INSTITUTO DE INFORMÁTICA, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CIÊNCIA DA COMPUTAÇÃO – ALGORITMOS E PROGRAMAÇÃO

João César de Paula

João Pedro Silveira e Silva

Relatório de desenvolvimento do jogo *Mouse Trap* para o trabalho final

PORTO ALEGRE

2018

INSTITUTO DE INFORMÁTICA, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CIÊNCIA DA COMPUTAÇÃO – ALGORITMOS E PROGRAMAÇÃO

Relatório de desenvolvimento do jogo *Mouse Trap* para o trabalho final

Projeto para o desenvolvimento de uma adaptação do jogo *Mouse Trap* para conclusão da disciplina de Algoritmos e Programação na Universidade Federal do Rio Grande do Sul, UFRGS.

Nomes: João César de Paula, João Pedro Silveira e Silva

PORTO ALEGRE

2018

Lista de ilustrações

Sumário

1 O Jogo

Neste trabalho foi desenvolvido uma adaptação do jogo Mouse Trap na linguagem C para exercícios dos conteúdos aprendidos em aula. O jogo possui uma dinâmica parecida com o famoso *PacMan*. Nele o jogador assume o papel de um rato e tenta escapar de diversos ratos enquanto se alimenta dos queijos. Para evoluir de nível e mudar de mapa é necessário capturar todos os queijos presentes no mapa. O jogador inicia com três vidas e cada vez que é capturado por um rato perde uma delas voltando a sua posição inicial.

Dentro do *Mouse Trap* é possível capturar ossos. Ao capturar um osso o rato se “transforma” em um cachorro onde passa a poder capturar os ratos. Isto lhe garante 50 pontos extras por cada captura. Cada queijo capturado gera 10 pontos e a pontuação se mantem entre os níveis.

Umas das maiores diferenças entre o *Mouse Trap* e o *PacMan* é a possibilidade de se mover portas. As portas abrem e fecham modificando os caminhos do mapa e podendo inclusive prender tanto o rato quanto os gatos.

Nesta implementação do jogo também é possível salvar jogos em andamento, e criar novos mapas em formato de texto.

2 Funcionalidades

O jogo foi criado com base nas seguintes funcionalidades implementadas:

- Exibição do mapa com as dimensões 27x11, sendo gerado a partir de um arquivo de texto.
- Implementação sem *delays*, de forma que o jogo possuí resposta imediata em termos de percepção humana para cada comando do usuário.
- Início no mapa de nível um, sendo possível evoluir de nível ao comer todos os queijos do mapa.
- Interações com o usuário através das teclas TAB, M ou m, abrem o menu, setas direcionais ou A, a, W, w, S, s, D, d, movimentam o rato dentro do mapa, b ou B para abrir ou fechar as portas.
- Um labirinto pode ter um número variado de portas, gatos e ossos. É recomendado, mas não obrigatório, o uso sete portas, quatro gatos e quatro osso.
- Os gatos se movimentam de forma a perseguir o rato pelo menor caminho possível. Quando o rato está no modo cachorro os gatos devem tentar se afastar do rato.
- As portas possuem dois estados: abertas ou fechadas. Quando pressionadas as teclas B e b as portas devem mudar de estado caso seja possível. As portas que não puderem se movimentar ficarão no mesmo estado. Os mapas devem ser planejados para possibilitar o movimento das portas, não devendo haver uma parede na diagonal inferior direita de cada porta. Dentro do mapa no arquivo de texto as portas são desenhadas no estado fechado.
- Se o rato for devorado e o jogador ainda possuir vidas extras o rato deve retornar a sua posição inicial no mapa. Os gatos também devem voltar as suas posições iniciais e o número de vidas do rato sofre o decréscimo de um.
- O jogo possui um placas com a pontuação atual do jogador, o número de gatos no jogo e a vida atual do jogador. Para cada queijo comido se adiciona dez pontos e para cada gato devorado se adiciona 50 pontos.
- Quando um rato é devorado ele retorna para a sua posição inicial e fica imune, porém imóvel, por três segundos.

- O modo cachorro ativado quando o usuário captura um osso dura cinco segundos.
- O jogo possuí três mapas implementados, porém podem ser criados mais mapas e adicionados a pasta “Levels” no mesmo diretório do executável.
- O menu do jogo é exibido ao se pressionar as teclas M, m ou TAB. Ele possuiu as seguintes opções:
    - “N”: inicia um novo jogo. Quando o usuário seleciona esta opção, um novo jogo é iniciado, a partir do primeiro nível e toda a pontuação e número de vidas do jogador também é resetado.
    - “C”: carrega um jogo salvo. Quando pressionado é aberto um menu com uma lista de jogos salvos. O usuário pode selecionar um jogo ou sair do menu. Caso selecione um jogo ele deve ser carregado sobre o jogo corrente e o menu deve ser fechado.
    - “S”: salvar jogo. Quando pressionado é criado um arquivo binário com o jogo atual dentro da pasta “SavedGames” no diretório do executável. O arquivo possuí a estrutura: “saved_dd_mm_aaa_hh_mm_ss”, onde *dd* representa o dia, *mm* o mês, *aaa* o ano, *hh* as horas, *mm* os minutos e *ss* os segundos em representação numérica de quando o jogo foi salvo. Ao se salvar o jogo o menu é fechado e o jogo retornado.
    - “Q”: finaliza o programa. O jogo é finalizado sem ser salvo.
    - “V”: voltar. Fecha o menu e volta ao jogo.
- O jogo deve requisitar o nome do usuário e manter ele salvo, inclusive caso o jogo atual seja reaberto.

3 Tecnologias

Foi escolhida a biblioteca Conio para criação da representação gráfica do jogo pela sua simplicidade e ao mesmo versatilidade. Isto viabilizou maior tempo de dedicação para a lógica do jogo e uma representação de fácil utilização.

Após a escolha da biblioteca e da analise de *game plays* para melhor compreensão do jogo foi iniciada a busca pelos conhecimentos necessário para a viabilização do jogo. Para isto foi analisado o material de apoio deixado pelos professores em aula, também foram utilizados forúns na internet como o StackOverFlow e sites com conteúdo extra e documentação de funções da biblioteca Conio.

Um exemplo do site utilizado foi *ProgrammingSimplified* (‘https://www.programmingsimplified.com/c/conio.h/’) que possui uma lista de funções da biblioteca junto com suas respectivas documentações. Outro exemplo é o *TutorialsPoint* (https://www.tutorialspoint.com/c_standard_library/) utilizado para pesquisa sobre manipulação de arquivos em C.

Também foi utilizado o material disponibilizado no Moodle pelos professores.

4 A Estruturação do Projeto

Com o intuito de evitar que o jogo fosse estruturado apenas em um arquivo que possuiria um número elevado de linhas foi pesquisado formas de se dividir o projeto em diferentes arquivos.

Durante a fase de pesquisa com diferentes IDEs (Integrated Development Environment, Ambiente de Desenvolvimento Integrado) escolhemos utilizar o formato de projeto *Console Application* do CodeBlocks pelo conhecimento prévio da ferramenta.

Foram utilizados os formatos de arquivo *Header*(.h) e *Sources*(.c) e o projeto foi dividido em três subpastas, que conforme **Ilustração 1** são:

"Utils": contém funções com que executam as funcionalidades do jogo;

"Consts": contém constantes que são utilizadas em mais de um arquivo dentro do projeto;

"Structs": contém os *Headers* com as definições de estruturas utilizadas em mais de um arquivo do projeto;

Dentro da subpasta "Consts" existem os seguintes arquivos:

"Boolean.h": define as constantes #TRUE e #FALSE com os valores um e zero respectivamente. Para facilitar reconhecimento de valores inteiros utilizados como booleanos;

"GamesStartData.h": define constantes para o nível e a quantidade de vidas.

"InputCharactersCode.h": Contém constantes para os valores inteiros das teclas utilizadas pelo usuário para realizar ações no jogo. Como exemplo temos a "TAB_CODE" que possui o valor 9 e a W_CHAR_CODE com o valor 87;

"MapCharacters.h" e "MapCharacters.c": definem as letras que presentam cada item no jogo, como a mouseCh que possui o valor "R" e representa o rato no vetor do mapa;

"MapColor.h": define constantes para as cores do mapa do jogo, como a WALL_COLOR que representa a cor das paredes do mapa e possui o valor um;

"MapDimensions.h": possui constantes que representam valores dimensionais do mapa, como o número de linhas e colunas definidos pelas constantes MAP_LINES e MAP_COLUMNS.

Para gerenciar os dados de forma mais organizada, utilizamos as seguintes estruturas:

“POSITION” (Position.h): representa uma posição dentro do mapa e possui dois inteiros “line” e “column”;

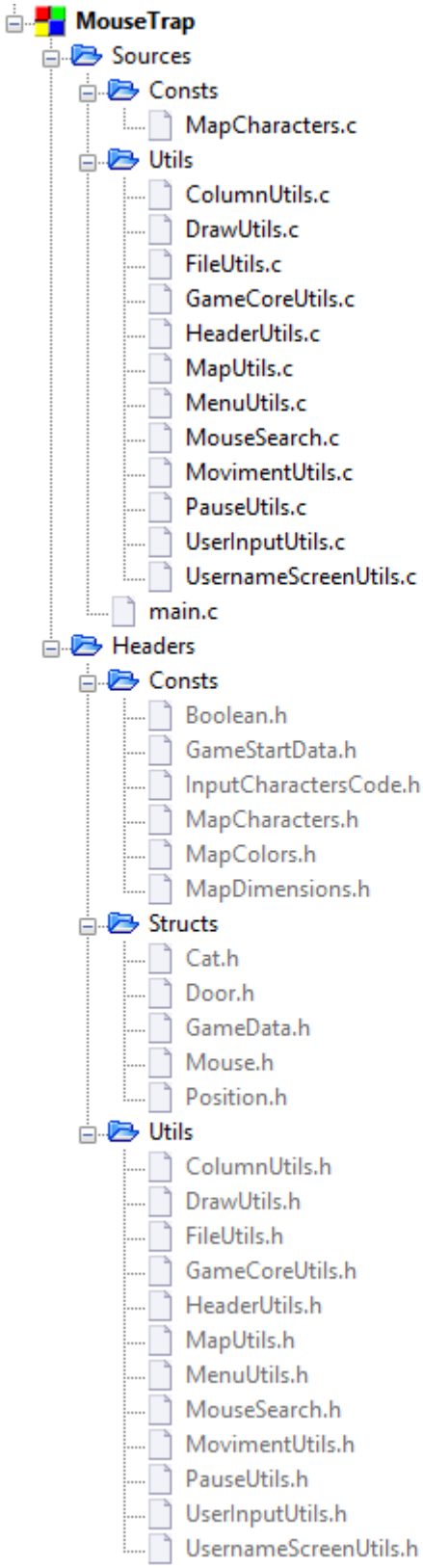
“CAT” (Cat.h): responsável pela representação de um gato no jogo. Possui uma posição atual, uma posição inicial, um “char” “overlaid” que é responsável por armazenar o item ao qual o rato se sobrepôs durante sua movimentação e um ponteiro direcionado para um outro rato na memória. Através desde ponteiro é gerada uma lista de gatos que pode ter variado número de elementos. Cada gato aponta para um próximo gato na lista;

“DOOR” (Door.h): Estrutura que representa uma porta no jogo. Possui uma posição (tipo Position), um “char” “overlaid” que representa um item ao qual a porta de sobrepôs ao se locomover e um inteiro booleano “opened” que mostra se a porta está aberta (um ou TRUE) ou fechada (zero ou FALSE);

“MOUSE” (Mouse.h): Representa o rato no jogo. Possui a posição atual e inicial do rato, um “char” “overlaid” com função igual ao “overlaid” do tipo Door, um tipo “clock_t” que guarda o horário em que o gato entrou no modo cachorro, o número de vidas, a direção atual ao qual o rato está andando e um booleano “isDog” que representa se o rato está no modo cachorro;

“GAMEDATA” (GameData.h): Representa um estado do jogo. Possuindo diversas informações sobre a execução do jogo. Algumas das informações mais importantes são um tipo “Mouse”, um ponteiro para o tipo “Cat” que aponta para o primeiro rato da lista de gatos e um ponteiro do tipo “Door” que aponta para a primeira porta da lista. A estrutura também armazena o mapa do jogo, uma matriz com dimensões 27 e 11 do tipo “char” ;

Ilustração 1 – Estrutura do projeto



Fonte: Autoria própria.

5 O Mapa

O mapa do jogo é estruturado em um arquivo de texto. Este mapa deve ficar no mesmo diretório do arquivo executável, dentro de uma subpasta “Levels” com a seguinte estrutura de nome: nivelN.txt, onde o “N” representa o número do nível (1,2,3,4,5...).
O texto do mapa é formado por 297 caracteres, sendo escritos em 11 linhas com 27 caracteres por linha. Podem ser utilizados os seguintes caracteres:

‘X’: representa uma parede;

‘M’: representa o rato, deve ser único;

‘Q’: representa o queijo;

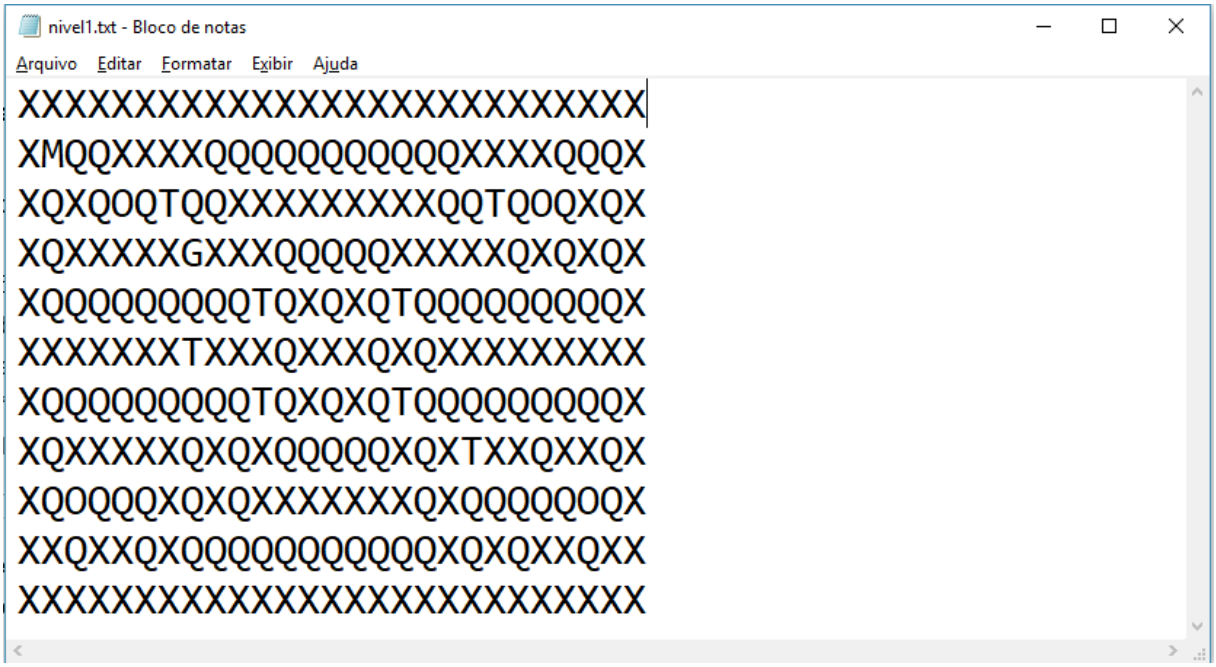
‘T’: representa uma porta, ao posicioná-la no mapa ela estará fechada, e é necessário que sua diagonal direita inferior possibilite seu movimento não podendo ser um ‘X’;

‘O’: representa um osso que transforma o rato em cachorro;

‘G’: representa um gato;

Também é possível utilizar espaços em branco, porém o recomendado é preenche-los com um queijo para o mapa possuir a maior pontuação possível e dificuldade a sua finalização.

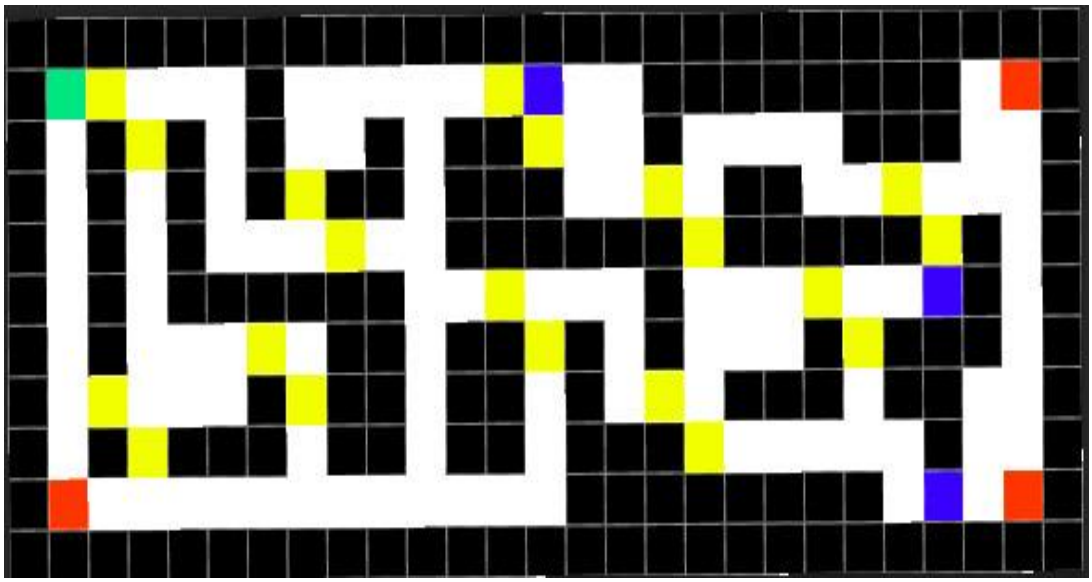
Ilustração 2 - Modelo de mapa.



Fonte: Autoria própria

A **Ilustração 2** representa um exemplo de mapa com sete portas e um gato. Estes números podem variar. No exemplo dado o arquivo representa o primeiro nível conforme o valor de ‘N’ em seu nome após a palavra nível.
Durante o desenvolvimento os mapas foram criados primeiramente em programas de edição para facilitar sua visualização e planejamento, conforme **Ilustração 3**.

Ilustração 3 – Mapa em desenvolvimento



Fonte: Autoria própria

O primeiro passo do desenvolvimento foi a leitura de um mapa em formato de texto e montar a conversão para uma matriz de duas dimensões, uma de tamanho vinte e sete e a outra de tamanho onze.
As funções de leitura e modifica de arquivo estão dentro do arquivo FileUtils.h e FileUtils.c, sendo as principais:
“void readMap(int ln, int col, GAMEDATA *data)”: recebe o tamanho do mapa e a estrutura do jogo, realiza a leitura do mapa e salva no ponteiro do tipo GAMEDATA.
“int saveGame(GAMEDATA data)”: recebe a estrutura com o estado do jogo e salva em um arquivo binário gerado a partir da data atual.

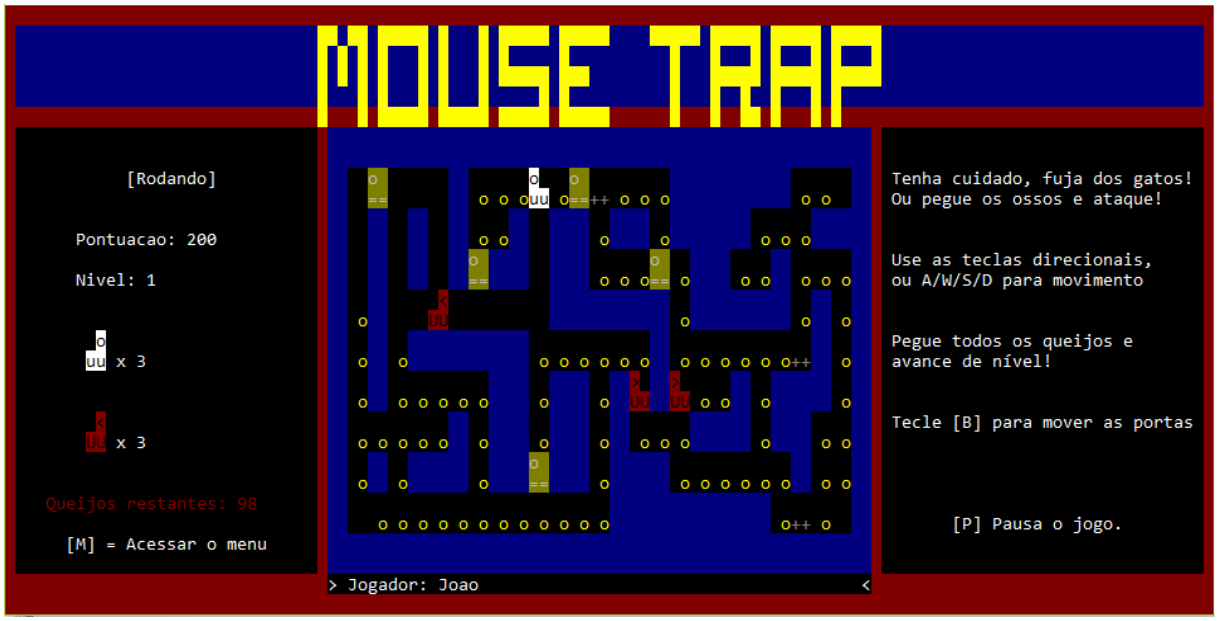
“int loadGame(GAMEDATA* data)”: gera um menu com opções de jogos salvos para o usuário, salva na estrutura do jogo o estado do jogo salvo selecionado, podendo

apenas retornar sem abrir um novo jogo, e retorna um inteiro informando o sucesso, falha ou cancelamento na operação.

6 Interface Gráfica e Telas do Jogo

Com a leitura do mapa finalizada e os dados convertidos para dentro da estrutura GAMEDATA se iniciou a conversão da matriz para uma representação gráfica. As funções base de desenho estão nos arquivos “DrawUtils.h” e “DrawUtils.c”.

Ilustração 4 – Interface gráfica



Fonte: Autoria própria.

A **Ilustração 4** mostra uma das telas principais do jogo onde são mostradas as informações atuais do jogo na coluna da esquerda, dicas do jogo na coluna da direita, mapa do jogo sendo executado no centro, nome do jogador no rodapé e título do jogo no cabeçalho.

Os itens do mapa são representados em um bloco de quatro caracteres, dois de altura e dois de comprimento. Outras estruturas do jogo, como os menus, o cabeçalho com o título e as colunas com as informações segue padrões de tamanho diferente. Seguem as funções criadas para a exibição do mapa e de outros itens em tela:

> “drawMono“: recebe uma posição inicial para o desenho (linha e coluna), um comprimento, um vetor de letras e posições coloridas, uma cor de fundo e uma cor para o caractere. Baseado no vetor de posições coloridas ele colore o fundo dos caracteres. Exemplo: { {0, 1}, {1, 1} } irá gerar um rato ou um gato que possui os dois blocos de baixo coloridos e apenas um dos de cima (no caso desde vetor o lado da direita seria colorido). A cor de fundo é dada como

entrada na função. O vetor de letras posiciona as letras ao fundo de cada bloco com a cor passada como parâmetro na função. O desenho inicial na posição inicial e se entende pelo comprimento passado de entrada. Para item do mapa, como citado anteriormente, o tamanho é 2x2. Para as letras que escrevem “Mouse Trap” o padrão é 5x5;

“drawMultiColor”: Versão melhorada da função anterior, mas que necessita de modificações para suprir todas as necessidades do jogo. Ela recebe a posição inicial, linha e coluna, do desenho, a altura e o comprimento, um vetor de inteiros que representam as cores de cada bloco, um vetor de caracteres que serão posicionados ao fundo de cada bloco e uma cor padrão para todas os caracteres de fundo. Exemplo: { {1, 1} {1,1} } irá gerar uma parede, pois os quatro blocos serão coloridos com a cor 1 que representa o azul;

6.1 Mapa, Colunas Laterais, Cabeçalho e Rodapé

As funções que geram os itens do mapa estão nos arquivos “MapUtils.h” e “MapUtils.c”. Existem diferentes funções para representar diferentes itens do mapa, porém todas utilizam a função “drawMono” ou “drawMultiColor” da “DrawUtils.h”.
As funções relativas a desenhar o cabeçalho com o título do jogo estão disponíveis nos arquivos “HeaderUtils.h” e “HeaderUtils.c”.
Funções criadas nos arquivos “ColumnUtils.h” e “ColumnUtils.c” desenham as colunas laterais e inferiores, além de exibir as informações e as dicas do jogo. Mais adiante serão apresentadas algumas funcionalidades das colunas laterais e inferior.

6.2 Captura Nome de Usuário

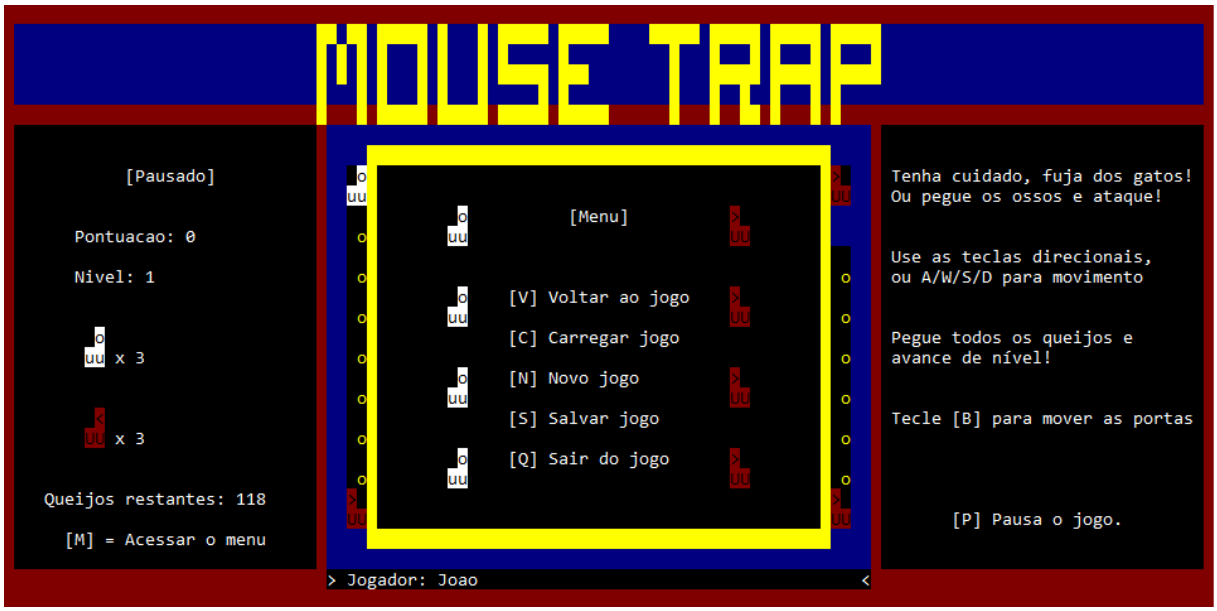
Ilustração 5 – Requisição nome de usuário.

Fonte: Autoria própria.

Após o desenho do mapa foi criada a funcionalidade de capturar o nome do usuário. Para isto foram criadas funções nos arquivos “UsernameScreenUtils.h” e “UsernameScreenUtils.c”. Quando o jogo é iniciado as funções responsáveis são chamadas e é gerada a tela conforme **Ilustração 5** requisitando a entrada do usuário.

6.3 Menu

Ilustração 6 – Menu aberto.



Fonte: Autoria própria.

Com o mapa representado passamos para a criação do menu principal, mostrado na **Ilustração 6**. O menu possui as funções: (V) Voltar ao jogo, (C) Carregar jogo, (N) Novo jogo, (S) Salvar jogo e (Q) Sair do jogo.

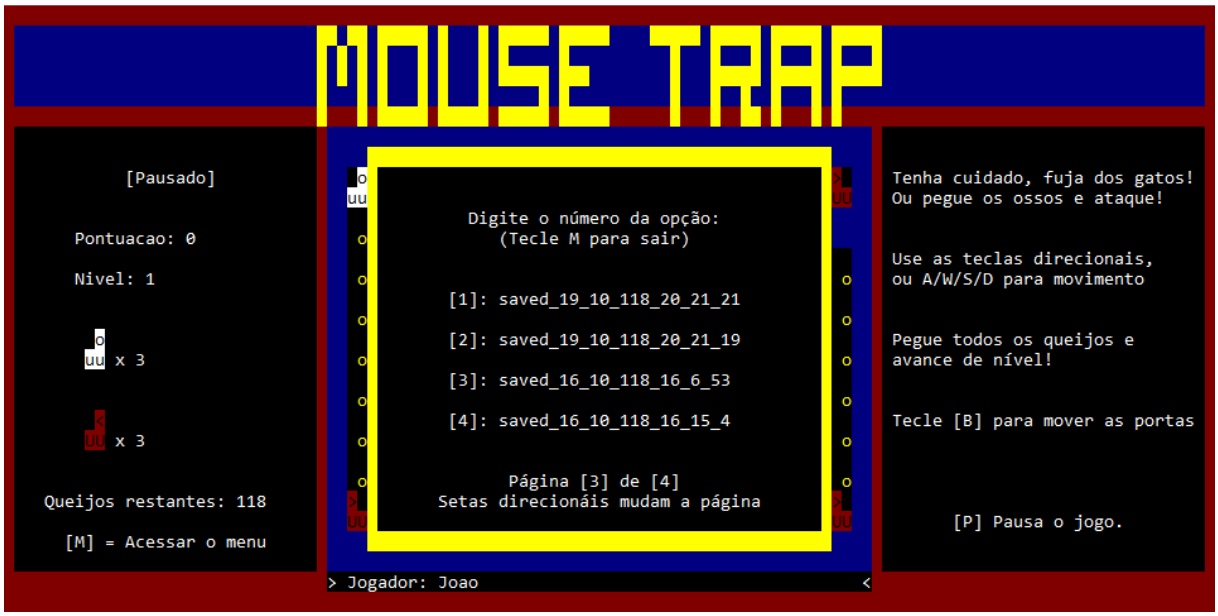
O menu foi criado para ser carregado na tela apenas quando a tecla “M”, “m” ou TAB fosse pressionada. Toda a lógica do menu foi criada nos arquivos “MenuUtils.h” e “MenuUtils.c”.

Quando o menu é aberto o jogo permanece pausado e ao ser fechado ele aguarda uma entrada do usuário para reiniciar. Algumas das principais funções que geram o menu são:

- “void openMenu(GAMEDATA* data)”: recebe um ponteiro para o estado do jogo e inicializa o menu.
- “void executeMenuChoice(GAMEDATA *data)”: aguarda uma entrada do usuário e toma uma decisão em resposta. Para interpretar a entrada do usuário a função utilizada os códigos de teclas definidos em “InputCharactersCode.h”.
- “void drawMenu()”: utiliza das funções dos arquivos “DrawUtils.h” e “DrawUtils.c” para desenhar um menu na tela.

6.3.1 Carregar Jogo

Ilustração 7 – Carregar jogo.



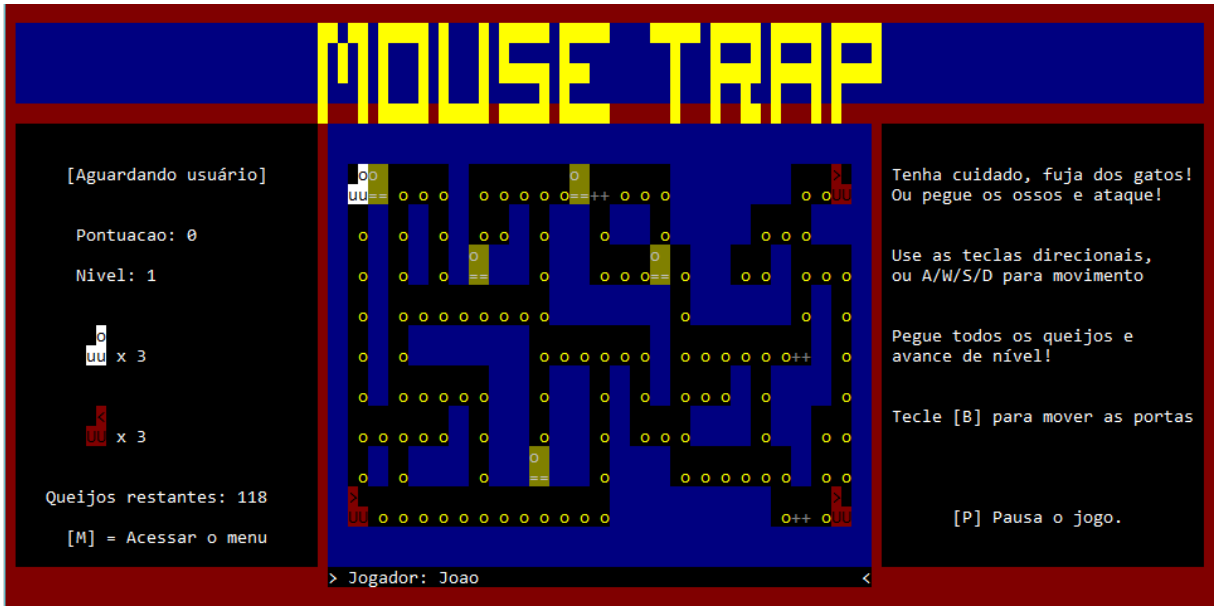
Fonte: Autoria própria.

Ao ser pressionada a opção “C” do menu é carregada a tela mostrada na **Ilustração 7** para a seleção do jogo salvo a ser carregado.

A tela foi desenvolvida de forma paginada para permitir a exibição de diversos arquivos com jogos salvos. Atualmente a tela está configurada para exibição de quatro itens por página, para modifica é necessário apenas mudar o valor de uma definição do programa. As setas direcionais modificam a página e as opções são carregadas como números inteiros de um até o tamanho da paginação.

Para sair do menu é necessário utilizar a tecla “M” ou “m”. Ao selecionar um jogo o menu é fechado e o jogo é imediatamente carregando, porém fica à espera de uma entrada do usuário para iniciar conforme é mostrado na **Ilustração 8** dentro da coluna esquerda, com a seguinte frase: “Aguardando usuário”.

Ilustração 8 – Aguardando usuário.



Fonte: Autoria própria.

6.4 Pausar Jogo

Ilustração 9 – Jogo Pausado

Fonte: Autoria própria.

Para pausar o jogo é necessário pressionar as teclas “P” ou “p”. Ao se pausar o jogo é carregado uma *modal* informando o estado do jogo e o atalho para voltar ao jogo conforme **Ilustração 9.** As funções que definem as funcionalidades relativas a pausa do jogo estão nos arquivos “PauseUtils.h” e “PauseUtils.c”.

É importante ressaltar que ao voltar ao jogo, pressionando novamente “P”, o jogo aguarda uma nova entrada do usuário para continuar, assim como ao carregar novo jogo ou ao sair do menu. Assim não há perigo da tela de pausa desaparecer e os gatos já começarem a andar.

Outra modificação em tela notada ao se pausar o jogo é a informação “Pausado” em vermelho na coluna direita. Basicamente este campo indica o estado atual do jogo.

7 Execução do Jogo

O jogo corrente se passa dentro da função “startGameCore” dentro dos arquivos “GameCoreUtils.h” e “GameCoreUtils.c”. Dentro destes arquivos estão localizadas diversas funções utilizadas para a execução do jogo, como:

- “void startGameCore(GAMEDATA* data)”: mantem o looping de jogo até o usuário sair do mesmo. Ela se repete constantemente e é a principal responsável por esperar as estradas do usuário durante o jogo, iniciar o movimento do rato e dos gatos e executar as funcionalidades do jogo conforme entrada do usuário.

- “void updateLevel(GAMEDATA* data)”: atualiza o nível do jogo. Ela é responsável por ler o chamar as funções que leem o mapa do próximo nível presentes em “FileUtils.h” e “FileUtils.c” e reiniciar o estado do jogo.

- “void verifyClash(GAMEDATA* data)”: verifica se houve contato entre os gatos e o rato e realiza as modificações no estado do jogo.

- “void restartWithNewLife(GAMEDATA* data)”: reinicializa as posições no mapa e as devidas informações do jogo quando o rato perde uma vida.

7.1 Movimentação

As funções relativas a movimentação do rato estão nos arquivos “MovimentUtils.h” e “MovimentUtils.c”.

Toda vez que o usuário pressionar uma tecla direcional ou uma tecla entra A, W, S e D o sentido de movimentação do rato irá mudar. A movimentação é feita com base no sentido da movimentação atual do rato contido no estado do jogo.

Nestes arquivos também estão contidas as funções responsáveis pela movimentação dos gatos.

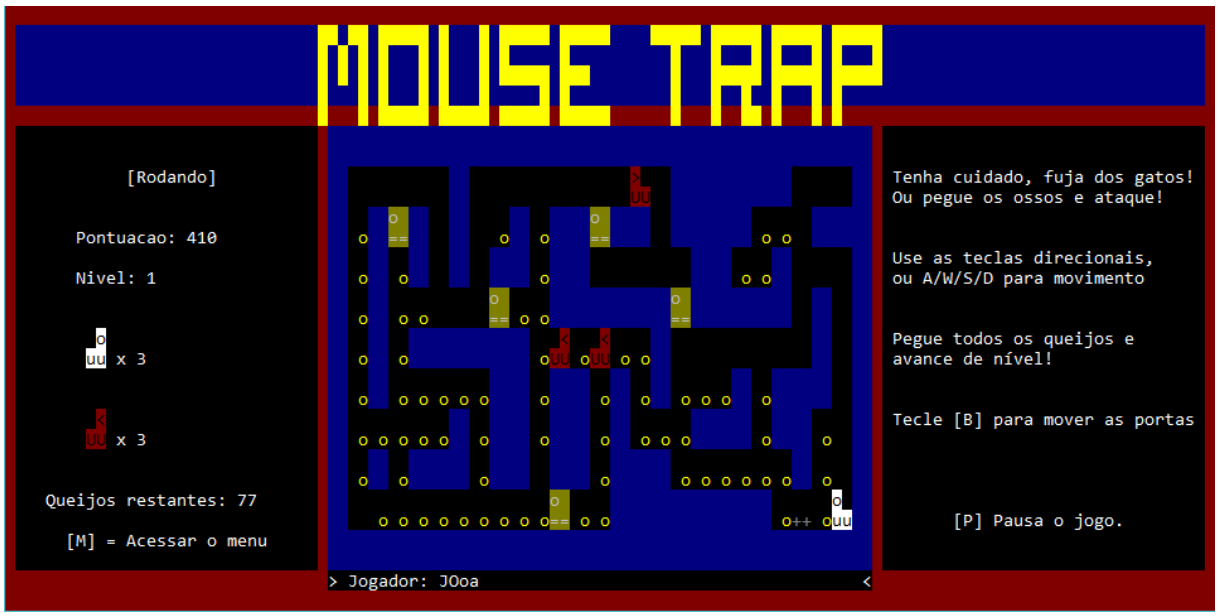
A movimentação de cada gato é feita baseada na matriz de mesmo tamanho do mapa do jogo “mapDistances”, presente na estrutura GAMEDATA. Esta matriz possui todas as distâncias dos pontos caminháveis até o rato. Cada gato toma a direção com a menor distância, todos baseados no mesmo vetor. Quando o rato está no modo cachorro os gatos tomam a direção com a maior distância.

A função “mapDistances” presente nos arquivos “MouseSearch.h” e “MouseSearch.c” percorre de forma recursiva todos os caminhos possíveis do jogo até o rato. Assim é gravado na matriz “mapDistances”gravando a distância de cada posição no mapa até

o rato, começando nos blocos próximos de distância um e indo até as últimas posições possíveis. Essa recursividade se inicia pelo rato e continua por todos os pontos de movimento possíveis entre direita, esquerda, cima e baixo.

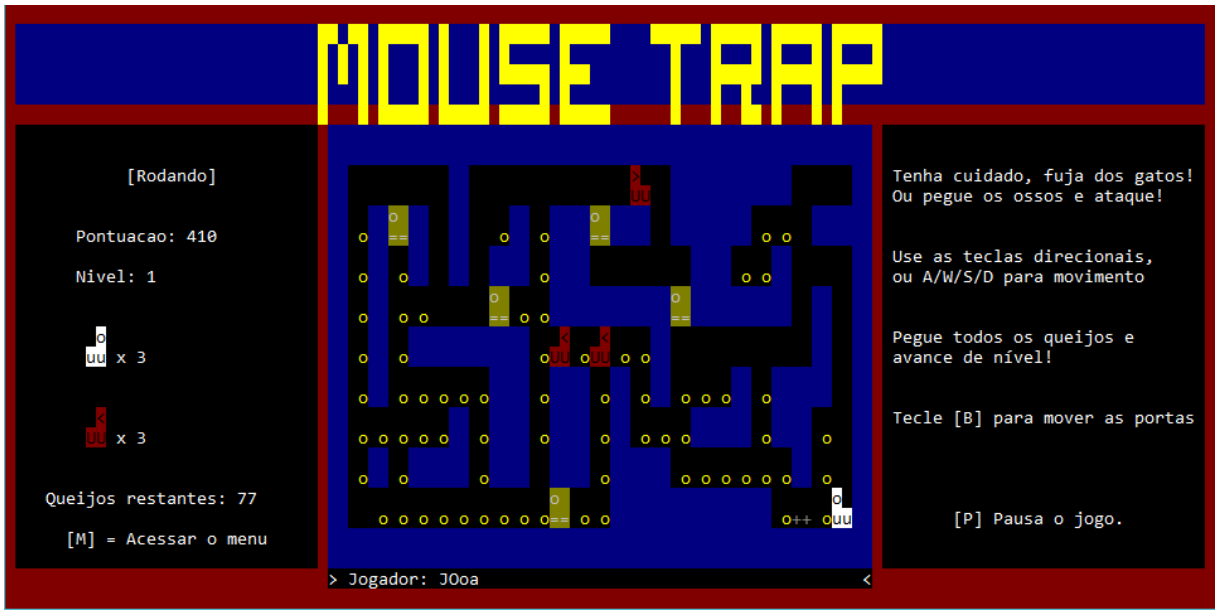
Nas ilustrações 10 e 11 é mostrado a movimentação de todos os ratos no jogo dentro de um curto período de tempo.

Ilustração 10 – Movimento dos gatos um.



Fonte: Autoria própria.

Ilustração 11 – Movimento dos gatos dois.



Fonte: Autoria própria.

7.2 Movimentação das Portas

A última das movimentações desenvolvidas por completo foi a movimentação das portas que seguem o padrão de deslocamento de uma casa para a direita e uma para baixo (caso elas estejam fechadas) ou uma casa para a esquerda e uma para cima voltando ao estado inicial (caso elas estejam abertas).

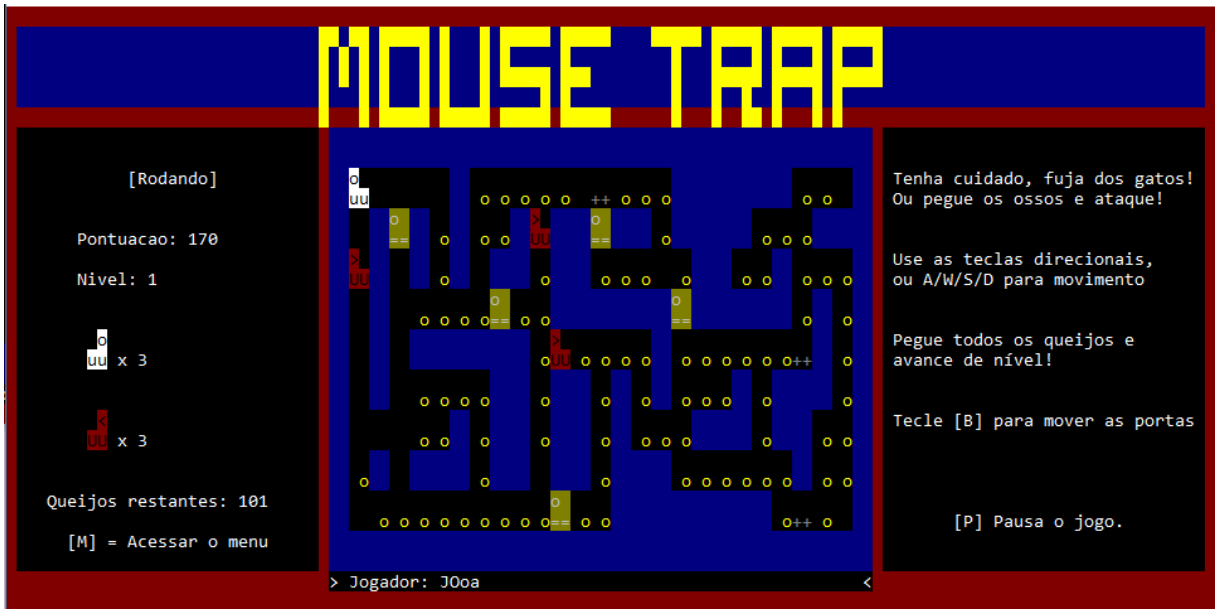
Cada porta se movimenta individualmente. Caso uma porta esteja impossibilitada de se movimentar, seja pela presença de um rato ou de um gato no seu ponto destino, ela não impede as outras portas de se movimentarem, mas ela permanecerá parada no próximo movimento para retornar a sincronia com as outras portas.

A movimentação das portas pode ser verificada nas ilustrações 12 e 13, sendo que na 12 elas estão fechadas e na 13 abertas.

Ilustração 12 – Portas fechadas.

Fonte: Autoria própria.

Ilustração 13 – Portas abertas.



Fonte: Autoria própria.

Conclusão

O presente trabalho apresentou os principais aspectos do desenvolvimento do projeto final da disciplina de Algoritmos e Programação baseado no jogo MouseTrap, desde as definições de estrutura, tecnologias e funcionalidades até sua implementação.
Durante a fase de pesquisa foi aprofundado e revisto o conteúdo dado em sala de aula, além de serem adquiridos novos conhecimentos para melhor implementação do jogo.