c” tem como funcionamento base os dois arrays: “array_pecas” e “garbage_array”. O “array_pecas” possui os 36 valores das peças, ele é utilizado para distribuir as peças do jogo, já o “garbage_array” possui todas as peças que algum jogador possui ou que já foram jogadas. Dessa forma é possível manter controle sobre o total de peças no jogo, visto que só pode conter 3 peças de cada tipo, no vetor “garbage_array” só pode existir três tipos de peças idênticas.

A distribuição é feita a partir da criação de dois loops, um para igualar a quantidade de peças do jogador [6] e outro para que os valores do “garbage_array” estejam de acordo com a quantidade de peças dentro do jogo. Para escolher uma peça é utilizado a função “*rand*” que foi alterada para gerar um número aleatório entre 0 e 35, os valores do vetor “*array_pecas*”, assim a função “*TrashTime*” é chamada de modo a colocar tal peça no vetor lixo ou proibir essa peça de ser entregue ao jogador porque já existe 3 peças iguais dentro do vetor lixo. Se a função “*TrashTime*” validar a peça ela é adicionada no lixo e entregue ao jogador.

A função “*TrashTime*” funciona primeiro percorrendo o vetor “garbage_array” com o valor que é passado como parâmetro em busca de valores idênticos dentro do vetor. Se ele encontrar um valor igual, um contador é incrementado. Case esse contador se igualar a 3, quer dizer que dentro do vetor já tem 3 peças iguais e essa peça atual é invalida, assim ele retorna 0, o que significa erro. Se esse contador não se igualar a 3 e o vetor todo for lido, significa que essa peça atual pode existir então ela é adicionada no vetor na última posição e retorna 1.

A função “*TrocaPeca*”, utilizada para realizar a troca da peça, funciona a partir da verificação da presença dessa peça no vetor da pessoa, depois disso ocorre a remoção de tal peça do vetor de lixo, ou seja, ela se torna reutilizável novamente, e uma nova peça é entregue ao jogador por meio da função “*rand*” e “*TrashTime*”. Assim, o valor da peça é entregue ao vetor de peças do usuário.

A função “*RefreshPecas*” faz com que após cada passagem de vez, é entregue ao usuário que passou a vez. Se o usuário realizou jogadas corretas nessa “vez dele”, significa que as peças que ele jogou estão com valores iguais a -1, assim elas não aparecem na parte de impressão e na função “*RefreshPecas*” ele modifica todas as posições do vetor que encontrar quando está como -1. Por fim, ele faz a verificação se a nova peça é valida conforme os requisitos do vetor de lixo e atribui para o jogador a nova peça.

2.2 Regras e Pontuação.

A análise para construção baseado no qwirkle seguiu os seguintes preceitos:

- Uma peça só pode ser cercada de peças da mesma forma, ou da mesma cor.
- Não pode haver duas peças iguais na mesma sequência.
- Se o jogador optou por fazer uma jogada seguindo determinada linha ou determinada coluna, ele só poderá mudar em sua próxima rodada.
- Não se pode jogar uma peça de cor diferente em uma sequência de mesma cor.
- Não se pode jogar uma forma diferente em uma sequência de mesma forma.
- A única peça que pode ser cercada de vazios será a da primeira jogada.

Com base nisso, iniciou-se a construção da sintaxe. Dentro das funções de verificações, é importante destacar que as regras se aplicam a partir da segunda peça jogada, já que a primeira poderá ser qualquer uma escolhida. A partir disso, todas as demais passam a valer.

Inicialmente é importante explicar o funcionamento do tabuleiro. Este é composto por valores inteiros apenas, sendo setados inicialmente como “0”. Caso o jogador faça a jogada de alguma peça, este valor é convertido para a respectiva associação em números. Por exemplo, inicialmente, a casa 0 0 tem seu valor 0. Se o jogador jogar “●” (círculo vermelho) neste local, ele passará a valer 35.

As associações seguem o seguinte formato: o primeiro dígito do int se refere a forma (um para losango, dois para quadrado, tres para círculo, quatro para estrela, cinco para cruz e seis para asterisco). O segundo dígito se refere a cor (um para azul, dois para branco, três para laranja, quatro para roxo, cinco para vermelho e seis para verde). Quando se escolhe uma peça, há a conversão da letra para o número e posteriormente a substituição do valor. Como citado, círculo vermelho terá o valor de “30” por ser circulo, e “5” por ser vermelho, obtendo-se 35.

Desta maneira, para a peça ser cercada apenas por valores válidos, fizemos uma função para analisar as peças adjacentes à peça jogada. Se a diferença dos valores adjacentes for maior que

cinco significa que a peça não é da mesma forma. Se o resto entre as divisões por 10 for diferente (se o segundo dígito for diferente) eles não são da mesma cor. Assim, se ambas as condições não forem satisfeitas, as peças ao redor são inválidas. Essa verificação ocorre inicialmente na função gameway.

Em seguida, a função tem-se a função same_seq. Ela analisa qual foi a escolha de jogada do jogador, nesse caso, se o jogador deseja seguir a sequência em linha ou em coluna. Após definida, qualquer jogada fora do local especificado será invalidada. Para tal, na primeira rodada de cada um dos jogadores, armazena-se a linha e a coluna escolhida. Na segunda jogada, comparamos a linha e coluna inicial com os novos valores. Caso a linha seja igual, uma variável “choice” receberá 1. Caso a coluna seja igual, a mesma variável receberá 2. Posteriormente, analisaremos os novos valores das novas linhas e colunas. Caso desrespeitem o valor anterior definido para choice, a jogada será invalidada.

A função away serve inicialmente para impedir bugs e trapaças. Para as demais funções, ignora-se as casas com valor 0, entretanto, não se pode jogar em um local completamente cercado por 0 - a não ser na primeira jogada. Assim, caso todas as casas ao redor da jogada sejam ou zero, ou o fim da matriz, a função definirá a jogada como inválida.

Além destas, a função gameway definiu literalmente o estilo de jogo, no caso, se a sequência será de cor ou de forma. Caso seja de forma, ela não aceitará uma cor diferente. Caso seja de uma cor diferente, não aceitará a mesma forma. Para tal, verificamos inicialmente as horizontais do local em que a peça foi jogada. Se as peças tiverem a mesma cor, a int gw será setada como um, caso tenham a mesma forma, gw será setado como 2. Caso aconteça de alguma peça desrespeitar o gw - ou seja, se houver uma peça em que a sequência é de cor (gw = 1), mas ela é de forma (gw = 2) - a função retornará o valor como invalido. Para percorrer toda a matriz, foi criado lin_verif, que percorre todos os valores de linha e coluna dependendo dos parametros mandados, até encontrar um 0 ou até ser o fim da matriz.

2.2.1 Pontuação

O cálculo da pontuação é feito no arquivo *“pontuacao.c”*, na função *“pontos()”*. A pontuação leva em conta 3 situações de “fim de vez” possíveis, a que nenhuma peça foi jogada, resultando em nenhuma alteração de pontos, a situação que uma peça foi jogada e a situação que mais de uma peça foi jogada.

Na situação de apenas uma peça jogada, o cálculo de pontos é feito analisando a casa da jogada nas 4 direções possíveis. Se na horizontal ou na vertical é detectada alguma peça adjacente,

elam passam a ser entendidas como sequência, contando-se a própria casa da jogada em cada uma das sequências por meio de uma flag.

Na situação de mais de uma peça jogada, é analisada a direção da sequência construída com as peças jogadas e é contabilizada a sequência final principal e todas as possíveis sequências perpendiculares, a cada peça jogada na vez, utilizando-se novamente uma flag para a casa analisada ser contada na sequência perpendicular.

2.3 Impressão do tabuleiro e Realocação

As funções de impressão e realocação do tabuleiro estão ambas no arquivo *"matrix_print.c"*. A função principal de impressão é a função *"matrix_print()"*, responsável por imprimir de modo formatado a matriz com as jogadas feitas até então. A formatação do print dos índices da matriz é feita pela função *"index_print()"*. A realocação, por sua vez, tem como função principal a *"ReallocMatrix()"*, responsável por aumentar a matriz do tabuleiro quando for chamada, dados dois comandos que dizem como ela deve ser expandida. Junto a essa função há a *"ShiftMatrix()"* responsável por deslocar as peças do tabuleiro se for necessário.

2.3.1 Princípio da matriz do tabuleiro

De início, é declarado na main um ponteiro duplo que apontará para a matriz responsável por guardar todas as jogadas feitas até então no jogo. São salvas também as variáveis de tamanho atual do tabuleiro em linhas e colunas e a sua origem relativa (onde na matriz está a casa (0,0) do tabuleiro) composta por duas coordenadas.

2.3.2 Funções de impressão do tabuleiro

A função principal de impressão utiliza de estruturas de *loop* para imprimir o tabuleiro formatado, alternando-se linhas de dados e linhas horizontais de separação.

De modo simplificado, a matriz é impressa com dois *loops* aninhados, o externo representando as linhas(i) e o interno as colunas(j). Sempre que *i=0* ou *j=0*, imprime-se o índice da coluna ou linha do tabuleiro, levando-se em conta o *i* ou *j* e a origem armazenada de modo que o índice (0,0) está na casa cujas coordenadas são as armazenadas na variável da origem.

O print dos índices é feito pela função secundária de print da matriz *"index_print()"*, que formata o espaçamento necessário para não deformar o tabuleiro, levando-se em conta os digitos que cada print vai ocupar.

Para os valores *i!=0* e *j!=0*, é impresso o símbolo que o valor na casa da matriz representa, utilizando-se a função *“printTab()”*.

2.3.4 Realocação

A realocação é fundamentada na utilização da função *“realloc()”*. A função principal possui todos os argumentos da função de impressão mais dois inteiros.

Os dois inteiros são os valores “add_row” e “add_col”, usados para se direcionar a realocação. Se “add_row” é 0, não é realocado mais nenhuma linha, se 1 é realocada uma linha embaixo, se -1 é realocada uma linha acima. Com “add_col”, se é 0, também não é realocada nenhuma coluna, se 1, é realocada uma coluna à direita, se -1, é realocada uma coluna à esquerda.

Na prática, a realocação é feita sempre com uma linha abaixo ou com uma coluna à esquerda. O que é feito nos casos de -1 é um deslocamento na direção da realocação, feito pela função *“ShiftMatrix()”*.

A função *“ShiftMatrix()”*, por sua vez, desloca os valores de uma coluna para a coluna à sua direita se *“add_col” = -1* e desloca os valores da uma linha para a linha abaixo se *“add_row” = 1*.

3. Links

a. **Repositório do github**
b. **Vídeo 1: solução em grupo**
c. **Vídeo 2: funcionamento do programa**