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

Alessandra Fernandes Lacerda, Enzo Trevisan Topanotti

[flacerdaale@gmail.com](mailto:flacerdaale@gmail.com), [enzotrevisantopanotti@gmail.com](mailto:enzotrevisantopanotti@gmail.com)

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

**Universidade Tecnológica Federal do Paraná - UTFPR**

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

**Resumo** – A disciplina de Técnicas de Programação do Curso Bacharelado em Engenharia da Computação possui como requisito o desenvolvimento de um software, no formato de um jogo de plataforma, para fins de aprendizado e aplicação dos métodos abordados durante o curso do disciplina, relativos principalmente à programação orientada a objetos em C++ e à modelagem de projetos. Neste trabalho, escolheu-se o jogo Innerspeaker, no qual o jogador enfrenta diversos tipos de inimigos em cenários que variam em cada fase. O jogo tem três fases que se diferenciam por dificuldades apresentadas para o jogador e por inimigos com características específicas. Para o desenvolvimento do jogo foram listados os requisitos e foi elaborado um diagrama de classes, usando como recurso a Linguagem de Modelagem Unificada (Unified Modeling Language - UML) para auxiliar na visualização de cada etapa necessária para o cumprimento de tais requisitos. Subsequentemente, utilizando-se a linguagem de programação C++, realizou-se o desenvolvimento do projeto, contemplando conceitos de Orientação a Objetos como Classe, Objeto, Relacionamento, Classe Abstrata, Biblioteca Padrão de Gabaritos (Standard Template Library - STL), Gabaritos, Polimorfismo e mais conceitos que serão relatados no decorrer do presente trabalho. Ao final da implementação, o jogo demonstrou sua funcionalidade de acordo com os requisitos e o projeto proposto inicialmente. Finalmente, destaca-se que o desenvolvimento descrito permitiu a consolidação dos tópicos estudados nessa disciplina, assim cumprindo com seu objetivo inicial.

**Palavras-chave ou Expressões-chave**: Trabalho Acadêmico Voltado à Implementação em C++, Desenvolvimento de Projeto em UML, Desenvolvimento de Jogo de Plataforma.

**Abstract** - The discipline of “Técnicas de Programação” of the Bachelor’s Degree in Computer Engineering has a requirement to develop a software, in the format of a platform game, for the purpose of learning and applying the methods approached during the course of the discipline, related mainly to object-oriented programming in C++ and project modeling. In this work, the game Innerspeaker was chosen, in which the player faces different types of enemies in scenarios that vary in each level. The game has three levels that are differentiated by difficulties presented for the player and by enemies with specific characteristics. For the development of the game, the requirements were listed and a class diagram was elaborated using the Unified Modeling Language(UML) to assist in the visualization of each step necessary to fulfill those requirements. Subsequently, using the C++ programming language, the development of the project was executed, contemplating concepts of Object Orientation such as Class, Object, Relationship, Abstract Class, Standard Template Library(STL), Templates, Polymorphism and more concepts that will be reported in the course of this paper. At the end of the implementation, the game demonstrated its functionality according to the requirements and the initially proposed project. Finally, it is highlighted that the described development allowed the consolidation of the topics studied in this discipline, thus fulfilling its initial objective.

**Key-words or Key-expressions**: Academic Work Focused on Implementation in C++, Project Development in UML, Platform Game Development.

**INTRODUÇÃO**

O presente relatório tem como objetivo descrever como foi o processo de desenvolvimento de um jogo de plataforma, sendo este um requisito da disciplina de Técnicas de Programação. O principal fim de tal desenvolvimento é a consolidação dos conceitos de Engenharia de *Software* e programação orientada a objetos vistos em sala de aula. Além disso, o trabalho tem como objetivo introduzir os alunos ao “mundo da engenharia”, sendo nos moldes de um projeto de Engenharia de *Software* completo, com planejamento, desenvolvimento, implementação e testes.

Como dito anteriormente, o objeto de estudo escolhido é um jogo de plataforma, que consiste em um jogo em duas dimensões no qual o jogador percorre plataformas pré definidas e enfrenta inimigos e obstáculos a fim de atingir o final da fase. No caso do presente trabalho, o jogo que foi desenvolvido teve como tema uma homenagem à banda Tame Impala e foi nomeado assim como o primeiro álbum da banda, *Innerspeaker*.

O desenvolvimento do projeto em si foi dado em várias etapas, começando pela compreensão dos requisitos e seguido pelo planejamento do que seria executado. Em seguida foi feito um primeiro diagrama de classe em UML para guiar e servir de base para o projeto a ser implementado. Toda a implementação foi dada aos moldes da programação orientada a objetos em C++ e os testes foram feitos a partir da execução do *software*.

Neste relatório, primeiramente será explicado o tema do jogo e como foi feito o projeto em si, com uma explicação breve da implementação do código e alguns desafios encontrados ao longo do desenvolvimento do *software*.

**EXPLICAÇÃO DO JOGO EM SI**

Toda a ideia do tema do jogo surgiu a partir de algumas músicas da banda Tame Impala, sendo que cada fase seria inspirada em uma música da banda, isso influenciando na trilha sonora (que seria a música que inspirou a fase), no cenário e nos inimigos presentes em cada fase. Na primeira fase, a música escolhida é *“Alter Ego”*, que trata de conflitos internos do eu lírico. O inimigo principal da fase é um alter ego do jogador, no caso foi usada a imagem de um fantasma do próprio jogador como seu alter ego para ficar mais clara a distinção entre personagens. Já na segunda fase, a música tema é *“Feels Like We Only Go Backwards”* e a influência dela é na mecânica da fase, que ocorre na direção contrária do eixo x em relação à fase anterior. A terceira fase, que é a fase do “chefão” é inspirada na música *“The Less I Know The Better”* e a influência é dado por meio do próprio “chefão”, que é Trevor, o rival do eu lírico da música (representado no vídeo oficial da música por um homem vestido de gorila). Além disso, todos os menus foram inspirados graficamente no vídeo oficial de *“The Less I Know The Better”*. Seguindo esse tema, foi considerado que a implementação de uma trilha sonora seria de extrema importância para a contextualização das fases e esse foi um ponto implementado com sucesso no projeto final.



Figura 1. Fase 1 - Alter Ego.



Figura 2. Fase 2 - Feels Like We Only Go Backwards



Figura 3. Fase 3 - “Chefão” - The Less I Know The Better.



Figura 4. Menu principal inspirado no vídeo de “The Less I Know The Better”.

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

No início do desenvolvimento deste projeto a dupla decidiu dar maior foco na engine do jogo, para assim criar uma base sólida que, de certa forma, facilitasse o cumprimento dos requisitos. Para isso foi utilizado o padrão de projeto Estado, baseado em alguns vídeos das referência 2, essa escolha tornou alguns problemas triviais, como o menu principal, o menu de fase e as fases em si. O funcionamento é bem simples, existe uma pilha de estados que cada estado criado é colocado nela, então o jogo inicia com um estado menu principal, nele podemos iniciar um estado fase ou um estado menu fase, assim por diante, o jogo fecha se a pilha de estados estiver vazia.

Para o gerenciamento de colisões entre os objetos do jogo, tendo em vista que foi escolhido trabalhar com a biblioteca gráfica SFML, foi utilizado como base a lógica da referência 1, que é uma colisão do tipo AABB, esse tipo de colisor trabalha com a diferença da distância entre a metade de um objeto com a do outro. Para ter maior controle da colisão foi usada a lógica de hitbox separadas do sprite, para visualizar essa hitbox durante o desenvolvimento do jogo ocorreu a manipulação algumas funcionalidades do SFML, como é possível visualizar na Figura 5. Outra funcionalidade que foi utilizada durante o desenvolvimento do jogo, foi imprimir a localização do mouse no prompt de comando, algo que é extremamente útil para “calibrar ” a posição de objetos estáticos como plataformas e botões.

Com relação a geração de inimigos e obstáculos aleatórios, sabendo que as plataformas não são randômicas, foram definidos intervalos onde eles podem ser inicializados. Para auxiliar uma função rand foi implementada, ela recebe um valor mínimo e máximo e gera um número aleatório entre eles. Ademais , seguindo o modelo UML exemplo, foram feitas uma lista de entidades (que são todos os objetos do jogo que utilizam diretamente a biblioteca gráfica) que utiliza template e listas de inimigos, plataformas e obstáculos que, para maior aprendizado do conteúdo, foram feitas respectivamente com *Vector, List e Set* da STL.



Figura 5. Hitbox visível.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **N**º. | **Requisitos Funcionais** | **Situação** | **Implementação** |
| **1** | Apresentar menu de opções aos usuários do Jogo. | Requisito previsto inicialmente e realizado. | Requisito cumprido via classe Estado Menu e seu respectivo objeto. |
| **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 Jogador. |
| **3** | Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas. | Requisito previsto inicialmente e realizado. | Requisito cumprido via classe Jogo e classe MenuDeFases. |
| **4** | Ter três tipos distintos de inimigos (o que pode incluir ‘Chefão’, vide abaixo). | Requisito previsto inicialmente e realizado. | Requisito cumprido, tendo em vista que na hierarquia da classe Personagens há quatro tipos de inimigos, sendo um o ‘Chefão’. |
| **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 cumprido, via função construtora da classe Fase, herdeira de EstadoFase, com o auxílio da função auxiliar rand, presente na classe mãe. |
| **6** | Ter inimigo “Chefão” na última fase. | Requisito previsto inicialmente e realizado | Requisito cumprido via classe Trevor. |
| **7** | Ter três tipos de obstáculos. | Requisito previsto inicialmente e realizado | Requisito cumprido, tendo em vista que na hierarquia da classe Obstáculo há três tipos de obstáculos. |
| **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 função construtora da classe Fase, herdeira de EstadoFase, com o auxílio da função auxiliar rand, presente na classe mãe. |
| **9** | Ter representação gráfica de cada instância. | Requisito previsto inicialmente e realizado. | Requisito cumprido via biblioteca gráfica SFML |
| **10** | Ter em cada fase um cenário de jogo com os obstáculos. | Requisito previsto inicialmente e realizado. | Requisito cumprido via classes Fase1, Fase2 e Fase3 herdeira de EstadoFase. |
| **11** | Gerenciar colisões entre jogador e inimigos. | Requisito previsto inicialmente e realizado. | Requisito cumprido via função gerenciaColisao pertencente ao EstadoFase. |
| **12** | Gerenciar colisões entre jogador e obstáculos. | Requisito previsto inicialmente e realizado. | Requisito cumprido via função gerenciaColisao pertencente ao EstadoFase. |
| **13** | Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (*ranking*). | Requisito previsto inicialmente e realizado. | Requisito cumprido na função atualizar() da classe Jogo. |
| **14** | Permitir Pausar o Jogo. | Requisito previsto inicialmente e realizado. | Requisito cumprido via classe MenuDePausa, o qual possui instância na classe EstadoFase |
| **15** | Permitir Salvar Jogada. | Requisito previsto inicialmente e não realizado. |  |

# **TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS**

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N.** | | Conceitos | | Uso | | Onde / O quê | |  |
|  | **1** | | **Elementares:** | | | | | |
|  | - Classes, objetos. &  - Atributos (privados), variáveis e constantes. &  - Métodos (com e sem retorno). | | Sim | | Todos .h e .cpp | |
|  | - Métodos (com retorno *const* e parâmetro *const*). &  - Construtores (sem/com parâmetros) e destrutores | | Sim | | Todos .h e .cpp | |
|  | - Classe Principal. | | Sim | | Main.cpp & Jogo.h/.cpp | |
|  | - Divisão em .h e .cpp. | | Sim | | No desenvolvimento como um todo. | |
|  | **2** | | **Relações de:** | | | | | |
|  | - Associação direcional. &  - Associação bidirecional. | | Sim | | Nas listas, por exemplo. | |
|  | - Agregação via associação. &  - Agregação propriamente dita. | | Sim | | A relação entre o Estado Fase e os Personagens | |
|  | - Herança elementar. &  - Herança em diversos níveis. | | Sim | | Nas classes que são herança da classe Entidade | |
|  | - Herança múltipla. | | Não | |  | |
|  | **3** | | **Ponteiros, generalizações e exceções** | | | | | |
|  | - Operador *this*. | | Sim | | Praticamente em todo código | |
|  | - Alocação de memória (*new* & *delete*). | | Sim | | Como quase todos os elementos de Fase são ponteiros para os tipos eles são alocados dinmicamente | |
|  | - Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via *Templates*). | | Sim | | Lista.h | |
|  | - Uso de Tratamento de Exceções (*try catch*). | | Não | |  | |
|  | **4** | | **Sobrecarga de:** | | | | | |
|  | - Construtoras e Métodos. | | Sim | | Classes que são heranças | |
|  | - Operadores (2 tipos de operadores pelo menos). | | Não | |  | |
|  | **Persistência de Objetos (via arquivo de texto ou binário)** | | | | | |
|  | **- Persistência de Objetos.** | | Sim | | Ranking | |
|  | **- Persistência de Relacionamento de Objetos.** | | Não | |  | |
|  | **5** | | **Virtualidade:** | |  | |  | |
|  | - Métodos Virtuais. | | Sim | | Utilizado em todas as classes filhas da classe Estado | |
|  | - Polimorfismo | | Sim | | Utilizada nas herdeiras da classe EstadoFase, Estado, Entidade, etc... | |
|  | - Métodos Virtuais Puros / Classes Abstratas | | Sim | | Utilizado na classe Estado | |
|  | - Coesão e Desacoplamento | | Sim | | No desenvolvimento como um todo. | |
|  | **6** | | **Organizadores e Estáticos** | | | | | |
|  | - Espaço de Nomes (*Namespace*) criada pelos autores. | | Não | |  | |
|  | - Classes aninhadas (*Nested*) criada pelos autores. | | Sim | | Classes  Animação e Elemento | |
|  | - Atributos estáticos e métodos estáticos. | | Parcial-mente | | Atributo relativo a altura da view | |
|  | - Uso extensivo de constante (*const*) parâmetro, retorno, método... | | Sim | | Utilizado na variável deltaTempo , como retorno da função estaPressionado da classe Botao | |
|  | **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 | | Utilizamos String como identificador nas classes Animacao e Botao. também foi utilizado Vector, List e Set do STL nas listas de Inimigos, Objetos e Plataformas. | |
|  | - Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa. | | Sim | | Utilizamos Mapa nas classes Animacao, Botao e praticamente todas as classes herdeiras de Estado. Para gerenciar os estados foi utilizada uma Pilha. | |
|  | **Programação concorrente** | | | | | |
|  | *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins*.* | | Não | |  | |
|  | *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetoscom uso de Mutex, Semáforos, **OU** Troca de mensagens. | | Não | |  | |
|  | **8** | | **Biblioteca Gráfica / Visual** | | | | | |
|  | - Funcionalidades Elementares. &  - Funcionalidades Avançadas como:   * tratamento de colisões * duplo *buffer* | | Sim | | Funcionalidades de Controle de Janela | |
|  | - Programação orientada e evento em algum ambiente gráfico.  **OU**  *- RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc). | | Sim | | Utilizamos a biblioteca gráfica SFML que funciona com eventos. | |
|  | **Interdisciplinaridades por meio da utilização de Conceitos de Matemática e/ou Física.** | | | | | |
|  | - Ensino Médio. | | Sim | | Utilizamos conceitos de Movimento Uniformemente variado da matéria de Física, na mecânica do pulo do jogador. | |
|  | - Ensino Superior*.* | | Sim | | Utilizamos conceitos de  Lista e Pilha, da disciplina de Estrutura de Dados 1. | |
|  | **9** | | **Engenharia de Software** | | | | | |
|  | - Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. & | | Sim | | Aplicado no projeto como um todo | |
|  | - Diagrama de Classes em *UML*. | | Sim | | Foi feito visando cumprir todos os requisitos. | |
|  | - Uso efetivo (quiçá) intensivo de padrões de projeto (particularmente GOF). | | Sim | | Foi Utilizado o padrão de projeto Estado. | |
|  | - Testes a luz da Tabela de Requisitos e do Diagrama de Classes. | | Sim | | Foram feitos testes com base no UML, montado a partir dos requisitos do projeto. | |
|  | **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 | | Via cópias manuais e backups na nuvem . | |
|  |  | | - Reuniões com o professor para acompanhamento do andamento do projeto. | | Sim | | Houveram 3 reuniões, nos dias:  06/05,11/06 e 19/06 | |
|  |  | | - Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. | | Não | |  | |
|  |  | | - Revisão do trabalho escrito de outra equipe e vice-versa. | | Não | |  | |

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

|  |  |  |
| --- | --- | --- |
| **No.** | **Conceitos** | **Situação** |
| **1** | **Elementares** | Classe e Objetos foram utilizado para seguir o conceito de coesão e desacoplamento. |
| **2** | **Relações** | As relações foram utilizadas para generalizar algumas classes, evitando repetições desnecessárias no código. |
| **3** | **Ponteiros, generalizações e exceções** | A maioria dos objetos foi criada dinamicamente para o uso mais eficiente da memória, para isso foram usados ponteiros.  Uma lista genérica foi criada com Template para haver maior flexibilidade nos seus métodos. |
| **4** | **Sobrecarga e persistência de Objetos** | A persistência de objetos por meio de arquivos foi feita no Ranking para facilitar o salvamento dos dados sem a criação de novas variáveis e/ou classes. |
| **5** | **Virtualidade** | Os métodos virtuais foram utilizados principalmente no personagens pois estes têm funções em comum mas que executam ações diferentes. |
| **6** | **Organizadores e Estáticos** | As classes aninhadas foram utilizadas para maior clareza em classes que seriam utilizadas para um único fim, como a classe Animacao, aninhada à classe Animador. |
| **7** | **STL e String OO** | Foram feitas listas usando set, vector e list para ampliar o aprendizado do uso destas funcionalidades da STL. |
| **8** | **Biblioteca Gráfica** | Foi utilizada a biblioteca gráfica SFML.d |
| **9** | **Engenharia de Software** | O uso do padrão de projetos Estado foi feito porque a equipe considerou esse método mais claro de implementar as fases e os menus do jogo. |
| **10** | **Execução de Projeto** | O controle de versões foi feito por cópias manuais pois a equipe considerou o método mais prático para troca de códigos entre si. |

# **REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS**

Comparando os desenvolvimentos do presente trabalho e dos trabalhos anteriores desenvolvidos na disciplina de Fundamentos de Programação 1(sem o conceito de orientação a objetos), pode-se perceber que a separação de um projeto em classes, aplicando o conceito de coesão e desacoplamento, torna o projeto como um todo mais claro, facilitando sua compreensão e possíveis resoluções de problemas durante seu desenvolvimento.

**DISCUSSÃO E CONCLUSÕES**

Ao fim do prazo de desenvolvimento, a maior parte dos requisitos foi cumprida. Alguns conceitos que seriam utilizados inicialmente não foram implementados por causa de atrasos no projeto devido a alguns problemas durante o desenvolvimento principalmente da colisão entre entidades. Também foram implementados itens que não eram requisitos para o trabalho, como a trilha sonora inserida em todas as etapas do jogo. Talvez se esse item tivesse sido deixado de lado, algum outro conceito pudesse ser implementado dentro do prazo. Além disso, a implementação do projeto como um todo foi de grande importância para consolidar os conceitos apresentados durante o curso da disciplina, devido aos requisitos principais para o jogo.

**DIVISÃO DO TRABALHO**

Tabela 4. Lista de Atividades e Responsáveis.

|  |  |
| --- | --- |
| **Atividades** | **Responsáveis** |
| **Levantamento de requisitos** | **Alessandra e Enzo** |
| **Diagrama de classes** | **Alessandra e Enzo** |
| **Programação em C++** | **Alessandra e Enzo** |
| **Implementação de Template e Listas** | **Alessandra** |
| **Implementação de Estados e Fases** | **Enzo** |
| **Implementação de Animações e Parte Gráfica(imagens, planos de fundo, imagens dos menus)** | **Alessandra** |
| **Implementação de Menus** | **Enzo** |
| **Implementação da lógica de colisões** | **Alessandra e Enzo** |
| **Implementação de músicas** | **Alessandra e Enzo** |
| **Escrita do Trabalho** | **Mais Alessandra que Enzo** |
| **Revisão do Trabalho** | **Mais Enzo que Alessandra** |

# 

# **REFERÊNCIAS CITADAS NO TEXTO**

# [1] VONCK, Hilze. SFML 2.4 For Beginners. <https://www.youtube.com/playlist?list=PL21OsoBLPpMOO6zyVlxZ4S4hwkY_SLRW9>

[2] SHARMA, Suraj. C++ SFML RPG**.**

<https://www.youtube.com/playlist?list=PL6xSOsbVA1ebkU66okpi-KViAO8_9DJKg>

**REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO**

[A] SFML C++ Tutorial - Music and Time. Acesso em 22/06/2019. Disponível em: <https://www.gamefromscratch.com/post/2015/11/05/SFML-CPP-Tutorial-Music-And-Time.aspx>

[B] sf::SoundSource Class Reference. Acesso em 22/06/2019. Disponível em: <https://www.sfml-dev.org/documentation/2.5.1/classsf_1_1SoundSource.php#ac43af72c98c077500b239bc75b812f03>

[C] std::normal\_distribution. Acesso em 20/06/2019. Disponível em: [http://www.cplusplus.com/reference/random/normal\_distribution](http://www.cplusplus.com/reference/random/normal_distribution/)

[D] C++ Program Code To Implement Singly Linked List Using Template Class. Acesso em 16/06/2019. Disponível em: <http://cncpp.divilabs.com/2013/12/c-program-code-to-implement-singly.html>