**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++. Ademais, o jogo desenvolvido dispõe de um ou dois jogadores – onde ambos podem jogar concomitantemente – o qual ou os quais terão que enfrentar algumas categorias de inimigos, diferentes a depender da fase, sendo que o mesmo apresenta duas fases – as quais oferecem dificuldades variadas. Não obstante, para o desenvolvimento do jogo presente foram considerados os requisitos inicialmente propostos pelo professor além de dispormos de 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*). Outrossim, em linguagem de programação C++, realizou-se o desenvolvimento das funções e implementações gráficas - com o auxílio da biblioteca SFML - 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*). Portanto, após a implementação dos requisitos e testes com os mesmos é possível salientar que houve uma funcionalidade conforme o esperado. Desta forma, é notória a contribuição do aprendizado bem como para nossa formação os conhecimentos adquiridos e aprimorados com o desenvolvimento do mesmo.

**Palavras-chave ou Expressões-chave:** Linguagem de programação Orientada a objetos (C++), Jogo de plataforma, Desenvolvimento de Jogos, 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. That being said, the learning contribution is notorious, as well as the knowledge acquired and improved with the development of this project.

**Key-words or Key-expressions:** Object-Oriented Programming language (C++), Platform game, Game development, 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, o mesmo é de fundamental importância para aplicação dos conceitos aprendidos no decorrer da disciplina, além de ampliar os conhecimentos e técnicas durante o desenvolvimento do mesmo.

Outrossim, visando a constituição do mesmo foi desenvolvido um jogo de plataforma com tema e mecânicas selecionadas pelos discentes e aprovadas pelo docente. Concomitantemente, cabe ressaltar que para a elaboração do mesmo foi buscado seguir os seguintes requisitos: programação orientada a objetos (POO), sendo que a mesmo foi o implementado na linguagem de programação C++, além de dispor do ciclo da Engenharia de Software e por fim utilizar da biblioteca gráfica SFML, de modo a auxiliar no desenvolvimento e implementações visuais. Não obstante, 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.

Ademais, no decorrer de sua implementação foram utilizados softwares, onde foi possível testar e aprimorar o funcionamento do mesmo. Não obstante, 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, onde o mesmo pode ser jogado por um ou dois jogadores. Ademais, quando o programa é executado é exibido o Menu Principal, onde no mesmo há acesso ao início do processo de jogar, através do botão jogar – de modo a determinar o número de jogadores bem como a fase a ser jogada – classificações, carregar jogo anterior e a opção de fechar o programa, 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 a seguinte tela –figura 03 – será apresentada ao usuário, com as opções: continuar o jogo, salvar o jogo ou retornar ao menu principal (menu inicial).

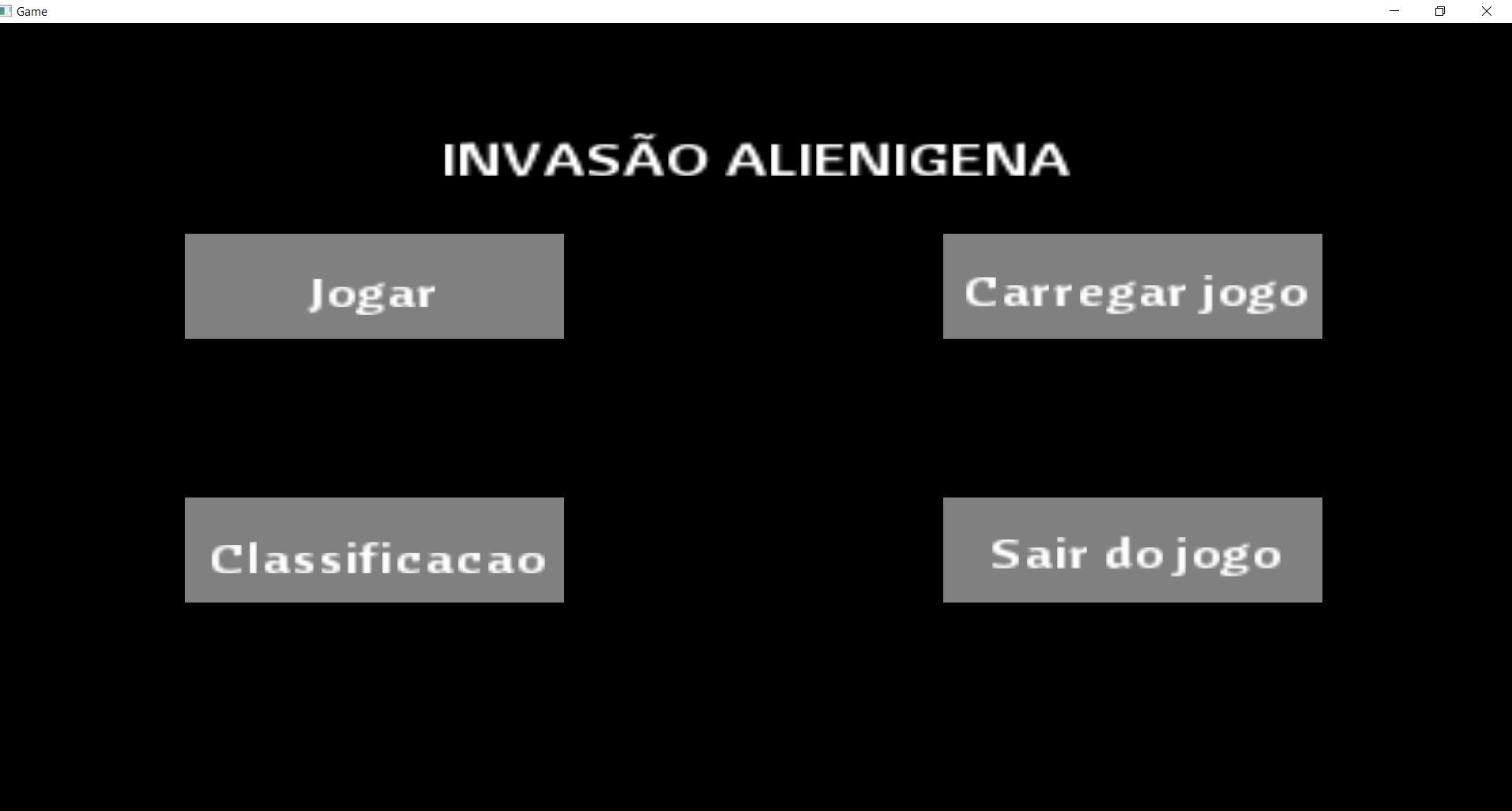
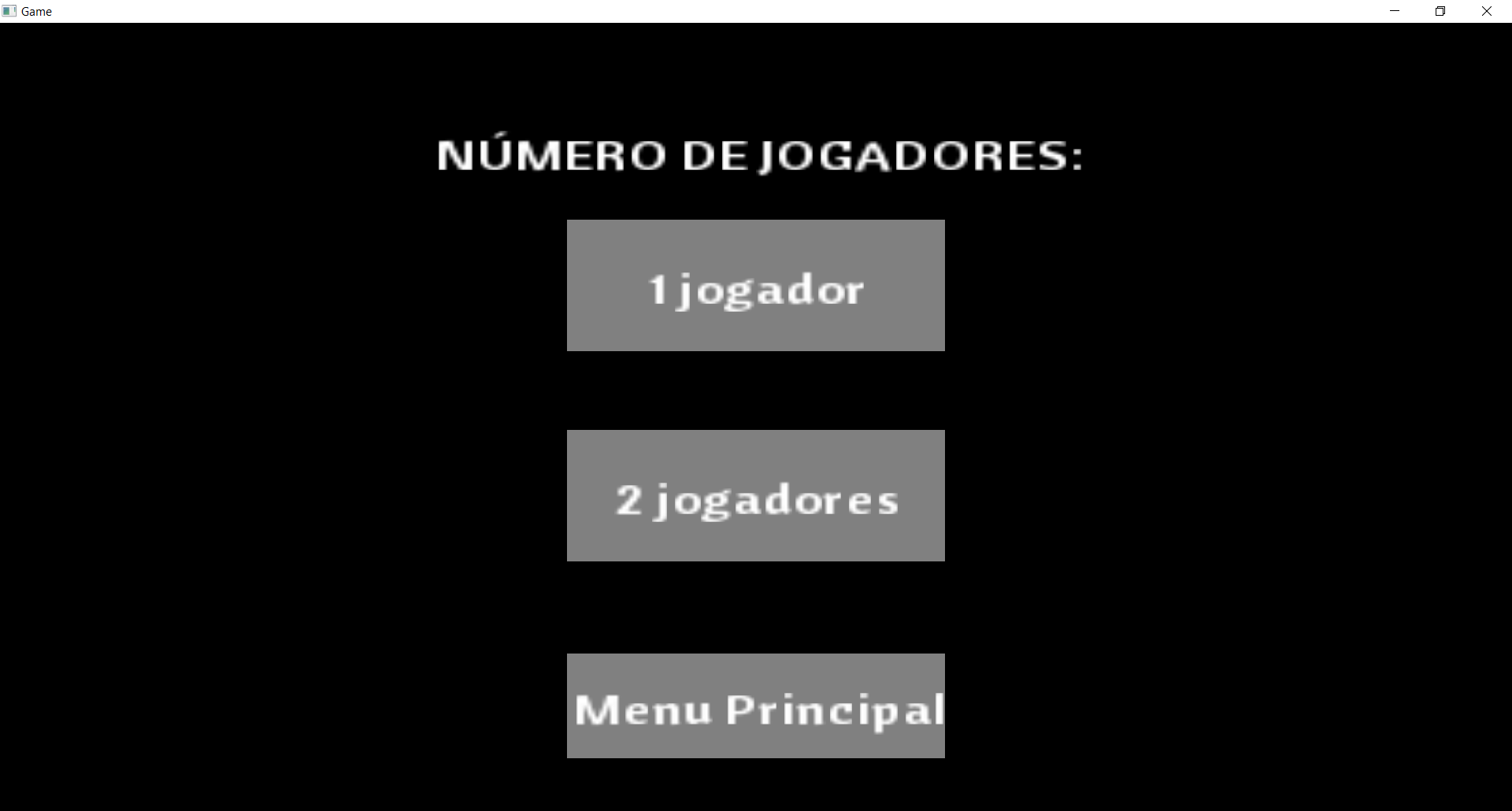
  

Figura 01: Menu Principal. Figura 02: Menu Jogador Figura 03: Menu Pausa.

Não obstante, a primeira fase é constituída por plataformas, onde o jogador ou jogadores, encontram determinados obstáculos a serem enfrentados de modo a chegar ao final da fase e avançar para a próxima. Sendo que, 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.

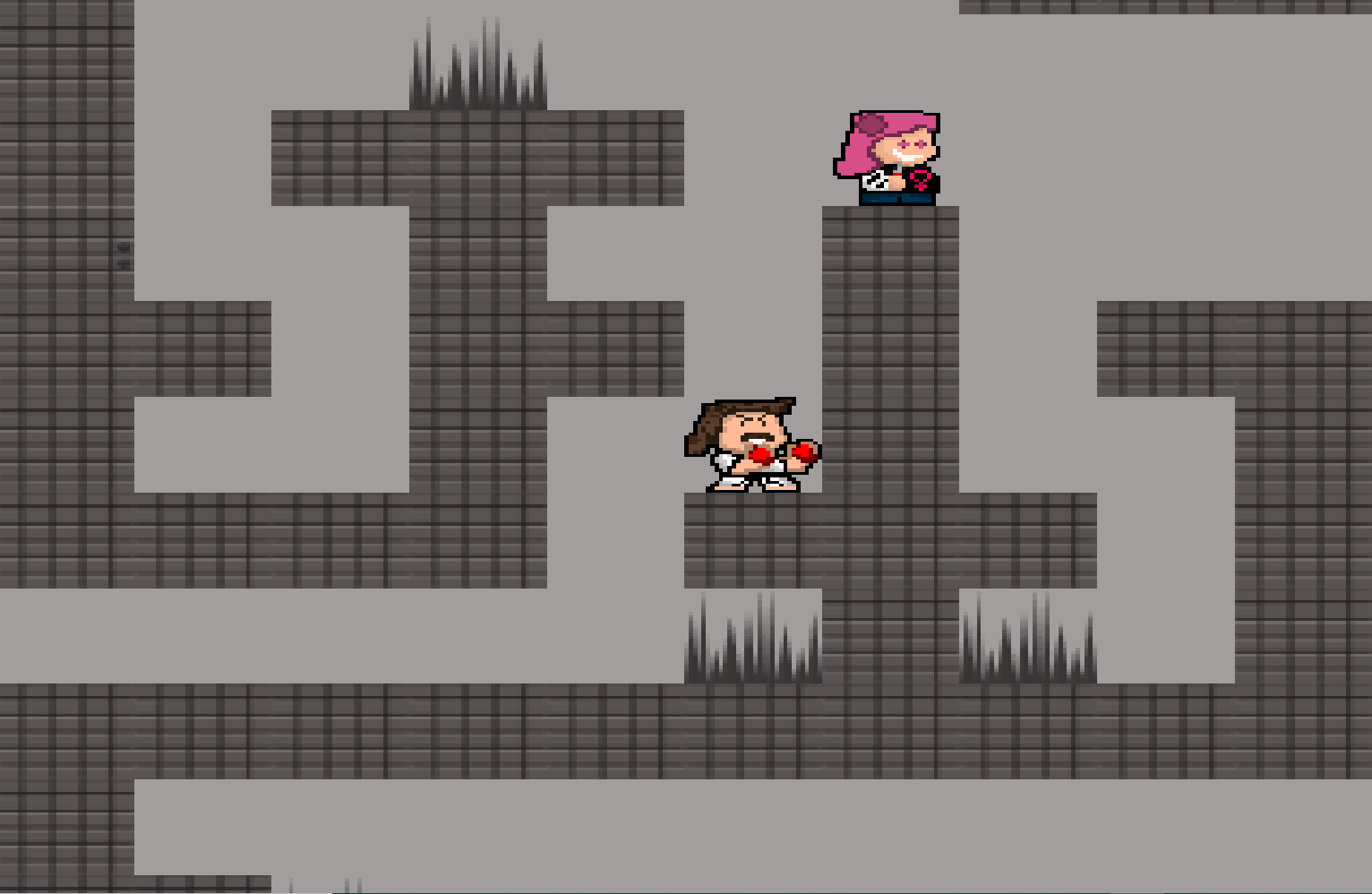
 

Figura 04: Manicômio (Primeira Fase). Figura 05: Manicômio (Primeira Fase).

Ademais, a segunda fase do jogo dispõe de semelhanças com a primeira – é constituída por plataformas, podendo ser jogada por um ou dois jogadores, além de dispor de obstáculos e inimigos semelhantes, como os espinho e o caixote para os obstáculos e LagartoVerde e Robotão – contudo, a mesma dispõe de características próprias. Dentre estas características, se destacam um novo obstáculo bem como um novo inimigo - denominado de buraco infinito, onde o jogador caso entre em contato com o mesmo é levado ao início da fase – recomeçando o jogo – e o novo inimigo é o Ciclope (chefão), encontrado no final da segunda fase sendo que o mesmo representa o último desafio da segunda fase, onde quando superado o jogador termina a fase corrente.

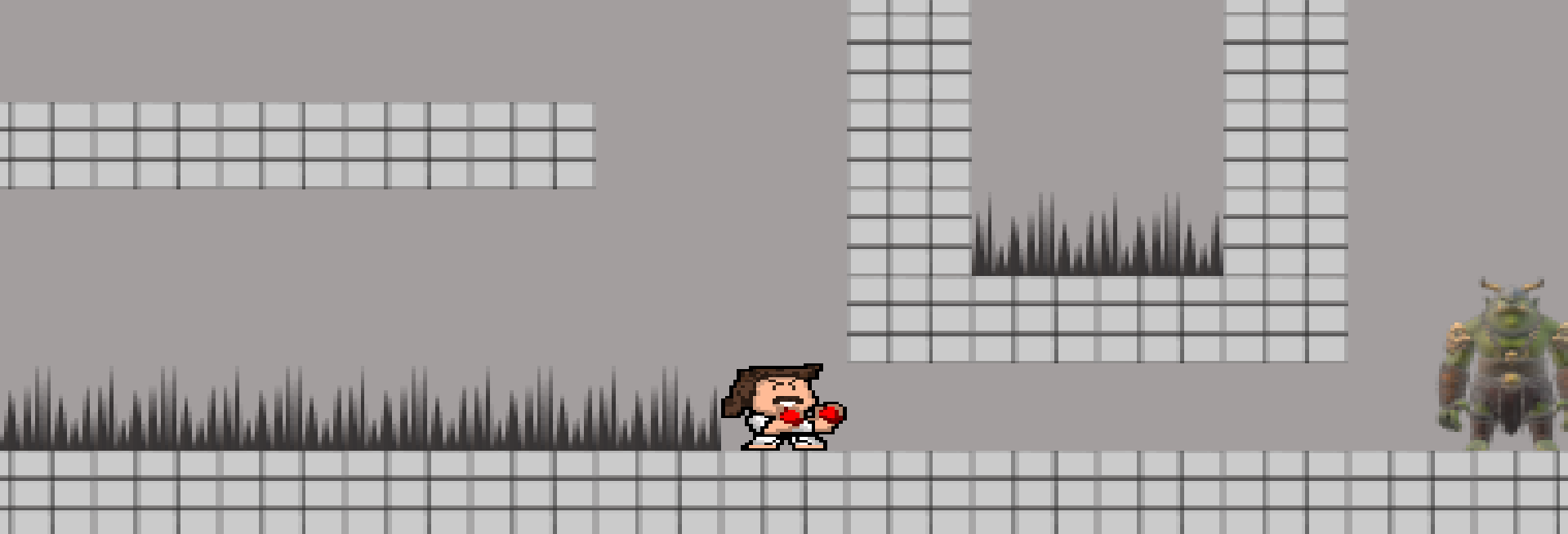
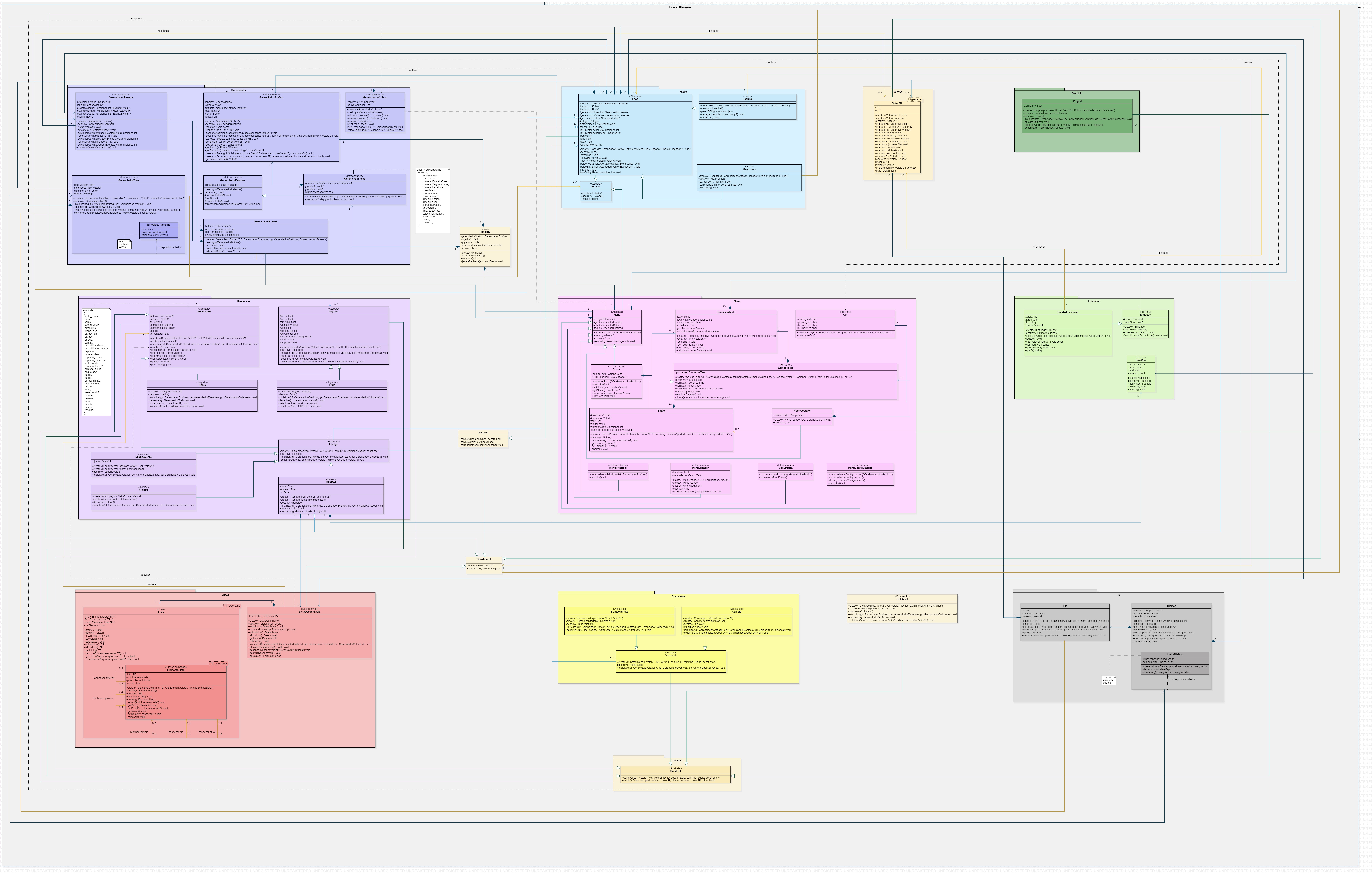
 

Figura 06: Hospital (Segunda Fase). Figura 06: Hospital (Segunda Fase).

**DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **N.** | **Requisitos Funcionais** | **Situação** | **Implementação** |
| 1 | Apresentar menu de opções aos usuários do Jogo. | Requisito previsto inicialmente e realizado. | Requisito cumprido via pacote Menu e seus respectivos 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 MenuConfiguracoes, onde 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 Desenhavel sendo que 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 instâncias por tipo. | Requisito previsto inicialmente e realizado. | Requisito realizado, sendo que o mesmo se encontra na presentes nas seguintes classes: Manicomio 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 – devido dispor de espinho presente no mapa. |
| 9 | Ter representação gráfica de cada instância. | Requisito previsto inicialmente e realizado. | Requisito cumprido via GerenciadorGrafico. |
| 10 | Ter em cada fase um cenário de jogo com os obstáculos. | Requisito previsto inicialmente e realizado. | Requisito cumprido via pacote de Tile e 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 GerenciadorColisao, bem como o pacote Desenhaveis. |
| 12 | Gerenciar colisões entre jogadores e obstáculos. | Requisito previsto inicialmente e realizado. | Requisito cumprido inclusive via classe Jogador, bem como GerenciadorColisao. |
| 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 através 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. |

 Figura 07: Diagrama de Classes UML.[[1]](#footnote-18990)

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, onde cada possui responsabilidades determinadas e sempre que possível foi minimizado as dependências entre elas. Portanto, 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 dispomos de uma main onde 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. Visando esse conceito, foi constituída as classes de gerenciadores – sendo que em sua maioria possuem como suporte a Standard Template Library (STL), onde dispõe as 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 definida o tamanho da janela, bem como a câmera e sua centralização, assim como métodos que desenham na tela – retângulos, texto, entre outros – ou seja, ela controla o que será apresentado na tela ao usuário.

Progredindo na explicação das classes gerenciadores, a próxima a ser explicada e elucidada é a classe GerenciadorEventos. A presente 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, como por exemplo: interações via teclado e mouse – as quais decorrem devido ao tratamento de eventos presentes, bem como aos métodos apropriados de adição e remoção de ouvintes.

Ademais, a classe GerenciadorColisão é de fundamental importância para o bom funcionamento do software pois a mesma serve como base para outra classe denominada de Colidivel – a qual será explicada mais a frente – sendo notória a relevância deste conceito para os jogadores, inimigos, obstáculos bem como o mapa do jogo. Sucedendo ao exposto, a classe apresentada – com auxílio de outra classe denominada Vetor2D, a qual será explanada nos tópicos decorrentes – verifica se os componentes analisados estão colidindo, através do método estaoColidindo que utiliza como base seus valores absolutos de posições bem como dimensões. Além do método verificarColisoes, o qual identifica colisões com o TileMap.

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

Não obstante, a classe GerenciadorTelas é a responsável por qual tela será apresentada ao usuário, de acordo com sua escolha. O método processarCodigo 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 execução correta, bem como o bom funcionamento do código, dispomos de uma classe que controla uma pilha de execuções, que ocorrem durante da execução do software. A classe responsável por isto é o GerenciadorEstado, onde com base no componente stack da STL, ao processar o código solicitado dispõe de recursos apropriados que empilham ou desempilham da pilha de execução.

Conduzindo-se ao fim das classes gerenciadores, terminaremos com a classe GerenciadorBotões. Onde a classe é encarregada por gerenciar a interação com usuário – através dos botões – identificando se a ação foi no campo determinado, e caso isso ocorra, novos métodos e classes serão acionados a fim de executar o solicitado.

Além disto, no desenvolvimento do trabalho foi utilizado o conceito de Templates o qual permite criar um conjunto de classes ou funções que estão relacionadas entre si, sendo seu principal objetivo manipular tipos de dados diferentes, ou seja, uma programação genérica. Para elucidar este conceito, tomaremos como base a classe Lista – onde o conceito foi implementado – entretanto é relevante ressaltar que ela dispõe de uma classe aninhada denominada ElementoLista, a qual possui como principal objetivo a navegação. Sendo assim, é factível que sua incumbência no software é de dispor de uma lista onde serão armazenados os elementos.

Contudo, a utilização de Templates no software não se limita apenas ao exposto acima. Com a finalidade de dispor do mesmo conceito elucidado sobre a utilização de Templates, foi desenvolvido a classe Vetor2D, entretanto, ela segue uma lógica distinta da explicitada acima. Na classe presente, o principal objetivo são as operações matemáticas. Logo, dispomos de operações de soma e subtração entre vetores bem como para inteiro, números decimais e doubles. Além disso, houve a implementação de módulo, versor e projeção ortogonal – tendo em vista que estes conceitos envolvem a interdisciplinaridade, a qual será discutida no decorrer da elucidação do projeto – e conceitos de Javascript Object Notation (JSON), o qual auxilia na transferência de dados.

Não obstante, com o intuito de trazer funcionalidade ao jogo – bem como o cumprimento dos 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 a definir como cada colisão deve ser tratada – respeitando a distância entre os centros das representações que estão colidindo, bem como define ações ao colidir com obstáculos. Da mesma forma, ocorre para 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.

Ademais, é notório que a junção de conceitos e a interdisciplinaridade traz um enriquecimento ao desenvolvimento do software, bem como para a formação dos discentes. Logo, tendo isto em vista o software dispõe de conceitos físicos e matemáticos intrínsecos, ou seja, este fato demonstra que apesar das disciplinas, a princípio, disporem de suas particularidades a sua junção é indispensável e essencial para representação da realidade e retratação de objetos da vida real, bem como é a proposta da programação orientada a objetos (POO).

Outrossim, quando pensamos em retratar a realidade, conceitos como gravidade, velocidade e entre outros, se tornam indispensáveis. Logo, com base no exposto é possível averiguar que a física se faz presente em conceitos necessários para a representação de movimento do jogo, contudo, quando analisamos mais a fundo é possível encontrar conceitos intrínsecos ao mesmo. Assim, quando dispomos de movimentos relativos entre dois corpos – no jogo, entre os jogadores bem como aos inimigos – a velocidade resultante entre eles é relativa a depender do referencial – bem como do sentido de movimento de ambos.

Igualmente ao exposto anteriormente, as aplicações da física no jogo não se limitam apenas a isto, sendo que a colisão é um conceito muito relevante bem como utilizado. Não obstante, analisando o software em outra vertente, podemos verificar que um dos seus obstáculos possuem relação com a física, com base na 1º Lei de Newton, sabemos que um objeto tende a permanecer em repouso ao menos que uma força externa atue sobre o mesmo, com base nisso podemos relacioná-lo com o caixote presente, pois o mesmo só apresenta movimento com a atuação do jogador sob ele, em um referencial inercial. Aliás, este obstáculo dispõe do conceito de queda livre, quando é empurrado para fora da plataforma.

Entretanto, é importante ressaltar que a interdisciplinaridade começa antes mesmo das primeiras linhas de código. Ao seguir o ciclo da Engenharia de Software, uma classe deve ser projetada bem como pensada antes de executar sua codificação. Assim, tendo isto em mente é fácil correlacionar o diagrama de classes com o teorema de conjuntos – presente na matemática – pelo simples fato de agregações e pertencimentos presentes.

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 – onde, cada posição recebe um elemento gráfico pré-determinado a fim de compor o cenário. Além disso, com o intuito de localizar e ter ciência das posições dos personagens, inimigos bem como os obstáculos são utilizadas as coordenadas cartesianas. Ademais, 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.

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

Por fim, a iteração que ocorre entre software e usuário é de grande relevância bem como necessária para melhor execução. Como representado acima, para o funcionamento do projeto desenvolvido é necessário dispor de interação – seja selecionando a quantidade de jogadores, qual fase é a desejada, entre outros. Tendo isto em vista, foi projetado 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 em Menu são: setCodigoRetorno e executar, onde respectivamente é retornado à ação selecionado pelo usuário bem como sua execução.

**TABELA DE CONCEITO UTILIZADOS E NÃO UTILIZADOS**

Tabela 2. 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). | Sim | Classe GerenciadorEventos, bem como com o auxílio da biblioteca gráfica SFML. |
| **Interdisciplinares por meio da utilização de Conceitos de Matemática e/ou Física.** | | |
| -Ensino Médio | Sim | Gravidade, velocidade, coordenadas cartesianas e teorema 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). |

Tabela 3. 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, possibilitando assim apontar para o objeto utilizado a fim de chamar um método. Não obstante, 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, ou seja, seguir a proposta da programação orientada a objetos. 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 professor responsável. |

**REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS**

O desenvolvimento de um software começa antes mesmo de projetar suas primeiras linhas de código, pois para saber qual é o melhor caminho a seguir, ou seja, qual estilo de programação se adequa ao problema a ser solucionado é necessário analisar os conceitos a serem implementados bem como a finalidade e funcionamento do programa. Pois, para cada situação pode-se aplicar uma categoria de programação diferente, por exemplo, a programação procedimental ou programação orientada a objetos, tomando como exemplos respectivamente as linguagens C e C++. Com base nisso, é possível elencar e fazer comparações sobre as mesmas.

O desenvolvimento de programas orientados a objetos consiste em um modelo onde se definem objetos – sistemas semelhantes a realidade, ou seja, 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. Enquanto que, a linguagem procedimental é um modulo de implementação imperativo, sendo assim são programas baseados no conceito de chamadas de procedimentos (um conjunto de passos declarados computacionalmente que devem ser executados). Ou seja, a linguagem C possui como base dois conceitos importantes: dados e algoritmos – sendo sua ênfase em algoritmos – enquanto que ao contrário da linguagem C a C++ possui como enfoque a resolução de problemas – bem como a linguagem procedural – mas, com uma visão centrada nos objetos a serem manipulados.

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. Não obstante, C++ apresenta vantagens em relação à linguagem C, tais como verificação de tipos e modularização, contento ainda 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, tomando como base que o C++ se baseia na linguagem C – devido ter herdado características da mesma – é possível afirmar que a mesma tem melhorias e é mais abrangente, sendo 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, como é proposta da engenharia – a idealização do objetivo final, bem como seus requisitos – a projeção do mesmo foi constituída pensando-se em quais conceitos seriam englobados, e 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**

Tabela 4. 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* | Anna Caroline |
| Implementação do Mapa | Anna Caroline |
| Inimigos, Obstáculos e Personagens | Anna Caroline |
| Gerenciadores | Anna Caroline |
| Coletáveis | Alfons |
| Sistema de pontuação | Mais Alfons que Anna Caroline[[2]](#footnote-18484) |
| Implementação de projeteis | Mais Alfons que Anna Caroline[[3]](#footnote-24610) |
| Tratamento das colisões | Mais Anna Caroline que Alfons |
| Tratamento de eventos | Anna Caroline |
| Jogadores múltiplos | Anna Caroline |
| Menu | Anna Caroline |
| Gravidade | Alfons |
| Controle de versão do código | Alfons e Anna Caroline |
| Listas | Anna Caroline |
| Vetor | Anna Caroline |
| Desenvolvimento da apresentação | Anna Caroline |
| 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 auxilio 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 CITADAS NO TEXTO**

[1] SIMÃO, J. M. Site da Disciplina de Técnicas de Programação, Curitiba – PR, Brasil, Acessado em 07/04/2021, às 15:00h.

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

**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.

1. Figura 07: Devido as proporções estabelecidas no documento, não é possível condicionar o digrama de classes a fim de dispor de uma nítida resolução. [↑](#footnote-ref-18990)
2. Idealização do sistema de pontuação realizada pelo Alfons, bem como sua implementação para os coletáveis sendo que a mesma foi expandida pela Anna a fim de colocar o sistema oposto aos espinhos e buraco negro. [↑](#footnote-ref-18484)
3. Criação da classe base para os projéteis realizada pela Anna, entretanto, sua funcionalidade e ajustes foram implementados pelo Alfons. [↑](#footnote-ref-24610)