TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO / SISTEMAS DE INFORMAÇÃO DA UTFPR:
Green Threat

Pedro Henrique Secchi, Pedro Scarpin Ribeiro
pedrohsecchi@gmail.com, pribeiro@alunos.utfpr.edu.br

Disciplina: **Técnicas de Programação – CSE20 / S71** – Prof. Dr. Jean M. Simão
Departamento Acadêmico de Informática – DAINF - Campus de Curitiba
Curso Bacharelado em: Engenharia da Computação / Sistemas de Informação
Universidade Tecnológica Federal do Paraná – UTFPR
Avenida Sete de Setembro, 3165 – Curitiba/PR, Brasil – CEP 80230-901

Resumo – Este documento apresenta os principais pontos abordados para a execução do projeto da disciplina de Técnicas de Programação. O projeto constitui-se de um jogo 2D implementado em linguagem orientada a objetos em C++. Para isso, foram estabelecidos alguns critérios que auxiliaram na execução e na avaliação do mesmo. Assim sendo, foi implementado o jogo Green Threat, onde o jogador controla um guerreiro que deve percorrer três fases enfrentando inimigos e desviando de obstáculos até chegar na última fase, na qual deve enfrentar um "chefão". Para melhor orientação e documentação, foi elaborado um diagrama de classes em UML (Unified Modeling Language - Linguagem de Modelagem Unificada). Com base nisso, foi-se utilizado de conceitos de orientação a objetos como classes, atributos e métodos, além das associações entre as classes. Juntamente houve a utilização da biblioteca SFML (Simple and Fast Multimedia Library - Biblioteca multimídia simples e rápida), a fim gerir a parte gráfica do projeto. Por fim, foram realizados testes para verificar se o jogo estava sendo executado de forma satisfatória e que os requisitos desejados foram realizados.

Palavras-chave ou Expressões-chave Trabalho Acadêmico com foco em C++; Linguagem Orientada a Objetos; Diagrama de Classes em UML; Biblioteca SFML.

Abstract – This document presents the main points discussed for the execution of the project of Programming Techniques discipline. The project consists of a 2D game implemented in object-oriented language in C++, for this, some criteria were established, helping in the execution and also being used by the teacher in its evaluation. Therefore, the game Green Threat was implemented and the player controls a warrior that must go through three levels, facing enemies and also dodging obstacles until reaching the last level, in which he must face a "boss". For better guidance, after discussions, a class diagram in UML (Unified Modeling Language) was developed. Based on this, object-oriented concepts such as classes, attributes and methods were used, in addition to associations between the classes. Furthermore, there was the use of the SFML library (Simple and Fast Multimedia Library - Simple and fast multimedia library), to manage the graphic part of the project. Finally, tests were performed to verify that the game as a whole was running satisfactorily and that the desired requirements were met.

Keywords or Key Expressions Academic work focused on C++; Object Oriented Language; UML Class Diagram; SFML Library.

INTRODUÇÃO

Esse documento tem como objetivo apresentar o projeto desenvolvido na matéria de Técnicas de Programação, que será utilizado como parâmetro avaliativo para o professor. Para isto, o trabalho, visa a aplicação dos conceitos aprendido pelos alunos no decorrer do semestre, bem como o aperfeiçoamento das habilidades dos mesmos.

O projeto requerido trata-se de um jogo em estilo plataforma 2D desenvolvido em linguagem orientada a objeto C++. Para tanto, o mesmo se embasa em alguns requisitos

estabelecidos pelo professor a fim de facilitar tanto a implementação quanto a avaliação do mesmo.

Baseado nos requisitos solicitados, foram feitas algumas análises por ambos os membros da equipe, o que possibilitou a melhor compreensão das demandas e melhores alternativas para atendê-las. Antes do início da implementação, foi modelado um diagrama de classe UML para demostrar com mais clarezas as necessidades inerentes ao projeto.

Ao decorrer deste documento, serão apresentados alguns tópicos importantes para o entendimento do processo de execução do projeto.

EXPLICAÇÃO DO JOGO EM SI

No jogo controlamos um guerreiro que tem como objetivo passar por todas as fases lutando contra os inimigos e evitando os obstáculos até que consiga escapar pela porta ao fim fase. O jogo é composto por três fases únicas, sendo que a última contém também um chefão, que traz um desafio superior aos demais.

Os obstáculos presentes no jogo são: a gosma que diminui a velocidade e não permite pular, os espinhos que causam dano e armadilhas que desaparecem ao pisar nelas, sendo necessário velocidade para que o jogador consiga escapar. Já de inimigos temos: o Goblin que causa dano ao encostar no jogador, o Bomber Goblin, que atira bombas para atingir o jogador, e o chefão que é mais resistente, o que acaba gerando um desafio maior para o jogador na fase final do jogo.

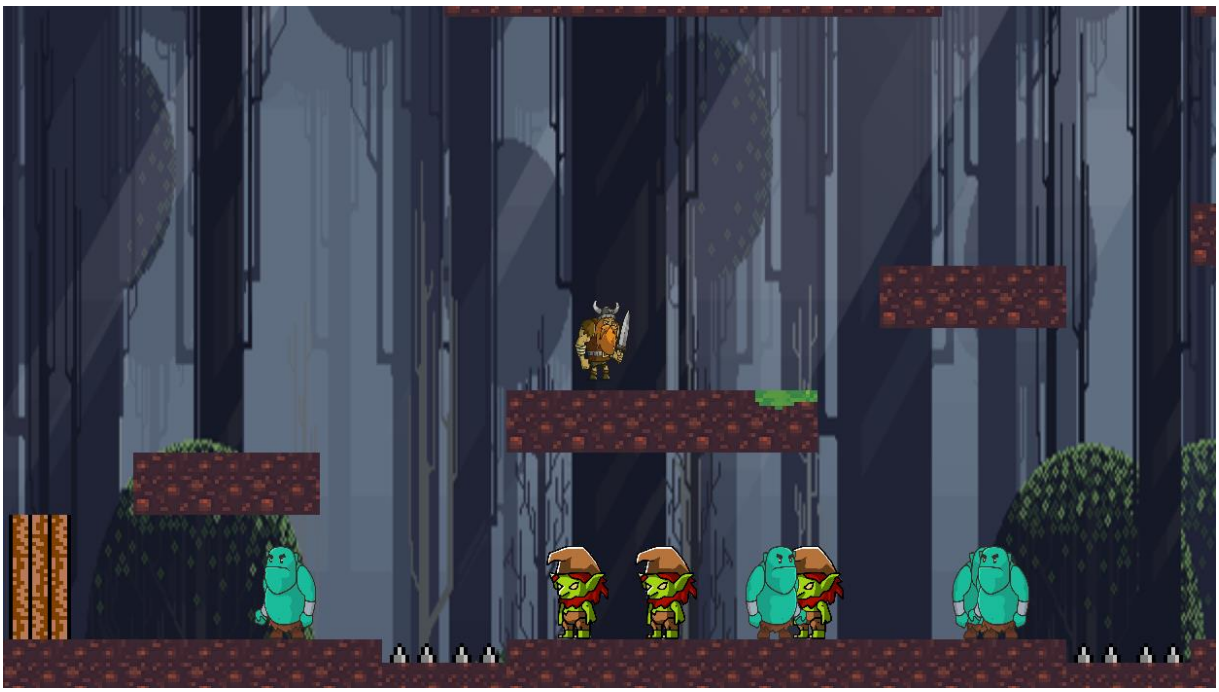

Figura 1: Level 1

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Para a implementação do jogo em linguagem orientada a objeto, chegou-se a um modelo de classes da UML como demostra a figura 2, partindo dos parâmetros estabelecidos anteriormente pelo professor, como a necessidade de ao menos um jogador, uma quantidade razoável de fases, inimigos e obstáculos, além da implementação do menu e gerenciamento das colisões.

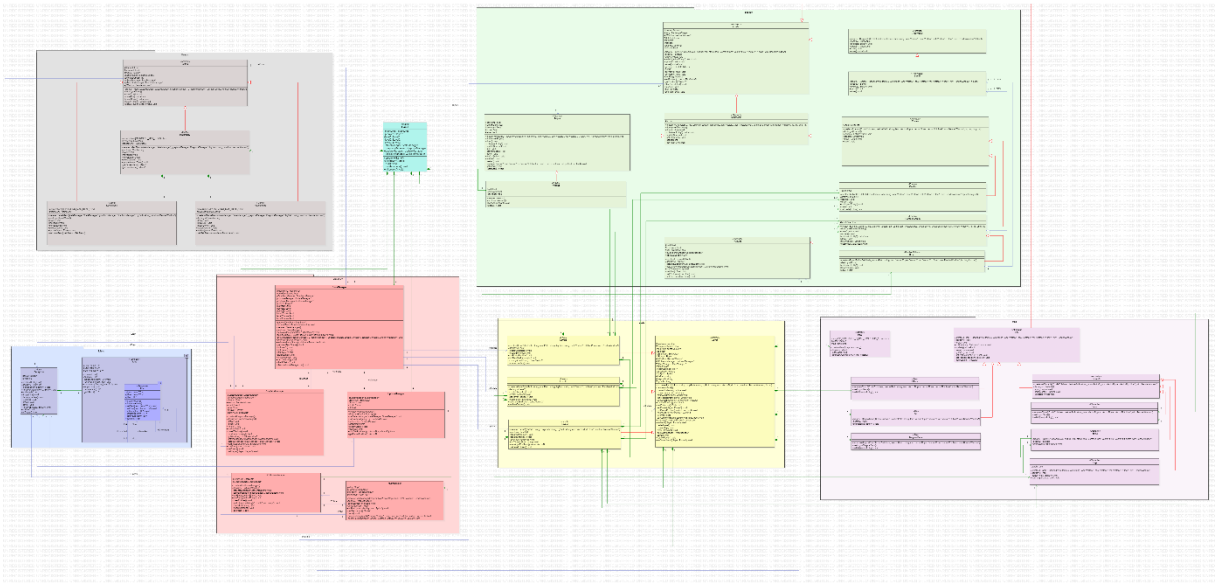

Figura 2. UML modelada para a realização do projeto.[1]

Toda a programação do jogo foi dividido em classes, o que facilitou no momento de programar, já que os atributos e métodos dos objetos que compunham o problema foram distribuídos cada qual conforme a necessidade, que por sua vez acaba por tornar um problema relativamente grande e complexo em problemas menores e mais simples. Com tudo, o ponto chave da implementação utilizando o paradigma orientado a objeto é a relação que cada classe tem com as demais.

Para melhor compreensão das necessidades, a tabela 1 apresenta de forma objetiva os requisitos solicitados bem como a situação que cada um se encontra, além de apontar quais classes e itens podem ser usados para verificar.

Tabela 1. Lista de Requisitos do Jogo e suas Situações.

N.	Requisitos Funcionais	Situação	Implementação
1	Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, ver colocação (ranking) de jogadores e demais opções pertinentes.	Requisito previsto inicialmente e realizado.	Requisito cumprido na classe Menu e em suas classes herdadas: LevelMenu, MainMenu, PauseMenu e Ranking.
2	Permitir um ou dois jogadores com representação gráfica aos usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante.	Requisito previsto inicialmente e realizado parcialmente – Faltou ainda o segundo jogador.	Requisito parcialmente cumprido inclusive via classe Warrior cujos objetos são agregados em jogo, podendo ser apenas um jogador, entretanto.
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas, via menu, nas	Requisito previsto inicialmente e realizado parcialmente – Faltou ainda a neutralização dos inimigos.	Requisito parcialmente cumprido nas classes Level1, Level2 e Level3, na respectiva sequência. Inimigos não são

1 Uma cópia da UML com qualidade superior foi enviada juntamente com esse relatório, para fins de consultas.

	quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa.		neutralizados pelo jogador.
4	Ter pelo menos três tipos distintos de inimigos, cada qual com sua representação gráfica, sendo que ao menos um dos inimigos deve ser capaz de lançar projetil contra o(s) jogador(es) e um dos inimigos dever ser um 'Chefão'.	Requisito previsto inicialmente e realizado.	Requisito cumprido nas classes Goblin, BomberGoblin e Boss, herdadas da classe Enemy. Tanto BomberGoblin quanto Boss herdam também da classe Thrower, o que os possibilita o lançamento de projéteis.
5	Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias e sendo pelo menos 3 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito cumprido via instância de quantidades aleatórias de objetos das classes Goblin e BomberGoblin.
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos causa dano em jogador se colidirem.	Requisito previsto inicialmente e realizado.	Requisito cumprido via instâncias das classes Spike, Slime e Trap, cada qual instanciada nas fases.
7	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório de instâncias (i.e., objetos), sendo pelo menos 3 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito cumprido via instâncias em quantidades aleatórias das classes Spike, Slime e Trap.
8	Ter em cada fase um cenário de jogo constituído por obstáculos, sendo que parte deles seriam plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe MapManager, Map e Tile.
9	Gerenciar colisões entre jogador para com inimigos e seus projeteis, bem como entre jogador para com obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe CollisionManager.
10	Permitir: (1) salvar nome do usuário, manter/salvar pontuação do jogador (incrementada via neutralização de inimigos) controlado pelo usuário e gerar lista de pontuação (ranking). E (2) Pausar e	Requisito previsto inicialmente e realizado parcialmente – Faltou ainda salvar nome do usuário e gerar lista de pontuação.	Requisito parcialmente cumprido via método save() em cada objeto de Entity, e utilização da classe PauseMenu.

	Salvar Jogada.		
Total de requisitos funcionais apropriadamente realizados.			**70%** (setenta por cento)

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

A tabela 2 demonstra alguns conceitos necessários e/ou relevantes para a implementação do projeto. Em sua maioria esses conceitos foram apresentados em aulas no decorrer do semestre, já os demais, tratam-se de melhorias no que se diz respeito ao software final e ao processo de execução do seu desenvolvimento.

Tabela 2. Lista de Conceitos Utilizados e Não Utilizados no Trabalho.

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	Game.h e Game.cpp
	- Divisão em .h e .cpp.	Sim	No desenvolvimento como um todo.
2	**Relações de:**		
	- Associação direcional. & - Associação bidirecional.	Sim	Um exemplo de associação direcional é o relacionamento de CollisionManager com MapManager. Já o relacionamento entre MainMenu e PauseMenu constitui um exemplo de associação bidirecional.
	- Agregação via associação. & - Agregação propriamente dita.	Sim	Um exemplo de agregação forte se dá entre MainMenu e Ranking. Já uma agregação fraca, entre EntityList e Entity.
	- Herança elementar. & - Herança em diversos níveis.	Sim	Um exemplo de herança elementar ocorre entre Tile e Block. Já entre as classes Tile, Obstacle e Slime, está presente uma herança em diversos níveis.
	- Herança múltipla.	Sim	A classe BomberGoblin herda tanto de Enemy quanto de Thrower, constituindo um exemplo de herança múltipla.

3	Ponteiros, generalizações e exceções		
	- Operador this para fins de relacionamento bidirecional.	Sim	Na maior parte das classes (de acordo com o código-fonte do programa). Um exemplo está presente na classe Player, no método *setCanJump()*.
	- Alocação de memória (new & delete).	Sim	Alocação de memória foi usada em diversos pontos do código. Um exemplo do uso de *new* está no método *createEnemies()*, na classe Level1. Já exemplos de delete estão na destrutora da mesma classe citada acima.
	- Gabaritos/Templates criada/adaptados pelos autores (e.g. Listas Encadeadas via Templates).	Sim	Uma lista template duplamente encadeada foi implementada na classe List, sendo que tal classe é classe base de EntityList.
	- Uso de Tratamento de Exceções (try catch).	Sim	Tratamento de exceções foram feitas ao longo de todo o código. Um bloco try/catch está presente no método *initMenu()*, pertencente a MainMenu.
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Sobrecarga de método no método *die()* pertencente a classe Player, e sobrecarga de construtora na classe Character.
	- Operadores (2 tipos de operadores pelo menos).	Sim	Sobrecarga do operador ! na classe CollisionManager e do operador + na classe Character.
	Persistência de Objetos (via arquivo de texto ou binário)		
	- Persistência de Objetos.	Sim	Ocorre persistência de objetos via arquivos de texto. Tal fato pode ser notado nas classes Entity e Level que possuem o método *save()* e *initSavedLevel()*, respectivamente.
	- Persistência de Relacionamento de Objetos.	Sim	Ocorre persistência de relacionamento de objetos via arquivo de texto. O relacionamento do Boss com seus projéteis é salvo e recuperado posteriormente.
5	Virtualidade:		
	- Métodos Virtuais.	Sim	Métodos virtuais são usados,

			por exemplo, na classe Entity, a qual possui os métodos virtuais *collide(), update(),* entre outros.
	- Polimorfismo	Sim	Na maior parte das classes (de acordo com o diagrama de classes e o código-fonte do programa). A título de exemplo, o método *createEnemies(),* que está presente tanto na classe Level1 quanto na classe Level3, pode ser citado. Ambas as classes herdam o método da superclasse Level, contudo executam instruções diferentes.
	- Métodos Virtuais Puros / Classes Abstratas	Sim	A classe Menu constitui um exemplo de classe abstrata. Já o método *moveUp*(), pertencente a mesma, exemplifica o uso de métodos virtuais puros.
	- Coesão e Desacoplamento	Sim	No desenvolvimento como um todo.
6	**Organizadores e Estáticos**		
	- Espaço de Nomes (Namespace) criada pelos autores.	Sim	O Namespace *entities,* em identifiers.h, usado para identificação das entidades ao longo do projeto.
	- Classes aninhadas (Nested) criada pelos autores.	Sim	A classe Lista possui a classe Item aninhada.
	- Atributos estáticos e métodos estáticos.	Não	
	- Uso extensivo de constante (const) parâmetro, retorno, método…	Sim	No desenvolvimento como um todo.
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	A classe String foi usada, por exemplo, no parâmetro da construtora da classe Entity a fim de receber o caminho da textura das entidades. Vector foi usado na classe MapManager para armazenar objetos da classe Tile.
	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa OU Multi-Mapa.	Sim	Map foi usado na classe MapManager.
	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	Houve uso das seguintes classes da biblioteca gráfica SFML: Texture, Sprite, Vector2f, RectangleShape, RenderWindow, View, Clock, Event, Font, Color, entre outras.
	- Programação orientada e evento em algum ambiente gráfico. OU - RAD – Rapid Application Development (Objetos gráficos como formulários, botões etc).	Sim	A biblioteca gráfica SFML oferece suporte aos recursos de evento.
	Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.		
	- Ensino Médio.	Sim	Noções de sistemas de coordenadas, vetores, operações envolvendo vetores, movimentos unidimensionais, entre outros.
	- Ensino Superior.	Sim	Uso de conceitos mais avançados relacionados a corpos, aceleração, gravidade, elasticidade, colisões, entre outros.
9	**Engenharia de Software**		
	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	Concepção da ideia e análise dos requisitos.
	- Diagrama de Classes em UML.	Sim	Etapa de planejamento e modelagem de projeto.
	- Uso efetivo e intensivo de padrões de projeto GOF.	Não	
	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	Bateria de testes a fim de checar o funcionamento correto de funcionalidades previstas e concordância com o modelo de classes.
10	**Execução de Projeto**		

	- Controle de versão de modelos e códigos automatizados (via SVN e/ou afins). & - Uso de alguma forma de cópia de segurança (backup).	Sim	O Git, um sistema de controle de versões distribuído, foi usado visando o gerenciamento de versões do código. Além disso, foi mantido um repositório remoto na plataforma de hospedagem de código-fonte GitHub[1], permitindo, assim, maior controle e segurança do projeto.
	- Reuniões com o professor para acompanhamento do andamento do projeto.	Não	Foram feitas 3 reuniões sincronamente: 04/08/2021, 11/08/2021 e 13/08/2021. Um resumo dos assuntos debatidos e uma cópia do diagrama de classes foram enviados ao término de cada encontro.
	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Não	
	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Equipe Diogo Da Silva Gouveia e Gustavo Esmanhotto Bareta
Total de conceitos apropriadamente utilizados.			**85%** (oitenta e cinco por cento)

A tabela 3 apresenta as justificativas e as necessidades da utilização dos conceitos apresentados anteriormente na tabela 2.

Tabela 3. Lista de Justificativas para Conceitos Utilizados.

N	Conceitos	Justificativa
1	**Elementares**	Classes, objetos, atributos e métodos foram utilizados a fim de seguir o paradigma de programação orientada a objetos.
2	**Relações**	Baseado na composição e interação entre objetos, o paradigma de programação orientada a objetos justifica o uso de relacionamentos.
3	**Ponteiros, generalizações e exceções**	A fim de modificar os dados da própria instância, o operador *this* foi usado. A alocação dinâmica de memória possibilitou alocar/desalocar blocos de memória de acordo com a necessidade, reservando ou liberando blocos durante a execução do programa. Gabaritos/Templates facilitaram a manipulação dos objetos durante o desenvolvimento do programa. O tratamento de ocorrências de condições que alteram o fluxo normal de execução do programa foi feito a fim de controlar a execução correta da aplicação.
4	**Sobrecarga e Persistência**	Sobrecarga de construtoras e métodos permitiram mais flexibilidade no processo de desenvolvimento do programa. Já a sobrecarga de operadores possibilitou operações com os objetos criados pela equipe. A persistência dos objetos e seus

		relacionamentos permitem o salvamento do progresso do jogador ao serem armazenados e recuperados entre as execuções da aplicação.
5	**Virtualidade**	Virtualidade foi empregada por permitir que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Assim, possibilitando o tratamento de vários tipos de maneira homogênea.
6	**Organizadores e Estáticos**	*Namespaces* foram empregados a fim de organizar o código e evitar ambiguidade. As classes aninhadas, por sua vez, foram usadas por serem uma maneira de agrupar logicamente as classes em um só lugar, aumentar encapsulamento e legibilidade. Já o qualificador *const* foi usado a fim de evitar inconsistências no código pela indicação que os dados são apenas para leitura.
7	**Standard Template Library (STL) e String OO**	A classe *String* foi usada visando facilitar a manipulação de cadeias de caracteres, como os caminhos para recursos do programa. O uso da *Standart Template Library* agilizou o atingimento de efeitos buscados no programa, como acessar e percorrer vários elementos usando iteradores.
8	**Biblioteca Gráfica / Visual**	A biblioteca gráfica *Simple and Fast Multimedia Library* (SFML), versão 2.5.1, foi usada por ser orientada a objetos, livre e fornecer uma interface simples para vários componentes multimídia.
9	**Engenharia de Software**	O processo de levantar requisitos, modelar, implementar e testar o projeto foi seguido a fim de garantir maior qualidade e correção de falhas do software construído.
10	**Execução de Projeto**	O uso da ferramenta Git e da plataforma GitHub possibilitaram a equipe desenvolver o projeto simultaneamente sem o risco de ter o progresso sobrescrito. Além disso, assegurou a integridade do projeto caso os repositórios locais fossem perdidos ou se fosse preciso buscar uma versão antiga do programa. Quanto às reuniões realizadas com o professor, essas nortearam o desenvolvimento geral do projeto.

REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS

Ao comparar o paradigma de programação orientada a objetos e o estruturado, é nítido que existem grandes diferenças. Entre elas, destaca-se a divisão em classes, que torna o projeto utilizando linguagem orientada a objetos mais organizado ao dividir os atributos e métodos em suas respectivas classes que representam os objetos que precisam ser abstraídos.

Outras características relevantes na orientação a objetos é o encapsulamento, que restringe outras classes de acessarem determinados atributos e métodos inerentes de um objeto de outra classe, o que torna ela mais segura. Além disso, temos as heranças que possibilitam que algumas classes derivem de outras permitindo o compartilhamento de atributos e métodos. Há também o polimorfismo que possibilita duas classes distintas terem o mesmo método em comum, porém com execuções diferentes.

Enquanto a programação estruturada se baseia em uma sequência de passos, podendo haver algumas escolhas e loops durante sua execução, a orientada a objetos utiliza principalmente as associações entre as classes, que são reflexo das relações dos objetos abstraídos.

DISCUSSÃO E CONCLUSÕES

Ao realizar esse projeto foi possível experimentar um ambiente de produção, com docente no papel de gerente de projeto, e também do cliente instituindo os requisitos. Assim, a equipe se organizou para verificar as necessidades do projeto, bem como a modelagem do diagrama de classes necessário. Assim sendo, o trabalho foi dividido e cada membro executou uma parte para contribuir no resultado final. Posteriormente, apresentar para o "gerente de projeto", que por sua vez sugeriu melhorias e correções de falhas.

Outro ponto que fica nítido são as principais diferenças da programação orientada a objetos e da programação estruturada, principalmente o fato de abstrair os objetos com seus atributos e métodos que estão presentes no problema para classes, o que torna softwares que utilizam esse paradigma mais fáceis de serem entendidos.

Apesar de o resultado não ser o almejado inicialmente, o projeto foi concluído de forma satisfatória. Para isso, investiu-se mais tempo em partes críticas do que em outras não tão vitais para a trabalho (como a animação de sprites).

DIVISÃO DO TRABALHO

A tabela 4 demostra as atividades empenhadas para a realização do projeto como todo, assim como o responsável da dupla que exerceu essa atividade.

Tabela 4. Lista de Atividades e Responsáveis.

Atividades	**Responsáveis**
Compreensão de Requisitos	Secchi e Ribeiro
Diagramas de Classes	Secchi e Ribeiro
Programação em C++	Secchi e Ribeiro
Implementação de Template	Ribeiro
Implementação da Persistência dos Objetos	Mais Ribeiro que Secchi
Arte (sprites)	Mais Secchi que Ribeiro
Mapa das fases	Mais Secchi que Ribeiro
Material para apresentação	Secchi
Escrita do Trabalho	Mais Secchi que Ribeiro
Revisão do Trabalho	Mais Ribeiro que Secchi

- Ribeiro trabalhou em 100% das atividades ou as realizando ou colaborando nelas efetivamente.
- Secchi trabalhou em 100% das atividades ou as realizando ou colaborando nelas efetivamente.

AGRADECIMENTOS

Agradecimentos ao Prof. Dr. J. M. Simão por seu ensino e orientações, aos alunos Diogo da Silva Gouveia e Gustavo Esmanhotto Bareta pela revisão do documento e aos artistas que disponibilizaram seus trabalhos gratuitamente.

REFERÊNCIAS CITADAS NO TEXTO

[1] Endereço virtual para o repositório remoto no GitHub.
Último acesso em 15/08/2021, às 16:59:
https://github.com/pedrosribeiro/SFMLJogo2D

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A] SIMÃO, J. M. Site da Disciplina de Técnicas de Programação, Curitiba – PR, Brasil.
Último acesso em 15/08/2021, às 17:02:
https://pessoal.dainf.ct.utfpr.edu.br/jeansimao/Fundamentos2/Fundamentos2.htm