UNIVERSIDADE FEDERAL DO RIO GRANDE DO RIO GRANDE DO SUL

TRABALHO FINAL DE ALGORITMOS E PROGRAMAÇÃO:
MOUSE TRAP

WELLINGTON MACHADO DE ESPINDULA, RAFAEL ANTÔNIO VENTURA TREVISAN

PORTO ALEGRE
2018

INTRODUÇÃO

O presente trabalho objetiva o desenvolvimento de um jogo similar ao clássico “Mouse Trap” do Atari®. Para o desenvolvimento, foi utilizada a linguagem de programação C e a biblioteca Conio 2.0 (LAI, H et al., 2018) para criar as interfaces gráficas. Assim, a partir desse trabalho, tem-se por objetivos paralelos, praticar todos os conceitos aprendidos na disciplina de Algoritmos e Programação - bem como a linguagem C -, exercitar as habilidades de pesquisa dos integrantes e a habilidade de trabalho em equipe. Por fim, nota-se que, por ser um jogo, esse trabalho torna a tarefa do exercício do desenvolvimento de software mais divertida e interessante.

Descrição básica do jogo:

O jogo desenvolvido chama-se “Lil Mouse”. Nesse jogo, o usuário controla um rato, que move-se a partir da entrada do usuário nas setas - ou nas teclas W, A, S e D -, e deve coletar todo o queijo de um labirinto. Existem também 4 gatos, que andam em direções aleatórias e tentarão “comer” o rato - isto é, farão o rato perder uma vida caso entrem em contato com ele. Entretanto, existem ossos no labirinto, que quando coletados pelo rato, o transformam temporariamente em um cachorro; assim, o rato em modo cachorro pode derrotar os gatos e mandá-los para as suas posições iniciais. Existem também portas móveis nesse labirinto que mudam de posição quando o jogador clica “B/b”, mudando a configuração do mapa.

Quando o jogador coleta todos os queijos de uma fase, ele passa para a próxima, e esse processo repete-se até que todas as fases acabem. Cada queijo coletado dá ao jogador 10 pontos, e cada gato derrotado da 50. O objetivo do jogo é conseguir a maior pontuação possível antes de perder as suas 3 vidas, ou mesmo conseguir passar todas as fases enquanto houverem vidas.

Figura 1.Interface do jogo desenvolvido. Fonte: Autoria Própria, 2018.

OBJETIVOS

Gerais:

Desenvolver jogo similar ao mouse trap, na linguagem C, utilizando funções, ponteiros, estruturas e arquivos de texto e binários.

Específicos:

- Desenvolver estruturas que suportem o desenvolvimento.
- Criar uma interface visual em modo texto (caracteres).
- Carregar o mapa do labirinto a partir de um arquivo texto nomeado “nivel01.txt”, onde o valor numérico indica o nível correspondente do mapa.
- Desenvolver uma interface que possibilite a interação do jogador com o jogo a partir de entrada no teclado, visto que tal interação não deve ser bloqueante; isto é, não deve atrapalhar as animações correntes do jogo.
- Desenvolver a movimentação dos objetos do jogo no mapa.
- Criar menu para realizar as funções de criar um novo jogo, salvar jogo, carregar jogo antigo e sair
- Desenvolver funções que possibilitem o armazenamento do estado de um jogo e que possa carregar um estado de jogo já salvo.
- Criar sistema de pontuação.
- Desenvolver interface sonora.

METODOLOGIA

Para cuprir os objetivos almejados pelo trabalho, foi utilizada a metodologia de *sotware* de prototipação - isto é, um modelo evolucional de construção de *software* na linguagem de programação C.

Ademais, para a construção da interface, fez-se necessário o uso da biblioteca externa Conio2.

Inicialmente, foram desenvolvidas as seguintes estruturas:

- Estruturas básicas (Figura 2):
    - POSICAO: Estrutura que armazena a posição de um elemento em um mapa;
    - DIRECAO: Estrutura que armazena a direção de movimento de um objeto (utilizado principalmente para a movimentação dos gatos);

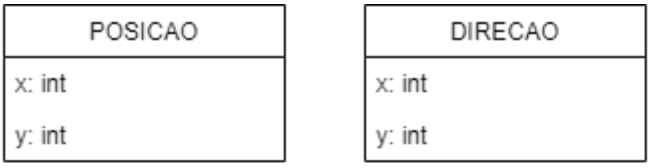

Figura 2. Diagrama de estruturas básicas do jogo. Fonte: Autoria Própria, 2018

- Objetos do jogo (Figura 3):
    - STR_RATO: Estrutura que armazena todas as propriedades de um rato no jogo, essas são: sua posição inicial no mapa de um nível, posição atual deste, quantidade de vidas e direção.
    - STR_GATO: Estrutura que armazena todas as propriedades de um gato no jogo, como nota-se na figura 3.

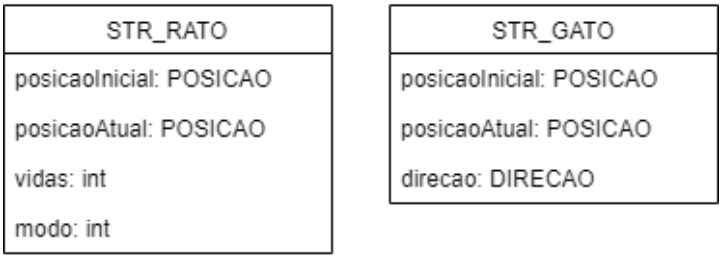

Figura 3. Diagrama de estruturas que representam os objetos do jogo. Fonte: Autoria Própria, 2018.

- MAPA (Figura 4): Estrutura utilizada para armazenar quais objetos inanimados do jogo estão disponíveis em uma determinada posição (a posição é determinada externamente por posições x e y de um vetor do tipo dessa estrutura, MAPA). Utiliza-se tal estrutura com uma matriz para assim armazenar o mapa carregado do arquivo de texto de cada nível.

Figura 4. Diagrama da estrutura MAPA. Fonte: Autoria Própria, 2018.

- DADOS (Figura 5): Estrutura utilizada guardar e carregar todas as informações de estado de um jogo.

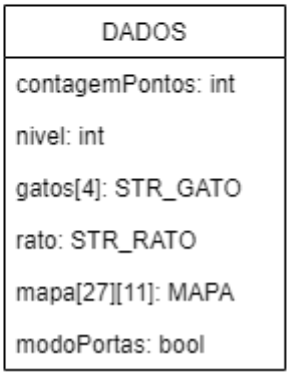

Figura 5. Diagrama da estrutura DADOS. Fonte: Autoria Própria, 2018.

Variáveis de ambiente:

Foram utilizadas as seguintes variáveis para implementar o ambiente do jogo:

- Variáveis de ambiente básicas:
  - **int contagemPontos:** Conta quantos pontos o usuário atual realizou.
  - **int nivel:** Armazena o nível em que o jogo se encontra.
  - **STR_GATO gatos[NUM_GATOS]:** Vetor de gatos do jogo. “NUM_GATOS” é uma constante definida no programa cujo valor é 4.
  - **STR_RATO lilmouse:** Rato do jogo.
  - **MAPA mapa[LIMITE_X][LIMITE_Y]:** Mapa do jogo que armazena o que existe em cada uma das posições do mapa, exceto rato e gatos - as posições dos gatos e do rato é dado por suas variáveis de posição atual no jogo. “LIMITE_X” e “LIMITE_Y” são constantes definidas no programa cujos valores são, respectivamente, 27 e 11.
- Variáveis de estado:
  - **bool modoPortas:** Variável que indica em que modo as portas se encontram abertas ou fechadas
  - **bool exibicaoMenu:** Variável que indica se o menu encontra-se exibido.
- Variáveis para alocação de objetos na tela:
  - **int xInicialMapa:** Variável que indica quanto deve-se deslocar o mapa para a direita para o centralizar.
  - **int yInicialMapa:** Variável que indica quanto deve-se deslocar o mapa para baixo para o centralizar.
  - **int alturaTela:** Variável que indica a altura da tela, em número de caracteres.
  - **int larguraTela:** Variável que indica a largura da tela, em número de caracteres.

Funções do jogo:

Para o desenvolvimento do jogo, implementou-se as seguintes funções:

- Funções de mudança de estados de jogo:
    - **void novoJogo():** Função que inicializa as variáveis de ambiente para a inicialização de um novo jogo.
    - **void mudaNivel():** Função que realiza as operações para a mudança de nível;
    - **void ganhou():** Função chamada quando o usuário ganha para exibir que o usuário ganhou na tela.
    - **void perdeu():** Função chamada quando o usuário perde para informar que perdeu.

- Funções de salvamento e carregamento de estado do jogo em arquivo de texto:
    - **void carregaJogo(DADOS dados):** Define as variáveis de ambiente utilizando os valores da variável do tipo DADOS passada como parâmetro.
    - **void getDados(DADOS *dados):** Utiliza as variáveis de ambiente para popular uma variável do tipo dados
    - **bool carregarDados(char *nomeArquivo, DADOS *dados):** Abre arquivo binário correspondente ao nome passado como parâmetro e armazena os dados de um jogo anterior em uma variável do tipo DADOS. Retorna verdadeiro se foi possível realizar as operações.
    - **bool salvarDados(char *nomeArquivo, DADOS dados):** Salva dados de um jogo em um arquivo binário. Retorna verdadeiro se foi possível realizar as operações.

- Funções de mapa
    - **bool carregarMapa(int nivel):** Carrega mapa de arquivo de texto dado o nível informado.

- **void exibeMapa():** Exibe o mapa na tela utilizando as variáveis de ambiente.

- Funções de locomoção:
  - **void moveGatos():** Função que percorre os gatos do vetor de gatos e os move.
  - **void moveGato(STR_GATO *gato):** Função que move um gato dada sua posição e sua direção. Caso a movimentação não seja possível, modifica a direção do deslocamento para uma que seja possível.
  - **bool moveRato(STR_RATO *rato, DIRECAO direcao):** Move o rato - caso esta movimentação seja possível - dada sua posição e direção do movimento. Retorna verdadeiro se foi possível mover o rato.
  - **bool existeParede(POSICAO posicao):** Retorna se existe parede no mapa na posição entrada.
  - **bool existeQueijo(POSICAO posicao):** Retorna se existe queijo no mapa na posição entrada.
  - **bool existeOsso(POSICAO posicao):** Retorna se existe osso no mapa na posição entrada.
  - **bool existePorta(POSICAO posicao):** Retorna se existe porta no mapa na posição entrada.
  - **bool existeGato(POSICAO posicao):** Retorna se existe gato no mapa na posição entrada.
  - **bool existeRato(POSICAO posicao):** Retorna se existe rato no mapa na posição entrada.
  - **void resetaGatos():** Função que percorre os gatos do vetor de gatos e os move para a posição inicial.
  - **void resetaGato(STR_GATO *gato):** Move gato para sua posição inicial do mapa.
  - **void resetaRato(STR_RATO *rato):** Move rato para sua posição inicial do mapa.

- **STR_GATO *retornaGato(POSICAO posicao):** Dada a posição, retorna um ponteiro para a variável gato que ocupa esta posição.
- **void colisaoRatoGato(STR_GATO *gato, STR_RATO *rato):** Dado um gato e um rato colidindo, realiza as operações de diminuir a vida de um gato, bem como reseta o gato e o rato no mapa.

- Funções de manipulação de tela
  - **void moveCursorMapa(POSICAO pObj, int xObjTela, int yObjTela):** Função cujo objetivo é, dada posição do objeto nas estruturas internas do programa, posicionar o cursor na posição que este deverá aparecer na tela - visto que o mapa na tela apresenta deslocamento para centralização e utiliza dois caracteres em x e dois em y para exibir um objeto. Os inteiros *xObjTela* e *yObjTela* são parâmetros que informam, respectivamente, quantas casas a mais o cursor deve se mover no eixo *x* e no eixo y.
  - **void exibeTela(int backgroundColor, int textColor, POSICAO p, char x1, char x2, char y1, char y2)**: Função genérica que exibe na tela um objeto do mapa dada sua posição. Para tanto, deverá ser informado com que cor de fundo esse objeto será exibido no mapa da tela, a cor do texto que deve ser exibido, e os caracteres que representam o objeto em x e em y.
  - **void exibeObjeto(POSICAO posicao):** Dada a posição, exibe o objeto que a ocupa.
  - **void exibePortaTela(POSICAO p):** Exibe uma porta na tela, dada sua posição.
  - **void exibeRatoTela(STR_RATO rato):** Exibe um rato na tela, dada sua posição.

- **void exibeGatos():** Função que percorre os gatos do vetor de gatos e os exibe.
- **void exibeGato(STR_GATO gato):** Exibe gato na tela, dada sua posição.
- **void moveRatoTela(STR_RATO rato, POSICAO pAnterior):** Executa rotina de movimentação de rato na tela, dado um rato e sua posição anterior. Assim, pode-se excluir o que havia na tela na posição anterior e modificar a tela para exibir o rato na nova posição.
- **void moveGatoTela(STR_GATO gato, POSICAO pAnterior):** Executa rotina de movimentação de gato na tela, dado um gato e sua posição anterior. Assim, pode-se repôr na tela o que havia na tela na posição anterior e modificar a tela para exibir o rato na nova posição.
- **void exibeMenu():** Exibe menu superior, pausando o andamento do jogo.
- **void escondeMenu():** Remove menu superior da tela, dando continuidade ao jogo.
- **void exibeMenuPrincipal():** Exibe menu inicial do jogo.
- **void exibePontuacao():** Exibe informações de pontuação, vida e teclas de atalho na tela.
- **void atualizaPortasTela():** Assim que modificada a posição das portas, atualiza a posição dessas na tela.
- **void excluiTela(POSICAO p):** Função genérica para remover elemento da tela, dada sua posição.

- **Funções auxiliares**
  - **void setDirecao(DIRECAO *direcao, int x, int y):** A partir de um ponteiro de uma variável do tipo direção, altera-se a direção de modo que esta esteja no intervalo [-1, 1].
  - **int retornaDirecaoAleatoria():** Retorna uma direção aleatória entre -1 e 1.

- **bool foramTodosQuiejosComidos():** Retorna se todos os queijos do mapa foram comidos.

Funcionamento do programa:

O jogo foi implementado de forma a realizar o fluxo presente na figura 6.

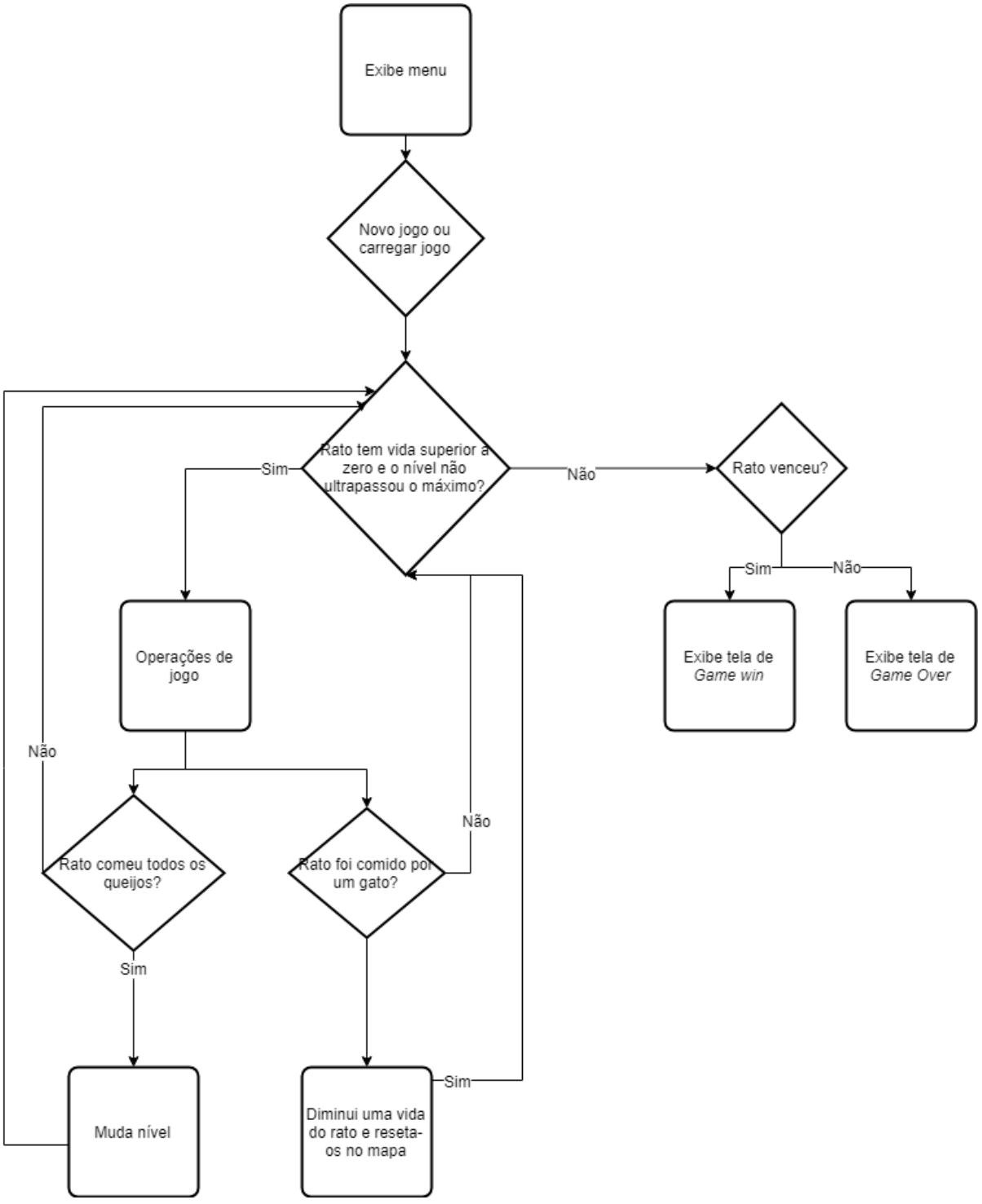

Figura 6. Diagrama de fluxograma do jogo. Fonte: Autoria Própria. 2018.

RESULTADOS

Como resultados, foi obtido um jogo desenvolvido na linguagem de programação C que cumpriu todos os objetivos almejados com o projeto.

CONCLUSÕES

Através desse trabalho foi possível perceber as dificuldades de estruturação de programas mais complexos , assim como a importância da manutenção de boas práticas de programação para que o programa seja mais fácil tanto de entender quanto de modificar.

REFERÊNCIAS

LAI, H et al. **CONIO 2.0.** Disponível em: <https://github.com/Fernando-Lafeta/Biblioteca-Conio-2>. Acesso em 21 de novembro de 2018.