Conteúdo

1. Resumo

Neste trabalho desenvolvemos uma versão simplificada de um jogo de computador bastante conhecido, o Campo Minado. Tendo em vista a finalidade didática da execução deste projeto, buscamos utilizá-lo como plataforma para pôr em prática variadas técnicas e ferramentas de planejamento e desenvolvimento de software. O projeto foi executado por uma equipe de quatro alunos no período de 2 de junho à 1 de julho.

2. Introdução

Neste trabalho desenvolvemos uma versão simplificada de um jogo de computador bastante conhecido, o **Campo Minado**. Tal jogo, conhecido também como **Minesweeper**, tem origens na década de 60, e consiste de um tabuleiro, inicialmente coberto, onde aleatóriamente são semeadas bombas, e demarcadas dicas numéricas à respeito do número de bombas nas redondezas de uma casa. O jogador deve a cada jogada indicar uma casa, e marcá-la como minada, ou "pisar" sobre a mesma, revelando seu conteúdo e, se vazia, de sua vizinhança. O objetivo é demarcar a localização de todas as bombas presentes no tabuleiro, utilizando as dicas reveladas, sem que se "pise" sobre uma casa minada.

Tendo em vista a finalidade didática da execução de tal projeto, buscamos utilizá-lo como plataforma para o exercício de variadas práticas em planejamento e desenvolvimento de software. Com isso estabelecemos um particionamento de etapas e tarefas na equipe, seccionando o projeto em módulos, cada um sob a responsabilidade de um integrante. Debatemos e definimos também um conjunto específico de ferramentas a ser utilizado, visando facilitar a comunicação e o desenvolvimento em grupo.

Dividiu-se a execução do projeto em três fases principais, **Planejamento**, **Desenvolvimento** e **Teste**.

3. Planejamento

Na primeira fase, o Planejamento, que representa o início do projeto como um todo, o foco esteve em definir as bases para as fases de desenvolvimento e teste, o que inclui a determinação de um **Cronograma**, a determinação de uma **Estrutura do Projeto**, a divisão das tarefas de desenvolvimento, e a escolha das **Ferramentas** em software utilizadas pela equipe.

3.1 Cronograma

Iniciando no dia 2 de junho, organizamos a primeira reunião da equipe de projeto, onde determinamos os primeiros tópicos a serem determinados, e a partir destes formulamos o cronograma apresentado abaixo.

Quarta-feira	Quinta-feira	Sexta-feira	Segunda-feira	Terça-feira
June 1st **1** Início da Fase de Planejamento	2nd **2** Reunião: Cronograma e estrutura do projeto	3rd **3**	6th **4** Reunião: Estrutura do projeto e ferramentas	7th **5** Reunião: Estrutura do projeto
8th **6** Reunião: Estrutura do projeto	9th **7**	10th **8**	13th **9** Início da Fase de Desenvolvimento	14th **10** Reunião: Escrita de código em grupo
15th **11**	16th **12**	17th **13**	20th **14**	21st **15**
22nd **16**	23rd **17**	24th **18** Primeiro Build funcional	27th **19** Início da Fase de Teste	28th **20**
29th **21**	30th **22** Conclusão do Projeto	July 1st **23**	4th **24**	5th **25**

3.2 Estrutura do Projeto

Durante as primeiras reuniões um dos principais tópicos foi a Estrutura do Projeto como um todo. Já que a idéia dominante nas discussões era a de como seria a secção do projeto e a divisão das tarefas de desenvolvimento, e já que este determina como será escrito o software todo, este mostrou-se como o tópico mais importante de toda a primeira fase da execução do projeto.

Logo no início, ficara decidido que o projeto teria uma estrutura modular, com unidades de código bem delimitadas, e relacionadas entre si por interfaces pré-definidas, de maneira que o código seja o mais flexível possível, mantendo ainda uma alta independência de implementação e uma relativa simplicidade de uso dos módulos.

Estas escolhas de design direcionaram o projeto para um estilo próximo à Programação Orientada à Objetos e a Programação Modular. Com isso, as posteriores reuniões foram reservadas apenas à determinação dos módulos e de suas interfaces de código. Ao fim do planejamento chegamos a uma organização composta por 3 módulos: **Mecânica de Jogo**, **Interface de Usuário** e **Estruturas de Dados**.

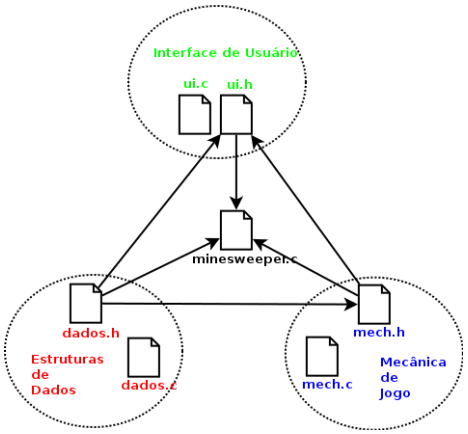

Figura 3.1: Módulos e Arquivos do Projeto

3.2.1 Estruturas de Dados

Neste módulo são implementadas estruturas de dados auxiliares, como vetores para coordenadas em 2-D, e estruturas de dados básicas do jogo, como o tabuleiro do campo minado, e suas interfaces de programação, tal como métodos para criação e destruição destas, e métodos para manipulação de seus estados internos e dados armazenados.

Estrutura "vec"

Esta estrutura é uma simples estrutura para armazenamento de coordenadas cartesianas 2-D inteiras no primeiro quadrante. Não possui métodos implementados.

Nome da estrutura	vec	
Nome do Campo	Tipo	Descrição
x	**unsigned**	Componente cartesiana em x
y	**unsigned**	Componente cartesiana em y

Estrutura "jogada_t"

Esta estrutura tem a função de servir como estrutura de dados auxiliar, transportando o código e coordenada de uma jogada realizada pelo usuário. Possui métodos *getters* e *setters*.

Nome da estrutura	jogada_t	
Nome do Campo	Tipo	Descrição
v	**vec**	Coordenada cartesiana do ponto da jogada
opjogada	**char**	Código da jogada

Métodos

- **char** *get_jogada*(**jogada_t** tab) : Retorna o código da jogada armazenada em uma estrutura **jogada_t**.
- **void** *set_jogada*(**jogada_t*** tab, **char** op) : Armazena um código de jogada em uma estrutura **jogada_t**.
- **vec** *get_coord*(**jogada_t** jogada) : Retorna a coordenada armazenada em uma estrutura **jogada_t**.
- **void** *set_coord*(**jogada_t*** jogada, **unsigned** x, **unsigned** y) : Armazena uma coordenada em uma estrutura **jogada_t**.

Estrutura "tabuleiro_t"

Nesta estrutura são armazenadas informações sobre o tabuleiro do campo minado, tais como suas dimensões, número de minas semeadas, e dois vetores armazenando o estado atual do tabuleiro, uma máscara das casas livres, e informações sobre as minas e dicas de suas localizações. Possui métodos para inicialização, destruição, *getters* e *setters*.

Nome da estrutura	tabuleiro_t	
Nome do Campo	Tipo	Descrição
m	**unsigned**	Número de linhas do tabuleiro
n	**unsigned**	Número de colunas do tabuleiro
q	**unsigned**	Número de minas semeadas
usr	**char***	Vetor máscara do tabuleiro
gabarito	**char***	Vetor das minas e dicas do tabuleiro

Métodos

Inicialização e Destruição

- **void** *inicializa_tab*(**tabuleiro_t*** tab) : Aloca espaço para os vetores da estrutura **tabuleiro_t** de acordo com as dimensões armazenadas.

- **void** *destroi_tab*(**tabuleiro_t*** tab) : Libera o espaço alocado para os vetores da estrutura **tabuleiro_t**.

Getters

- **unsigned** *get_m*(**tabuleiro_t** tab) : Retorna a dimensão vertical do tabuleiro.
- **unsigned** *get_n*(**tabuleiro_t** tab) : Retorna a dimensão horizontal do tabuleiro.
- **unsigned** *get_q*(**tabuleiro_t** tab) : Retorna o número de minas do tabuleiro.
- **char** *get_usr*(**tabuleiro_t** tab, **vec** v) : Retorna o conteúdo de uma determinada coordenada **v** do tabuleiro máscara.
- **char** *get_gabarito*(**tabuleiro_t** tab, **vec** v) : Retorna o conteúdo de uma determinada coordenada **v** do tabuleiro das minas e dicas.

Setters

- **void** *set_m*(**tabuleiro_t*** tab, **unsigned** m) : Armazena a dimensão vertical do tabuleiro.
- **void** *set_n*(**tabuleiro_t*** tab, **unsigned** n) : Armazena a dimensão horizontal do tabuleiro.
- **void** *set_q*(**tabuleiro_t*** tab, **unsigned** q) : Armazena o número de minas do tabuleiro.
- **void** *set_usr*(**tabuleiro_t*** tab, **vec** v, **char** usr) : Armazena um caractere **usr** em uma determinada coordenada **v** do tabuleiro máscara.
- **void** *set_gabarito*(**tabuleiro_t*** tab, **vec** v, **char** gabarito) : Armazena um caractere **gabarito** em uma determinada coordenada **v** do tabuleiro das minas e dicas.

3.2.2 Mecânica de Jogo

Neste módulo são implementadas funções relacionadas ao funcionamento do jogo, tal como rotinas para processar condições de vitória, ou rotinas para determinar os resultados de uma jogada efetuada pelo usuário. Não possui estruturas próprias. Depende do módulo **Estruturas de Dados**.

Função CriarTabuleiroUsr

Sintaxe **void** *CriarTabuleiroUsr*(**tabuleiro_t*** usr);
Preenche o tabuleiro máscara da estrutura **tabuleiro_t** apontado por **usr** com asteriscos, colocando-no no estado inicial.

Função ColocarBombas

Sintaxe **void** *ColocarBombas*(**tabuleiro_t*** gabarito);
Semeia o número de bombas configurado na estrutura **tabuleiro_t** apontada por **gabarito**.

Função AvaliarVizinhos

Sintaxe **void** *AvaliarVizinhos*(**tabuleiro_t*** gabarito);
Preenche o tabuleiro máscara da estrutura **tabuleiro_t** apontado por **gabarito** com dicas da presença de bombas na vizinhança de cada casa.

Função Revela

Sintaxe **void** *Revela*(**tabuleiro_t*** gabarito, **vec** v);
Dada uma coordenada **v**, e uma estrutura **tabuleiro_t** apontado por gabarito, revela as casas vizinhas à **v** no tabuleiro máscara da estrutura, tendo como fronteira, casas contendo dicas diferentes de zero.

Função ExecutaJogada

Sintaxe **void** *ExecutaJogada*(**tabuleiro_t*** gabarito, **jogada_t*** jogada, **int*** p);
Verifica o resultado de uma jogada, sendo **gabarito** um ponteiro para uma estrutura **tabuleiro_t**, **jogada** um ponteiro para a estrutura **jogada_t** da jogada a ser realizada, e **p** um ponteiro para a variável contador do número de minas marcadas corretamente.

Função ConfereRevelados

Sintaxe **int** *ConfereRevelados*(**tabuleiro_t** tab);
Testa se o tabuleiro máscara da estrutura **tabuleiro_t** apontada por **tab** já foi completamente revelado. Retorna 1 se **tab** foi completamente revelado, 0 no caso contrário.

3.2.3 Interface de Usuário

Neste módulo são implementadas funções relacionadas à interface do usuário com o jogo. Nisto inclui-se por exemplo a exibição do tabuleiro em seu estado atual na tela, ou a captura dos comandos e jogadas do usuário. Depende do módulo **Estruturas de Dados**.

Função menu

Sintaxe **int** *menu*();
Imprime uma mensagem de início do jogo seguida de um menu de opções. Retorna o número da opção digitada.

Função opseguir

Sintaxe **void** *opseguir*(**tabuleiro_t*** p, **int*** i);
Chama a função **menu()**, trata a opção retornada e captura dimensões para um novo tabuleiro. **p** é ponteiro para a estrutura **tabuleiro_t** a ser configurada, e **i** é ponteiro para uma **int** onde será armazenada a opção escolhida(1 se o jogo deve começar e 0 se deve sair do jogo).

Função fimDeJogo

Sintaxe **void** *fimDeJogo*(**int** i, **tabuleiro_t** tab);
Imprime uma mensagem de fim de jogo adequada à condição de término da partida. **i** é o número de minas descobertas, **tab** é o tabuleiro da partida.

Função lerJogada

Sintaxe **void** *lerJogada*(**jogada_t*** jog, **tabuleiro_t** tab);
Captura uma jogada digitada pelo usuário. **jog** é o ponteiro para a estrutura **jogada_t** onde deve ser armazenada a jogada, **tab** é o tabuleiro da partida.

Função imprimirTabuleiro

Sintaxe **void** *imprimirTabuleiro*(**tabuleiro_t** tab);
Imprime o tabuleiro **tab** formatado na tela.

3.3 Ferramentas

Visando simplificar e eliminar algumas dificuldades do processo de desenvolvimento, escolhemos fazer uso de algumas ferramentas em software.

3.3.1 Git

O **Git** é um sistema de controle de versão distribuído. Este sistema permite que a equipe simplesmente clone os arquivos do projeto armazenado em um repositório on-line, trabalhe em simultâneo, localmente e independentemente, em suas cópias, fazendo todas as modificações necessárias, enviando ao fim suas alterações ao repositório sem grandes problemas no momento de integrar as modificações. Isto se deve a uma das facilidades do sistema de versionamento, que trata automaticamente quase todos os conflitos causados pelas modificações, além da facilidade adicionada de todos os integrantes sempre poderem receber o projeto em sua forma mais atual sem necessitarem cloná-lo por completo novamente. Com o **Git** ainda há a possibilidade de reverter modificações específicas ou ainda de desenvolver uma versão alternativa do projeto em paralelo com a original, sem afetar o andamento do restante do projeto.

O repositório do projeto pode ser encontrado em `https://github.com/K17K47/minesweeper`

3.3.2 make

Reconhecendo o número de arquivos presentes no projeto, realizar uma compilação manualmente se torna uma operação laboriosa, e passível de erros que

afetam o desempenho do processo de teste e desenvolvimento. Assim optamos por usar o **GNU make**, uma versão de uma ferramenta de automação de compilação, que por meio de um pequeno script, que para projetos simples como o atual não precisa ser reescrito, mesmo com intensas modificações, é capaz de reduzir o ato de compilação, neste projeto, de 5 comandos no terminal para apenas 1.

3.3.3 LaTeX

Visando facilitar também a escrita da documentação e do relatório do projeto, optamos por escrevê-lo em LaTeX, um sistema de diagramação de textos, amplamente utilizado na escrita de documentos oficiais, artigos científicos e livros devido à sua alta qualidade tipográfica. Este sistema reduz a escrita de um texto formatado à escrita de um arquivo em uma linguagem de descrição de alto nível, desacoplando a aparência do texto de seu conteúdo, e permitindo sua compilação em diversas plataformas, mantendo ainda a mesma aparência. Este documento foi escrito com o LaTeX 2_ε.

4. Conclusão

T