

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
SISTEMAS DE COMPUTAÇÃO

RELATÓRIO DO TRABALHO V

Rafael Dalonso Schultz
11800945
Ricardo Monteiro Barbosa
11800604

São Carlos
Julho de 2020

DESCRIÇÃO DA SOLUÇÃO

I. Recursos utilizados

A codificação do jogo tentou seguir os passos do procedimento de criação do jogo original: tabuleiro, peças, jogadores, jogada, dinâmica e pontuação. Ela se deu de forma organizada, por meio de diversos arquivos que ajudaram a estruturar o problema. A seguir, lista-se cada um dos arquivos (ou a pasta que os contém) e sua descrição.

- **.../headers**

Pasta que contém os ficheiros **.h**, cabeçalhos que organizam informações para cada um dos arquivos do programa. Fazem parte delas os arquivos-cabeçalho *blocks.h*, *board.h*, *buffer.h*, *colors.h*, *game.h*, *infoBag.h*, *infoBlock.h*, *infoBlockRelation.h*, *infoGame.h*, *infoPlayer.h*, *output.h* e *players.h*, cada um deles referindo-se aos arquivos abordados a seguir

- **main.c**

Arquivo que contém a função principal, responsável por chamar as funções essenciais ao bom funcionamento do programa

- **blocks.c**

Arquivo que contém as funções que lidam com as peças – tanto na sacola, quanto na mão do jogador. Algumas delas são: **short verifyBlock(Game *g, Block b)**, **short verifyPlayerHand(Player p, Block b)**, **void removeBlockFromHand(Player *players, short player_number, Block b)** e **void changeBlock(Game *g, Player *players, short player_number, Block b)**, esta última exemplificada em Código 1

```
void changeBlock(Game *g, Player *players, short player_number, Block b){
    removeBlockFromHand(players, player_number, b);

    short letter_pos = b.letter - 'A';
```

```

short number_pos = b.number - 1;

(g->bag.blocksControl[letter_pos][number_pos])--;
(g->bag.blocks_number)++;

for(short i = 0; i < 6; i++){
    if(players[player_number].tiles[i].letter == '\0'){
        players[player_number].tiles[i] = drawOneBlock(g);
        break;
    }
}

```

Código 1: Função trocadora de peças – o jogador solicita a troca e a chamada da função assim efetua, removendo a peça desejada da mão do usuário e pegando uma nova

- **board.c**

Arquivo que descreve o tabuleiro do jogo. Possui as funções **void createBoard(Game *g)** e **void eraseBoard(Game * g)**, esta última chamada ao fim da função principal *main*, para liberar o espaço de memória. Importante notar que, ao início da confecção do trabalho, o grupo tinha calculado um tamanho máximo possível para o tabuleiro, e iria seguir o método de inicializar uma matriz toda em zero com este número máximo nos limites, e mostrando apenas as peças jogadas. Entretanto, como não são muitas casas, o grupo percebeu que poderia utilizar *realloc* sem grandes problemas no código, evitando usos desnecessários de memória com uma matriz pré-definida

- **player.c**

Arquivo que lida com os jogadores, partindo de sua definição (quantidade, nome). Possui as funções **void deletePlayers(Game g, Player *p)** e **Player * initializePlayers(Game *g)**

- **game.c**

Arquivo que lida com a jogabilidade em si. Possui funções verificadoras, pontuadoras e de rodadas dos jogadores. Algumas delas: `int verifyEndGame(Game *g, Player *players)`, que verifica se o jogo já finalizou, `int makeMoviment(Game *g, Block b, int lin, int col, short *f_lin, short *f_col)`, que realiza um movimento possível, `int playerTurn(Game *g, Player *players, short player_number, char isCheatMode)`, que define a vez de um jogador e `void gameRounds(Game *g, Player *p)`, que inicia o jogo e mantém a alternância de jogadores em cada rodada e é exemplificada no Código 2.

```
void gameRounds(Game *g, Player *p){
    do{
        cheatModeMenu();
        char isCheatMode = fgetc(stdin);
        cleanBufferEnter();

        if(isCheatMode == 's' || isCheatMode == 'n') {
            isCheatMode -= 32;
        }

        if(isCheatMode == 'S' || isCheatMode == 'N'){
            short result = 0;

            int i = 0;
            while(!result){
                playerTurn(g, p, i, isCheatMode);
                i = (++i) % (g->n_players);

                if(g->bag.blocks_number == 0){
                    result = verifyEndGame(g, p);
                }
            }
            break;
        } else {
```

```

        invalidOption(0);
    }
} while(1);
}

```

Código 2: Verifica se o *cheat mode* deve ser ativado, limpa o *buffer* de entrada até um enter, verifica opções válidas e executa comandos até que o resultado do jogo seja definido. No caso de sua finalização, o laço de repetição é quebrado

- **buffer.c**

Arquivo que lida com a memória temporária do programa

- **colors.c**

Arquivo com os códigos de cores (representadas também por números)

- **output.c**

Arquivo que contém as funções que interagem com o jogador (*feedback* ao usuário)

II. Funcionalidade do programa

O funcionamento do programa segue a proposta do jogo norte americano, mas com algumas adaptações para simplificar a jogabilidade. No início do jogo, por exemplo, ao contrário do original, em que o primeiro jogador é aquele que possui maior linha inicial (e, no caso de todos terem o mesmo tamanho de linha, começa o jogador mais velho), o jogo em C é iniciado pelo jogador número 0, ou seja, o primeiro a escrever seu nome.



Figura 1: Tela inicial do jogo

O usuário preenche, então, o número de jogadores e seus nomes, pelos quais serão identificados em cada jogada, além de optar ou não pelo *cheat mode* (“modo trapaça”). Assim que preenchidos, é mostrado o tabuleiro inicial – de linha 0 e coluna 0, com espaço para alocar a primeira peça –, as peças de cada jogador e as opções para o jogador número 0:

```
QWIRKLE!!

Numero de jogadores: 2
Nome do Jogador #0: A
Nome do Jogador #1: B
Cheat Mode (S/N): N

TABULEIRO

  0
0  [ ]  0
  0

PEÇAS

- A: D1 B5 D1 B2 C1 B1
- B: C2 B4 B4 E1 F2 A4

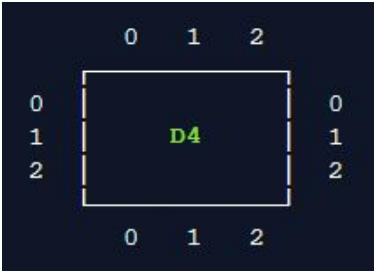
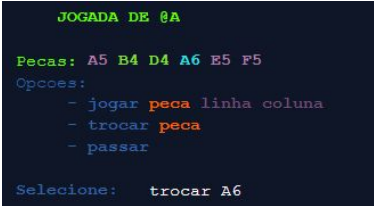


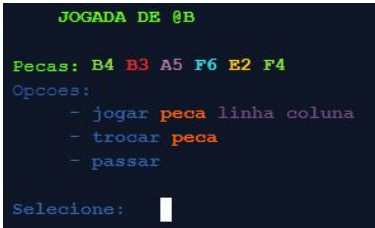
JOGADA DE 0A

Pecas: D1 B5 D1 B2 C1 B1
Opcoes:
- jogar peca linha coluna
- trocar peca
- passar

Selecione: 
```

Figura 2: Tabuleiro inicial e opções de jogada

O jogador 0, nomeado por A no exemplo, escolhe uma das opções:

Opção 1: jogar peça	Opção 2: trocar peça	Opção 3: passar
<p>O jogador escolhe a peça que deseja iniciar sua jogada, e, no primeiro caso, a única posição que ela poderia ocupar (0,0):</p>  <p>Figura 3.1: No exemplo, o jogador escolheu a peça D4 para iniciar o jogo</p>  <p>Figura 3.2: Tabuleiro expandido, possibilitando a escolha de posições para jogadas futuras</p>	<p>Insatisfeito com alguma peça em sua mão, o jogador executa o comando:</p>  <p>Figura 4.1: No exemplo, o jogador escolheu trocar a peça A6</p> <p>A peça é então trocada de modo aleatório por outra presente na “sacola” (segundo o jogo real):</p>  <p>Figura 4.2: Note que a peça A6 escolhida foi trocada pela peça B4, ao passo que as restantes foram mantidas</p>	<p>Esse comando é executado quando o usuário deseja finalizar sua jogada:</p>  <p>Figura 5.1: No exemplo, o jogador 0 (“A”) resolveu concluir sua jogada e passar para o jogador 1 (“B”)</p> <p>Em seguida, o próximo jogador recebe o tabuleiro atualizado com a jogada de A e essas mesmas três opções:</p>  <p>Figura 5.2: Opções de jogada do próximo usuário, que aparecem logo após a figura atualizada do tabuleiro</p>

A pontuação e a dinâmica do jogo foram feitas conforme as regras do original. A ordem da jogada individual segue a alocação de peças, pontuação da jogada e retirada de peças da “sacola” para manter 6 peças em mão. Não se pode alocar fichas individuais, todas devem estar “coladas”, isto é, formando filas entre si.

Além disso, tal como o original, o programa permite colocar múltiplas filas na vez do jogador, desde que tenham a mesma cor ou forma e mesma linha (tocam-se), e esse jogador é pontuado somando os valores unitários das peças existentes na fila mais a peça que colocou. Por fim, caso seja detectado um **QWIRKLE!**, isto é, o preenchimento completo de uma fila com seis cores ou letras, o usuário responsável por completar recebe um bônus de seis pontos.

O jogo continua até não houver mais blocos na bolsa. Então, continua até um dos jogadores não possuir mais nenhuma ficha em sua mão. Esse jogador recebe, também, um bônus de seis pontos. Finalmente, são verificadas as pontuações, e vence o jogador que mais pontuou.

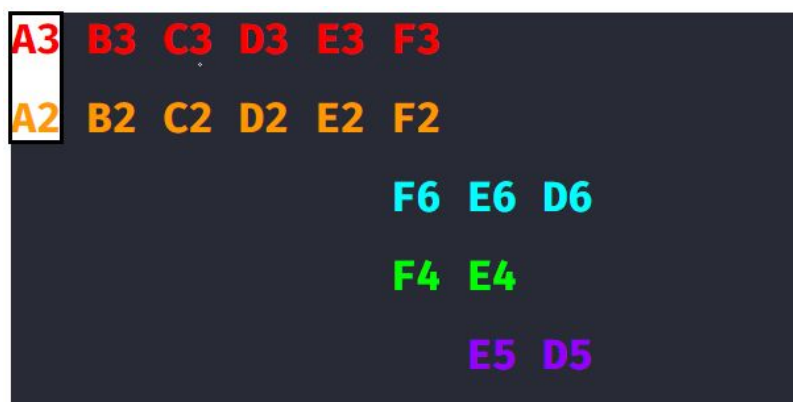


Figura 6: exemplo prático de funcionamento do programa. Nele, o jogador que colocou as peças A3 e A2 ganhou 26 pontos, da fileira laranja (+6), do quirkle na fileira laranja (+6), da fileira vermelha (+6), do quirkle na fileira vermelha (+6), e da linha vertical em branco (+2)

III. Tratamento de erros

Conforme solicitado pelo enunciado, o programa faz tratamento dos erros desde o início para responder a possíveis erros do usuário. Segue uma lista de algumas mensagens de erros presentes no código e seus objetivos:

- **Peça inserida inválida!**

Mensagem de erro recebida quando o jogador tenta alocar uma peça não existente (exemplo: usuário escreve **jogar D9 0 0**. A peça D9 não existe, portanto, o jogador recebe essa mensagem, e deve tentar novamente)

- **Você não possui essa peça!**

Mensagem de erro recebida quando o jogador tenta alocar uma peça que não possui em mãos

- **Posição inválida!**

Mensagem de erro recebida quando o jogador tenta alocar uma peça em uma posição inválida – seja uma casa já ocupada, seja uma posição da maior do que a matriz atual

- **Opção inválida!**

Mensagem de erro recebida quando o jogador não segue à risca as instruções de opção. Sem escrever na formatação correta, ele deve tentar novamente, até acertar a opção que deseja (exemplo: o usuário gostaria de jogar a peça F2 na posição 1 2, mas, em vez de escrever **jogar F2 1 2**, acaba por escrever **joga F2 1 2** ou **jogar peca F2 1 2**, formas erradas. Ele recebe a mensagem e tenta novamente, seguindo os padrões colocados)

- **Você pode realizar apenas um tipo de jogada por rodada!**

Mensagem de erro recebida quando o jogador tenta realizar mais de um tipo de jogada. Por exemplo, após ter realizado a jogada de uma peça, insere a opção de trocar uma peça, algo não permitido pelo jogo original. Ele recebe a mensagem de erro e deve escolher entre jogar novamente uma peça ou passar sua jogada ao próximo usuário

- **Movimento inválido!**

Mensagem de erro recebida quando o jogador tenta realizar um movimento não permitido pelas regras do jogo, como colocar em uma fileira de letras roxas uma letra verde, ou em uma fileira de letras A colocar uma letra B. Ele recebe a mensagem de erro e deve fazer um movimento correto

- **Não há mais peças para serem distribuídas!**

Mensagem de erro recebida quando a “sacola” não possui mais peças a serem distribuídas, próximo ao fim do jogo. Após isso, o jogador que não tiver mais peças na mão recebe os seis pontos bônus e o jogo termina

LINKS DO TRABALHO

[Repositório no GitHub](#)

[Vídeo da solução em grupo](#)

[Vídeo da apresentação do programa em funcionamento](#)