TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA UTFPR: DUNGEON CRAWLER

Breno Moura de Abreu
Eduardo Marcelino Jr.
breno2601@gmail.com, eduardo.marcelinojr@gmail.com

Disciplina: **Técnicas de Programação - CSE20** - Prof. Dr. Jean M. Simão
Departamento Acadêmico de Informática - DAINF - Campus Curitiba
Curso Bacharelado em: Sistemas de Informação
Universidade Tecnológica Federal do Paraná - UTFPR
Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

Resumo - Como trabalho final da disciplina de Técnicas de Programação é exigido dos alunos o desenvolvimento de um *software*, no modelo de um jogo de plataforma com o intuito de aplicar as técnicas de engenharia de *software* aprendidas na disciplina em questão utilizando, principalmente, a programação orientada a objetos na linguagem C++. No jogo, o jogador enfrenta inimigos em um cenário com diversos tipos de obstáculos com o objetivo de chegar no fim desta com a maior quantidade de pontos. O jogo apresenta duas fases com obstáculos diferentes. Foram levantados os requisitos do projeto e elaborado uma modelagem usando o Diagrama de Classes e Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML). A linguagem de programação C++ foi utilizada para o desenvolvimento do código com o auxílio da biblioteca gráfica *Allegro* versão 5, aplicando os conceitos avançados de programação orientada a objetos como Classe Abstrata, Polimorfismo, Persistência de Objetos em Arquivos e uso da Biblioteca Padrão de Gabaritos (*Standard Template Library* - STL). O *software* foi implementado e testado, resultando em um trabalho que cumpre a maior parte dos requisitos previstos, cumprindo com o objetivo do trabalho em auxiliar o aprendizado dos alunos na área de engenharia de *software.*

Palavras-chave: Relatório para o Trabalho Final de Técnicas de Programação, Trabalho Acadêmico Voltado para a Implementação em C++, Relatório do Desenvolvimento de Jogo de Plataforma para Aprendizado de Engenharia de Software.

Abstract - As the final work of *Técnicas de Programação* subject is required of students the development of a software, in the model of a platform video game with the objective of apply the software engineering techniques learned using, mainly, object oriented programing in C++ language. In the game, the player will face enemies in a level with different obstacles with the objective of reaching the end of the stage with the highest number of points. The game features two levels with different obstacles. The requirements of the project have been raised and a modeling was developed using Unified Modeling Language (UML). C++ programing language and the graphic library Allegro 5 was used to develop the code, applying advanced object oriented programing concepts like Abstract Classes, Polymorphism, Persistence of Objects using Files and the use of the Standard Template Library (STL). The software was developed and tested, resulting in a work that meets most of the requirements, fulfilling the main objective of helping the students to learn software engineering techniques.

Key-words: Final Work Report of *Técnicas de Programação,* Academic Work Aimed at the Implementation in C++, Platform Video Game Development Report for Software Engineering Learning.

INTRODUÇÃO

Para os alunos do curso de Sistemas de Informação da UTFPR, na disciplina Técnicas de Programação, é dado, como trabalho final e compondo 50% da nota total da matéria, um trabalho que visa aplicar os conhecimentos de engenharia de *software* dos estudantes. O projeto tem como objetivo a criação de um *software* que cumpra com todos os requisitos previstos pelo professor. O professor irá avaliar o conteúdo aprendido e aplicado pelos alunos a fim de julgar se os mesmos tem, ou não, o mínimo de conhecimento na área para que seja possível dar continuidade ao curso.

O trabalho consiste no desenvolvimento de um *software* no modelo de um jogo de plataforma contendo inimigos e obstáculos, criado em programação orientada a objeto, mais especificamente na linguagem C++. O jogo precisa cumprir determinados requisitos como por exemplo a inclusão de objetos no cenário que interagem com o jogador, funções como salvar e carregar o jogo, a opção de se ter dois jogadores simultaneamente e a implementação de duas ou mais fases. O jogo em si segue os padrões de um jogo de plataforma comum, tendo sistema de colisão, gravidade, vida e pontos, que juntos tornam o jogo desafiador.

Para desenvolver o jogo foi feito o levantamento de requisitos que visa observar quais são as exigências do projeto e quais recursos devem ser utilizados para que se possa desenvolvê-lo. Após o levantamento, foi feita a modelagem do programa no diagrama de classes em UML com o objetivo de definir qual a relação entre as classes e como as mesmas atuam dentro do programa. Feito isso iniciou-se a implementação do código, feito em C++ com o auxílio da biblioteca gráfica *Allegro* 5 para que fosse possível desenvolver o jogo em si. Ao finalizar o código, foram executados testes para validar o programa e conferir se esse está de acordo com todos os requisitos pré estipulados.

O trabalho conta conta com uma explicação do jogo em si, mostrando telas do jogo e conceitos utilizados. Além disso há uma seção específica para mostrar o desenvolvimento do jogo usando a orientação a objetos que irá conter os requisitos mínimos para o projeto e o diagrama de classes. Apresenta também uma tabela com conceitos utilizados e não utilizados que demonstra quais técnicas de engenharia de *software* foram, ou não, aplicadas pelos membros da equipe e o porquê. Há também uma seção para comparar a forma procedimental e orientada a objetos na programação e uma seção voltada para a discussão e conclusão do trabalho. O trabalho ainda contempla uma seção onde é demonstrado a divisão do trabalho dos membros no desenvolvimento de cada área específica.

EXPLICAÇÃO DO JOGO

O jogo, denominado “Dungeon Crawler”, é um jogo de plataforma que retrata as aventuras de um guerreiro que explora cavernas em busca de tesouros. Porém, deve enfrentar diversos inimigos como ghouls, demônios e necromancers, além de sobreviver à obstáculos que estão em seu caminho. O jogo apresenta as seguintes especificações:

- Duas fases que podem ser jogadas sequencialmente ou escolhidas pelo jogador.
- Pode-se jogar com um ou dois jogadores ao mesmo tempo
- Três tipos de inimigos diferentes, sendo um deles o chefão que existe apenas na segunda fase e é instanciado somente uma vez. Os outros dois inimigos são instanciados mais de cinco vezes por fase.
- Três tipos de obstáculos, instanciados mais de cinco vezes por fase.
- Forma de salvar e carregar jogadas.
- Ranking de melhores pontuações.

Como é previsto para um jogo do gênero, o personagem tem a capacidade de atacar seus inimigos e destruí-los usando projéteis de energia, mas seus inimigos também tem a capacidade de feri-lo. O Ghoul ataca a uma pequena distância e não produz projéteis visíveis, dessa forma a simular um ataque corpo-a-corpo; o Demônio atrira projéteis assim como o jogador; e o Necromancer não possui um ataque direto contra o jogador, mas tem a capacidade de conjurar ghouls.

Além disso, são encontrados alguns obstáculos, sendo eles: chão com espinhos, que tira dano do jogador quando há contato; plataforma móvel, é uma plataforma que se move para baixo constantemente; e projéteis de energia, que são projéteis de movimento vertical com direção para cima e são produzidos constantemente em um intervalo de tempo.

A seguir algumas imagens do jogo funcionando.

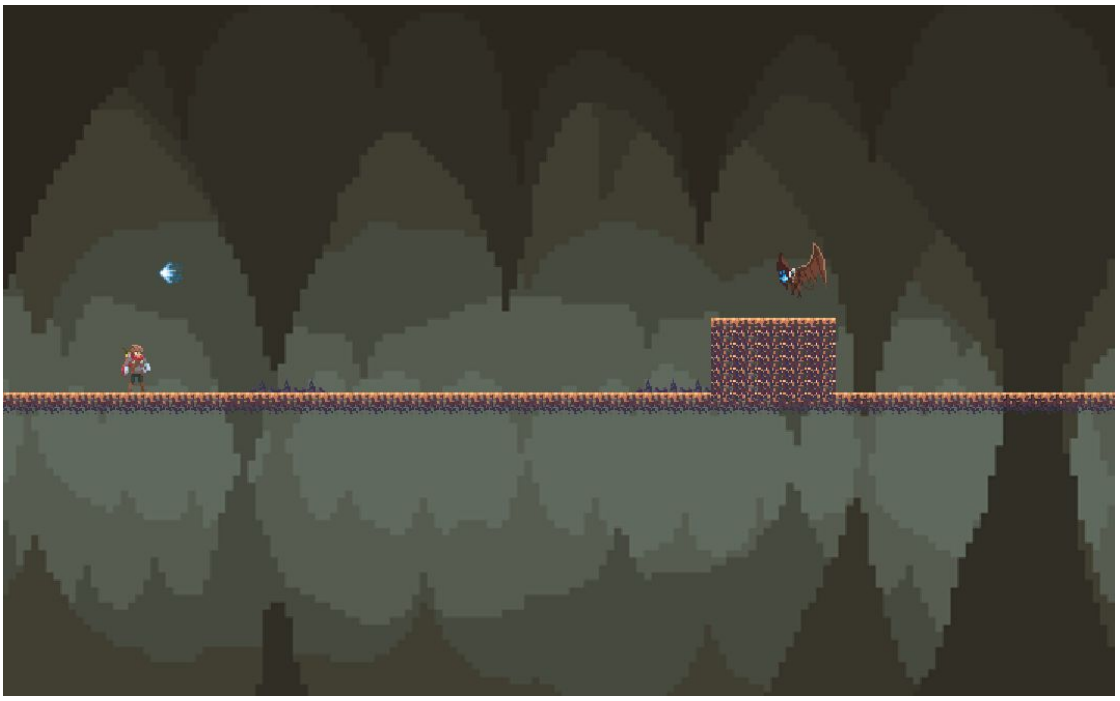

Figura 1. Representação gráfica do jogador, projétil, demônio e chão com espinhos.

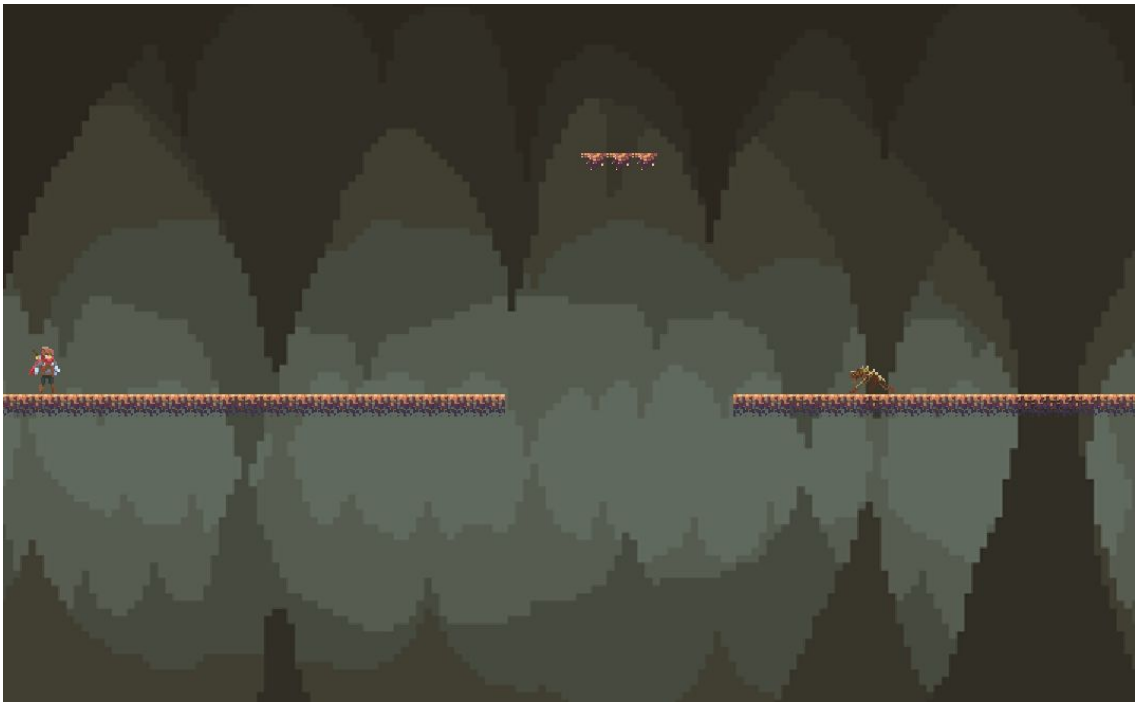

Figura 2. Representação gráfica do jogador, plataforma móvel e ghoul.

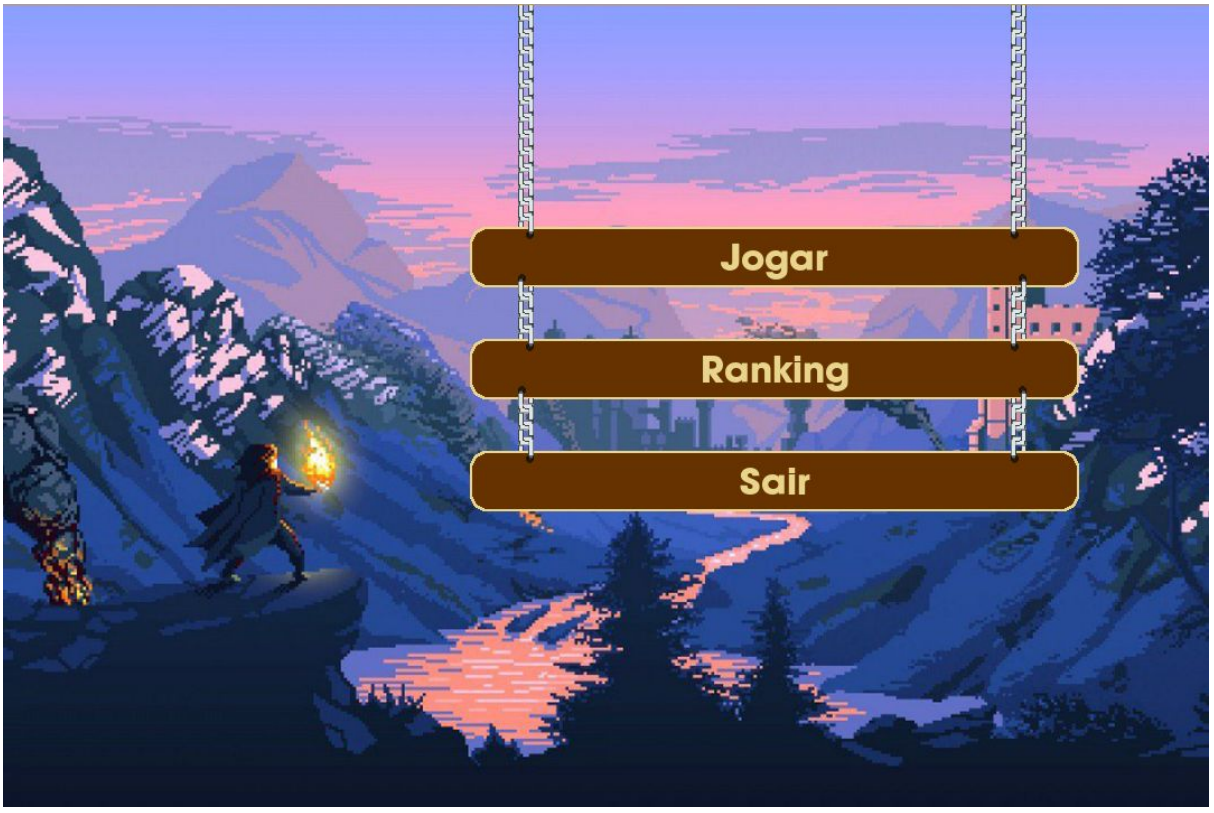

Figura 3. Tela inicial do jogo

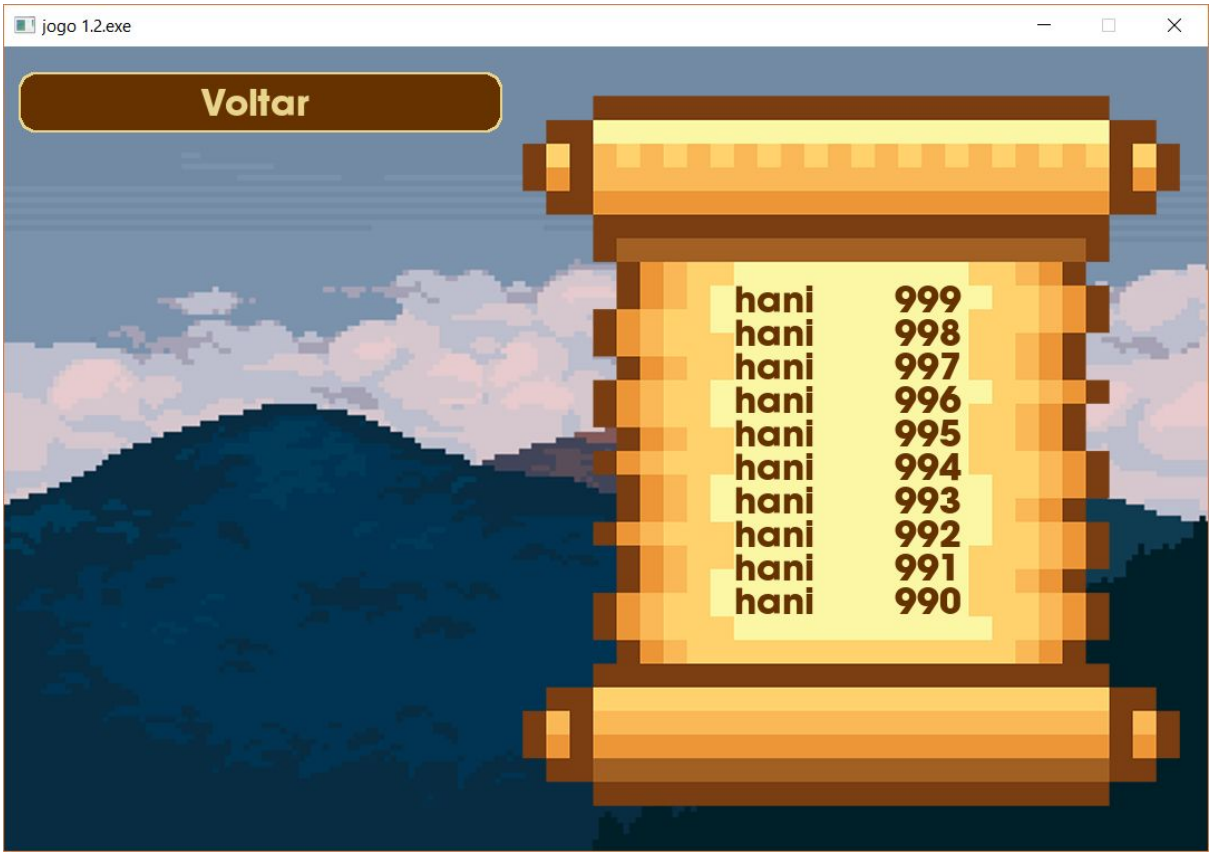

Figura 4. Ranking do jogo

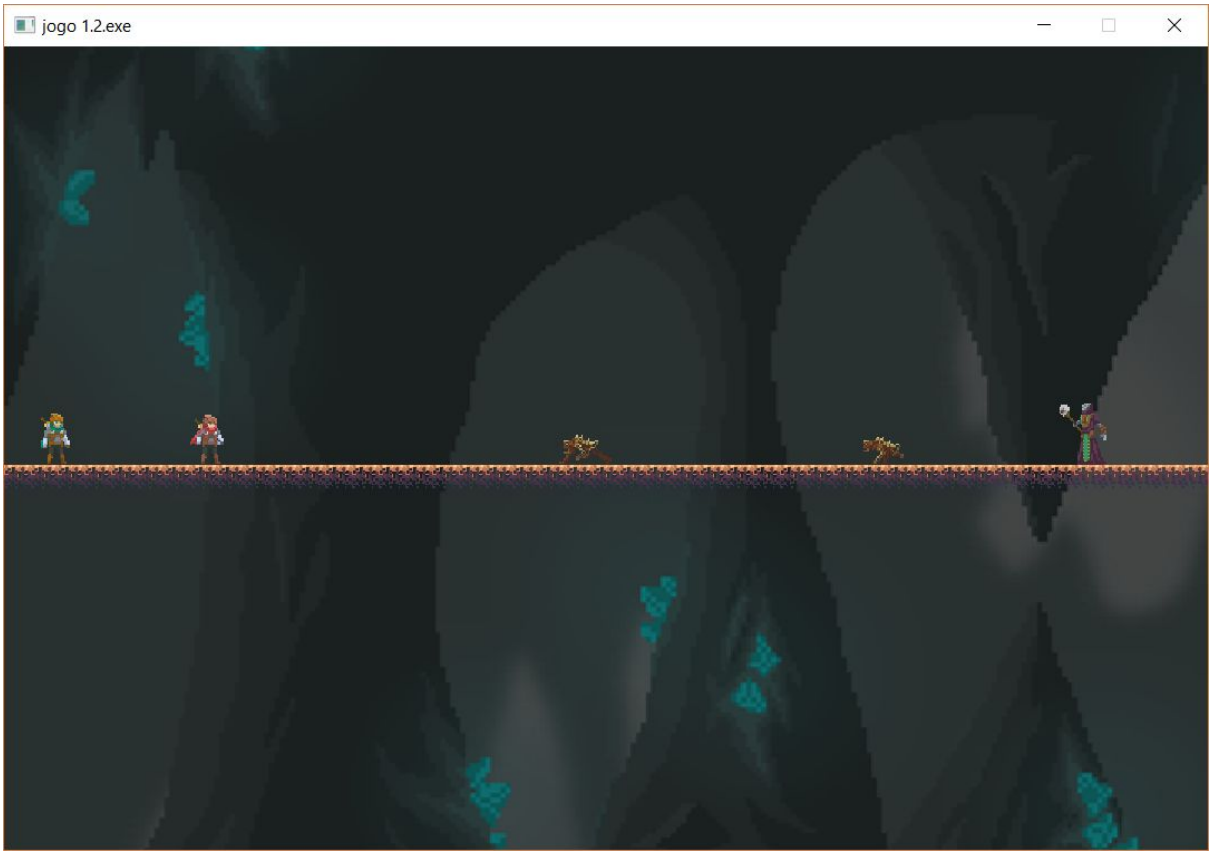

Figura 5. Boss do Jogo e segundo jogador

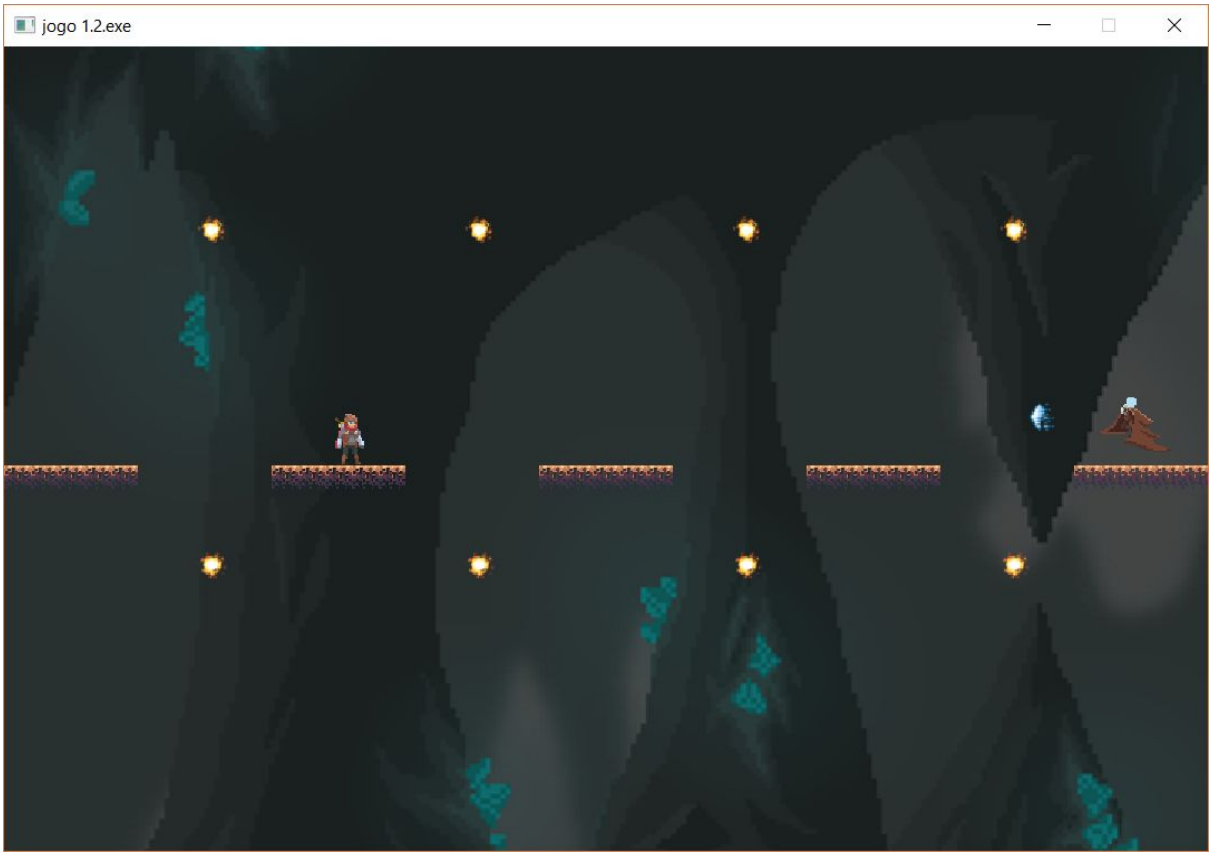

Figura 6. Obstáculo Bola de fogo e demônio

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

A tabela a seguir lista os requisitos do trabalho em questão, sua situação (realizado, parcialmente realizado ou não realizado) e em que parte do código o requisito foi feito.

Tabela 1. Lista de Requisitos do Jogo e suas Situações

N.	Requisitos Funcionais	Situação	Implementação
1	Apresentar menu de opções aos usuários do Jogo	Requisito previsto inicialmente e realizado	Requisito cumprido via classe Menu.
2	Permitir um ou dois jogadores aos usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante.	Requisito previsto inicialmente e realizado	Requisito cumprido através das classes Jogador1 e Jogador2, em que seus objetos são instanciados na classe Jogo. O Jogador2 é criado somente se necessário.
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas.	Requisito previsto inicialmente e realizado	Requisito cumprido via classes Fase1 e Fase2, sendo que podem ser escolhidas através da classe Menu.
4	Ter três tipos distintos de inimigos (o que pode incluir 'Chefão', vide abaixo).	Requisito previsto inicialmente e realizado	Requisito cumprido via classes Ghoul, Demônio e Necromancer.
5	Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias e sendo pelo menos 5 instâncias por tipo.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe GeradorFase.
6	Ter inimigo "Chefão" na última fase	Requisito previsto inicialmente e realizado	Requisito cumprido via classe Necromancer.

7	Ter três tipos de obstáculos.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe Espinho, Plataforma e Atirador.
8	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório de instâncias (i.e., objetos) sendo pelo menos 5 instâncias por tipo.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe GeradorFase.
9	Ter representação gráfica de cada instância.	Requisito previsto inicialmente e realizado	Requisito cumprido via funções biblioteca Allegro 5 utilizadas pelas classes derivadas de Entidade e pela classe Projetil.
10	Ter em cada fase um cenário de jogo com os obstáculos.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe GeradorFase que irá gerar o padrão de obstáculos e inimigos que será utilizado pelas classes Fase1 e Fase2
11	Gerenciar colisões entre jogador e inimigos.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe GerenciadorColisao.
12	Gerenciar colisões entre jogador e obstáculos.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe GerenciadorColisao.
13	Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (*ranking*).	Requisito previsto inicialmente e realizado	Requisito cumprido via classes Jogo, Ranking e Menu, através de persistência de objetos.
14	Permitir Pausar o Jogo	Requisito previsto inicialmente e realizado	Requisito cumprido via classes Menu e Jogo.
15	Permitir Salvar Jogada.	Requisito previsto inicialmente e realizado	Requisito cumprido via classe Menu e Jogo.

A imagem a seguir é do diagrama de classes UML feito em StarUML. Cada módulo será explicado posteriormente.

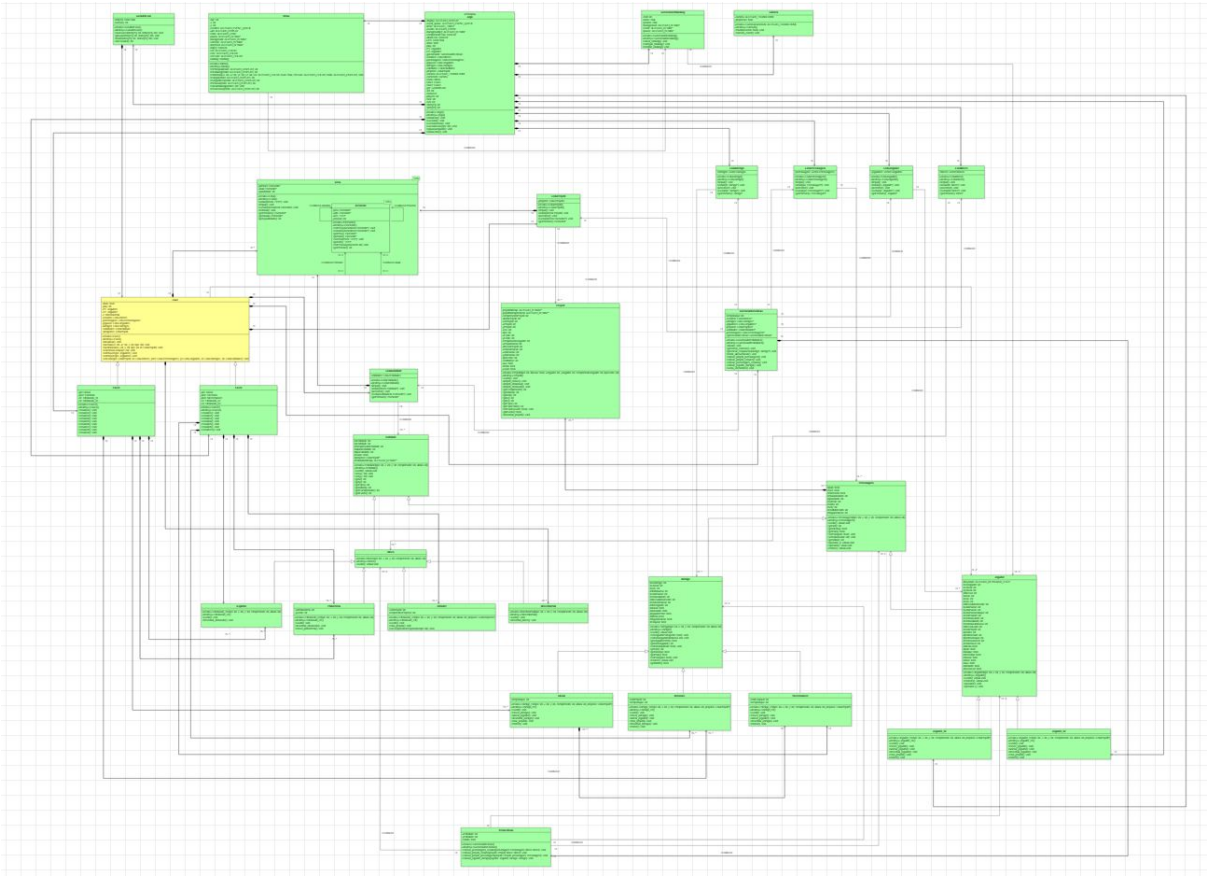

Figura 5. Diagrama de Classes UML

Infelizmente não é possível observar com clareza o UML nesse documento, mas uma descrição de cada módulo será provida a seguir.

Módulo Entidade:

Contém a classe abstrata Entidade que possui as informações básicas que todos os objetos das classes derivadas possuem como suas coordenadas, tipo e tamanho do elemento. Todas as classes derivadas de Entidade são partes do jogo que possuem representação gráfica, como os blocos que compõe o cenário, os inimigos, os obstáculos e os jogadores.

A classe Entidade deriva as classes Bloco e Personagem.

A classe Bloco dá origem à classe dos blocos que formam o chão e as classes dos três obstáculos, e a classe Personagem deriva outras duas classes a Inimigo, que dá origem às classes dos três inimigos, e a classe Jogador, que dá origem às classes dos dois jogadores. O terceiro inimigo possui a habilidade de criar dinamicamente objetos da classe Ghoul.

Ainda há a classe Projétil responsável pelos atributos e métodos dos projéteis que serão lançados pelos personagens, ou que serão criados como obstáculos.

Módulo Listas:

Aqui foi feita uma lista usando a ferramenta *template* que tem a função de armazenar objetos de diferentes classes, excluir e incluir elementos. Dentro da mesma há uma classe aninhada denominada Elemento que servirá para guardar os endereços dos objetos em si de cada tipo.

Outras duas classes foram criadas a fim de possibilitar o percorrimento das listas: a classe ListaEntidades e a ListaProjeteis, contendo objetos de Entidade e Projéteis respectivamente.

Nesse módulo foi usado a biblioteca STL *vector* para criar as listas: ListaInimigo, ListaPersonagem, ListaBloco e ListaJogador, com funções similares às descritas acima mas sem a opção de percorrer a lista, visto que tal função não seria necessária para o funcionamento do código.

Módulo de Interação entre Elementos

Possui duas classes principais classes, a GerenciadorColisao e a GerenciadorEntidades que irão possibilitar a colisão entre personagens e cenários além de testar a aproximação de um jogador com um inimigo.

Módulo de Fases:

Composta pelas classes GeradorFase, que irá gerar uma fase e salvar a mesma em um arquivo de texto, a classe Fase, que ainda não foi feita no código em si, e suas duas classes derivadas Fase1 e Fase2 que irão carregar uma fase e incluir os elementos criados nas respectivas listas.

Módulo de Gerenciamento do Jogo:

Aqui estão as classes relativas ao gerenciamento do jogo, como a classe Jogo que irá criar todas as classes básicas para o funcionamento do jogo, a classe Menu que permite a comunicação com o usuário, a classe Câmera que define a proporção da tela em relação a suas dimensões e a classe Ranking que permite salvar e mostrar as pontuações dos jogadores.

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

A tabela a seguir mostra os conceitos utilizados e não utilizados no projeto.

Tabela 2. Lista de conceitos requeridos no projeto, seu uso, ou não uso, e onde foram aplicados no projeto.

N.	Conceitos	Uso	Onde / O quê
1	**Elementares:**		
	- Classes, objetos, - Atributos (privados), variáveis e constantes - Métodos (com e sem retorno).	Sim	Todos .h e .cpp
	- Métodos (com retorno *const* e parâmetro *const*). - Construtores (sem/com parâmetros) e destrutores	Sim	Todos .h e .cpp
	- Classe Principal.	Sim	Main.cpp e Jogo.h/.cpp
	- Divisão em .h e .cpp.	Sim	No desenvolvimento como um todo.
2	**Relações de:**		
	- Associação direcional. - Associação bidirecional.	Sim	Diversas classes com a função de utilizar um objeto de outra classe. Não foi usada Associação bidirecional.
	- Agregação via associação - Agregação propriamente dita.	Sim	Classe Fase e duas derivadas, classe Jogo .
	- Herança elementar. - Herança em diversos níveis.	Sim	Classe Entidade e suas derivadas

			e classe Fase e suas derivadas.
	- Herança múltipla.	Não	
3	**Ponteiros, generalizações e exceções**		
	- Operador *this*.	Sim	Classe Entidade.h
	- Alocação de memória (*new* & *delete*).	Sim	Classe GerenciadorColisao, classe Lista
	- Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via *Templates*).	Sim	Classe Lista.
	- Uso de Tratamento de Exceções (*try catch*).	Não	
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Classe Fase1 e Fase2
	- Operadores (2 tipos de operadores pelo menos).	Sim	Classe Personagem e classe Jogador
	Persistência de Objetos (via arquivo de texto ou binário)		
	- Persistência de Objetos.	Sim	Salvar fases e ranking
	- Persistência de Relacionamento de Objetos.	Não	
5	**Virtualidade:**		
	- Métodos Virtuais.	Sim	Classe Personagem e classe Jogador
	- Polimorfismo	Sim	Classe entidade e suas derivadas
	- Métodos Virtuais Puros / Classes Abstratas	Sim	Classe entidade e suas derivadas

	- Coesão e Desacoplamento	Sim	Todo o desenvolvimento
6	**Organizadores e Estáticos**		
	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Classe Camera
	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Classe Lista
	- Atributos estáticos e métodos estáticos.	Sim	Classes Ghoul, Demônio e Necromancer
	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	Todo o desenvolvimento em geral
7	**Standard Template Library *(STL)* e String OO**		
	- A classe Pré-definida *String* ou equivalente. - *Vector* e/ou *List* da *STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores)	Sim	Classe GeradorFase, classes de Inimigos (String). *Vector* nas classes de lista: ListaInimigo, ListaPersonagem, ListaJogador e ListaBloco.
	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **ou** Multi-Mapa.	Sim	Classe ListaEntidade
	Programação concorrente		
	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **ou** Win32API ou afins.	Não	
	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **ou** Troca de mensagens.	Não	
8	**Biblioteca Gráfica / Visual**		

	- Funcionalidades Elementares. - Funcionalidades Avançadas como: • tratamento de colisões • duplo *buffer*	Sim	Via biblioteca Allegro 5, colisões tratadas na classe GerenciadorColisao. Funcionalidades usadas como manipulação de bitmaps, receber estados do teclado e mouse.
	- Programação orientada e evento em algum ambiente gráfico. OU - *RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).	Sim	Usado Allegro 5
	Interdisciplinaridades por meio da utilização de Conceitos de Matemática, Física etc		
	- Ensino Médio.	Sim	Conceitos de geometria analítica, gravidade e colisões
	- Ensino Superior.	Sim	Conceitos de estruturas de dados
9	**Engenharia de Software**		
	- Levantamento e rastreabilidade de cumprimento de requisitos.	Sim	Através dos documentos providos pelo professor
	- Diagrama de Classes em *UML*.	Sim	Feito em StarUML
	- Uso intensivo de padrões de projeto (GOF).	Sim	Uso do *Mediator* na classe GerenciadorColisao, *builder* na GeradorFase e *facade* na classe *main* .

	- Testes a luz da Tabela de Requisitos	Sim	Testes feitos com base no requisitos previstos
10	**Execução de Projeto**		
	- Controle de versão de modelos e códigos automatizado (via SVN e/ou afins) ou manual (via cópias manuais). - Uso de alguma forma de cópia de segurança (backup).	Sim	Versões diferentes do programa foram salvas e enviadas para o Google Drive
	- Reuniões com o professor para acompanhamento do andamento do projeto.	Não	2 Reuniões: 28/05 e 25/06
	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Não	0 Reuniões
	- Revisão do trabalho escrito de outra equipe e vice-versa.	Não	

A tabela a seguir apresenta justificativas para o uso, ou não uso, dos conceitos apresentados na tabela 2.

Tabela 3. Lista de justificativas para Conceitos Utilizados e Não Utilizados no projeto.

N.	Conceitos	Situação
1	Elementares	Classes e Objetos foram utilizados porque o desenvolvimento do projeto é orientado a objetos. Métodos constantes foram utilizados para evitar possíveis mudanças de variáveis e manipulações indesejadas das mesmas. Divisão em .h e .cpp utilizado como padrão de arquivos de projetos em linguagem C++.

2	Relações	Associação foi utilizada para que uma classe pudesse receber informações de outra classe. Agregação foi utilizada para a instância de novos objetos. Herança foi utilizada em classes que poderiam compartilhar os mesmos atributos e métodos. Herança múltipla não foi utilizada pois em nenhum momento houve a necessidade de utilizar a técnica, todas as necessidades nessa área puderam ser satisfeitas com o uso de herança e herança em diversos níveis.
3	Ponteiros, generalizações e exceções	Operador *this* foi utilizado para evitar acesso de memórias indevidas na classe Entidades. Alocação de memória foi utilizada para criar novos objetos de forma dinâmica no jogo, pois os cenários são criados de forma aleatória. O *template* foi utilziado para a criação de uma lsita genérica para evitar a criação de novas listas a cada vez que fosse necessário guardar um conjunto de objetos de diferentes classes. O *try catch* não foi utilizado por recomendação do próprio professor.
4	Sobrecarga	Sobrecarga de operadores foi utilizado para incrementar ou decrementar a pontuação e vida dos personagens. Sobrecarga de método foi utilizado para a criação de métodos que não necessariamente precisam receber uma quantidade fixa

		de atributos, podendo assim ter aplicabilidades diferentes dependendo da quantidade de atributos recebidos.
5	Virtualidade	Os itens apresentados nessa categoria foram todos utilizados em sua maioria nas classes derivadas de Elemento(classe abstrata em si), com o objetivo de tornar possível que os métodos de mesmo nome tivessem funções diferentes, dependendo da classe que a implementa. O uso de coesão e desacoplamento é de extrema importância para a programação orientada a objetos pois permite que diferentes classes podem ser usadas em diferentes códigos, sem que haja uma conexão muito forte entre elas, permitindo que cada classe ou conjunto de classes tenham certa independência.
6	Organizadores e Estáticos	Namespace foi utilizado na classe câmera apenas para teste da ferramenta em questão. Classe aninhada foi usada dentro da classe Lista, pois esse conjunto de classes possui uma ligação muito forte. A ferramenta constante foi amplamente utilizada para definir atributos que não variam durante a execução do jogo, e evitar que a mesma varie por alguma razão , além disso, foi utilizada em métodos para garantir que o valor retornado ou recebido não se modifique. Estáticos foram usados para que as classes de inimigos tenham

		um nome que pode ser acessado por qualquer objeto dessa classe.
	Persistência de Objetos	Persistência de objetos foi utilizado para salvar o os cenários de jogo e salvar o ranking em arquivos de texto. Persistência de relacionamentos de objetos não foram utilizados pois no caso em questão não houve necessidade de salvar a relação entre objetos.
7	Standard Template Library e String	Classe *string* utilizada para facilitar a manipulação de cadeias de caracteres. Set usado para que seja possível identificar quais tipos de elementos existem numa lista. *Vector* usado na criação de classes com objetivo de listagem.
	Programação Concorrente	Não foram utilizadas por recomendação do professor.
8	Interdisciplinaridade	Conceitos de estruturas de dados foi utilizado para criar as listas. Conceito básico de física para tratar colisões e gravidade. Conceito de geometria analítica para melhorar o entendimento do plano cartesiano em que a parte gráfica do jogo existe.
9	Engenharia de software	Todo o código foi feito e testado com base nos requisitos apresentados no artigo relatório. Diagrama de classe foi feito para definir a relação entre as classes além dos principais métodos e atributos de cada classe. O padrão de projeto *mediator* foi utilizado na classe GerenciadorColisao

		para conectar objetos de diferentes classes e definir como os mesmos se relacionam entre si.
10	Execução do Projeto	Backups do projeto foram feitos com a ferramenta Google Drive, e diferentes versões foram salvas nas máquinas de cada um dos integrantes da dupla. Tanto reuniões com o professor quanto com o monitor não foram possíveis por irresponsabilidade dos integrantes da dupla. Não foi possível revisar o trabalho de outra dupla pois os integrantes se encontraram com pouco tempo para a execução de seu próprio projeto, não possibilitando tempo para a revisão do trabalho de outra equipe

REFLEXÃO COMPARATIVA

A partir da realização do trabalho, foi possível observar as principais diferenças entre o desenvolvimento orientado a objetos e o desenvolvimento procedimental. Com a divisão do programa em classes da programação orientada a objetos e os conceitos de desacoplamento, é possível criar programas mais complexos e melhores de serem modificados. Além disso, essa lógica permite observar o programa como algo mais próximo da vida real, sendo que cada objeto possui seus próprios atributos e métodos, isso ajuda na compreensão e aplicação da lógica no código.

DISCUSSÃO E CONCLUSÕES

Em conclusão, o trabalho final infelizmente não apresentou todos os requisitos técnicos - utilização de recursos do C++ - propostos, mas foi possível desenvolver um produto final cumprindo com todas as especificações necessárias.

O projeto em questão foi de fundamental importância para o entendimento prático da matéria, visto que foi proposto um trabalho com requerimentos mais próximos dos

apresentados em problemas reais, assim melhorando a percepção dos alunos em relação à utilização do conhecimento aprendido para a resolução de problemas.

Não fosse por falta de planejamento e organização dos integrantes, seria possível concluir o trabalho com uma qualidade superior, aplicando mais técnicas de forma a criar um código mais limpo e funcional, além de cumprir com a visão ideal do professor para o projeto.

Foi possível perceber o quão complexo um trabalho pode ser, com diversas soluções diferentes para o mesmo problema, aparições de erros e *bugs* não previstos na idealização do código obrigando os desenvolvedores a achar soluções, ter que trabalhar em grupo para encontrar a melhor forma de resolver problemas, criar um projeto a partir de requisitos previamente propostos. Não há dúvidas que os integrantes cometeram muitos erros ao longo do desenvolvimento do projeto, mas foi possível aprender com as falhas e perceber soluções antes não pensadas.

DIVISÃO DO TRABALHO

Apresentado a seguir, está uma tabela que descreve a divisão do trabalho entre os dois membros da equipe.

Tabela 4. Lista de Atividades e Responsáveis

Atividades	**Responsáveis**
Levantamento de Requisitos	Breno e Eduardo em geral
Diagrama de Classes	Mais Breno que Eduardo
Programação em C++	Breno e Eduardo em geral
Implementação de Template	Breno
Implementação de Persistência dos Objetos	Eduardo
Implementação da sobrecarga de Métodos	Eduardo
Implementação da sobrecarga de Operadores	Breno
Implementação de Virtualidade	Mais Breno que Eduardo
Implementação de Organizadores	Breno e Eduardo em geral
Implementação de String	Eduardo
Implementação do Vector (STL)	Breno
Implementação das funcionalidades do Allegro	Breno e Eduardo em geral

Escrita do Trabalho	Mais Breno que Eduardo
Revisão do Trabalho	Breno e Eduardo em geral