TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR: LAGGANDINO – RELATÓRIO DO PROJETO

Mário Cordeiro Junior
mario.2018@alunos.utfpr.edu.br

Disciplina: **Técnicas de Programação – CSE20** / S73 – Prof. Dr. Jean M. Simão
Departamento Acadêmico de Informática – DAINF - Campus de 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 - A disciplina de Técnicas de Programação possui como parte de sua estrutura o aprendizado prático de técnicas de engenharia de software. Para isso, a atividade final exige o desenvolvimento de um jogo de plataforma, aplicando os conceitos de programação orientada a objetos em C++ aprendidos ao longo das aulas. Para tal, o jogo desenvolvido foi o *Laggandino Game*, que se passa em um cenário fantasioso, no qual o jogador enfrenta obstáculos e inimigos para chegar ao fim das fases. Tendo no total duas fases, diferenciadas por cenários, em que a última possui um chefão a ser derrotado para finalizar o jogo. Os requisitos de desenvolvimento considerados foram propostos textualmente, assim como a modelagem (análise e projeto) considerada, em que o Diagrama de Classes em Linguagem de Modelagem Unificada (*Unified Modeling Language - UML*) foi feito tendo como base um diagrama genérico, previamente proposto. Seguidamente, foi iniciado o desenvolvimento utilizando a linguagem de programação C++, como também uma biblioteca externa com suporte gráfico Simple and Fast Multimedia Library (SFML), que permitiu a utilização de conceitos de Orientação a Objetos como Classes Abstratas, Namespaces, Polimorfismo, Heranças e Sobrecarga de Operadores. Posteriormente, foi utilizado um processo de testes manuais, a fim de verificar a integridade do código e o cumprimento dos requisitos previamente exigidos. Por fim, considera-se que o desenvolvimento em questão permitiu cumprir o objetivo de aprendizado visado.

Palavras-chave ou Expressões-chave: Artigo-Relatório para o Trabalho em Técnicas de Programação, Jogo Plataforma utilizando Biblioteca Gráfica em C++, Orientação a Objetos com C++ e Biblioteca Gráfica, Trabalho Final da Disciplina Técnicas de Programação.

Abstract - *The Programming Techniques course has, as part of its structure, the practical learning of software engineering techniques. For such, the final activity requires the development of a platform game, applying the concepts of object-oriented programming in C++ learned throughout the classes. Towards this end, the developed game was Laggandino Game, which takes place in a fantasy setting, in which the player faces obstacles and enemies to reach the end of the stages. Having in total two stages, differentiated by scenery, in which the last one has a boss to be defeated to finish the game. The development requirements considered were proposed textually, as well as the modeling (analysis and design) considered, in which the Class Diagram in Unified Modeling Language (UML) was made based on a previously proposed generic diagram. Then, development was started using the C++ programming language, as well as an external library with Simple and Fast Multimedia Library (SFML) graphic support, which allowed the use of Object Oriented concepts such as Abstract Classes, Namespaces, Polymorphism, Inheritance and Operator Overload. Subsequently, a manual testing process was used in order to verify the integrity of the code and the fulfillment of the previously required requirements. Finally, it is considered that the development in question allowed the achievement of the intended learning objective.*

Key-words or Key-expressions: Article-Report for the Final Project in the Subject Programming Techniques, Platform Game using C++ Graphics Library, Object Oriented Programming with C++ and Graphics Library, Final Project of the Subject Programming Techniques

INTRODUÇÃO

Este documento se trata do artigo-relatório para o jogo *Laggandino Game*, desenvolvido por Mário Cordeiro Junior, representando um quesito avaliativo parcial para a disciplina

Técnicas de Programação, do Prof. Dr. Jean M. Simão, no curso de Bacharel em Sistemas de Informação da Universidade Tecnológica Federal do Paraná (UTFPR). Para o registro dos procedimentos realizados para o desenvolvimento do projeto, bem como para consolidar as evidências do seguimento de boas práticas, esse documento visa evidenciar o cumprimento dos objetivos definidos para a elaboração do jogo.

Dentre os diversos fatores desempenhando as diretrizes atribuídas a este sistema, o trabalho objetiva a concretização do aprendizado em aula, abrangendo os conceitos estudados ao longo do período acadêmico. Dessa forma, o *Laggandino Game* possui complexidade de forma a utilizar diversos recursos da linguagem C++, além da biblioteca gráfica Simple and Fast Multimedia Library (SFML). Assim, estabelece-se a oportunidade de ampliar conceitos e aprimorar habilidades através do cumprimento dos requisitos solicitados.

O método utilizado para este trabalho fundamenta-se no ciclo clássico de Engenharia de Software, constituído por quatro etapas, cada uma com sua devida relevância. As duas primeiras englobam o planejamento, em que há o levantamento de requisitos, além da elaboração do UML. Seguidamente, as duas etapas finais retratam a execução do projeto, com a codificação em programação orientada a objetos, e finalizando em estágio de testes. Dessa forma, é concretizado um regime consistente de estruturação de software, uma vez que cada fase do ciclo é sistematizada coerentemente.

Para as próximas seções, primeiro será realizada a explicação do jogo em si, e então, será exposta a parte de desenvolvimento do jogo. Haverá, também, a exposição de tabelas de conceitos utilizados, reflexões e considerações do projeto. Uma vez apresentado do que se trata o conteúdo presente neste documento, segue a dissertação do projeto.

EXPLICAÇÃO DO JOGO EM SI

O nome “Laggandino Game” idealizou-se com uma temática envolvendo o jogo de palavras com “lag” (referindo-se a um “atraso”) e “dino” (referindo-se a dinossauros), utilizando a intenção cômica de associar um grupo de animais extintos à uma ação atrasada. Como o nome sugere, os protagonistas do jogo são dois dinossauros, capazes de se movimentar e disparar projéteis, que objetivam matar seus adversários, e, por fim, o chefão.

Ao iniciar o programa, é exibido o menu principal de opções, no qual o usuário pode escolher jogar com um ou dois jogadores, carregar uma jogada anterior, exibir o ranking ou sair. A navegação das opções é feita com as setas do teclado e a seleção é feita com a tecla ENTER.

Figura 1. Menu Principal.

Feita a seleção no menu principal, a fase “Montanha” é iniciada. A jogabilidade segue os clássicos de plataforma da era 32 bits, em que o jogador, enquanto em cima de uma plataforma, tem livre mobilidade na horizontal, pode pular e é afetado pela aceleração da gravidade. Como forma de ataque, é disparado um projétil de fogo que descreve movimento retilíneo e tem duração de meio segundo.

Compondo o cenário de jogo, tem-se diversos dinossauros inimigos. Sendo eles: Andino, inimigo comum que possui movimentação horizontal padrão e pula aleatoriamente; Atiradino, que não se movimenta e atira projéteis de fogo para a esquerda; Chefedino, que representa o desafio final da fase, tem tamanho e velocidade maiores que o Andino e mais vidas que os outros inimigos comuns.

Figura 2. Fase Floresta com os dois jogadores.

Como parte adicional do cenário, as fases agregam tipos de obstáculos, que diferem na maneira que interagem com o jogador: espinhos, matam o jogador ao contato e aparecem em ambas as fases; Pedras, que podem ser movidas, sofrem ação da gravidade e são exclusivas da montanha; Galhos, que são incinerados quando entram em contato com o projétil disparado pelo jogador.

Diferente de outras entidades, inimigos possuem vidas, sendo necessários múltiplos acertos de projétil do jogador para abatê-los. De maneira análoga, quando jogadores sofrem dano, seja por contato com inimigo ou por ameaças do ambiente, estes perdem vida e são reposicionados na fase, de forma que se algum dos dois ficar sem vida, o jogo acaba e volta para o menu principal.

Em qualquer momento, o jogador pode apertar a tecla ESC para adentrar o menu de pause, acessando assim as opções de: voltar para a fase, salvar a jogada atual, verificar o ranking, carregar uma jogada anterior, ou voltar ao menu principal. Após chegar à última plataforma da montanha, o jogador é levado à fase floresta, que por sua vez ao ser completada, o jogo é finalizado, então o usuário vai para a tela de registro, na qual digita seu nome e sua pontuação é salva no ranking geral dos campeões.

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Ao longo do seguimento, estruturam-se as ideias do jogo (tratadas na seção anterior) consoantemente às exigências designadas dentro do âmbito avaliativo do que a disciplina institui. Dessa maneira, o trabalho foi desenvolvido iniciando-se com a compreensão dos

requisitos fornecidos para o desenvolvimento do jogo. Oportunamente, esse conjunto de requisitos está apresentado na Tabela 1 tal qual foram apresentados inicialmente, sendo que também consta a situação deles e onde foram contemplados. Para tanto, segue-se abaixo a tabela com as respectivas informações.

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

N.	Requisitos Funcionais	Situação	Implementação
1	Apresentar graficamente menu de opções aos usuários do Jogo.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe base Menu e suas respectivas classes especializadas MenuPrincipal e MenuPause.
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.	Requisito cumprido via classe Jogador e suas derivadas Vita e Tard cujos objetos são agregados nas duas fases.
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas, via menu, nas quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Fase e suas derivadas Montanha e Floresta que podem ser jogadas de maneira sequencial.
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 projéteis contra o(s) jogador(es) e um dos inimigos deve ser um 'Chefão'.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Inimigo e suas derivadas Andino, Atiradino, AtiradinoThread e ChefeDino sendo que os três últimos atiram projéteis no(s) jogador(es).
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 classe FabricaFase e suas derivadas FabricaMontanha e FabricaFloresta, sendo que essas fábricas instanciam aleatoriamente os inimigos nas fases, sendo pelo menos 3 inimigos de cada.
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Obstáculo e suas derivadas Espinho, Pedra e Galho, sendo Espinho o obstáculo que causa dano.
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 classe Obstáculo e suas derivadas Espinho, Pedra e Galho.
8	Ter em cada fase um cenário de jogo constituído por obstáculos, sendo que parte deles seriam	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Plataforma que molda o cenário da fase.

	plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores.		Aleatoriamente em cima de cada plataforma, podem ser instanciados inimigos e/ou obstáculos.
9	Gerenciar colisões entre jogador para com inimigos e seus projéteis, bem como entre jogador para com obstáculos. Ainda, todos eles devem sofrer o efeito da gravidade no âmbito deste jogo de plataforma vertical e 2D.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe GerenciadorColisoes que gerencia as colisões entre as entidades da fase.
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 **Salvar** Jogada.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe GerenciadorPersistencias que é chamada por classes derivadas de Estado para salvar a jogada. Para pausar o jogo é usada a classe MenuPause, e para o ranking e cadastro de usuário é usada a classe Menu. Para a pontuação se faz o uso da classe Fase.
Total de requisitos funcionais apropriadamente realizados. *(Cada tópico vale 10%, sendo que para ser contabilizado deve estar realizado efetivamente e não parcialmente)*			**100%** (cem por cento).

Durante o desenvolvimento do projeto, foram considerados como base os requisitos funcionais listados na Tabela 1 e o diagrama de classes em UML fornecido pelo professor[1]. O diagrama foi desenvolvido, com a supervisão do professor, até chegar em sua versão final na Figura 3. Tal modelo possibilita desta forma, ter uma visão clara dos objetivos e organização geral do trabalho, bem como os caminhos a se seguir na etapa de implementação do código.

Como fundamento do projeto tem-se o agrupamento de classes chamado Controladoras, no qual tem em seu cerne a classe Jogo, que é a gerenciadora principal do sistema. Para a execução das diversas funcionalidades, a classe principal utiliza o padrão de projeto State, agregando desta forma uma classe PilhaEstados, que organiza as derivadas da classe abstrata Estado e garante o polimorfismo, uma vez que a classe Principal chama a função executar do estado mais acima da pilha, os menus e as fases, e tem como núcleo uma pilha padrão da Biblioteca Padrão de Gabaritos (Standard Template Library - STL).

As principais funções das fases são realizar a construção do cenário e gerenciar interações entre as entidades que o compõem. Para realizar a primeira tarefa foi utilizado o padrão de projeto Factory Method, de forma que o estado mais atual da pilha chama a construção da fase e de seus componentes, tal construção pode ser feita de duas formas, criando uma novo jogo do início ou começando a partir de um anterior.

Ao começar uma nova jogada, um arquivo de plataformas é lido e, em cima de cada uma delas, é instanciado um inimigo ou obstáculo de tipo aleatório específico para a determinada fase que se está construindo. Caso o usuário opte por iniciar a partir de um jogo anterior, são lidos arquivos de todas as entidades previamente instanciadas, bem como os diferentes atributos do jogador, como quantidade de pontos e vidas.

A fim de realizar as relações entre os elementos, toda fase agrega um Gerenciador_Colisões[2] e uma lista de entidades, classe que tem como cerne uma lista

encadeada de entidades, de forma que a primeira utiliza a segunda para realizar colisões entre os objetos, técnica que só é possível devido à especialização de atributos de controle (podeMorrer, podeMatar e empurrão), feita de acordo com as necessidades de cada classe derivada de entidade. O modelo de detecção de colisão adotado foi o AABB².

Com o intuito de garantir maior variedade e exploração de conceitos, foram criados diversos tipos de inimigos, obstáculos e plataformas para compor o cenário, que a fim de garantir maior coesão, realizam polimorfismo ao especializarem as funções de imprimir e executar da classe abstrata Entidade, bem como os atributos específicos para o gerenciamento de colisões, possibilitando o tratamento de forma genérica pelas classes controladoras da fase.

Pelo fato de ter sido usada uma biblioteca gráfica e para garantir maior desacoplamento, todas as entidades agregam as classes de apoio CorpoGráfico e Animadora³, que tratam de tudo relacionado à representação gráfica de texturas e animações. De maneira análoga e como parte da infraestrutura geral do projeto tem-se a classe GerenciadorGráfico, conhecida por derivadas de entidade e estados, que gerencia estruturas de base como janela, fundo e câmera.

Além disso vale ressaltar que cada um dos diferentes tipos de personagens e obstáculos contemplam conceitos interdisciplinares anteriormente estudados. Exemplos de aplicação de conceito são: Andinos capazes de pular e desta forma serem afetados pela aceleração da gravidade, além de realizar movimento parabólico ao pular; Atiradinos que não andam, mas atiram projéteis que realizam movimento uniformemente variado com velocidade decrescente; e pedras, que diminuem a velocidade do jogador ao serem empurradas, caracterizando conceitos de atrito.

A fim de realizar maior exploração de conceitos, a técnica multithreading foi implantada no inimigo Atiradino, para tal foram criadas as classes PThread, que provém a infraestrutura geral de threads usando Posix Thread, e AtiradinoThread que deriva do inimigo comum e da classe de infraestrutura. Desta forma os inimigos derivados de Thread atiram, controlados por mutex, um a cada três segundos. A implementação feita das threads é uma adaptação dos slides do professor.

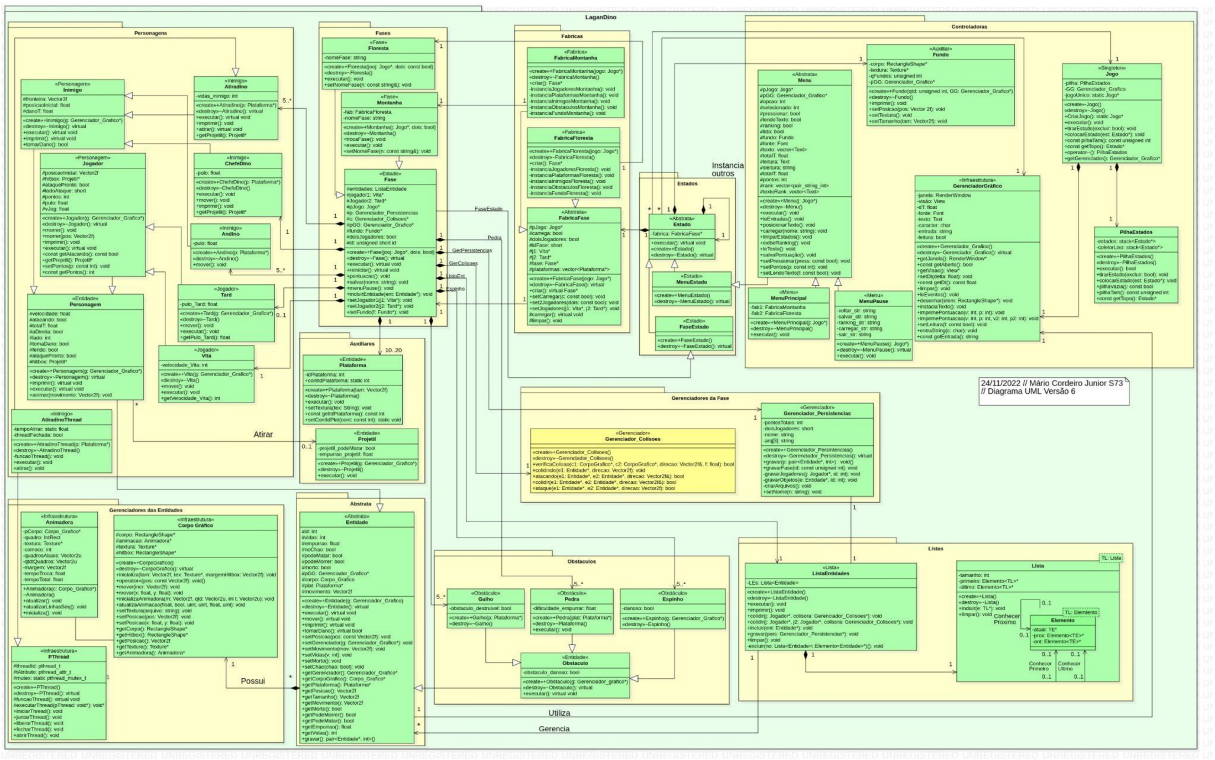

Figura 3. Diagrama de Classes de base em UML.

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

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

	N.	Conceitos	Uso	Onde / O quê
	1	**Elementares:**		
	1.1	- Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno).	Sim	Todos .h e .cpp.
	1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Todos .h e .cpp.
	1.3	- Classe Principal.	Sim	Main.cpp & Principal.h/.cpp.
	1.4	- Divisão em .h e .cpp.	Sim	No desenvolvimento como um todo.
	2	**Relações de:**		
	2.1	- Associação direcional. & - Associação bidirecional.	Sim	No desenvolvimento como um todo.
	2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	No desenvolvimento como um todo.
	2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	No desenvolvimento como um todo.
	2.4	- Herança múltipla.	Não	AtiradinoThread.h.
	3	**Ponteiros, generalizações e exceções**		
	3.1	- Operador *this* para fins de relacionamento bidirecional.	Não	No desenvolvimento como um todo. Ex Corpo_Grafico.cpp.
	3.2	- Alocação de memória (*new* & *delete*).		No desenvolvimento como um todo. Ex FabricaFase.cpp.
	3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).		Lista.h.
	3.4	- Uso de Tratamento de Exceções (*try catch*).		Gerenciador_Persistencias.cpp.
	4	**Sobrecarga de:**		
	4.1	- Construtoras e Métodos.		No desenvolvimento como um todo.
	4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).		Corpo_Grafico.h & PilhaEstados.h
	---	**Persistência de Objetos (via arquivo de texto ou binário)**		
	4.3	- Persistência de Objetos.		Gerenciador_Persistencias.cpp, FabricaFase.cpp.
	4.4	- Persistência de Relacionamento de Objetos.		Gerenciador_Persistencias.cpp.
	5	**Virtualidade:**		
	5.1	- Métodos Virtuais Usuais.		No desenvolvimento do projeto como um todo.
	5.2	- Polimorfismo.		No desenvolvimento do projeto como um todo.

	5.3	- Métodos Virtuais Puros / Classes Abstratas.		No desenvolvimento do projeto como um todo.
	5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.		No desenvolvimento do projeto como um todo.
	6	**Organizadores e Estáticos**		
	6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.		Todos os .h e .cpp.
	6.2	- Classes aninhadas (*Nested*) criada pelos autores.		Lista.h.
	6.3	- Atributos estáticos e métodos estáticos.		PThread.h, Jogo.h, AtiradinoThread.h e Plataforma.h.
	6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...		No desenvolvimento do projeto como um todo.
	7	**Standard Template Library *(STL)* e String OO**		
	7.1	- A classe Pré-definida *String* ou equivalente. **&** - *Vector* e/ou *List* da *STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores)		No desenvolvimento do projeto como um todo. Ex Menu.h e Menu.cpp.
	7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.		PilhaEstados.h e PilhaEstados.cpp
	---	**Programação concorrente**		
	7.3	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins.		PThread.h, PThread.cpp, AtiradinoThread.h e AtiradinoThread.cpp.
	7.4	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.		PThread.h, PThread.cpp, AtiradinoThread.h e AtiradinoThread.cpp.
	8	**Biblioteca Gráfica / Visual**		
	8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: ● tratamento de colisões ● duplo *buffer*		Biblioteca SFML 2.5.1: Gerenciador_Grafico.h & Gerenciador_Grafico.cpp.
	8.2	- Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive) em algum ambiente gráfico. **OU** - *RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).		Gerenciador_Grafico.cpp, Menu.cpp, Vita.cpp, Tard.cpp, Jogo.cpp.
	---	**Interdisciplinaridades via utilização de Conceitos de <u>Matemática Contínua e/ou Física</u>.**		
	8.3	- Ensino Médio Efetivamente.		Álgebra, Funções, Gravidade, Probabilidade, Geometria plana, Geometria Analítica, Movimento acelerado.
	8.4	- Ensino Superior Efetivamente.		Lógica, Estrutura de dados, Geometria Analítica, Cálculo 1, Velocidade Instantânea.
	9	**Engenharia de Software**		
	9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &		Diagrama de Classes como apoio, e impressão do modelo para trabalho com o fim de melhor atender os requisitos. Outrossim, foram marcadas reuniões com o monitor e o professor para sanar as eventuais dúvidas.

	9.2	- Diagrama de Classes em *UML*.		Largamente utilizado na implementação do projeto, sendo constantemente atualizado tendo em sua totalidade 6 versões.
	9.3	- Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões.		Singleton, Composite, Factory Method, State.
	9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.		Foram utilizadas ferramentas para debugar e testar o código como gdb e valgrind, sempre visando atender os requisitos satisfatoriamente.
	10	**Execução de Projeto**		
	10.1	- Controle de versão de modelos e códigos automatizados (via github e/ou afins). & - Uso de alguma forma de cópia de segurança (*i.e.*, *backup*).		Utilizando a ferramenta Git e uma de umas respectivas interfaces acompanhado de um repositório online chamado GitHub.
	10.2	- Reuniões com o professor para acompanhamento do andamento do projeto.		Reuniões nos dias: 03/11, 10/11, 17/11, 24/11.
	10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.		Reuniões online nos dias: 7/11, 14/11, 21/11
	10.4	- Revisão do trabalho escrito de outra equipe e vice-versa.		Equipe formada pelos alunos Pedro Lucas e Ian Neves.
	Total de conceitos apropriadamente utilizados.			**100%** (cem por cento).

Tabela 3. Lista de Justificativas para Conceitos Utilizados.

No.	**Conceitos**	***Listar apenas os utilizados* Situação**
1	**Elementares**	Classe, objetos, atributos, variáveis, constantes e métodos foram utilizados, por serem fundamentais para a programação orientada a objetos. Métodos com retorno const ou parâmetro const foram utilizados principalmente para o bom funcionamento do código em geral.
2	**Relações**	Herança e associações foram utilizadas para criar uma relação entre as classes.
3	**Ponteiros, generalizações e exceções**	O Operador this foi utilizado porque foi extremamente necessário. A alocação de memória foi utilizada para aproveitamento de memória e melhor desempenho do programa. Gabaritos e try catch foram utilizados somente como exemplos para cumprir a tabela de conceitos.
4	**Sobrecarga e Persistência**	Sobrecarga de métodos e construtoras foram utilizadas pois se fez muito necessário para o desenvolvimento do projeto. Sobrecarga de operadores foi utilizada somente como exemplo para cumprir a tabela de conceitos. Persistência foi utilizada para manter dados gravados fora da execução do programa.
5	**Virtualidade**	A virtualidade foi utilizada porque é fundamental para o paradigma OO.
6	**Organizadores e Estáticos**	Namespace e Nested foram utilizados porque se fez necessário para organizar o código. Estáticos foram utilizados porque se fez extremamente necessário para o funcionamento do programa. Constantes foram utilizadas porque são necessárias para o bom desenvolvimento do código.
7	**Standard Template Library (STL) e String OO**	Classes da STL foram utilizadas porque foram necessárias para facilitar o desenvolvimento de algumas soluções.

8	**Biblioteca Gráfica / Visual**	A Biblioteca Gráfica e suas funcionalidades foram utilizadas porque foram extremamente necessárias para a viabilização do desenvolvimento do jogo.
9	**Engenharia de Software**	O ciclo da engenharia de software foi utilizado para melhor planejar o desenvolvimento, tornando-o mais organizado e poupando tempo.
10	**Execução do Projeto**	O versionador de arquivos foi utilizado para registrar o progresso do desenvolvimento e tornar fácil a restauração do código em caso de erros graves. As reuniões com o professor foram úteis para mensurar os avanços do projeto, bem como o esclarecimento de dúvidas.

REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS

Um desenvolvimento procedimental, devido a sua simplicidade de execução, demandaria um grande grau de dificuldade quando se trata de implementações de código em escalas mais robustas e complexas, como foi no jogo desenvolvido.

Em contrapartida, um desenvolvimento voltado a POO é muito mais consistente com suas múltiplas funcionalidades, em virtude das características referentes às propriedades de organização, encapsulamento e reutilização. Além disso, o uso de classes auxilia na capacidade de abstração do código, visto que a relação entre elas possui comportamentos análogos à realidade.

DISCUSSÃO E CONCLUSÕES

O desenvolvimento do trabalho como um todo possibilitou ao aluno aplicar, como demonstrado na Tabela 3, toda a gama de conceitos referente ao paradigma de Programação Orientada a Objetos.

Inicialmente, na etapa de modelagem (análise e projeto) foram levantados diversos requisitos, referentes tanto à execução do diagrama de classes em UML quanto à implementação do código em si. Em tempo foram corrigidas e aperfeiçoadas, em reuniões iniciais com o professor, certas particularidades postuladas no diagrama de classes base[1] e no modelo para elaboração artigo-relatório[3], a fim de melhor atender as necessidades do projeto.

Referente à elaboração do diagrama de classes do projeto e posteriormente implementação do código, foram realizadas múltiplas reuniões para acompanhar o desenvolvimento do trabalho, com o intuito de proporcionar ao aluno uma situação real de engenharia de software, no qual o professor e o monitor da disciplina representam profissionais de maior hierarquia.

Salienta-se que todos os requisitos foram cumpridos, com seus respectivos conceitos utilizados, seja em ampla escala ou em situações específicas. Em relação a isso, destaca-se que a implementação de padrões de projeto foi especialmente benéfica, dado que tal prática é de suma importância para o paradigma OO e seu conhecimento representa amadurecimento das habilidades necessárias a um bom programador. Nesse sentido, conclui-se que a execução e resultado do trabalho foram, considerando o intuito de aprendizado, muito satisfatórios.

CONSIDERAÇÕES PESSOAIS

Tendo em vista o projeto como um todo, considera-se a experiência como muito gratificante do ponto de vista didático e acadêmico, principalmente quando se considera o nível de autonomia que foi necessário para o desenvolvimento do trabalho, tanto na parte de usar uma ferramenta de modelagem, neste caso StarUML, quanto na de aprender a usar uma

biblioteca gráfica, conhecimentos que certamente apresentam grande enriquecimento no aprendizado e formação profissional do aluno de Sistemas de Informação.

DIVISÃO DO TRABALHO

Assim como definido na última reunião do projeto, realizada em 24 de novembro de 2022, o membro Pedro Enzo foi removido do grupo, devido à falta de participação no desenvolvimento do jogo, tanto na realização (modelagem e escrita de código) quanto na colaboração (revisão de código e testes). A remoção do membro foi alinhada com o Prof. Dr. Jean M. Simão, que validou a decisão.

Tabela 4. Lista de Atividades e Responsáveis.

Atividades	Responsáveis
Compreensão de Requisitos	Mário Cordeiro Junior
Diagramas de Classes	Mário Cordeiro Junior
Programação em C++	Mário Cordeiro Junior
Implementação de *Template*	Mário Cordeiro Junior
Implementação da Persistência dos Objetos	Mário Cordeiro Junior
Tratamento de Colisões	Mário Cordeiro Junior
Gerenciador Gráfico	Mário Cordeiro Junior
Menus	Mário Cordeiro Junior
Inimigos	Mário Cordeiro Junior
Obstáculos	Mário Cordeiro Junior
Jogadores	Mário Cordeiro Junior
Fases	Mário Cordeiro Junior
Escrita do Trabalho	Mário Cordeiro Junior
Revisão do Trabalho	Mário Cordeiro Junior

Mário Cordeiro Júnior trabalhou em 100% das atividades citadas, realizando-as efetivamente.

AGRADECIMENTOS

Agradeço ao monitor Giovani Limas Salvi pelos vídeos de explicações e links para melhor entendimento de alguns assuntos.

REFERÊNCIAS CITADAS NO TEXTO

[1] SIMÃO, J. M. Diagrama de Classes de Base, Curitiba – PR, Brasil, Acessado em 22/11/2022, às 16:00:

http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/TopicosTrab/DiagramaJogoModelo.jpg

[2] VONCK, Hilze. SFML 2.4 For Beginners - 12: Collision Detection (AABB). 2016. (19m36s). Acesso em: 08/10/2022. Disponível em: <https://www.youtube.com/watch?v=l2iCYCLi6MU & list= PL21OsoBLPpMOO6zyVlxZ4S4hwkY_SLRW9 & index=13> .

[3] VONCK, Hilze. SFML 2.4 For Beginners - 9: Animation. 2016. (19m36s). Acesso em: 09/10/2022. Disponível em:

<https://www.youtube.com/watch?v=Aa8bXSq5LDE&list=PL21OsoBLPpMOO6zyVlxZ4S4hwkY_SLRW9&index=11&t=0s>.

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A] BEZERRA, E. Princípios de Análise e Projeto de Sistemas com UML. Editora Campus. 2003. ISBN 85-352-1032-6.

[B] HORSTMANN, C. Conceitos de Computação com o Essencial de C++, 3ª edição, Bookman, 2003, ISBN 0-471-16437-2.

[C] DEITEL, H. M.; DEITEL, P. J. C++ Como Programar. 5ª Edição. Bookman. 2006.

[D] SIMÃO, J. M. Site das Disciplina de Fundamentos de Programação 2, Curitiba – PR, Brasil, Acessado em 09/10/2022, às 15:15:

http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/Fundamentos2.htm.