TRABALHO PARA A DISCIPLINA DE FUNDAMENTOS DE PROGRAMAÇÃO 2 DO CURSO DE ENGENHARIA ELETRÔNICA DA UTFPR: *MODELO PARA COMPOSIÇÃO DO TRABALHO*

José Geraldo Mirachi, Thiago Henrique Pincinato

josemirachi@alunos.utfpr.edu.br, thiago.h1993@hotmail.com

Disciplina: **Fundamentos de Programação 2** / S11 – Prof. Dr. Jean M. Simão

**Departamento Acadêmico de Eletrônica – DAELN** - Campus de Curitiba

### Universidade Tecnológica Federal do Paraná - UTFPR

Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

**Resumo** – A disciplina de Fundamentos de Programação 2 exige o desenvolvimento de um *software*, 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 Sonic Boom, onde o jogador terá que chegar até o final da fase passando por alguns inimigos e obstáculos. O jogo possui quatro fases que se diferenciam por dificuldades para o jogador, sendo que a última fase é onde o jogador desafia o chefão do jogo. Para o desenvolvimento do jogo foram levantados textualmente os requisitos, estabelecidos os casos de uso e elaborado uma modelagem (análise e projeto) usando recursos como Diagrama de Classes e Diagrama de Atividades em *Unified Modeling Language* (*UML*). Subsequentemente, em linguagem de programação C++, realizou-se o desenvolvimento que contemplou os conceitos usuais de Orientação a Objetos, bem como alguns conceitos avançados como Polimorfismo, Classe Abstrata, Gabaritos, Persistências de Objetos por Arquivos, Sobrecarga de Operadores e Standard Template Library (STL). Após a implementação, os testes e uso do jogo demonstraram sua funcionalidade conforme os requisitos e o projeto. Por fim, salienta-se que o projeto permitiu cumprir o objetivo de aprendizado visado.

**Palavras-chave ou Expressões-chave:** Unified Modeling Language (UML), Programação Orientado a Objetos, Engenharia de Software.

**Abstract** - This document shows a model for the manuscript to the academic work of *Fundamentos de Programação 2* as well as it presents general instructions about this academic work. With respect to the abstract contents, it must give a general explanation about the work. Precisely, the abstract must shortly present the work motivation and context, its study object (ordinarily a game), its development process, and the obtained results. An instance of abstract would be:

**. . .**

**Key-words or Key-expressions** (maximum four, not exceeding three lines)**:** Paper Model to the Academic Work of Programming Course, Academic Work Related to C++ Implementation, Internal Rules for Work Elaboration, Examples of Elements for the Work of a Programming Course.

# INTRODUÇÃO

Este presente artigo-relatório tem como objetivo relatar o desenvolvimento de um software/jogo, utilizando os princinpios de Engenharia de Software e a programação orientada à objeto. Para tal implementou-se o jogo com a linguagem de programação C++ , onde todos os conceitos explicados durante a disciplina de Fundamentos de Programação 2 foram utilizados.

O objetivo de estudo deste projeto é a implementação de um jogo no estilo plataforma aplicando, da melhor forma possível, os conceitos de orientação à objetos. O jogo deve ser implementado de forma que sua programação fique desacoplada e encapsulada .

Para realização deste trabalho, baseando-se no ciclo clássico de Engenharia de Software, primeiramente foi levantado os requisitos do jogo, conforme mostra a tabela 1. Após o levantamento dos requisitos foi efetuado a modelagem do jogo utilizando os diagramas em UML, tais como Diagrama de Atividades, Diagrama de Casos e Usos e Diagrama de Classes.

As seções subsequentes explicam como é o funcionamento do jogo em si, como seria sua implementação na versão procidemental e na versão orientada a objeto, comparando os dois tipos de implementação.

# EXPLICAÇÃO DO JOGO EM SI

Sonic Boom é um jogo de plataforma em deslocação lateral (2D), cuja jogabilidade centra-se na capacidade de Sonic correr em alta velocidade através de níveis que incorporam molas, quedas altas e loops verticais. Os níveis têm perigos na forma de robots (inimigos), dentro dos quais o Dr. Robotnik aprisionou animais. Destruir um robot liberta o animal dentro dele, mas tal não é necessário para completar o jogo. O jogador deve evitar filas de espinhos afiados, cair em poços sem fundo, ser esmagado por paredes e plataformas movediças e a morte por afogamento. O principal ataque de Sonic é a sua própria rotação que, quando se enrola, fica como uma bola e gira rapidamente causando danos aos inimigos. O ataque pode ser feito quando o jogador salta ou rola no chão.

Espalhados pelos níveis estão alguns anéis de ouro, e coletando 100 destes o jogador é recompensado com uma vida extra.

O jogo está divido em três zonas (Green Hill, Marble, Spring Yard, Labyrinth, Star Ligh e Scrap Brain), cada uma com seus próprios estilos visuais e inimigos. Após completar os três níveis o jogador confronta o Dr. Robotnik numa luta de chefe.

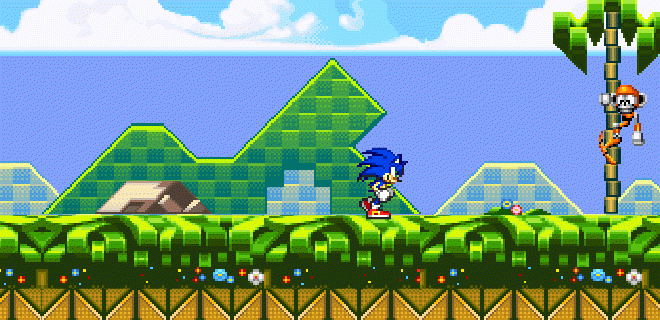


Figura 1. Fase 01 do Jogo



Figura 2. Sonic combatendo Dr. Robotnik

# DESENVOLVIMENTO DO/DE JOGO NA VERSÃO PROCEDIMENTAL

Idealmente, o jogo ou *software* em questão teria sido implementado em disciplina anterior sob o viés de abordagem procedimental (i.e. ‘orientação’ a procedimentos ou funções), o que permitiria subseqüentemente tecer reflexão comparativa entre a programação (e mesmo projeto) procedimental e a orientada a objetos.

Isto dito, nesta seção se deveria discorrer a explicação resumida do desenvolvimento do jogo ou *software* em questão caso ele já tivesse sido implementado anteriormente sob o viés de abordagem procedimental em linguagem C (ou mesmo em outra linguagem ‘procedimental’).

Entretanto, caso o jogo ou *software* em questão não tenha sido previamente implementado sob a abordagem procedimental, pode-se então discutir e mesmo apresentar (sucintamente) algum outro jogo ou *software* feito anteriormente, utilizando abordagem procedimental.

Isto também permitiria mostrar que se teria experiência prévia para tecer reflexão comparativa entre a abordagem procedimental e a orientada a objetos. Tal comparação se daria na seção chamada COMPARAÇÃO ENTRE DESENVOLVIMENTOS explicada mais ao final deste modelo de artigo-relatório.

Nesta explicação em questão, sobre desenvolvimento procedimental, aconselhar-se-ia:

* Listar sucintamente os requisitos funcionais do jogo/*software* visado.
* Utilizar um fluxograma para explicar a dinâmica do jogo/*software* visado e mesmo utilizar parte do código. Este último, caso realmente se faça necessário e enquadrado em uma moldura com se fosse uma figura.
* Utilizar, até mesmo, um ‘diagrama de blocos’ e/ou ‘diagrama de fluxo de dados’ para explicar os módulos existentes (e.g. funções ou conjunto de funções) e suas conexões.
* Valorizar as ‘sofisticações’ que tenham sido realizadas, como uma eventual função que permita ao jogador humano enfrentar um dado ‘jogador artificial’.
* **Evitar frases que não reflitam a realidade** como “implementou-se a inteligência artificial”. Este exemplo dado não reflete a realidade porque certamente os autores não implementaram todo conteúdo da disciplina/domínio de Inteligência Artificial. Neste caso o correto seria: “implementou-se uma técnica ou solução que pode ser classificado como uma solução (elementar) de Inteligência Artificial”.

O projeto apresentado, em sua versão procedimental, ou seja, orientado a procedimentos ou funções, é apresentado abaixo através de um fluxograma.



Figura 3. Fluxograma Versão Procedimental

# DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Nesta seção se deve discorrer a explicação do desenvolvimento do jogo/*software* utilizando orientação a objeto, culminando na programação em C++. A explicação deve-se 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 listar (normalmente textualmente) os **requisitos funcionais** levantados para o jogo/*software* em questão (certamente aproveitando os esforços da APS1). Os requisitos devem estar enquadrados em uma tabela de duas colunas onde a primeira coluna traz os requisitos e a segunda a sua situação (*status*) que pode ser ‘realizado’, ‘semi realizado’, ‘abandonado’ etc. **A Tabela 1 exemplifica o exposto definindo, ademais, os requisitos mínimos que cada jogo deve ter**. Ainda, os requisitos também devem ser convertidos em Casos de Uso e expressos por meio de Diagramas de Casos de Uso em *UML*.

Para o desenvolvimento do jogo, em sua versão orientada a Objetos, foram levantados alguns requisitos funcionais conforme é mostrado na Tabela 1.

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

|  |  |  |
| --- | --- | --- |
| N. | Requisitos Funcionais | Situação |
| 1 | Apresentar menu de opções aos usuários do Jogo | Requisito previsto inicialmente e realizado |
| 2 | Permitir um ou dois jogadores aos usuários do Jogo |  |
| 3 | Disponibilizar ao menos três fases que podem ser jogadas sequencialmente ou selecionadas. |  |
| 4 | Ter seis tipos distintos de inimigos |  |
| 5 | Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias. |  |
| 6 | Ter inimigo “Chefão” na última fase |  |
| 7 | Ter quatro tipos de obstáculos. |  |
| 8 | Ter em cada fase entre um e quatro tipos de obstáculos com número aleatório de obstáculos. |  |
| 9 | Ter representação gráfica de instâncias. |  |
| 10 | Ter em cada fase um cenário de jogo com obstáculos. |  |
| 11 | Gerenciar colisões entre jogador e inimigos. |  |
| 12 | Gerenciar colisões entre jogador e inimigos. |  |
| 13 | Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (*ranking*). |  |
| 14 | Permitir Pausar o Jogo |  |
| 15 | Permitir Salvar Jogada. |  |
| 16 |  |  |

Isto feito a explicação do desenvolvimento segue devendo-se:

* “Converter” requisitos em Casos de Uso e expressá-los por meio de Diagramas de Casos de Uso em *UML*.
* Utilizar um Diagrama de Classes em *UML* para explicar as classes e suas relações (que normalmente atendem aos requisitos).
* Utilizar um Diagrama de Atividades (“Fluxograma”) em *UML* para explicar a dinâmica do programa e utilizar até mesmo parte do código (caso isto realmente se faça necessário).
* Utilizar demais diagramas da *UML* **caso** o(s) aluno(s) os saiba(m) ou esteja(m) fazendo esforços para apreendê-los.
* A luz dos diagramas, explicar o desenvolvimento de maneira sucinta e suficiente.
* Valorizar as ‘sofisticações’ que tenham sido realizadas, como uma eventual função que permita ao jogador humano enfrentar um dado ‘jogador artificial’.
* Valorizar a interdisciplinaridade como a aplicação de conceitos de física e matemática aprendidos em disciplinas do ensino médio e (preferencialmente) em disciplinas da graduação.

Para a implementação em C++ deve-se usar uma biblioteca gráfica (*e.g*. *Allegro*, *TCL*/*TK*, *OpenGL* ou outra), pois isto valoriza o trabalho esteticamente além de demonstrar a capacidade de ‘pesquisa’ e aprendizado. Não deixe de valorizar esta capacidade de pesquisa e aprendizado nesta seção.

Esta seção em questão é muito importante no trabalho e será corrigida com muita atenção pelo Professor. Pede-se, por fim, que todos os autores revisem cuidadosamente a versão final do trabalho 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.

Após o levantamento de forma textual dos requisitos, foi elaborado os diagramas de atividades e de casos e usos em UML, conforme mostrado nas figuras 4 e 5.

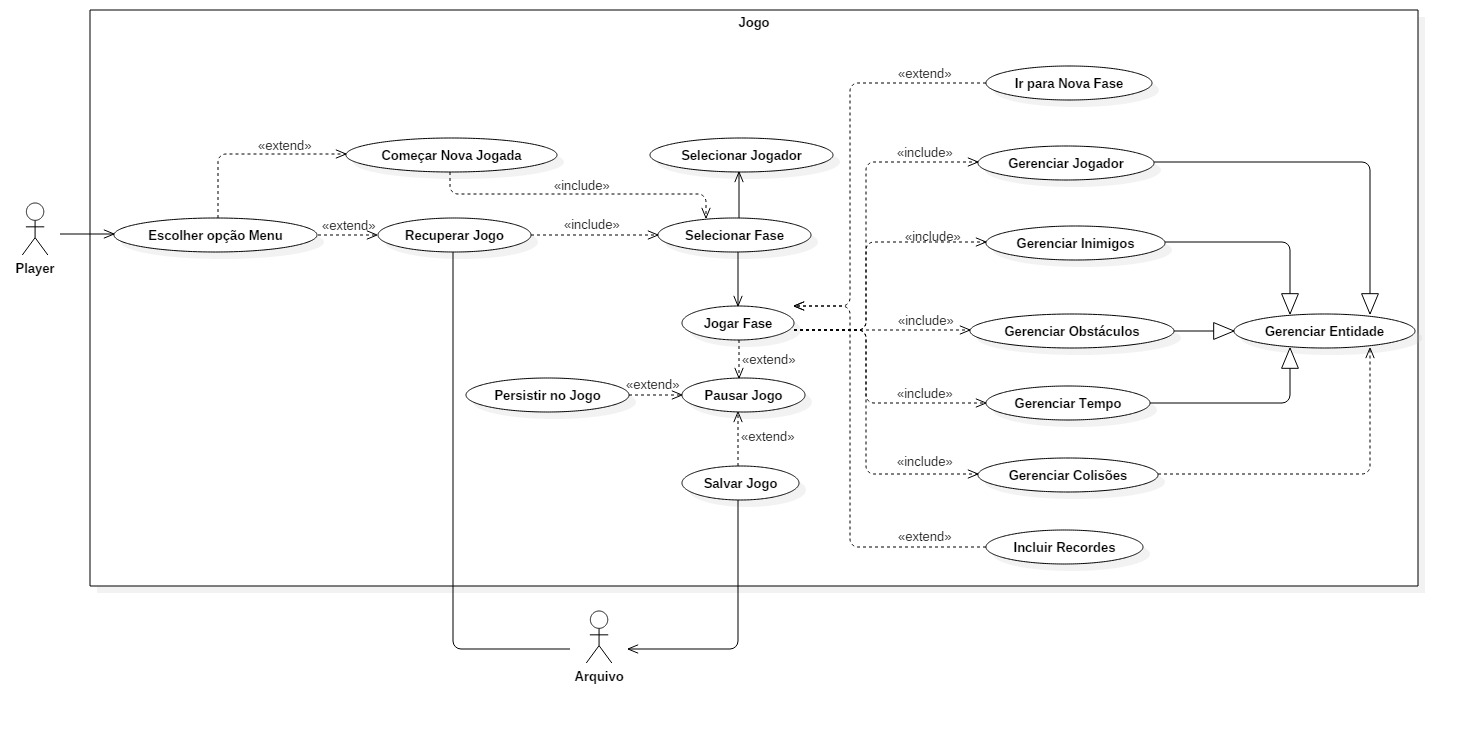


Figura 4. Diagrama de Casos e Usos

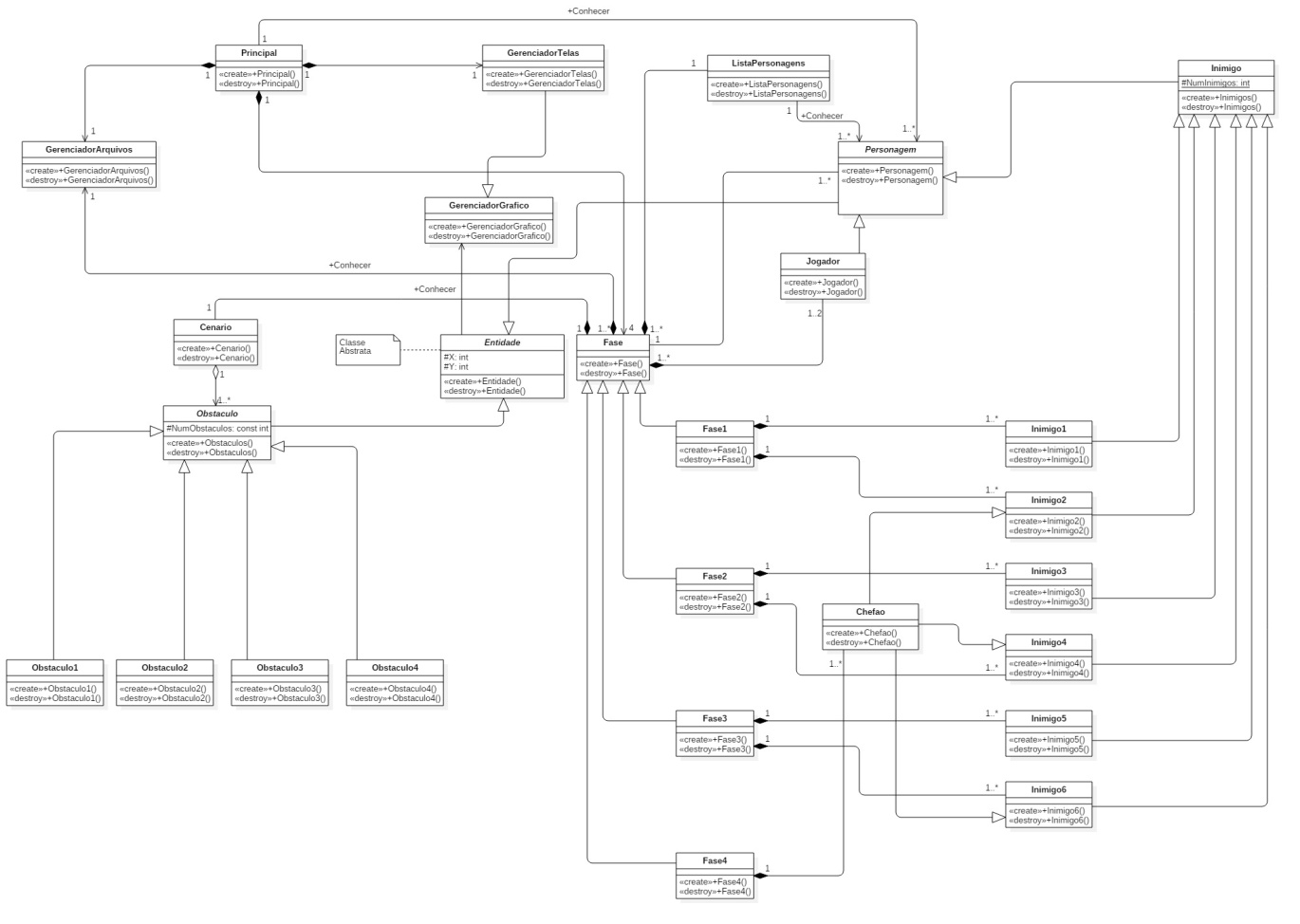


Figura 5. Diagrama de Classes

# 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 tal qual a Tabela 2. Deve-se também apresentar outra tabela justificando o uso ou não uso, tal qual a Tabela 3.

Oportunamente, todas as tabelas que venham a ser utilizadas deverão ser numeradas seqüencialmente com algarismos arábicos, conforme o exemplo abaixo:

Para realização do jogo foram utilizados alguns conceitos aprendidos durante o semestre na disciplina de Fundamentos de Programação 2. Tais conceitos são apresentados na tabela 2.

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **N.** | | Conceitos | | Uso | | Onde | |
| **1** | | **Elementares:** | | | | | |
| - Classes, objetos, | | Sim | | Todos .h e .cpp | |
| - Atributos (privados), variáveis e constantes | | Sim | | Todos .h e .cpp | |
| - Métodos (com e sem retorno). | | Sim | | Todos .h e .cpp | |
| - Métodos (com retorno *const* e parâmetro *const*). | | Sim | | Todos .h e .cpp | |
| - Construtores (sem/com parâmetros) e destrutores | | Sim | | Todos .h e .cpp | |
| - Classe Principal. | | Sim | | Main.cpp & Principal.h/.cpp | |
| - Divisão em .h e .cpp. | | Sim | | No projeto. | |
| **2** | | **Relações de:** | | | | | |
| - Associação | |  | |  | |
| - Agregação via associação | |  | |  | |
| - Agregação propriamente dita. | |  | |  | |
| - Herança elementar. | |  | |  | |
| - Herança em diversos níveis. | |  | |  | |
| - Herança múltipla. | |  | |  | |
| **3** | | **Ponteiros, generalizações e exceções** | | | | | |
| - Operador *this* | |  | |  | |
| - Alocação de memória (*new* & *delete*) | |  | |  | |
| - Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via Templates) | |  | |  | |
| - Uso de Tratamento de Exceções | |  | |  | |
| **4** | | **Sobrecarga de:** | | | | | |
| - Construtoras e Métodos. | |  | |  | |
| - Operadores (2 tipos de operadores pelo menos) | |  | |  | |
| **Persistência de Objetos** | | | | | |
| - Texto via Arquivos de Fluxo | |  | |  | |
| - Binário | |  | |  | |
| **5** | | **Virtualidade:** | |  | |  | |
| - Métodos Virtuais. | |  | |  | |
| - Polimorfismo | |  | |  | |
| - Métodos Virtuais Puros / Classes Abstratas | |  | |  | |
| - Coesão e Desacoplamento | |  | |  | |
| **6** | | **Engenharia de Software** | | | | | |
| **- Levantamento de Requisitos Textualmente e Tabelado**  (Ou por meio equivalente como Diagrama de Requisitos da SysML) | |  | |  | |
| **- Levantamento de Casos de Uso e sua expressão por meio de Diagrama de Casos de Uso em *UML*** | |  | |  | |
| **- Diagrama de Classes em *UML*** | |  | |  | |
| **- Diagrama de Atividades em *UML*** | |  | |  | |
| **- Outros diagramas em *UML*, Diag. de Estados, Diag. de Seqüência, Diag. de Pacotes etc.**  - E/ou outros diagramas estabelecidos, como Diag. de Fluxo de Dados (DFD) ou Diag. em SysML (Diag. de Requisitos,. de Blocos etc). | |  | |  | |
| **7** | | Biblioteca Gráfica | | | | | |
| - Funcionalidades Elementares. | |  | |  | |
| - Funcionalidades Avançadas como:   * tratamento de colisões * duplo *buffer* * *especificar aqui outras* | |  | |  | |
| *Obs.: especificar quais funcionalidades.* | | | | | |
| **Interdisciplinaridades por meio da utilização de Conceitos de Matemática, Física etc** | | | | | |
| - Ensino Médio(especificar quais Conceitos aqui) | |  | |  | |
| - Ensino Superior(especificar quais Conceitos aqui) | |  | |  | |
| **8** | | **Organizadores:** | | | | | |
| Espaço de Nomes (*Namespace*) criada pelos autores. | |  | |  | |
| Classes aninhadas. | |  | |  | |
| **Estáticos e String:** | |  | |  | |
| Atributos estáticos e chamadas estáticas de métodos. | |  | |  | |
| A classe Pré-definida *String* ou equivalente. | |  | |  | |
| **9** | | **Standard Template Library *(STL)*** | | | | | |
| *Vector da STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores). | |  | |  | |
| *List da STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores). | |  | |  | |
| Pilhas, Filas, Bifilas, Filas de Prioridade, Conjuntos, Multi-Conjuntos, Mapas **ou** Multi-Mapas\*. | |  | |  | |
| \*Obs.: Listar apenas os utilizados | | | | | |
| **10** | | **Uso de Conceito Avançado no tocante a Orientação a Objetos.** | | | | | |
| **Ou Padrões de Projeto:** GOF  **Ou Programação orientada a eventos e visual:** Objetos gráficos como formulários, botões etc (Listar apenas os utilizados)  **Ou Programação concorrente:** Threads (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **ou** Win32API ou afins (com ou sem uso de Mutex, Semáforos, **ou** Troca de mensagens).  ***Ou******API de Comunicação em Rede:*** *Cliente Servidor.* | |  | |  | |

Os conceitos utilizados (ou não) para modelagem do jogo, conforme listados na tabela 2, são justificados na tabela 3.

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

|  |  |  |
| --- | --- | --- |
| No. | Conceitos | *Listar apenas os utilizados* Situação |
| 1 | **Elementares** | Classe e Objetos foram utilizado porque ...  Atributos, Variáveis e Constantes foram utilizados porque ...  . . . |
| 2 | **Relações** | Associação foi utilizado porque.... |
| ... | . . . | ... |
| 10 | **Uso de conceitos avançados.** | Padrões de Projeto não foi utilizado porque.... |

# COMPARAÇÃO ENTRE DESENVOLVIMENTOS

Nesta seção se deve apresentar uma comparação geral entre o desenvolvimento orientado a objetos e o desenvolvimento procedimental segundo a percepção dos desenvolvedores nos trabalhos realizados no âmbito da disciplina de Fundamentos de Programação 1 e Fundamentos de Programação 2 (ou até no âmbito de outras disciplinas se for pertinente).

# DISCUSSÃO E CONCLUSÕES

Esta seção deverá apresentar uma reflexão sobre o desenvolvimento e os resultados obtidos. Certamente uma conclusão bem elaborada ajuda na avaliação do Professor. Por sua vez, a avaliação do trabalho como um todo (pelo Professor) será baseada em:

* Quantidade e qualidade de conceitos utilizados na elaboração do *software*, envolvendo particularmente os bons princípios de Orientação Objetos, como organização, encapsulamento e reutilização, todos baseados no princípio de coesão e desacoplamento doutrinados em classe.
* Complexidade do problema e respectiva quantidade e qualidade de projeto e código o que, novamente, envolve a correção na aplicação dos princípios da Orientação a Objetos (i.e. coesão, desacoplamento, encapsulamento, organização e reutilização), o número de classes, o número e a forma de relacionamento, a complexidade algorítmica etc.
* Qualidade do trabalho escrito e da apresentação.

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 (sobretudo o primeiro e o segundo). Não obstante, esta porcentagem é relativa, pois (por exemplo) um item muito bem desenvolvido pode até compensar (um pouco) outro não tão bem desenvolvido em proporções diferentes deste referencial dado. Lembrar ainda que, além das *APS*, fazer seu projeto ser acompanhado pelo Professor (a medida que avança) é fundamental.

# CONSIDERAÇÕES PESSOAIS

Caso o aluno ou os alunos desejam expressar algum sentimento relativo, por exemplo, aos aprendizados e dificuldades encontrados, isto deve ser feita nesta seção opcional. Nesta seção pode-se até utilizar primeira pessoa, entretanto seria melhor a forma impessoal.

Neste sentido, **todas as demais seções devem ser escritas de forma impessoal** (o que significa não usar primeira e segunda pessoa singular ou plural – em suma não usar “eu” ou “nós” no texto). Ademais, salienta-se que o trabalho deve ser redigido linguagem correta, na forma culta e sem exageros poéticos, com textos não prolixos e bem encadeados.

# 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.

Tabela 4. Lista de Atividades e Responsáveis.

|  |  |
| --- | --- |
| Atividades. | Responsáveis |
| Levantamento de Requisitos | Fulano e Ciclano |
| Diagramas de Classes | Fulano e Ciclano |
| Programação em C++ | Fulano e Ciclano em geral |
| Implementação de *Template* | Fulano |
| Implementação da Persistência dos Objetos... | Ciclano |
| ... |  |
| Escrita do Trabalho | Mais Fulano que Ciclano |
| Revisão do Trabalho | Mais Ciclano que Fulano |

# AGRADECIMENTOS

Havendo agradecimentos de ordem profissional, estes deverão vir antes das referências.

Aqui se pode 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 09/09/2015, às 15:15:

<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