TRABALHO PARA A DISCIPLINA DE   
TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE   
SISTEMAS DE INFORMAÇÃO DA UTFPR:   
*AQUI NOME DO JOGO ~~– MODELO & ESPECIFICAÇÃO DO TRABALHO~~*

Gabriel Felipe Miguel Machado, Yago Augusto Constantino Ribeiro

gabmac@alunos.utfpr.edu.br, yagoribeiro@alunos.utfpr.edu.br

Disciplina: **Técnicas de Programação – ICSE20** / 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 exige o desenvolvimento de um *software* de plataforma, no formato de um jogo, para fins de aprendizado de técnicas de engenharia de *software*, particularmente de programação orientada a objetos em C++. Para tal, neste trabalho, escolheu-se o jogo **Brasileirinho++**, no qual o jogador enfrenta inimigos em dois cenários distintos. O jogo tem duas fases que se diferenciam por dificuldades para o jogador. Para o desenvolvimento do jogo foram considerados os requisitos textualmente propostos e elaborado modelagem (análise e projeto) via Diagrama de Classes em Linguagem de Modelagem Unificada (*Unified Modeling Language* - *UML*) usando como base um diagrama assaz genérico e prévio proposto. Subsequentemente, em linguagem de programação C++, realizou-se o desenvolvimento que contemplou os conceitos usuais de Orientação a Objetos como Classe, Objeto e Relacionamento, bem como alguns conceitos ditos avançados, como por exemplo, Classe Abstrata, Polimorfismo, Gabaritos, Persistências de Objetos por Arquivos, Sobrecarga de Operadores e Biblioteca Padrão de Gabaritos (*Standard Template Library* - *STL*). Depois da implementação, os testes e o uso do jogo feitos pelos próprios desenvolvedores demonstraram sua funcionalidade conforme os requisitos e o modelagem elaborada. Por fim, salienta-se que o desenvolvimento em questão permitiu cumprir o objetivo de aprendizado visado.

**Palavras-chave ou Expressões-chave** (máximo quatro itens, não excedendo três linhas)**:** Artigo-Relatório ~~Modelo~~ para o Trabalho em Técnicas de Programação, Trabalho Acadêmico Voltado a Implementação em C++, ~~Normas Internas para Elaboração de Trabalho~~, ~~Exemplo de Conteúdos de Trabalho de Técnicas de Programação~~.

# INTRODUÇÃO

Os trabalhos não poderão ser entregues ao Professor de maneira impressa. Assim sendo, apenas a versão digital será aceita. Na verdade, é necessário enviar o trabalho escrito em formato digital em “.doc(x)” e “.pdf” para o e-mail do professor (*jeansimao* ‘arroba’ *utfpr.edu.br*). Também é necessário enviar as implementações respectivas, diagrama(s) de projeto e demais materiais de suporte, como apresentação “.ppt” (e respectivos pdf) usados para apresentar o trabalho em classe. Para tal, deve-se utilizar o link do github no qual deve estar o projeto. Por fim, o(s) diagrama(s) de classes deve(m) ser entregue(s) de maneira impressa, além de maneira digital.

Uma vez explicado o necessário à introdução em si, este presente documento-modelo de artigo-relatório apresenta demais seções necessárias que o trabalho deve conter, bem como seus conteúdos. Estas indicações de conteúdos mostram o que se faz necessário contemplar em cada seção, além de explicar alguns itens de formatação de elementos contemplados.

Este trabalho se deu como uma forma de avaliação da disciplina de Técnicas de Programação da UTFPR, no qual teve como objetivo a realização de um jogo de plataforma na linguagem de programação C++. A fim de aprender, principalmente, o paradigma de Orientação a Objetos. Portanto, foi necessário o uso de diversos requisitos e conceitos para o projeto, os quais serão expostos e explicados neste documento nas referentes seções.

Dessa forma, o jogo, como acordado com o professor Dr. Jean M. Simão, deve ser em estilo 2D em plataforma. Assim, foi usado a IDE Visual Studio, além da biblioteca externa gráfica SFML. Ademais, foi permitido a possibilidade de expandir o projeto além dos critérios pedidos, como a implementação de áudio, som e animações, as quais não foram utilizadas no trabalho.

Por conseguinte, para a realização da avaliação, foi aplicado o ciclo de Engenharia de Software. Isto é, o levantamento de requisitos e conceitos, análise e interpretação do trabalho via diagrama de classes em UML, implementação em C++, e testes pelo uso do software.

Por fim, serão apresentados as seguintes seções: “explicação do jogo em si” (), “desenvolvimento do jogo na versão orientada a objetos” (), “tabela de conceitos utilizados e não utilizados” (), “discussão e conclusões” (), “divisão do trabalho” (), e “agradecimentos profissionais” ().

# EXPLICAÇÃO DO JOGO EM SI

O jogo NOME começa pelo menu principal (conforme a figura 1), no qual o(s) jogador(es) podem escolher entre: ou clicar no botão “Jogar” e entrar no menu de jogadores, ou clicar no botão “Ranking” e entrar no menu de ranking, ou clicar no botão “Sair” e fechar a janela do jogo (ou apertando a tecla ESC).

Ao entrar no menu ranking, poderá ser visto as maiores pontuações cadastradas de diferentes jogadores (conforme a figura 2). Mas também, é possível clicar no botão “Voltar” e reentrar no menu principal (ou apertando a tecla ESC).

No menu de jogadores (conforme a figura 3.1), é possível escolher entre jogar com um ou dois jogadores, no mesmo computador, e escrever um nome para cada (com limite de caracteres). Após o preenchimento da(s) caixa(s) de texto(s), é possível clicar no botão “Confirmar nome(s) e entrar no menu de fases. Se apertado a tecla ESC, volta ao menu principal.

Sobre o personagem jogador (conforme a figura 3.2), é possível controlá-lo usando teclas do teclado, caso haja dois jogadores, a funcionalidade será a mesma para ambos, entretanto as teclas serão diferentes. Portanto, o personagem poderá andar no eixo X, pular e atacar. Se ele sofrer dano, receberá “knockback”, isto é, dependendo da posição dele e do atacante, receberá uma força no eixo X para se afastar do local.

No menu de fases (conforme a figura 4), é necessário escolher em qual fase entrar, ou seja, na primeira fase, “a floresta”, ou na segunda fase, “o castelo”. Ao clicar no botão “Confirmar fase”, o(s) jogador(es) entrarão na fase escolhida. Se apertado a tecla ESC, volta ao menu de jogadores.

Na fase “a floresta” (conforme a figura 5), o(s) jogador(es) é/são criado(s) em uma posição pré-determinada, o(s) qual/quais deve(m) derrotar inimigos de dificuldade fácil (cavaleiros) e médio (mortos-vivos), além de terem cuidado com obstáculos fáceis (plataforma) e médio (barra mágica). Ademais, todos esses personagens devem ficar em cima de plataformas, se não, pela ação da gravidade, ao encostarem na borda inferior da janela, morrem. Se um jogador for criado e ele morrer, o jogo termina e cria a tela de fim de jogo. Se dois jogadores forem criados e os dois morrerem, o jogo termina e etc.

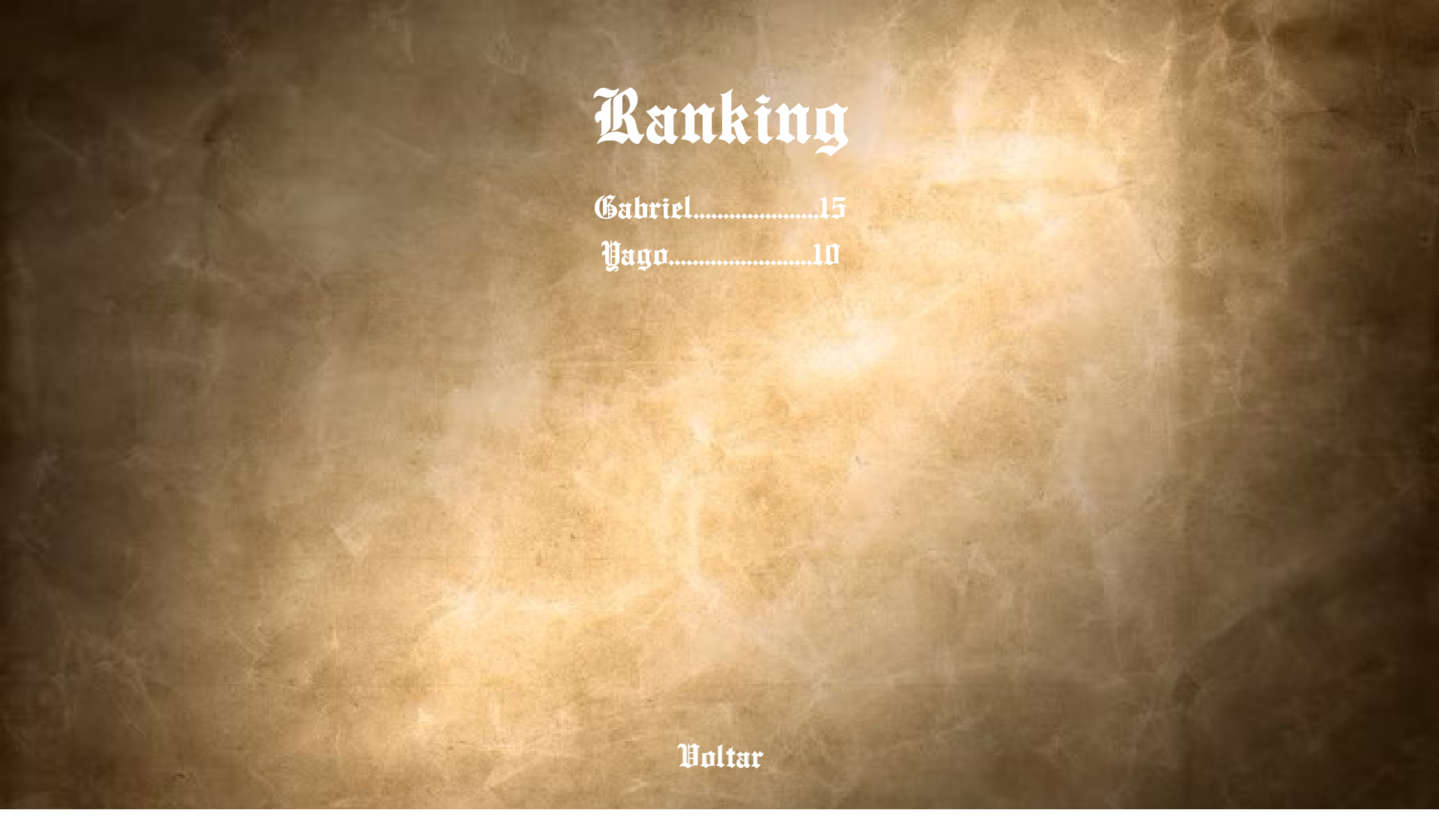


Figura 2. Menu de ranking

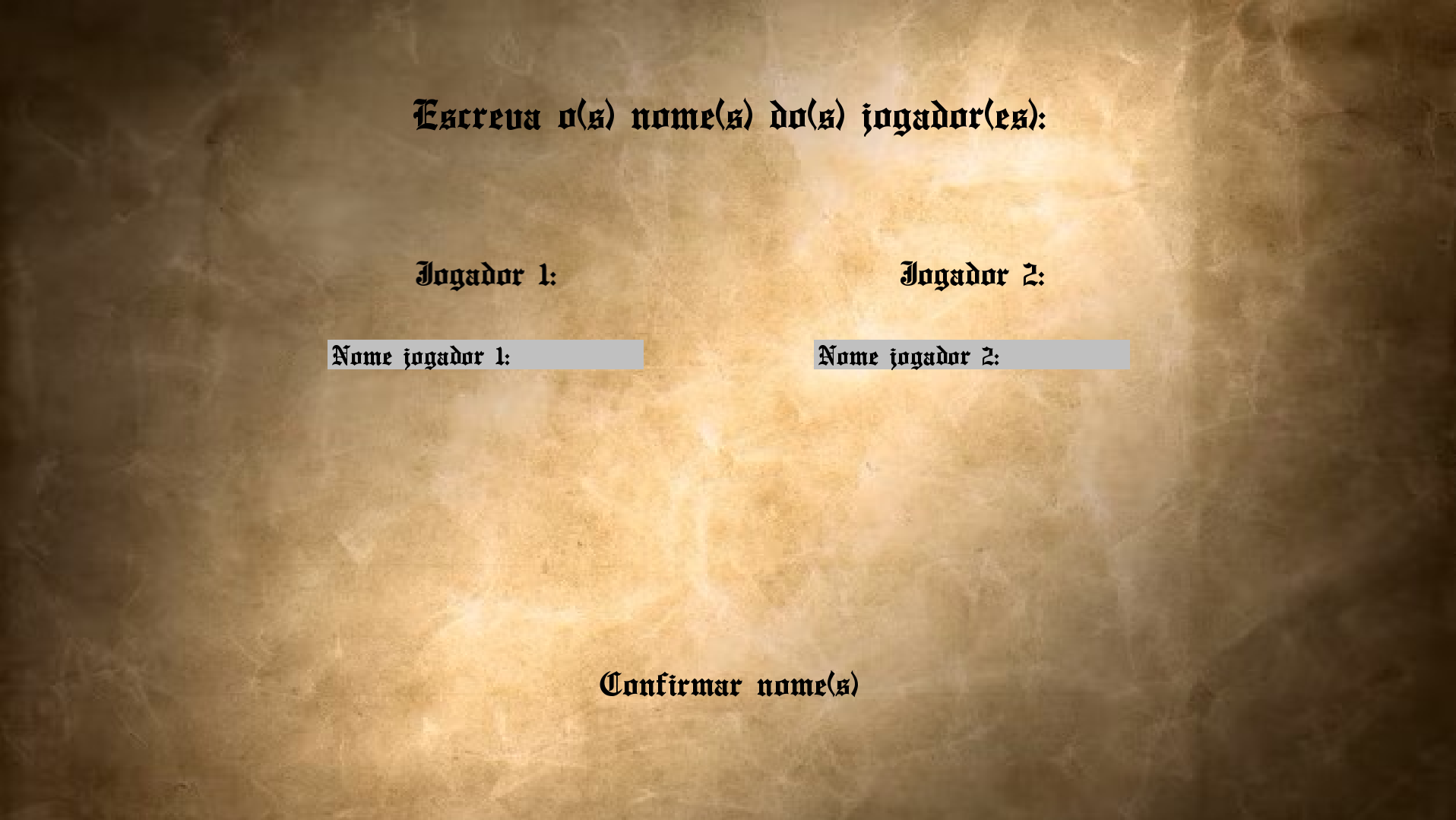


Figura 3.1. Menu de jogadores



Figura 3.2. Jogador



Figura 4. Menu de fases

A gravidade, no jogo, é uma aceleração constante que sempre afeta, até um valor máximo, todas as entidades. No caso dos obstáculos, eles sofrem uma contraforça que tem o mesmo valor em módulo da gravidade, a fim de mantê-los estáticos no lugar.

É importante ressaltar que em ambas as fases, as plataformas e os inimigos possuem um valor aleatório de instâncias. Sendo o valor mínimo para cada é 3 (três) e o valor máximo varia para cada entidade. Assim, na fase “a floresta”, o máximo de cavaleiros é 8 (oito), o de mortos-vivos é 5 (cinco), o de barras mágicas é 4 (quatro) e o de plataformas é 7 (sete). Já na fase “o castelo”, o máximo de cavaleiros, plataformas (o número mínimo também) e espinhos é diferente para cada tamanho de tela do computador, mas o número máximo de magos é 4 (quatro).

Sobre a mecânica do cavaleiro (conforme a figura 6), é um inimigo que, se colidir com um jogador, dará 1 (um) de dano. Além da movimentação (andar em linha reta), que depende da distância, no eixo X, de sua geração, ou seja, após a entidade percorrer um número de pixel’s da janela, ela andará pelo lado contrário e repetirá o processo indefinidamente.

Sobre o morto-vivo (conforme a figura 7), é um inimigo que dará 2 (dois) de dano e que sempre perseguirá o jogador mais próximo, isso se estiver dentro do raio de alcance, visão. Caso o jogador pulasse e estivesse sendo perseguido, a entidade também pularia, isto para se manter no mesmo eixo Y do jogador.

Sobre a barra mágica (conforme a figura 8), como a plataforma, é um obstáculo que se mantêm estático no cenário. Assim, sua funcionalidade se resume a paralisar, por um tempo, um jogador que colida com a barra. Porém, esse efeito ocorre somente nas laterais da barra, caso o jogador encoste na parte de cima, o mesmo não ficará paralisado.

Na fase “o castelo” (conforme a figura 9), o(s) jogador(es) deve(m) matar inimigos de dificuldade fácil (cavaleiros) e difícil (magos), além de terem cuidado com obstáculos fáceis (plataforma) e difícil (espinho). Quanto a tela de fim de jogo, ocorre o mesmo caso da fase “a floresta”.



Figura 6. Cavaleiro



Figura 7. Morto-vivo



Figura 8. Barra Mágica



Figura 9. O castelo

Sobre o mago (figura 10), há dois estados diferentes, se ele tiver com mais da metade da vida, dará 1 (um) de dano a um jogador, senão

causará 2 (dois) e se moverá mais rápido. Ao contrário do morto-vivo, o mago não pula, mas segue o jogador mais perto no eixo X. Além de poder lançar uma bola de fogo em direção ao jogador mais perto, o qual ao tocá-lo acarretá-la em um dano equivalente ao do arremessador. O mago somente atirará novamente quando o projétil não estiver na janela.

Sobre o projétil (figura 11), além de dar dano a um personagem, sofre a força da gravidade, e para alcançar o alvo é necessário o atirador aplicar uma força no eixo Y, causando um lançamento oblíquo. Assim, na equação (1) estará descrito como é feito para calcular tal força, considerando o ângulo do lançamento como 45º (quarenta e cinco graus).

Sobre o espinho (figura 12), sendo um obstáculo, fica estático na janela, além de que se um jogador colidir por qualquer lado, sofrerá 1 (um) de dano.

Na tela de fim de jogo (figura 13), o(s) jogador(es) pode(ram) entrar no menu principal e salvar a pontuação obtida ao clicar(em) o botão “Voltar ao menu” ou apertarem a tecla ESC.

Finalmente, nas fases, é possível, ao apertar a tecla ESC, entrar no menu de pause (figura 14). Nela, há duas opções: ou clicar no botão “Retomar” (ou apertar ESC) e voltar para onde todos os personagens estavam, sem perda de pontuação e de memória; ou clicar no botão “Salvar e sair” e salvar a pontuação já obtida, sair da fase e entrar no menu principal.



Figura 10. Mago

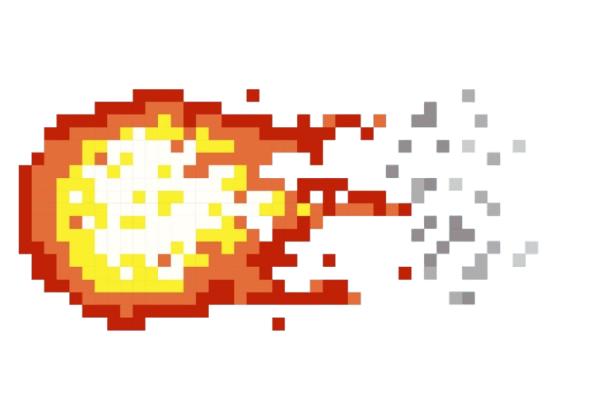
.

Figura 11. Projétil (bola de fogo)

(1)

# 

Figura 12. Espinho



# Figura 13. Tela de fim de jogo



# Figura 14: menu de pause (no castelo)

# DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Isto posto, nesta presente seção se deve discorrer a explicação do desenvolvimento do jogo/*software* utilizando orientação a objeto**, começando pelos requisitos e avançando para a modelagem, salientado o projeto em diagrama(s) de classes em UML,** e culminando na programação em C++. **A explicação deve ser feita de maneira tal a NÃO ser um relatório técnico repleto de detalhes, mas que seja capaz de sintetizar e valorizar os recursos técnicos utilizados (*i.e.*, sucinto e suficiente).**

Nesta explicação, deve-se primeiramente ter texto introdutório sucinto que leva a listar textualmente em tabela os **requisitos funcionais** definidos para o jogo/*software* em questão, os quais são justamente os requisitos definidos na Tabela 1 neste presente documento. Isto disto, os requisitos devem estar enquadrados em uma tabela de duas colunas na qual a primeira coluna traz os requisitos e a segunda coluna a sua situação (*status*) que pode ser ‘realizado’, ‘semi-realizado’ e ‘não realizado, usando sistemas de cores ademais (*i.e.*, verde feito, amarelo semifeito e vermelho não feito).

Ainda, quando o requisito estiver como ‘realizado’ ou ‘semi-realizado’ faz-se absolutamente necessário indicar sucintamente quais classes ou conjuntos de classes (*e.g.*, via pacotes) que realizaram cada requisito no preenchimento tabela, no tocante ao campo ‘Implementação’ da Tabela 1. Eventualmente, pode-se também nomear objetos que se julguem pertinentes, se não for suficiente apenas nomear suas classes. **Em tempo, cada requisito será contabilizado no âmbito de avaliação apenas quando estiver completamente ‘realizado’**.

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, no qual pode se escolher fases, escolher ver colocação (*ranking*) de jogadores e escolher demais opções pertinentes (previstas nos demais requisitos). | **REALIZADO** | Cf. as classes Menu, MenuFases, MenuJogadores e MenuRanking e seus respectivos objetos, com suporte da SFML. |
| 2 | Permitir um ou dois jogadores com representação gráfica aos usuários do Jogo, sendo que no último caso é para que os dois joguem de maneira concomitante. | **REALIZADO** | Cf. classe Jogador cujos objetos são agregados em jogo e inicializados cf. classe MenuJogadores. |
| 3 | Disponibilizar ao menos duas fases distintas que podem ser jogadas sequencialmente ou selecionadas, via menu, nas quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa. | **REALIZADO** | Cf. classes XYZ, no pacote W, sendo que permitem . . . |
| 4 | Ter pelo menos três tipos distintos de inimigos, cada qual com sua representação gráfica, sendo que ao menos um deles deve poder lançar projetil contra o(s) jogador(es) e um dos inimigos dever ser um ‘chefão’. | **REALIZADO** | Cf. pacote . |
| 5 | Ter a cada fase ao menos dois tipos de inimigos (um deles exclusivo nela) com número aleatório de instâncias, podendo ser várias instâncias (definindo um máximo) e sendo pelo menos 3 instâncias para cada tipo que estiver na fase. | **PARCIALMENTE** | Cf. . . . |
| 6 | Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem. | **REALIZADO** | Cf. . . . |
| 7 | Ter em cada fase ao menos dois tipos de obstáculos (um deles exclusivo nela) com número aleatório (definindo um máximo) de instâncias (*i.e.*, objetos), sendo pelo menos 3 instâncias por tipo. | **REALIZADO** | Cf. . . . |
| 8 | Ter em cada fase um cenário de jogo constituído por obstáculos, sendo que parte deles **devem ser** plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores. Em cada fase, só poder ter um tipo coincidente de imigo e um tipo coincidente de obstáculo (que é a plataforma) em relação as demais fases. | **REALIZADO** | Cf. . . . |
| 9 | Gerenciar colisões entre jogador para com inimigos e seus projeteis, bem como entre jogador para com obstáculos. Ainda, todos eles devem sofrer o efeito de alguma ´gravidade´ no âmbito deste jogo de plataforma vertical e 2D. | **REALIZADO** | Cf. . . . |
| 10 | Permitir: (1) salvar nome do usuário, manter/salvar pontuação (incrementada via neutralização de inimigos) do jogador controlado pelo usuário e gerar lista de pontuação (*ranking*). **E** (2) Pausar e **Salvar/Recuperar** Jogada. | **NÃO** realizado. | Requisito NÃO realizado. |
| **Total de requisitos funcionais apropriadamente realizados.**  *(Cada tópico realizado efetivamente vale 10%)* | | | **80%** (oitenta por cento). |
| Os requisitos dependem em algo uns dos outros, na chamada interdependência de requisitos. | | | |

Isto feito, após a tabela de requisitos, a explicação textual objetiva do desenvolvimento segue, devendo-se para tal:

* Utilizar Diagrama(s) de Classes em *UML* para explicar os pacotes com as classes e suas relações, que DEVEM atender aos requisitos.
* Utilizar OBRIGATORIAMENTE como base o Diagrama de Classes proposto na FIGURA 2, completando-o, melhorando-o, expandindo-o etc[[1]](#footnote-2).
* A luz do(s) diagrama(s), explicar o desenvolvimento de maneira sucinta e suficiente no texto. Assim, em poucos parágrafos (*i.e.*, algo como meia-página, não mais que uma página enfim) deve-se explicar as principais classes, à luz do agrupamento de classes em pacotes e como elas se inter-relacionam.
* Em tempo, valorizar as ‘sofisticações’ que tenham sido realizadas, como comportamento mais elaborado de inimigos.
* Valorizar a interdisciplinaridade como a aplicação de conceitos de física e matemática aprendidos em disciplinas do ensino médio e preferencialmente em disciplinas da graduação.
* Deixar no diagrama apenas o que de fato foi implementado em C++ ou, ao menos, bem indicar o que foi efetivamente implementado por sistema de cores das classes (verde feito, amarelo semi-feito e vermelho não feito).

Para a implementação em C++ orientado a objetos (OO), que é a expressão técnica do projeto em diagrama(s) de classes em UML, deve-se usar a biblioteca gráfica OO ***SFML***, pois isto valoriza o trabalho esteticamente além de demonstrar a capacidade de uso de bibliotecas e suas interfaces. Não deixar de valorizar esta atividade de uso da biblioteca gráfica nesta seção. Em tempo, no site da disciplina e/ou no perfil dela no moodle, há exemplos pedagógicos prontos com *SFML*.

Esta seção em questão é muito importante e fundamental no trabalho, sendo que será corrigida com muita atenção pelo professor. Pede-se, por fim, que todos os autores revisem cuidadosamente a versão final do trabalho (como um todo) para evitar erros de português, digitação e/ou formatação. Na verdade, além disto, uma equipe poderia revisar o trabalho escrito da outra e vice-versa para fins de aprimoramento mútuo, aconselhando-se que o artigo-relatório esteja em um documento on-line (e.g., *google docs*) para facilitar as atividades pertinentes. **Por fim, o trabalho será corrigido à luz do diagrama de classe impresso, sendo que o que não estiver no diagrama já de prima será considerado como não feito.**

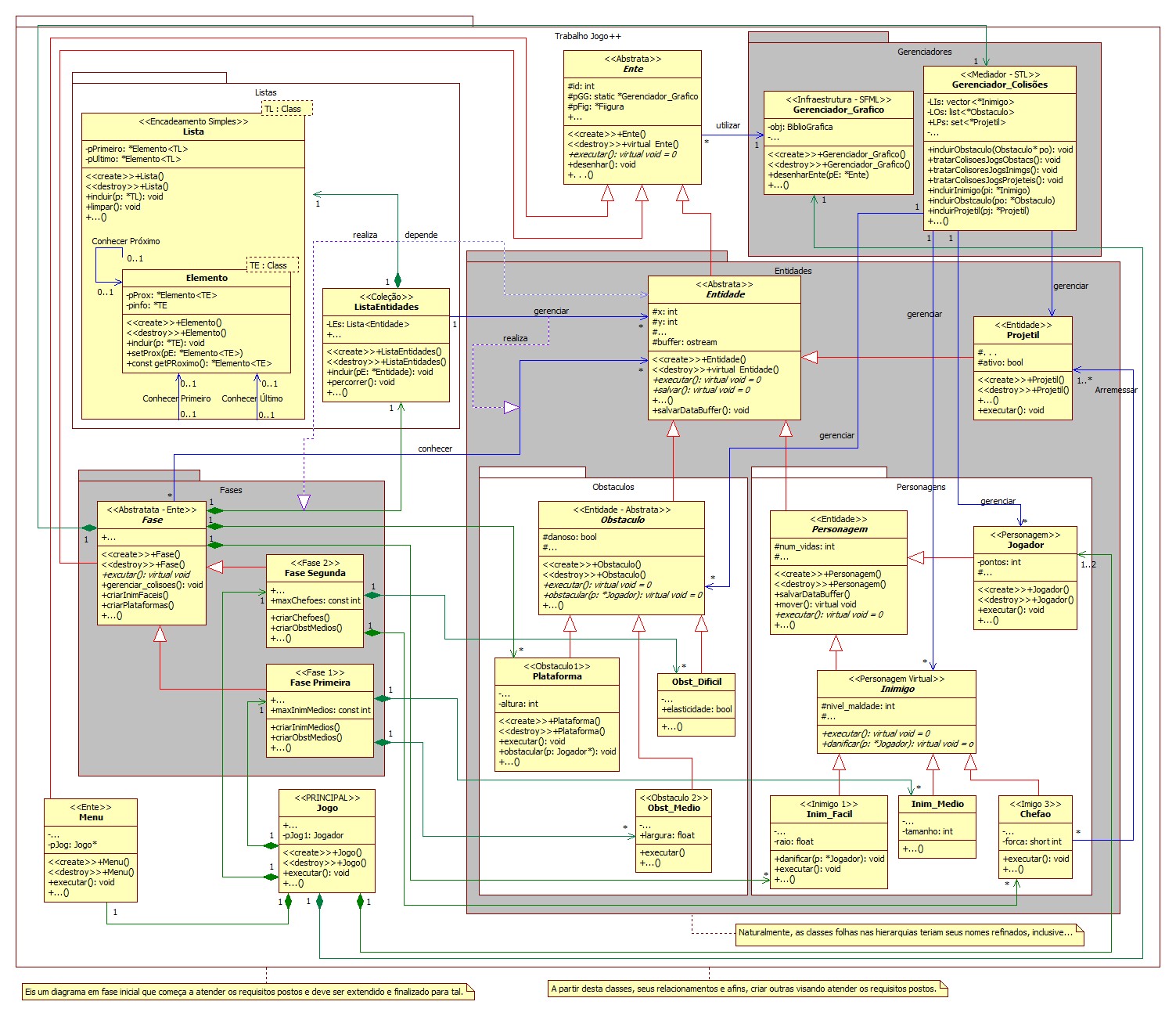


Figura 2. Diagrama de Classes de base em UML – Padrão Arquitetural do Jogo a ser seguido!

# TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Nesta seção, em relação aos conceitos aprendidos, deve-se apresentar uma tabela de conceitos utilizados e não utilizados conforme a Tabela 2, sendo que a coluna de ‘Conceitos’ dessa tabela **NÃO** pode ser alterada, absolutamente. Oportunamente, todas as tabelas que venham a ser utilizadas deverão ser numeradas sequencialmente com algarismos arábicos, conforme o exemplo abaixo**. Ainda, NUNCA se começa seção de artigo-relatório diretamente com a tabela, sem antes apresentar texto a introduzindo.** Em verdade, toda seção deve ser começada por texto apropriado, antes de apresentar elementos outros como tabelas, figuras etc.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **N.** | Conceitos | Uso | Onde / O quê / Justificativa em uma linha |
| **1** | **Elementares:** | | |
| 1.1  **&** | - Classes, objetos. &  - Atributos (privados), variáveis e constantes.  - Métodos (com e sem retorno). | Sim | - Todos .h e .cpp, como nas classes nos *namespaces* X e Y.  - Classe, Objetos, Atributos e Métodos foram utilizados porque são conceitos elementares na orientação a objetos. |
| 1.2  **&** | - Métodos (com retorno *const* e parâmetro *const*).  - Construtores (sem/com parâmetros) e destrutores | Sim | - Na maioria dos .h e .cpp, como nas classes nos *namespaces* W e Z.  - A constância pertinente evita mudanças equivocadas, construtores são mandatórios para inicializar atributos e destrutores pertinentes para finalizações como desalocações. |
| 1.3 | - Classe Principal. | Sim | - Main.cpp & Principal.h/.cpp  - Uma classe Principal é mais ‘purista’ em termos de OO. |
| 1.4 | - Divisão em .h e .cpp. | Sim | - No desenvolvimento como um todo, como nas classes nos *namespaces* A e B.  - Permite organizar as classes e afins que compõem o sistema. |
| **2** | **Relações de:** | | |
| 2.1 | - Associação direcional. &  - Associação bidirecional. | Sim | - Em vários dos .h e .cpp, como nas classes nos *namespaces* W e Z.  - Associação direciona é fundamental para .... |
| 2.2  **&** | - Agregação via associação.  - Agregação propriamente dita. | Sim | - Em vários dos .h e .cpp, como nas classes nos *namespaces* W e Z.  - ... |
| 2.3  **&** | - Herança elementar.  - Herança em vários níveis. | Sim | - Em alguns dos .h e .cpp, como nas classes nos *namespaces* W e Z.  - ... |
| 2.4 | - Herança múltipla. | Sim | - Precisamente nos .h e .cpp, das classes C, D e F.  - ... |
| **3** | **Ponteiros, generalizações e exceções** | | |
| 3.1 | - Operador *this* para fins de relacionamento bidirecional | Sim | - Precisamente nos .h e .cpp, das classes X, Y e Z.  - ... |
| 3.2 | - Alocação de memória (*new* & *delete*). | Sim | - ...  - ... |
| 3.3 | - Gabaritos/*Templates* criada/adaptados pelos autores para Listas. | Sim | - ...  - ... |
| 3.4 | - Uso de Tratamento de Exceções (*try catch*). | Sim | - ...  - ... |
| **4** | **Sobrecarga de:** | | |
| 4.1 | - Construtoras e Métodos. | Sim | . . . |
| 4.2 | - Operadores (2 tipos de operadores pelo menos). |  | Foi usado o *operator==* e o *operator++* no PPP.h. ***[Especifar qual e onde aqui!]*** |
| --- | **Persistência de Objetos (via arquivo de texto ou binário)** | | |
| 4.3 | - Persistência de Objetos. |  | . . . |
| 4.4 | - Persistência de Relacionamento de Objetos. |  | . . . |
| **5** | **Virtualidade:** | | |
| 5.1 | - Métodos Virtuais Usuais. |  | . . . |
| 5.2 | - Polimorfismo. |  | . . . |
| 5.3 | - Métodos Virtuais Puros / Classes Abstratas. |  | . . . |
| 5.4 | - Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto (mais de 5 padrões). |  | . . . |
| **6** | **Organizadores e Estáticos** | | |
| 6.1 | - Espaço de Nomes (*Namespace*) criada pelos autores. |  | . . . |
| 6.2 | - Classes aninhadas (*Nested*) criada pelos autores. |  | . . . |
| 6.3 | - Atributos estáticos e métodos estáticos. |  | . . . |
| 6.4 | - Uso extensivo de constante (*const*) parâmetro, retorno, método... |  | . . . |
| **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) |  | . . . |
| 7.2 | - Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa. |  | . . . |
| --- | **Programação concorrente** | | |
| 7.3 | *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins*.* |  | . . . |
| 7.4 | *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetoscom uso de Mutex, Semáforos, **OU** Troca de mensagens. |  | . . . |
| **8** | **Biblioteca Gráfica / Visual** | | |
| 8.1  **&** | - Funcionalidades Elementares.  - Funcionalidades Avançadas como: tratamento de colisões e duplo *buffer* |  | *Especificar aqui quais funcionalidades.* |
| 8.2  **OU** | - Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive, via padrão de projeto *Observer*) em algum ambiente gráfico.  *- RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc). |  | . . . |
| --- | **Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.** | | |
| 8.3 | - Ensino Médio Efetivamente. |  | Especificar quais conceitos aqui. |
| 8.4 | - Ensino Superior Efetivamente*.* |  | Especificar quais conceitos aqui. |
| **9** | **Engenharia de Software** | | |
| 9.1 | - Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. |  | . . . |
| 9.2 | - Diagrama de Classes em *UML*. |  | . . . |
| 9.3 | - Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões. |  | . . . |
| 9.4 | - Testes à luz da Tabela de Requisitos e do Diagrama de Classes. |  | . . . |
| **10** | **Execução de Projeto** | | |
| 10.1  **&** | - Controle de versão de modelos e códigos automatizado (via github).  - Uso de alguma forma de cópia de segurança (*i.e.*, *backup*). |  | Especificar qual modo.  **INFORMAR AQUI O LINK DO GITHUB.** |
| 10.2 | - Reuniões com o professor para acompanhamento do andamento do projeto.  **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]** |  | Especificar quantidade e **quando**, sendo o mínimo de 2 reuniões a serem marcadas conforme instrução a ser dada (normalmente via planilha on line a ser indicada). Estas duas reuniões só depois das 4 reuniões com os monitores / Peteco (vide item justo abaixo). Ainda, após cada reunião, enviar e-mail para o professor, com cópia para seu colega de dupla, relatando sucintamente a reunião. |
| 10.3 | - Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.  **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]** |  | Deve haver 4 reuniões com o monitor (de algo com meia-hora cada**)** **e/ou curso com o pessoal do PETECO** (de algo com duas horas pelo menos**), especificando quando**. Ainda, após cada reunião ou curso, enviar e-mail para o professor, com cópia para seu colega de dupla e monitor, relatando sucintamente a reunião/curso. |
| 10.4  **&** | - Escrita do trabalho e feitura da apresentação  - Revisão do trabalho escrito de outra equipe e vice-versa. |  | Especificar quem fez o quê.  Especificar qual equipe. |
| **Total de conceitos apropriadamente utilizados.**  (Cada grande tópico vale 10% do total de conceitos. Assim, caso se tenha feito metade de um tópico, então ele valeria 5%.) | | | **70%** (setenta por cento).  (Naturalmente e obviamente, este tipo de observação aqui entre parênteses deve ser retirada dos relatórios.) |

Ressalta-se que as legendas das figuras devem ser colocadas abaixo, enquanto as legendas das tabelas devem ser colocadas acima das mesmas. Em tempo, entre tabelas também deve sim ter texto apropriado. Idem entre figuras, as quais sempre são intercaladas por textos pertinentes.

# DISCUSSÃO E CONCLUSÕES

Esta seção deverá apresentar reflexão sobre o desenvolvimento e os resultados obtidos. Certamente uma conclusão bem elaborada auxilia na avaliação do professor. **Outro item ainda mais fundamental para a avaliação são as reuniões com os monitores/petecos e com o professor, sendo que o trabalho não pode ser entregue sem elas terem ocorrido, bem entendido**.

Por sua vez, a avaliação do trabalho como um todo pelo professor será baseada em:

* Quantidade e qualidade dos requisitos funcionais cumpridos na elaboração do *software*, à luz do conjunto e qualidade da modelagem e códigos, resultando em número apropriado de classes e objetos, número/forma apropriada de relacionamentos e apropriada complexidade algorítmica. Naturalmente, isto tudo envolve particularmente os bons princípios de Orientação Objetos, como organização, encapsulamento e reutilização, todos baseados no princípio de coesão e desacoplamento doutrinados na disciplina.
* Quantidade e qualidade dos conceitos utilizados na elaboração do *software*, inclusive em termos de modelagem e realização de código, o que novamente envolve a correção na aplicação dos princípios da Orientação a Objetos (*e.g.*, coesão, desacoplamento, encapsulamento, organização e reutilização), além da utilização apropriada de cada conceito em si, naturalmente.
* O conteúdo das reuniões e as evoluções a partir delas, bem como a qualidade do trabalho escrito, da apresentação, de diagramas, de códigos e afins.

Em termos gerais, pode-se considerar o primeiro item com um peso de 35%, o segundo com um peso de 35% e o terceiro com um peso de 30%, lembrando que eles estão inter-relacionados. Não obstante, esta porcentagem é relativa, pois (por exemplo) um item muito bem desenvolvido pode eventualmente até compensar (em algo) outro não tão bem desenvolvido em proporções diferentes deste referencial dado. Lembrar ainda e novamente que fazer o projeto ser acompanhado pelo professor e monitor(es), à medida que avança, é **fundamental**.

# DIVISÃO DO TRABALHO

Esta seção deverá ter uma tabela salientando quem desenvolveu cada classe/módulo do *software* e realizou demais atividades como as de ‘engenharia de *software’*, a redação do trabalho escrito, a revisão da redação do trabalho e a preparação da apresentação do trabalho. A tabela 4 pode e mesmo deveria ser melhorada à luz das tabelas de requisitos e conceitos.

Tabela 4. Lista de Atividades e Responsáveis.

|  |  |  |
| --- | --- | --- |
| **N.** | Atividades | Responsáveis |
| **1** | **Elementares: AMBOS** | |
| 1.1 | - Classes, objetos. & - Atributos (privados), variáveis e constantes. &  - Métodos (com e sem retorno). | Fulano e Ciclano |
| 1.2 | - Métodos (com retorno *const* e parâmetro *const*). &  - Construtores (sem/com parâmetros) e destrutores | Fulano e Ciclano |
| 1.3 | - Classe Principal. | Fulano e Ciclano |
| 1.4 | - Divisão em .h e .cpp. | Fulano e Ciclano |
| **2** | **Relações de: MAIS FULANO** | |
| 2.1 | - Associação direcional. & - Associação bidirecional. | Mais Fulano que Ciclano |
| 2.2 | - Agregação via associação. & - Agregação propriamente dita. | Mais Fulano que Ciclano |
| 2.3 | - Herança elementar. & - Herança em vários níveis. | Mais Fulano que Ciclano |
| 2.4 | - Herança múltipla. | Mais Fulano que Ciclano |
| **3** | **Ponteiros, generalizações e exceções: MAIS FULANO** | |
| 3.1 | - Operador *this* para fins de relacionamento bidirecional. | Fulano |
| 3.2 | - Alocação de memória (*new* & *delete*). | Fulano |
| 3.3 | - Gabaritos/*Templates* criada/adaptados pelos autores para Listas. | Fulano |
| 3.4 | - Uso de Tratamento de Exceções (*try catch*). | Ciclano |
| **4** | **Sobrecarga de: CICLANO** | |
| 4.1 | - Construtoras e Métodos. | . . . |
| 4.2 | - Operadores (2 tipos de operadores pelo menos) | . . . |
| --- | **Persistência de Objetos (via arquivo de texto ou binário):** | |
| 4.3 | - Persistência de Objetos. | . . . |
| 4.4 | - Persistência de Relacionamento de Objetos. | . . . |
| **5** | **Virtualidade: . . .** | |
| 5.1 | - Métodos Virtuais Usuais. | . . . |
| 5.2 | - Polimorfismo. | . . . |
| 5.3 | - Métodos Virtuais Puros / Classes Abstratas. | . . . |
| 5.4 | - Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto (mais de 5 padrões). | . . . |
| **6** | **Organizadores e Estáticos: . . .** | |
| 6.1 | - Espaço de Nomes (*Namespace*) criada pelos autores. | . . . |
| 6.2 | - Classes aninhadas (*Nested*) criada pelos autores. | . . . |
| 6.3 | - Atributos estáticos e métodos estáticos. | . . . |
| 6.4 | - Uso extensivo de constante (*const*) parâmetro, retorno, método... | . . . |
| **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) | . . . |
| 7.2 | - Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa. | . . . |
| --- | **Programação concorrente: . . .** | |
| 7.3 | *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins*.* | . . . |
| 7.4 | *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetoscom uso de Mutex, Semáforos, **OU** Troca de mensagens. | . . . |
| **8** | **Biblioteca Gráfica / Visual: . . .** | |
| 8.1 | - Funcionalidades Elementares. **&**  - Funcionalidades Avançadas como: tratamento de colisões e duplo *buffer* | . . . |
| 8.2 | - Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive, via padrão de projeto *Observer*) em algum ambiente gráfico. **OU** *- RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc). | . . . |
| --- | **Interdisciplinaridades via uso de Conceitos de Matemática Contínua e/ou Física: . . .** | |
| 8.3 | - Ensino Médio Efetivamente. | . . . |
| 8.4 | - Ensino Superior Efetivamente*.* | . . . |
| **9** | **Engenharia de Software: . . .** | |
| 9.1 | - Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. | . . . |
| 9.2 | - Diagrama de Classes em *UML*. | . . . |
| 9.3 | - Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, + de 5 padrões. | . . . |
| 9.4 | - Testes à luz da Tabela de Requisitos e do Diagrama de Classes. | . . . |
| **10** | **Execução de Projeto: . . .** | |
| 10.1 | - Controle de versão de modelos e códigos automatizado (via github). &  - Uso de alguma forma de cópia de segurança (i.e., backup). | . . . |
| 10.2 | - Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO A ENTREGA DO TRABALHO]** | . . . |
| 10.3 | - Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA]** | . . . |
| 10.4 | - Escrita do trabalho e feitura da apresentação **&**  - Revisão do trabalho escrito de outra equipe e vice-versa. | . . . |

Aqui, após a tabela deve-se **obrigatoriamente** constar o quanto cada um trabalhou no projeto em termos de realização (*e.g.*, modelagem e escrita de código) e colaboração (*e.g.*, revisão de código e testes):

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

- Ciclano trabalhou em 20% das atividades as realizando ou colaborando nelas efetivamente.

# AGRADECIMENTOS PROFISSIONAIS

Havendo agradecimentos de ordem profissional, como ajuda de monitores e/ou ao pessoal do PETECO, estes deverão vir antes das referências. Neste sentido, aqui se pode também salientar e agradecer caso outra equipe tenha revisado o trabalho.

# REFERÊNCIAS CITADAS NO TEXTO

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

[2] STADZISZ, P. C. Projeto de Software usando UML. Apostila CEFET-PR 2002.

[http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/EngSoftware/Apostila%20UML%20-%20Stadzisz%202002.pdf](http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/EngSoftware/Apostila UML - Stadzisz 2002.pdf)

[3] SIMÃO, J. M. Site das Disciplina de Fundamentos de Programação 2, Curitiba – PR, Brasil, Acessado em 20/06/2021, às 20:32 -

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

# 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.

1. Eventualmente, pode-se utilizar outros diagramas UML caso os conheça ou os tenha estudado. [↑](#footnote-ref-2)