TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR: *INVASÃO ALIENÍGENA*

Alfons Carlos Cesar Heiermann de Andrade, Anna Caroline de Oliveira Sousa
alfons@alunos.utfpr.edu.br, annacaroline@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 – A disciplina de Técnicas de Programação, lecionada pelo Professor Dr. Jean Marcelo Simão, exige o desenvolvimento de um *software* de plataforma, no formato de um jogo, para fins de aprendizado e aprimoramento de técnicas de engenharia de *software*, particularmente de programação orientada a objetos em C++. O jogo desenvolvido apresenta duas fases, com dificuldades variadas, e dispõe de um ou dois jogadores, os quais podem jogar concomitantemente e terão que enfrentar algumas categorias de inimigos diferentes, a depender da fase. Para o desenvolvimento do presente jogo, foram considerados os requisitos inicialmente propostos pelo professor, além de se ter à disposição uma modelagem de classe, a fim de projetar e idealizar as classes a serem constituídas usando como recurso o Diagrama de Classes em Linguagem de Modelagem Unificada (*Unified Modeling Language – UML*). Com isso, em linguagem de programação C++ e com o auxílio da biblioteca SFML, realizou-se o desenvolvimento do software contemplando os conceitos usuais de Orientação a Objetos como Classe, Objeto e Relacionamento, assim como também alguns conceitos avançados como Classe Abstrata, Polimorfismo, Gabaritos, Sobrecarga de Operadores e Biblioteca Padrão de Gabaritos (*Standard Template Library - STL*). Desse modo, após a implementação dos requisitos e testes com os mesmos, é possível salientar que houve uma funcionalidade conforme o esperado.

Palavras-chave ou Expressões-chave: Estudo de Linguagem de programação Orientada a objetos (C++), Um jogo de plataforma, Desenvolvimento de um Jogo, Uso da biblioteca gráfica SFML.

Abstract – The subject Programming Techniques taught by Professor Dr. Jean Marcelo Simão requires the development of a platform software, in the format of a game, for learning and improving software engineering techniques, particularly object-oriented programming in C++. Besides, the game developed has one or two players - where both can play concurrently - who will have to face some types of enemies, different depending on the phase, and it has two phases - which offer varying difficulties. Nevertheless, for the development of this game, the requirements initially proposed by the professor were considered, in addition to having a class modeling to design and idealize the classes to be constituted using the Class Diagram in Unified Modeling Language (UML) as a resource. On the other hand, in C++ programming language, the functions, and graphical implementations were developed - with the help of the SFML library - contemplating the usual concepts of Object Orientation such as Class, Object and Relationship, as well as some advanced concepts such as Abstract Class, Polymorphism, Templates, Operators Overloading and Standard Template Library (STL). Therefore, after implementing the requirements and testing them, it is possible to point out that the functionality was as expected.

Key-words or Key-expressions: Study of Object-Oriented Programming language (C++), A platform game, Developming a Game, Using the SFML graphical library.

INTRODUÇÃO

Este documento apresenta o trabalho desenvolvido para a disciplina Técnicas de Programação, lecionada pelo Professor Dr. Jean Marcelo Simão, no curso de Engenharia de

Computação na UTFPR. Ademais, foi desenvolvido um jogo de plataforma com tema e mecânicas selecionadas pelos discentes e aprovadas pelo docente.

Concomitantemente, para o desenvolvimento e realização do mesmo foi necessário a compreensão dos requisitos solicitados e concretização da modelagem do presente projeto, através de um diagrama de classe (UML), onde foram levantados os requisitos e as principais relações entre as classes que seriam constituídas.

Aliás, cabe ressaltar que para a elaboração do mesmo foi buscado seguir os seguintes requisitos: dispor do ciclo da Engenharia de Software, programação orientada a objetos (POO), sendo que a mesmo foi o implementado na linguagem de programação C++, além de utilizar a biblioteca gráfica SFML, de modo a auxiliar no desenvolvimento e implementações visuais. Sendo que, no decorrer de sua implementação, o jogo foi testado e aprimorado o seu funcionamento.

Além disto, as seções que seguem explicam o jogo em si, bem como a sua implementação e desenvolvimento.

EXPLICAÇÃO DO JOGO EM SI

O jogo “Invasão Alienígena” é um jogo de plataformas composto por duas fases distintas, o qual pode ser jogado por um ou dois jogadores. Quando o programa é executado é exibido o Menu Principal, dispondo de algumas opções como demostrado na figura 01. Além disso, durante o processo de seleção das características do jogo, caso o usuário deseje voltar ao menu principal também é possível através do botão Menu Principal, como demonstrado na figura 02.

Outrossim, após inicializado a execução, caso o usuário deseje pausar o jogo é necessário pressionar tecla ESC, sendo que quando a mesma é pressionada será apresentado ao usuário algumas opções, como representado pela Figura 03.

Figura 01: Menu Principal.

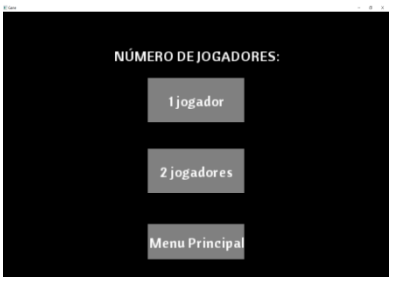

Figura 02: Menu Jogador

Figura 03: Menu Pausa.

A primeira fase é constituída por plataformas, na qual o jogador ou jogadores, encontram determinados obstáculos a serem enfrentados. Tal fase é representada pelas Figuras 04 e 05. Ademais, na fase presente há duas categorias de inimigos diferentes – LagartoVerde e o Robotão, além de que a mesma é composta por alguns obstáculos sendo os

mesmos: espinhos, caixote – sucedendo que tanto inimigos quanto obstáculos possuem instanciações distintas.

Figura 04: Manicômio (Primeira Fase).

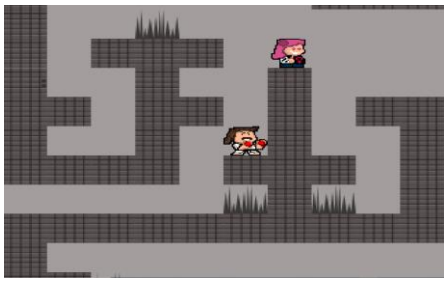

Figura 05: Manicômio (Primeira Fase).

Além disso, a segunda fase do jogo dispõe de semelhanças bem como características próprias. Tal fase pode ser observadas nas Figuras 06 e 07. Dentre estas características, se destacam um novo obstáculo bem como um novo inimigo, denominados de buraco infinito e Ciclope (chefão) respectivamente, encontrado no final da segunda fase representando o último desafio da fase.

Figura 06: Hospital (Segunda Fase).

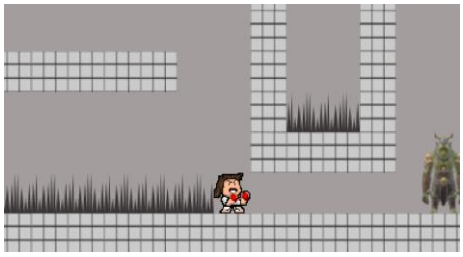

Figura 07: Hospital (Segunda Fase).

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

O desenvolvimento do Jogo se deu à luz dos requisitos propostos pelo Professor da disciplina. Tais requisitos são apresentados na Tabela 1. Nesta tabela há também a situação do requisito no tocante a sua realização. Caso tenha sido realizado, ainda é indicado em que parte do sistema isto ocorreu. De antemão, faz-se contar que foram cumpridos 86,7% dos requisitos propostos.

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	Requisito cumprido via pacote Menu e seus respectivos

		realizado.	objetos.
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 via classe MenuJogador, bem como o GerenciadorTelas.
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas.	Requisito previsto inicialmente e realizado.	Requisito cumprido via pacote Fase, bem como MenuConfigurações, em que o usuário pode escolher a fase.
4	Ter três tipos distintos de inimigos (o que pode incluir 'Chefão', vide abaixo), sendo que pelo menos um dos inimigos deve ser capaz de lançar projetil contra o(s) jogadores(es).	Requisito previsto inicialmente e (realizado).	Requisito realizado, sendo que os inimigos estão contidos no pacote Desenhável e o inimigo responsável pelo projétil é o Robotão.
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 por tipo.	Requisito previsto inicialmente e realizado.	Requisito realizado, sendo que o mesmo se encontra presente nas seguintes classes: Manicômio e Hospital.
6	Ter inimigo "Chefão" na última fase.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Ciclope e seu objeto.
7	Ter três tipos de obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido via pacote Obstáculo, bem como no pacote de Tile.
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 pacote Obstáculos, bem como TileMap – pelo fato de dispor de espinho presente no mapa.
9	Ter representação gráfica de cada instância.	Requisito previsto inicialmente e realizado.	Requisito cumprido via GerenciadorGráfico.
10	Ter em cada fase um cenário de jogo com os obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido via pacote de Tile, inclusive via pacote Obstáculo.
11	Gerenciar colisões entre jogadores e inimigos, bem como seus projeteis (em havendo).	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe GerenciadorColisão, bem como o pacote Desenháveis.

12	Gerenciar colisões entre jogadores e obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe Jogador, bem como GerenciadorColisão.
13	Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação *(ranking).*	Requisito previsto inicialmente e parcialmente realizado.	Requisito parcialmente realizado, sendo que a pontuação é contabilizada por meio das seguintes classes: Jogador e Coletáveis.
14	Permitir Pausar o Jogo.	Requisito previsto inicialmente e realizado.	Requisito cumprido, inclusive via GerenciadorEventos e GerenciadorEstados, bem como seus objetos.
15	Permitir Salvar Jogada.	Requisito previsto inicialmente e não realizado.	Não realizado.

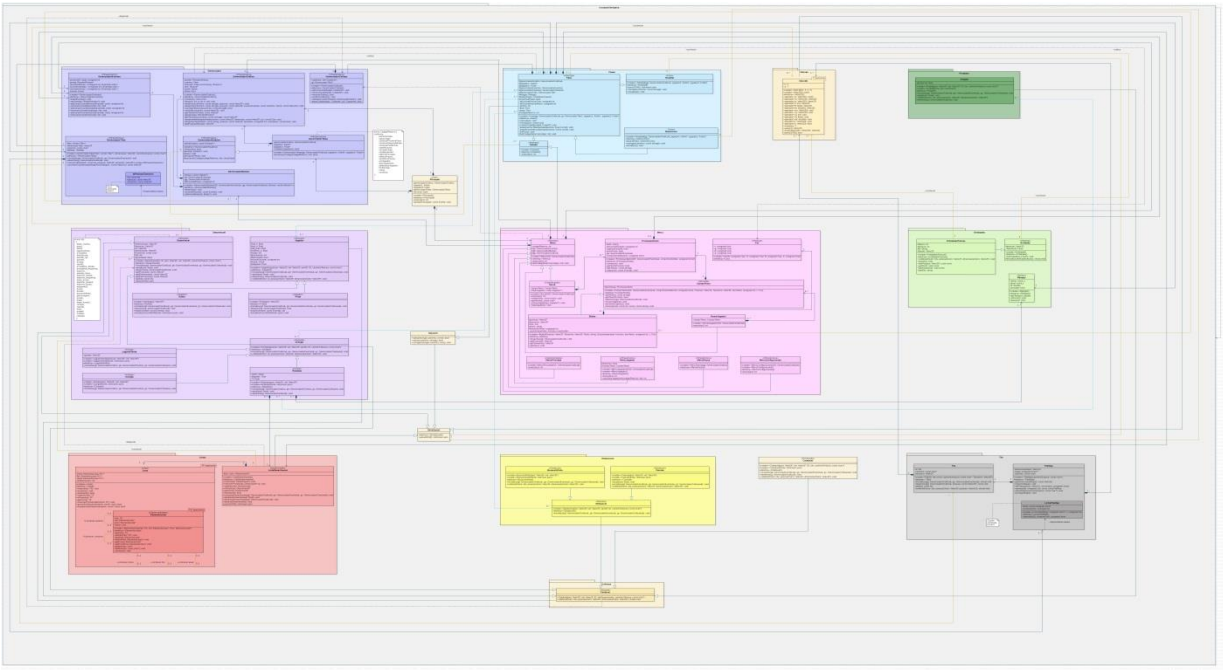

Tabela 2: Diagrama de Classes UML.[1]

O *software* projetado e desenvolvido segue os princípios da Engenharia de Software, bem como a maioria dos requisitos solicitados. Além disso, visando os princípios de coesão e desacoplamento, ele foi fracionado em categorias, em que cada uma possui responsabilidades determinadas e, sempre que possível, foram minimizadas as dependências entre elas. Portanto,

[1] Tabela 2: Devido às proporções estabelecidas no documento, não é possível condicionar o digrama de classes, a fim de dispor de uma nítida resolução.

dispomos dos seguintes conjuntos: Gerenciadores, Inimigos, Jogadores, Listas, Menus, Obstáculos, Projeteis e Tiles – os quais serão exemplificados e discorridos os seus funcionamentos a seguir.

Tendo em vista que o programa projetado e desenvolvido é orientado a objetos, o mesmo deve dispor de desacoplamento. Assim, o programa apodera-se de uma classe, a qual além de ser referenciada como "Principal", faz jus a sua intitulação. Na classe presente, os principais objetos que existem no sistema são instanciados, aliás, além do exposto, dispõe-se de uma main na qual é implementado somente um objeto da classe Principal, executando nosso programa desta forma.

O desenvolvimento de *software* requer controle, bem como o gerenciamento de suas classes. Tendo em vista esse conceito, foram constituídas as classes de gerenciadores, que, em sua maioria, possuem como suporte a Standard Template Library (STL), e dispõem das subcategorias: botões, colisões, estados, eventos, gráfico, telas e tile, a fim de ter coesão entre si. Com o intuito de elucidar os seus respectivos funcionamentos, assim como suas finalidades, partiremos do pressuposto da ordem cronológica de desenvolvimento – tendo em vista haver uma razão lógica para esta decisão.

A classe Gerenciador Gráfico possui como principal objetivo o controle de toda parte gráfica presente no jogo "Invasão Alienígena". Na presente classe é definido o tamanho da janela, bem como a câmera e sua centralização, e métodos que desenham na tela (retângulos, texto, entre outros).

Progredindo na explicação das classes de gerenciadores, a próxima a ser explicada é a classe GerenciadorEventos. Essa classe é de fundamental importância para o *software*, pois permite a interação do usuário com o jogo, tratando os eventos que ocorrem durante sua utilização, por exemplo: interações via teclado e *mouse*, as quais decorrem do tratamento de eventos presentes, bem como dos métodos apropriados de adição e remoção de ouvintes.

Além dessas classes, a classe GerenciadorColisão é muito importante para o bom funcionamento do *software*, pois serve como base para outra classe, denominada de Colidivel (que será explicada mais para a frente), cujo conceito é de notória relevância para os jogadores, inimigos, obstáculos bem como o mapa do jogo. Sendo assim, a classe apresentada, com o auxílio de outra classe denominada Vetor2D (que será explanada nos tópicos seguintes), verifica se os componentes analisados estão colidindo, através do método estãoColidindo, que utiliza como base seus valores absolutos de posições, bem como dimensões. Além disso, há o método verificarColisões, o qual identifica colisões com o TileMap.

Avançando nas classes de gerenciadores, prosseguiremos a explanação com a classe GerenciadorTile. Essa classe é necessária para o gerenciamento do mapa do jogo, através dos seguintes métodos: inicializar e checarColisões, que desenha o mapa na tela ao usuário, bem como gerencia as colisões com suas paredes. Aliás, apesar de dispormos do GerenciadorColisão e seus métodos, a função presente nesta classe é capaz de indicar o sentido da colisão, a fim de ser tratada, e, caso ocorra fora do mapa, será ignorada.

A classe GerenciadorTelas, por sua vez, é a responsável pela tela que será apresentada ao usuário, de acordo com sua escolha. O método processarCódigo é imprescindível, pois, de acordo com o retorno passado como parâmetro, é possível elencar qual tela deve ser apresentada e, consequentemente, suas demais funcionalidades.

Prosseguindo, para a execução correta, bem como o bom funcionamento do código, dispomos de uma classe que controla uma pilha de execuções que ocorrem durante a execução do *software*. A classe responsável por isso é o GerenciadorEstado, o qual, com base no componente *stack* da STL, ao processar o código solicitado, dispõe de recursos apropriados que empilham ou desempilham a pilha de execução.

Finalizando as classes de gerenciadores, terminaremos com a classe GerenciadorBotões, a qual é encarregada por gerenciar a interação com o usuário (por meio dos botões), identificando se a ação ocorreu no campo determinado. Caso isso ocorra, novos métodos e classes serão acionados, a fim de executar o solicitado.

Além disso, no desenvolvimento do trabalho, foi utilizado o conceito de *Templates*, sendo seu principal objetivo manipular tipos de dados diferentes, ou seja, uma programação genérica. Para abordar esse conceito, tomaremos como base a classe Lista, cujo conceito foi implementado, entretanto, é relevante ressaltar que essa lista dispõe de uma classe aninhada, denominada ElementoLista, a qual possui como principal objetivo a navegação. Sendo assim, é factível que sua incumbência no *software* seja a de dispor de uma lista na qual serão armazenados os elementos.

Contudo, a utilização de *Templates* no *software* não se limita apenas ao exposto acima, para isso, foi desenvolvida a classe Vetor2D, que segue uma lógica distinta da explicitada acima. Nessa classe, o principal objetivo consiste nas operações matemáticas. Logo, dispomos de operações de soma e subtração entre vetores, bem como para números inteiros, decimais e *doubles*. Além disso, houve a implementação de módulo, *versor* e projeção ortogonal, tendo em vista que esses conceitos envolvem a interdisciplinaridade, a qual será discutida no decorrer do projeto, e o conceito de Javascript Object Notation (JSON), o qual auxilia na transferência de dados.

Com o intuito de trazer funcionalidade ao jogo, bem como de cumprir os requisitos solicitados, houve o desenvolvimento de jogadores para o presente *software*. A classe Jogadores foi elaborada a fim de ser uma classe abstrata, a qual serve de base para as seguintes classes: Kahlo e Frida (sendo respectivamente o primeiro e o segundo jogador). Assim, a classe Jogador condiciona suporte para que os personagens sejam desenhados na tela, bem como auxilia na definição do modo como cada colisão deve ser tratada, respeitando a distância entre os centros das representações que estão colidindo, e define as ações ao colidir com os obstáculos. Da mesma forma, ocorre com a classe Inimigos, que também é abstrata, dispondo das seguintes classes derivadas: Ciclope, LagartoVerde e Robotão, sendo que este é responsável pelo lançamento de projéteis.

Além disso, é evidente que a interdisciplinaridade traz um enriquecimento ao desenvolvimento do *software*. Sendo assim, o *software* dispõe de conceitos físicos e matemáticos, ou seja, a sua junção é indispensável e essencial para a representação da realidade e retratação de objetos da vida real, bem como da proposta de programação orientada a objetos (POO).

Igualmente, quando pensamos em retratar a realidade, conceitos como gravidade, velocidade, entre outros, se tornam indispensáveis. Dessa forma, com base no exposto, é possível averiguar que a física se faz presente em conceitos necessários para a representação de movimento do jogo. Assim, como dispomos de movimentos relativos entre dois corpos – no jogo, entre os jogadores e entre os inimigos – a velocidade resultante entre eles é relativa, dependendo do referencial, bem como o sentido do movimento de ambos.

Da mesma forma, as aplicações da física no jogo não se limitam apenas a isso, sendo que a colisão é um conceito relevante e utilizado. Com base na 1ª Lei de Newton, sabemos que um objeto tende a permanecer em repouso se pelo menos uma força externa atuar sobre o mesmo. Assim, é factível relacioná-lo com o caixote presente, pois o mesmo só apresenta movimento com a atuação do jogador sobre ele, em um referencial inercial. Aliás, esse obstáculo dispõe do conceito de queda livre, quando é empurrado para fora da plataforma.

Ao seguir o ciclo da Engenharia de *Software*, uma classe deve ser projetada e pensada antes de executar sua codificação. Assim, tendo isso em mente, é fácil correlacionar o diagrama de classes com a teoria de conjuntos – presente na matemática.

Posto que os conceitos matemáticos se expandem para as representações gráficas igualmente, os mapas que constituem os cenários presentes em cada fase são construídos com base em uma matriz – cada fase dispõe de uma matriz distinta – na qual cada posição recebe um elemento gráfico predeterminado, a fim de compor o cenário. Além disso, com o intuito de localizar e ter ciência das posições dos personagens, inimigos e obstáculos, são utilizadas as coordenadas cartesianas. Além disso, dispomos da implementação de módulo, ou seu valor absoluto, assim como *versor* e projeção ortogonal – ambos relacionados a vetores – sendo sua implementação na classe Vetor2D.

No entanto, a implementação do cenário do jogo ocorre na classe TileMap, que dispõe de métodos apropriados, com a finalidade de imprimir o mapa e carregá-lo – denominadas respectivamente de imprimirMapa e CarregarMapa. Seguindo a mesma lógica, é possível correlacionar com um dos obstáculos presentes no jogo, "os espinhos", pois são implementados juntamente e da mesma forma. Além disso, dispomos de mais outros dois obstáculos (buraco infinito e caixote) que possuem a classe Obstáculos como base e é abstrata.

Por fim, a iteração que ocorre entre *software* e usuário é de grande relevância, bem como necessária para melhor execução. Tendo isso em vista, foi projetada a classe Menu, a qual é abstrata e serve como base para as demais classes: Menu Principal, Menu Configuração, Menu Pausa e Nome Jogador. Contudo, é importante ressaltar que os principais métodos presentes no Menu são: setCódigoRetorno e executar, que são, respectivamente, retorno à ação selecionada pelo usuário e execução.

TABELA DE CONCEITO UTILIZADOS E NÃO UTILIZADOS

O desenvolvimento do Jogo se deu à luz dos conhecimentos e conceitos aprendidos no decorrer da disciplina. Tais conceitos são apresentados na Tabela 3. Nesta tabela há também a situação do conceito no tocante a sua realização. Caso tenha sido realizado, ainda é indicado em que parte do sistema isto ocorreu. De antemão, faz-se contar que foram cumpridos 87,5% dos conceitos propostos.

Tabela 3. Lista de Conceitos Utilizados e Não Utilizados no Trabalho.

N.	Conceito	Uso	Onde/ O quê
1	**Elementos:**		

	-Classe, objetos & -Atributos (privados), variáveis e constantes. & -Métodos (com e sem retorno).	Sim	Em todos .h e .cpp
	-Métodos (com retorno *const* e parâmetro *const*) &. - Construtores (sem/com parâmetros) e destrutores.	Sim	Em todos .h e .cpp
	-Classe Principal.	Sim	Classe Principal.h/ Principal.cpp e main.cpp
	-Divisão em .h e .cpp	Sim	No desenvolvimento como um todo.
2	**Relações de:**		
	-Associação direcional. & -Associação bidirecional.	Sim	Associação direcional entre Lista e ElementoLista e bidirecional entre as classes GerenciadorTelas e Menu, por exemplo.
	-Agregação via associação. & -Agregação propriamente dita.	Sim	Agregação via associação na classe Fase e propriamente dita na classe ListaDesenhaveis, por exemplo.
	-Herança elementar. & -Herança em diversos níveis.	Sim	Elementar presenta entre as classes CampoTexto e Botao, por exemplo. Outrossim, diversos níveis temos entre a classe Jogador e Colidivel.
	-Herança múltipla.	Sim	Classe Fase.
3	**Ponteiros, generalizações e exceções**		
	-Operador *this.*	Sim	No desenvolvimento como um todo.
	-Alocação de memória *(new & delete).*	Sim	Em diversos momentos do desenvolvimento.
	-Gabaritos/*Templates* criadas/adaptados pelos autores (e.g. Listas Encadeadas via *Templates*).	Sim	Classes Vetor2D e Lista.
	-Uso de Tratamento de Exceções *(try catch).*		Classe GerenciadorGrafico.
4	**Sobrecarga de:**		
	-Construtoras e Métodos.	Sim	Sobrecarga de construtora na classe Coletaveis e de método na classe GerenciadorGráfico, por exemplo.
	-Operadores (2 tipos de operadores pelo menos).	Sim	Classe Vetor2D.
	Persistência de Objetos (via arquivo de texto ou binário)		
	-Persistência de Objetos.	Não	Não foi utilizado no desenvolvimento do software.
	-Persistência de Relacionamentos de Objetos.	Não	Não foi utilizado no desenvolvimento do software.
5	**Virtualidade:**		
	-Métodos Virtuais.	Sim	Classe Botão, por exemplo.
	-Polimorfismo.	Sim	Classe Fase, por exemplo.
	-Métodos Virtuais Puros/ Classes Abstratas.	Sim	Classe Colidivel possui função virtual pura e Classe Fase é abstrata, por exemplo.
	-Coesão e Desacoplamento.	Sim	No desenvolvimento como um todo.

6	**Organizadores e Estáticos**		
	-Espaços de Nomes *(Namespace) criada pelos autores.*	Sim	No desenvolvimento como um todo.
	-Classes aninhadas *(Nested)* criada pelos autores.	Sim	Classe Lista, sendo a classe aninhada a classe ElementoLista.
	-Atributos estáticos e métodos estáticos.	Sim	Atributo estático na classe GerenciadorEventos.
	-Uso extensivo de constantes *(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	Na classe CampoTexto, por exemplo, é utilizada a classe String. Além disso, a classe GerenciadorBotoes faz a utilização de Vector.
	-Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, MultiConjunto, Mapa **OU** Multi-Mapa.	Sim	GerenciadorEstados, por exemplo, faz a utilização de pilha.
	Programação concorrente		
	-*Threads* (Linha de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins.	Não	Não foi utilizado no desenvolvimento do software.
	-*Threads* (Linha de Execução) no âmbito de Orientação a Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.	Não	Não foi utilizado no desenvolvimento do software.
8	**Biblioteca Gráfica/ Visual**		
	-Funcionalidade Elementares. & -Funcionalidades Avançadas como: • Tratamento de colisões • Duplo *buffer*	Sim	Com o auxílio da biblioteca gráfica SFML.
	-Programação orientada e evento em algum ambiente gráfico. **OU** -*RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).	Não	Não foi utilizado no desenvolvimento do software.
	Interdisciplinares por meio da utilização de Conceitos de Matemática e/ou Física.		
	-Ensino Médio	Sim	Gravidade, velocidade, coordenadas cartesianas e a teoria de conjuntos.
	-Ensino Superior.	Sim	Operações matemáticas com vetores bem como conceitos de física.
9	**Engenharia de Software**		
	-Compreensão, melhoria e rastreabilidade de cumprimento de requisito.	Sim	Durante o processo de desenvolvimento, bem como com auxílio do professor e monitores.
	-Diagrama de Classes em *UML*.	Sim	No desenvolvimento como um todo.
	-Uso efetivo (quiçá) intensivo de padrões de projeto (particularmente GOF).	Não	Não foi utilizado no desenvolvimento do software.

	-Testes a luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	No desenvolvimento como um todo.
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	No desenvolvimento como um todo, utilizando um repositório no Github.
	-Reuniões com o professor para acompanhamento do andamento do projeto.	Sim	12/04/2021 - Durante o horário da aula. 19/04/2021 - Durante o horário da aula. 26/04/2021 - Durante o horário da aula. 03/05/2021 - Durante o horário da aula. 06/05/2021 - Horário distinto ao da aula.
	-Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Sim	19/03/2021 - Monitor: Lucas Skora 01/04/2021 - Monitor: Lucas Skora 06/04/2021 - Monitor: Lucas Skora 08/04/2021 - Monitor: Lucas Skora 15/04/2021 - Monitor: Lucas Skora 20/04/2021 - Monitor: Lucas Skora 22/04/2021 - Monitor: Augusto Correa 23/04/2021 - Monitor: Lucas Skora 23/04/2021 - Monitor: Augusto Correa 26/04/2021 - Monitor: Lucas Skora 27/04/2021 - Monitor: Lucas Skora 29/04/2021 - Monitor: Lucas Skora 03/05/2021 - Monitor: Lucas Skora
	-Revisão de trabalho escrito de outra equipe e vice-versa.	Sim	Dupla João Vitor Caversan dos Passos (ra: 2192969) e Guilherme Toshio Saito (ra: 2155133).

Ademais, faz-se necessário justificar o motivo e porque a utilização ou não utilização de cada conceito que foi contemplado na realização do trabalho. Tais conceitos bem como suas justificativas encontram-se na Tabela 4.

Tabela 4. Lista de Justificativa para Conceitos Utilizados.

No.	Conceitos	Situação
1	**Elementares**	Conceitos elementares de Programação Orientada a Objetos (POO) foram utilizados, bem como classes, atributos e métodos.
2	**Relações**	Foram utilizados conceitos de herança, agregação e associação – sendo direcionais e bidirecionais.
3	**Ponteiros, generalizações e exceções**	Durante a execução foi utilizado o ponteiro this, o qual representa a instância da classe utilizada no momento. Além disto, generalizações foram utilizadas – em Templates, como mencionado no presente relatório – pois agrega simplicidade e reutilização ao código. Além disso, visando o tratamento de exceções foram utilizados try/catch, ou seja, os erros gerados em tempo de

		execução são controlados bem como seu tratamento eficiente.
4	**Sobrecargas e Persistências**	Sobrecargas de métodos, funções e operadores foram utilizados visando o polimorfismo. Sendo assim, dispomos de uma maior reutilidade do código, pois, por exemplo para os operadores aritméticos é possível alterar seu significado, ou seja, partimos de uma expressão complexa a algo simples e intuitivo.
5	**Virtualidade**	Funções virtuais são utilizadas para auxiliar na reutilizada do código, pois uma mesma função pode ter declarações distintas em suas classes derivadas, sendo que o mesmo é uma parte importante do POO, polimorfismo das classes e funções.
6	**Organizadores e Estáticos**	A utilização de namespace foi embasada com o intuito de garantir uma organização do código, bem como evitar possíveis conflitos de nomes – métodos. Outrossim, a utilização de membros estáticos visa que a variável criada fique disponível a todos os objetos, ou seja, implica em economia de memória bem como em tempo de processamento.
7	**STL, String e Programação Concorrência**	A STL é uma biblioteca que aumenta sensivelmente o poder de programação em C++, a qual proporciona soluções que envolvem o processamento de estrutura de dados, ou seja, auxiliou na organização estrutural. Além disto, apesar de ser evidente a importância da programação concorrente – devido aumento de desempenho e melhor modelagem – o mesmo não foi possível implementar.
8	**Biblioteca Gráfica e Interdisciplinaridades**	A biblioteca gráfica utilizada para o desenvolvimento do jogo presente é a SFML, sendo que a mesma foi utilizada durante todo o desenvolvimento auxiliando assim, nas implementações gráficas presente e afins. Concomitantemente, a interdisciplinaridade é de extrema de importância, pois, sem a mesma seria praticamente impossível dispor de uma retratação próxima à realidade. Assim, os conceitos utilizados relacionam a matemática e a física juntamente a programação.
9	**Engenharia de Software**	O ciclo da Engenharia de Software foi utilizado durante todo o desenvolvimento do presente projeto, sendo os mesmos: levantamento de requisitos, análises, implementações de diagramas bem como de testes. Ademais, é notória a importância da utilização dos padrões de projetos – a fim de melhorar e solucionar problemas, bem como condicionar um programa mais coeso e desacoplado – entretanto, não foi possível realizar a implementação do mesmo.
10	**Execução de Projeto**	Durante a execução e elaboração do projeto foi utilizado um controle de versão, o qual visa garantir o histórico bem como todas as versões do projeto. Outrossim, reuniões com intuito de aprimoramento bem como acompanhamento do desenvolvimento foram realizadas

		com os monitores e o professor responsável.

REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS

O desenvolvimento de programas orientados a objetos consiste em um modelo onde se definem objetos – sistemas semelhantes a realidade. Com base nisto, é possível aproximar este modelo do mundo real ao mundo virtual através da composição e interação entre os objetos apresentados, utilizando conceitos de classes, métodos e atributos. Ou seja, possui uma visão centrada em objetos a serem manipulados. Enquanto que, a linguagem procedimental é um modulo de implementação imperativo. Sendo que, possui como base dois conceitos importantes: dados e algoritmos – com ênfase em algoritmos.

Ademais, a linguagem orientada a objetos, é uma linguagem de propósitos gerais com uma tendência para a programação de sistemas, a qual suporta programação de baixo-nível nos moldes tradicionais, além de abstração de dados e orientação a objetos. Além disso, tomando como exemplo as linguagens C e C++, é possível averiguar que o C++ apresenta vantagens em relação à linguagem C, tais como verificação de tipos e modularização. Aliás, possui melhorias que incluem constantes simbólicas, função inline, argumentos-default para funções, sobrecarga de funções e métodos, operadores de gerenciamento de armazenamento livre, polimorfismo, herança, encapsulamento, referência e programação genérica.

Portanto, com base no que foi exposto é possível averiguar que cada linguagem tem suas especificações e apesar de serem comparadas, cada problema pode se adequar melhor a uma determinada linguagem. Desta forma, é possível afirmar que a orientação a objetos tem melhorias e é mais abrangente, entretanto, é possível notar que ambas possuem semelhanças bem como suas peculiaridades e benefícios uma relacionada a outra.

DISCUSSÃO E CONCLUSÕES

O aprendizado condiciona a novas perspectivas bem como agrega uma nova visão sobre o assunto evidenciado. Da mesma forma, é possível correlacionar o exposto ao desenvolvimento do presente software, pois é factível a agregação tanto profissional como intelectual que o mesmo condiciona. A programação orientada a objetos, proporciona uma nova visão – diferente da proposta apresentada pela procedural, até então trabalhada no presente curso de graduação – evidenciado a importância do processo de planejamento bem como a projeção de softwares.

Todavia, é imprescindível ressaltar que o ato contínuo de aprendizado – seja, com funcionalidades da biblioteca gráfica, aprofundando conceitos, através de pesquisas e com orientações dos monitores e professor – proporcionaram um aprendizado inestimável. Além de que, o mesmo ampliou a concepção da interrelação que há entre diversos campos de conhecimentos, ou seja, viabilizou a compreensão tornando possível o conhecimento como um todo e não como partes fragmentadas.

Portanto, a projeção do mesmo foi constituída pensando-se em quais conceitos seriam englobados. Isto, tornou apto o aprimoramento dos conceitos aprendidos no decorrer da

presente disciplina bem como toda experiência e desenvoltura absorvida durante a execução do trabalho condicionaram em um conhecimento incalculável aos discentes.

DIVISÃO DO TRABALHO

O desenvolvimento do software se deu de forma conjunta. Outrossim, é importante salientar que as atividades realizadas e seus responsáveis são apresentados na Tabela 5. De antemão, faz-se contar que foram cumpridos em uma porcentagem total do trabalho, envolvendo requisitos e conceitos, 87,1% do esperado.

Tabela 5. Lista de Atividades e Responsáveis.

Atividades	**Responsáveis**
Levantamento de Requisitos	Alfons e Anna Caroline
Diagrama de Classes	Anna Caroline
Programação em C++	Alfons e Anna Caroline
Implementação de *Template* e do Mapa	Anna Caroline
Inimigos, Obstáculos e Personagens	Anna Caroline
Gerenciadores e Menu	Anna Caroline
Coletáveis	Alfons
Sistema de pontuação	Mais Alfons que Anna Caroline
Implementação de projeteis	Mais Alfons que Anna Caroline
Tratamento das colisões	Mais Anna Caroline que Alfons
Jogadores múltiplos	Anna Caroline
Gravidade	Alfons
Controle de versão do código	Alfons e Anna Caroline
Listas e Vetor	Anna Caroline
Desenvolvimento da apresentação e escrita do trabalho	Anna Caroline
Revisão do Trabalho	Alfons e Anna Caroline

AGRADECIMENTOS

Agradecimentos aos monitores Lucas Eduardo Bonacio Skora e Augusto Mudrei Correa, pelo auxílio durante o desenvolvimento do presente projeto – bem como solucionado nossas dúvidas – e ao Professor J. M. Simão pelos apontamentos que auxiliaram no aprimoramento durante o desenvolvimento do software. Ademais, aos colegas discentes João Vitor Caversan dos Passos e ao Guilherme Toshio Saito.

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A] SAADE, Joel. Programando em C++. Editora Novatec. 2003. ISBN 85-7522-042-X.

[B] DEITEL, H.M., DEITEL P.J. C++: Como programar. Editora Pearson. 2015 (8º impressão). ISBN 978-85-4301-373-2.

[C] SIMÃO, J. M. Site da Disciplina de Técnicas de Programação, Curitiba – PR, Brasil, Acessado em 07/04/2021, às 15:00h. https://pessoal.dainf.ct.utfpr.edu.br/jeansimao/Fundamentos2/Fundamentos2.htm

[D] Imagens – LagartoVerde, Ciclope e Caixote, utilizadas da biblioteca de distribuição gráfica paint 3D. Acessado em 01/04/2021, às 10:30h.