Gustavo Valandro da Rocha	00318514	Turma G
Lucas Klement Reinehr	00326910	Turma H

BOW AND ARROW

CONTEÚDOS DO REPOSITÓRIO

- Arquivos de código e *headers* escritos por nós, nas pastas *src* e *include*, respectivamente.
- Arquivos de texto, usados como fonte dos gráficos, e arquivos *.ncr* e *.pdc*, que contém atributos extras dos gráficos (cores, etc.) na pasta *materiais*.
- *Headers* da biblioteca PDCursesMod[1], na pasta *libs*, e uma *dll* do módulo *wincon* (Windows Console) da biblioteca compilada para computadores 64 bits[2], na raiz. Foi usado o *release* 4.2.0.
- Arquivos binários do jogo e do editor usado para criar os gráficos, compilados para Linux 64 bits e Windows 64 bits.
- Arquivos de projeto do Codeblocks, com alvos de compilação para Linux e Windows, além de uma Makefile.

COMPILAÇÃO E USO

As instruções assumem que o diretório de trabalho é a raiz do repositório. A versão de Linux assume que a versão do ncurses para *wide chars* e a biblioteca do servidor de janelas X11 estão instaladas. A versão de Windows requer que a *dll* do PDCurses esteja presente no mesmo diretório que o programa. Ao iniciar, o jogo procura os gráficos *.txt* na pasta *materiais* e, se não os encontrar, fecha com erro.

Windows: *$(CC) -o bow-and-arrow-windows src/*.c -Iinclude -lm -L. -lpdcurses -Ilibs*
Linux: *$(CC) -o bow-and-arrow-linux src/*.c -Iinclude -lm -lncursesw -lX11*

[1] A página do projeto pode ser encontrada em <https://www.projectpluto.com/win32a.htm>.
[2] Instruções para compilação com compiladores 32 bits estão disponíveis em <https://github.com/Bill-Gray/PDCursesMod/blob/master/wincon/README.md>. Arquivos de atributos *.pdc* podem não ser compatíveis com a versão de 32 bits.

BUGS CONHECIDOS

Foi observado que, em alguns casos, a versão de Windows falha ao tentar carregar os arquivos de atributos *.pdc*. Imaginamos que isso seja causado pelo programa tentar acessar esses arquivos enquanto são escaneados pelo antivírus. Geralmente, tentativas subsequentes de abrir o programa funcionam. Se continuarem falhando, basta mover ou deletar os arquivos *.pdc* da pasta *materiais*.

ESTRUTURA DO CÓDIGO

main.c

Carrega os materiais do jogo (gráficos, placar) e configura o terminal. Define o *loop* principal do jogo, que é executado até o jogo chegar ao fim. O jogo pode ter diversos estados: *MENU*, *PLACAR*, etc. Cada um tem uma função associada a ele, que é chamada em cada iteração do loop pela função *main*. Com exceção do estado *JOGO*, que recebe um arquivo próprio por ser mais complexo, cada função de estado é definida em *main.c*. Além de chamar a função do estado, o *loop* principal também faz o processamento de entradas, garante que o terminal tenha as dimensões corretas, calcula a passagem de tempo entre iterações, atualiza a tela e limita o FPS do programa, evitando o uso excessivo de CPU. Alguns estados têm estruturas associadas a eles; isso foi feito para limitar o *namespace* das variáveis usadas nessas estruturas.

Biblioteca curses (ncurses/PDCursesMod)

O jogo usa *ports* da biblioteca curses: PDCursesMod no Windows, ncurses no Linux. Essas bibliotecas são, em grande parte, intercompatíveis, e gerenciam a impressão de texto na tela e a leitura de entradas, entre outras funções do terminal. Todas as impressões feitas no terminal pelo programa são feitas em um *buffer* criado pela biblioteca. Essa buffer é enviado para a tela de uma vez só no final de cada iteração do *loop* principal, através da função *do_update*.

Configs.h

Contém constantes que podem ser usados por diversos arquivos. Constantes usadas por apenas um arquivo foram definidas no arquivo que as usa.

Jogo.c e Jogo.h

Define a função para a atualização de um quadro do jogo. O estado *JOGO* é dividido em diversos subestados: *EM_JOGO, PAUSADO*, etc. Cada um tem uma função associada a ele, chamada pela função *atualizarQuadroDoJogo*. Cada objeto do jogo (arqueiro, flecha, etc.) é uma instância da estrutura *OBJETO*, que contém informação como o tipo do objeto, sua posição, seu gráfico, etc.

Em cada quadro do jogo, cada objeto ativo em *vetObjetos* é atualizado, se movendo de acordo com sua velocidade e com o tempo passado desde o último do quadro. Dependendo do objeto, eles podem ter sua posição limitada à área do jogo, serem desativados se saírem dessa área, ou voltarem ao outro lado se saírem dessa área. Após isso, eles são desenhados. Essa etapa também realiza a checagem de colisão, pois a colisão dos objetos depende de seus gráficos. A checagem é feita com a ajuda dos vetores *posFlecha* e *posMonstros*, que guardam as posições em que esse objetos estão presentes.

Placar.c e Placar.h

Define uma estrutura para entradas no placar e funções para carregar arquivos de placar e inserir novas entradas neles.

Grafico.c e Grafico.h

Define uma estrutura que guarda informações sobre um gráfico (suas dimensões, tamanho na memória, imagem e outros atributos como cores, caracteres sublinhados, etc.) Define também funções para carregar gráficos de arquivos em disco e

descarregá-los da memória. Finalmente, define funções para desenhar esses gráficos na tela, realizando checagem de colisões no processo, se necessário.

TerminalIO.c e TerminalIO.h

Definem funções para inicializar o terminal e novas janelas dentro dele, funções de conveniência para formatação e processamento de texto (especificamente UTF-8) e funções para processar a entrada do usuário. A função *processarEntrada* processa todo o *buffer* de entrada do curses, e é usada uma vez em cada iteração do *loop*. A função *teclaPressionada* usa o estado instantâneo do teclado para checar se alguma tecla está pressionada, e pode ser chamada a qualquer momento durante a execução do programa. Ela é usada quando o pré-processamento da entrada realizada pelo terminal não é desejada, como, por exemplo, durante o jogo, quando o movimento do jogador deve ter velocidade constante e não variar de acordo com a velocidade de entrada do terminal.

Timer.c e Timer.h

Definem funções para medir a passagem de tempo real ao longo da execução do programa, usadas para que o movimento dos objetos do jogo ocorra com velocidade constante. Também definem uma função para interromper a execução do programa por uma quantidade de tempo, usada para limitar o uso de CPU.