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

Daniel Augusto Pires de Castro, Francisco Cardoso Becheli
danielcastro@alunos.utfpr.edu.br, franciscobecheli@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
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 Ludi, no qual o jogador enfrenta inimigos em diferentes cenários. O jogo possui duas fases que se diferenciam por requerer diferentes habilidades do jogador, além de elementos – tais como entidades – com texturas realizadas por uma artista visual qualificada. Para o desenvolvimento, 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 genérico e previamente proposto. Subsequentemente – em Linguagem C++ – realizou-se o desenvolvimento, contemplando os conceitos usuais de Orientação a Objetos como Classe, Objeto e Relacionamento, bem como alguns conceitos avançados como Classe Abstrata, Polimorfismo, Gabaritos, Persistências de Objetos por Arquivos, Sobrecarga de Operadores e Biblioteca Padrão de Gabaritos (*Standard Template Library - STL*). Posteriormente, os testes e uso do jogo feitos pelos próprios desenvolvedores demonstraram sua funcionalidade conforme os requisitos e a modelagem elaborada. Portanto, salienta-se que tal desenvolvimento permitiu cumprir o objetivo de aprendizado visado.

Palavras-chave ou Expressões-chave: 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**.

Abstract - *The Programming Techniques course requires the development of a platform software, in game format, for learning goals of software engineering techniques, specifically object oriented programming in C++. For such, the game Ludi was chosen, in which the player faces enemies in different scenarios. The game has two levels which require the player different skills, besides elements — such as entities – with textures made by a qualified visual artist. For the development, the textually proposed requisites were considered– and a modeling (analysis and project) was elaborated via Class Diagram in Unified Modeling Language - UML using as a base a generic previously proposed diagram. Subsequently – in C++ programming language – the development– was realized , contemplating the usual Object Oriented concepts as Class, Object and Relationship, as well as advanced concepts such as Abstract Class, Polymorphism, Templates, Object Persistence through Files, Operator Overloading and Standard Template Library - STL. Posteriorly, the tests made by the developers themselves showed their functionality according to the requirements and modeling elaborated. Therefore, it should be noted that such development allowed to accomplish the targeted learning objective.*

Key-words or Key-expressions: 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 documento se trata de um relatório referente ao jogo "Ludi", desenvolvido em termos de código por D. A. P. de C. e F. C. B., e cujo processo artístico foi realizado por M. P. C.. Em especial, concebe-se a necessidade de registrar tais procedimentos, com o intuito de consolidar

Comentado [JMS1]: Sds,

Esta é a correção do trabalho referente à 2ª parcial. Salienta-se que as críticas são sempre construtivas, sendo os textos de críticas diretos apenas por motivo de ganho de tempo.

Cordialmente

Att.

Prof. J. M. Simão.

Texto/Apresentação:	2,5 / 3,0.
Conceitos:	2,9 / 3,5
Sistema:	3,5 / 3,5

Nota do trabalho como um todo: 8,9 / 10,0.

NOTA (100%): Daniel 8,9 / 10,0

NOTA (100%): Francisco 8,9 / 10,0

Formatado: Fonte: Negrito, Cor da fonte: Vermelho

Formatado: Inglês (Estados Unidos)

Formatado: Fonte: Negrito, Cor da fonte: Vermelho, Inglês (Estados Unidos)

Formatado: Inglês (Estados Unidos)

Formatado: Cor da fonte: Vermelho

o que foi executado durante essa conjuntura. Ademais, o trabalho em sua totalidade corresponde a um quesito avaliativo parcial da disciplina Técnicas de Programação do ~~p~~Professor J. M. S., no curso de Engenharia da Computação da UTFPR. Para tanto, serão apresentados tópicos relacionados ao conteúdo discutido pela matéria, à luz dos- bons princípios de projeto e POO quando contextualizados no âmbito de elaboração de um jogo.

Diversos fatores desempenham as diretrizes de ensino atribuídas a este sistema. Dentre eles, o trabalho em si objetiva essencialmente a concretização do aprendizado, no que abrange os conceitos estudados ao longo do período acadêmico. Dessa maneira, o jogo escolhido para ser implementado possui complexidade tal que permite utilizar diversos recursos da linguagem, sobretudo os ensinados em classe. Por consequência, estabelece-se a oportunidade de ampliar conceitos e aprimorar habilidades através do cumprimento dos requisitos solicitados.

O método utilizado no decurso deste trabalho fundamenta-se no ciclo clássico de Engenharia de Software, constituído por quatro etapas cada qual com sua devida relevância. As duas primeiras fases **abordam o planejamento** – com seu devido ambiente de UML – em que há o levantamento de requisitos, seguindo para a análise/projeto. Logo após, as duas subsequentes **retratam a execução** – com seu ambiente de POO para codificação –, as quais operam a implementação/codificação, finalizando em estágio de testes. Assim, é concretizado um regime consistente de estruturação de software, uma vez que as fases do ciclo são sistematizadas coerentemente.

Para as seções subsequentes, a explicação do jogo em si se transcorrerá preliminarmente, dando seguimento à parte de desenvolvimento do jogo na versão orientada a objetos. Outrossim, haverá a exposição de tabelas de conceitos utilizados, reflexões e considerações do projeto, além da divisão de tarefas entre os membros. Portanto, uma vez apresentado do que se trata o conteúdo presente neste documento, segue-se o secionamento dos tópicos alusivos à **dissertação** do projeto.

EXPLICAÇÃO DO JOGO EM SI

O jogo "Ludi" idealizou-se com uma temática voltada a princípios lúdicos – justificando seu nome – que simbolizam a imaginação de uma criança ao percorrer espaços de seu cotidiano. A partir disso, o objetivo do jogador é atravessar ambientes repletos de plataformas, obstáculos e inimigos em cada fase. Os dois personagens jogáveis consistem em uma fazendeira e um bruxo os quais – ao longo do percurso pré-estabelecido nos cenários – são capazes de se movimentar e disparar projéteis, com o intuito de enfrentar as adversidades em seus caminhos.

Figura 1. Fazendeira, Bruxo e seus projéteis, respectivamente.

Inicialmente, o jogo é aberto através de um menu que direciona opções tal como quantidade de jogadores, análise de pontuação, carregamento de jogo salvo e seleção de fases (as quais serão explanadas a seguir).

A fase do quintal é constituída por elementos orgânicos, representando um lugar extenso repleto de árvores e arbustos. Dentre os inimigos, estão inclusos pássaros voando horizontalmente – que atiram nozes em direção ao jogador – e monstros estáticos cobertos de folhas, os quais periodicamente se expandem ou contraem ao ficar vulneráveis. Os obstáculos compreendem plataformas conjuntamente com teias de aranha – que retardam o movimento do jogador – e espinhos (os quais provocam dano). No final do cenário, por conseguinte, há um

Formatado: Cor da fonte: Vermelho
Formatado: Fonte: Negrito, Cor da fonte: Vermelho
Formatado: Cor da fonte: Vermelho
Formatado: Fonte: Negrito, Cor da fonte: Vermelho
Formatado: Cor da fonte: Vermelho
Formatado: Cor da fonte: Vermelho
Formatado: Cor da fonte: Vermelho
Formatado: Fonte: Negrito, Cor da fonte: Vermelho
Formatado: Cor da fonte: Vermelho
Comentado [JMS2]: Figuras devem ser citadas explicitamente no texto.
Formatado: Cor da fonte: Vermelho
Formatado: Cor da fonte: Vermelho

obstáculo do tipo porta que ao ser colidido transporta os personagens jogáveis para a fase subsequente.

Figura 2. Fase Quintal, seus inimigos e obstáculos.

Posteriormente, a fase do quarto integra uma temática escura e sombria, formada por um ambiente fechado com brinquedos espalhados, cômodas e luminárias ao fundo (dando contraste ao que foi visto na anterior). Há fantasmas se movimentando em zigue e zague vertical – sem colidir com plataformas –, além dos mesmos monstros estáticos encontrados no quintal, porém com texturas diferentes. Assim como na fase anterior, plataformas e espinhos – de texturas diferentes –, além das mesmas teias de aranhas formam os obstáculos.

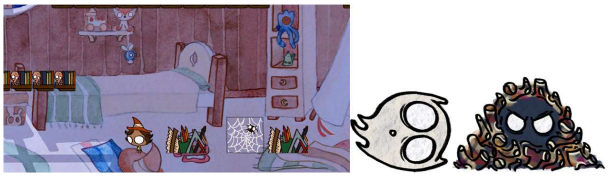

Figura 3. Fase Quarto, seus inimigos e obstáculos.

Para finalizar o caminho (e consequentemente o jogo) é necessário enfrentar o Bicho Papão – o inimigo chefão – que possui múltiplos estágios, atacando o jogador por meio de projéteis.

Figura 4. Chefão e seu projétil

Após toda essa jornada, é possível salvar pontuação – que foi incrementada a partir da neutralização de inimigos do jogo –, sair, ou até mesmo jogar novamente. Assim, todo esse sistema de fases e menus compõe elementos efetivamente aplicáveis para uma interatividade dentro da proposta esperada. Outrossim, ressalta-se o processo de textura e ilustração realizado por M. P. C. (a qual esteve presente na maior parte da criação de ideias) permitindo atribuir um caráter autêntico ao projeto. O jogo em si – quando referente a esses aspectos dissertados anteriormente – correspondeu, portanto, às expectativas dos desenvolvedores.

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Ao longo do seguimento, estruturam-se as ideias do jogo (tratadas na seção anterior) consoantemente às exigências designadas dentro do âmbito avaliativo do que a disciplina institui. Dessa maneira, o trabalho foi desenvolvido iniciando-se com a compreensão dos requisitos fornecidos para o desenvolvimento do jogo. Oportunamente, esse conjunto de requisitos está apresentado na Tabela 1 tal qual foram apresentados inicialmente, sendo que também consta a situação deles e onde foram contemplados. Para tanto, segue-se abaixo a tabela com as respectivas informações.

Formatado: Cor da fonte: Vermelho

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

N.	Requisitos Funcionais	Situação	Implementação
1	Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, ver colocação (*ranking*) de jogadores e demais opções pertinentes.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Menu e suas respectivas derivadas, cujos objetos foram implementados na classe principal Jogo, com suporte da SFML.
2	Permitir um ou dois jogadores com representação gráfica 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 inclusive via classe Jogador cujo objeto é agregado em Jogo.
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas, via menu, nas quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa.	Requisito previsto inicialmente e realizado.	Requisito foi realizado completamente porque a classe Fase deriva as classes Quintal e Quarto, que constituem diferentes cenários.
4	Ter pelo menos três tipos distintos de inimigos, cada qual com sua representação gráfica, sendo que ao menos um dos inimigos deve ser capaz de lançar projétil contra o(s) jogador(es) e um dos inimigos deve ser um 'Chefão'.	Requisito previsto inicialmente e realizado.	Requisito foi realizado completamente como se observa na hierarquia da classe Inimigo em que há quatro tipos de inimigos, sendo um deles o Pássaro que atira projéteis e o Bicho Papão de 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 3 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via função rand() no instante de inicializar inimigos em cada Fase.
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem.	Requisito previsto inicialmente e realizado.	Requisito foi realizado completamente como se observa na hierarquia da classe Obstáculo há quatro tipos de obstáculos, sendo

			um deles o Espinho que causa dano ao colidir.
7	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório de instâncias (i.e., objetos), sendo pelo menos 3 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via função rand() no instante de inicializar obstáculos em cada Fase.
8	Ter em cada fase um cenário de jogo constituído por obstáculos, sendo que parte deles seriam plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores.	Requisito previsto inicialmente e realizado.	Requisito cumprido ao inicializar plataformas e Cenário em cada Fase.
9	Gerenciar colisões entre jogador para com inimigos e seus projéteis, bem como entre jogador para com obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Gerenciador de Colisões que invoca método virtual colidir() em cada Personagem polimorficamente.
10	Permitir: (1) salvar nome do usuário, manter/salvar pontuação do jogador (incrementada via neutralização de inimigos) controlado pelo usuário e gerar lista de pontuação (*ranking*). **E** (2) Pausar e **Salvar** Jogada.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classes de Menus e funções de objetos ofstream e ifstream.
Total de requisitos funcionais apropriadamente realizados. ~~*(Cada tópico vale 10%, sendo que para ser contabilizado deve estar realizado efetivamente e não parcialmente)*~~			**100%** (cem por cento).

Após a apresentação dos requisitos com os acompanhamentos pertinentes, salienta-se que naturalmente foi elaborado um projeto do sistema. Em verdade, além dos requisitos já havia um modelo genérico de classes, à luz desses quesitos, que também serviu de base para o projeto em questão. À vista disso, o projeto se trata de um diagrama de classes detalhado em UML, o qual correlaciona as ideias convenientes à POO, estabelecendo classes com seus devidos relacionamentos, características e hierarquias. A implementação do jogo é descrita, por conseguinte, através dos componentes retratados nesses diagramas.

Analisando o diagrama em UML, a classe Jogo é tida como a principal (possuindo uma tela e um visualizador, ou *view*, administrados pela classe GerenciadorGrafico). Além disso, ressalta-se a classe abstrata Ente – oriunda etimologicamente do latim "ser" – que deriva três outras classes abstratas as quais atuam como pilares no sistema: Menu, Fase e Entidade. Dessa maneira, a classe principal contém objetos de classes derivadas delas: os tipos de menus (para fins de administração ou seleção de estados de jogo), os personagens jogáveis (Fazendeira e Bruxo), bem como as duas fases (Quintal e Quarto). Seguem-se nos próximos itens mais detalhes relacionados a essas classes pilares, tal como suas relações de herança com suas devidas propriedades.

Enfatiza-se a relevância da classe Entidade tratando-se de um eixo genérico que herda classes com suas respectivas representações gráficas (controladas pela classe GerenciadorGrafico). Assim, a derivada Personagem – caracterizada por se atualizar e possuir níveis específicos de complexidade – é outra abstrata a qual deriva as classes Jogador e Inimigo,

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

sendo este também classe base para tipos de inimigos diferentes. Já a entidade Obstáculo particulariza-se por conter derivadas não atualizáveis que interagem com os jogadores ao colidir, entretanto. Ademais, outras entidades isoladas consistem nas classes Cenario (que só desempenha a função de ser desenhada) e Projétil, o qual também se atualiza (não possuindo complexidades como a Personagem).

Com finalidades sistemáticas, os menus de jogo coordenam a ordem do que é inicializado de acordo com eventos e estados pré-definidos no Jogo. À vista disso, atribui-se uma associação bidirecional à relação entre Menu e Jogo. Dentre as funcionalidades mais significativas, destaca-se a opção de salvar jogo – obtida pela classe MenuPause –, administrar pontuações – através da MenuColocacao (editável ou não dependendo do acesso) –, e carregar jogo novo ou salvo (operado pela MenuPrincipal). O sistema de recuperação e salvamento é exercido dentro das fases, sendo que a classe Entidade possui um método virtual para salvar polimorficamente cada entidade necessária, o que permite restaurar informações contidas em arquivos (ao escolher carregar jogo).

Logo que é selecionado um novo jogo, inicializa-se individualmente a fase, conhecendo os jogadores criados pela classe principal. Cada entidade de fase é inicializada de acordo com o solicitado, sendo que todas são adicionadas em uma classe ListaEntidades – enquadrada consoante ao seu template Lista – a qual é responsável principalmente por atualizar, desenhar ou salvar cada entidade. Os personagens são inseridos em uma classe ListaPersonagens à parte, a qual está dentro da classe GerenciadorColisoes (que inclui a ListaEntidades também) conhecida pela própria fase. Os inimigos e obstáculos (excetuando chefão, porta e plataformas) são instanciados aleatoriamente, enquanto o jogador é posicionado no início da fase, finalizando a etapa de inicialização.

A classe GerenciadorColisoes calcula o módulo das distâncias entre os centros das entidades e a intersecção de coordenadas dos personagens e entidades para estabelecer a colisão deles de diferentes maneiras. Quando um personagem colide com a entidade, é chamado um método virtual das entidades que define sua reação à colisão. Uma plataforma colidindo em um personagem irá impedir com que ele a atravesse, enquanto um inimigo colidindo em um jogador irá decrementar a vida deste. Assim, estabelece-se uma lógica a qual permite simular efeitos físicos entre entidades.

Os projéteis são disparados por três personagens derivados da classe Atirador, sendo projéteis inimigos os disparados pelos Pássaros e Chefão e amigáveis os disparados pelos Jogadores, caracterizados por possuírem um ponteiro para a fase. O colidir do projétil verifica se um personagem amigável colidiu com um não amigável e vice e versa (com o intuito de decrementar a vida do personagem), desalocando a memória do projétil instantaneamente. Caso a vida de um personagem chegue a zero, ele também é desalocado. O controle da memória desalocada é efetuado pela Lista.

Por fim, cumpriu-se o desenvolvimento do jogo, que esteve constantemente sendo testado a cada implementação de funcionalidade ou conceito dado. É nesse âmbito que a quarta etapa do processo de Engenharia de Software esteve presente, uma vez que foram utilizadas ferramentas de debug a fim de conferir dados relativos aos elementos do sistema, além de inúmeras discussões com o monitor para direcionar a resolução dos problemas enfrentados. Isto posto, ressalta-se a importância da fase de testes para refinar imprecisões no código, **bem como avaliar métodos e mecanismos.**

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Fonte: Negrito, Cor da fonte: Vermelho

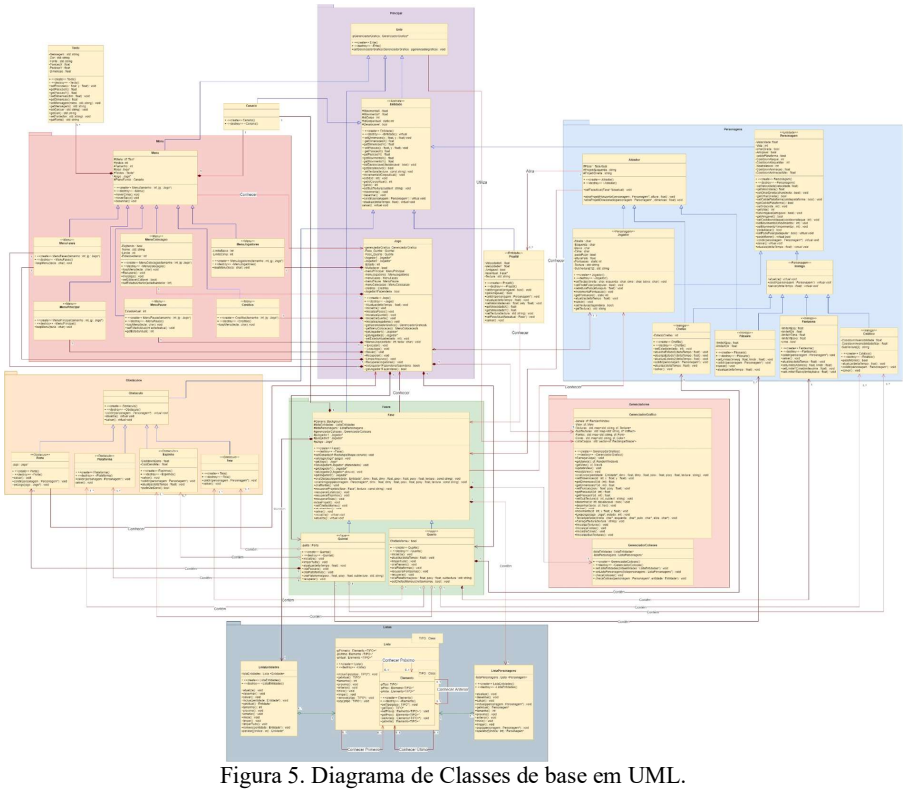

Figura 5. Diagrama de Classes de base em UML.

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Texto?

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	No desenvolvimento como um todo
	- 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	No desenvolvimento como um todo.

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

	- Agregação via associação. & - Agregação propriamente dita.	Sim	Classes Quarto e Quintal agregam objetos da classe Espinho
	- Herança elementar. & - Herança em diversos níveis.	Sim	Classe Entidade elementar, que chega a três níveis de herança.
	- Herança múltipla.	Sim	Classe Atirador
3	**Ponteiros, generalizações e exceções**		
	- Operador *this* para fins de relacionamento bidirecional.	Sim	Jogo.h/.cpp com Menu.h/.cpp
	- Alocação de memória (*new* & *delete*).	Sim	Inclusão e exclusão de entidades via classe ListaEntidades
	- Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via *Templates*).	Sim	Template de de Lista Encadeada adaptada da prova do professor[1]
	- Uso de Tratamento de Exceções (*try catch*).	Sim	Cálculo do versor da direção do projétil de pássaro para evitar divisão por 0.
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	**Meio**	**Método desenhar da classe GerenciadorGrafico**
	- Operadores (2 tipos de operadores pelo menos).	Sim	Classe gabarito Lista
	Persistência de Objetos (via arquivo de texto ou binário)		
	- Persistência de Objetos.	Sim	Salvamento polimórfico de cada entidade.
	- Persistência de Relacionamento de Objetos.	Sim	Salvamento polimórfico de cada entidade.
5	**Virtualidade:**		
	- Métodos Virtuais.	Sim	colidir(); movimentar(); ...
	- Polimorfismo	Sim	Colisão e movimento de Personagem; ...
	- Métodos Virtuais Puros / Classes Abstratas	Sim	Classe Entidade; ...
	- Coesão e Desacoplamento	Sim	No desenvolvimento como um todo.
6	**Organizadores e Estáticos**		
	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Organizando menus do jogo.
	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Classe ListaEntidades, adaptada de uma prova anterior do professor[1]
	- Atributos estáticos e métodos estáticos.	Sim	Id de entidade atual. Pontuação simultânea de jogadores.
	- Uso extensivo de constante (*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	Classe Texto utilizando strings. Vector da classe Texto em menus do jogo.

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Fonte: Negrito, Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	Map para associar strings a cores, texturas e fontes do SFML dentro do Gerenciador Gráfico.
	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 Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.	Não	
8	**Biblioteca Gráfica / Visual**		
	- Funcionalidades Elementares. & - Funcionalidades Avançadas como: ● tratamento de colisões ● duplo *buffer*	Sim	Classe GerenciadorGrafico trata janela do jogo. Classe GerenciadorColisoes trata as colisões.
	- Programação orientada e evento em algum ambiente gráfico. **OU** - *RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).	Sim	Classe GerenciadorGrafico trata os eventos.
	Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.		
	- Ensino Médio.	Sim	Implementação de gravidade da Cinemática, para reproduzir movimentos análogos à realidade para o Personagens.
	- Ensino Superior.	Sim	Módulo e versor de vetores da Geometria Analítica para direcionar projéteis à posição do Jogador.
9	**Engenharia de Software**		
	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	No desenvolvimento como um todo.
	- Diagrama de Classes em *UML*.	Sim	Plataforma Draw.io
	- Uso efetivo e intensivo de padrões de projeto *GOF*.	Não	
	- Testes à 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). & - Uso de alguma forma de cópia de segurança (backup).	Sim	Git e GitHub do repositório do D.A.P.C[2] e do F.C.B[3]
	- Reuniões com o professor para acompanhamento do andamento do projeto.	Sim	Reuniões dias 29/07, 03/08; 06/08 e 12/08. Total de reuniões: 4.
	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Sim	Reuniões com A. M. C: dias 15/07, 02/08, 05/08,

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

			06/08, 09/08, 10/08 e 13/08; Reunião com L. B. V: dia 13/07. Reuniões com M. K. L.: dias 04/08 e 11/08; Total de reuniões: 10
	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Pedro Foresti Leão e Carolina de Souza Fernandes
Total de conceitos apropriadamente utilizados. ~~(Cada grande tópico vale 10% do total de conceitos. Assim, por exemplo, caso se tenha feito metade de um tópico, então valeria 5%.)~~			~~**100%** (setenta por cento).~~ 82,5%

Texto?

Tabela 3. Lista de Justificativas para Conceitos Utilizados.

No.	Conceitos	Situação
1	**Elementares**	Classe, objetos, atributos e métodos foram utilizados para que o desenvolvimento respeitasse o Paradigma OO.
2	**Relações**	Herança e associações foram utilizadas para criar uma relação entre as classes.
3	**Ponteiros, generalizações e exceções**	A alocação de memória dinâmica foi utilizada para criar objetos em tempo de execução, como projéteis. O tratamento de exceções foi útil para prevenir erros, como a divisão por 0.
4	**Sobrecarga e Persistência**	A sobrecarga de operadores foi utilizada para auxiliar na simplicidade e clareza do código, enquanto a sobrecarga de métodos para permitir diferentes tratamentos para diferentes parâmetros passados para uma função.
5	**Virtualidade**	A virtualidade foi utilizada para tratar todos os objetos derivados de uma mesma classe de forma geral, a partir de apontamentos para tal, sendo todos tratados genericamente em listas.
6	**Organizadores e Estáticos**	Namespaces foram utilizados para melhor organizar o código e torná-lo mais compreensível.
7	**Standard Template Library *(STL)* e String OO**	Para melhor organizar o código e facilitar a alocação e desalocação de memória. Strings foram efetivadas para manipulação de textos, e atuaram em conjunto com Mapas.
8	**Biblioteca Gráfica / Visual**	A biblioteca gráfica foi utilizada na implementação visual do jogo, como a impressão de entidades na tela, além do tratamento de eventos.
9	**Engenharia de Software**	O ciclo da engenharia de software foi utilizado para melhor planejar o desenvolvimento, tornando-o mais organizado e poupando tempo.
10	**Execução de Projeto**	O versionador de arquivos foi utilizado para registrar o progresso do desenvolvimento e tornar fácil a

Formatado: Cor da fonte: Vermelho

		restauração do código em caso de erros graves. As reuniões com o professor foram úteis para mensurar os avanços do projeto, bem como o esclarecimento de dúvidas.

REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS

Um desenvolvimento procedimental, devido a sua **simplicidade de execução**, demandaria um grande grau de dificuldade quando se trata de implementações de código em escalas mais robustas e complexas, tal como foi no jogo. Por outro lado, um desenvolvimento voltado a POO é muito mais consistente com suas múltiplas funcionalidades, em virtude das características referentes às propriedades de organização, encapsulamento e reutilização. Ademais, o uso de classes auxilia na capacidade de abstração do código, já que a relação entre elas possui comportamentos análogos à realidade. Portanto, evidencia-se que um projeto realizado em prol de POO provou ser mais conveniente do que em uma versão procedimental.

Formatado: Fonte: Negrito, Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

Formatado: Cor da fonte: Vermelho

.

DISCUSSÃO E CONCLUSÕES

Dentre os resultados obtidos, é ressaltado que a capacidade de trabalho em equipe trouxe bons desempenhos. Isso se refere a questões alusivas à comunicação interpessoal e condução de ideias conjuntas, uma vez que é fundamental estabelecer uma relação conveniente entre os integrantes. Ademais, outro efeito alcançado constituiu o cumprimento das expectativas mediante a um período tão curto de atividades acadêmicas realizadas em grupo, as quais englobam uma densidade de assuntos abordados. Por isso, salienta-se a relevância da capacitação concernente à resiliência e disposição de tarefas.

Evidencia-se que este trabalho auxiliou no desenvolvimento de novas aptidões e qualificações aos membros da equipe. Além do mais, ao longo da execução houve avanços na performance de técnicas tal como a disciplina propõe. Dessa forma, foram alicerçadas experiências pertinentes em projetos as quais irão implicar na formação de competências não só na parte de POO mas também generalizadamente na área de computação, visto que há grande demanda vinculada às qualificações de cooperação e disposição de ofícios pré-designados. Por conseguinte, o caráter disciplinador relacionado a este trabalho será benéfico para os desenvolvedores no escopo profissional.

DIVISÃO DO TRABALHO

Tabela 4. Lista de Atividades e Responsáveis.

Atividades.	Responsáveis
Compreensão de Requisitos	Francisco e Daniel
Diagramas de Classes	Francisco e Daniel
Programação em C++	Francisco e Daniel
Implementação de *Template*	Francisco e Daniel
Implementação da Persistência dos Objetos...	Francisco e Daniel
Tratamento de Colisões	Mais Francisco que Daniel
Gerenciador Gráfico	Mais Daniel que Francisco
Menus	Francisco e Daniel
Inimigos	Mais Francisco que Daniel

Obstáculos	Francisco e Daniel
Jogadores	Mais Daniel que Francisco
Fases	Francisco e Daniel
Escrita do Trabalho	Mais Daniel que Francisco
Revisão do Trabalho	Mais Francisco e Daniel

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

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

AGRADECIMENTOS

Agradecimentos ao Prof. Dr. Jean M. Simão pelas reuniões e pelos conteúdos disponibilizado em sua página da disciplina[1]. Agradecemos também os monitores Augusto Mudrei Correia, Matheus Kunnen Ledesma e Lucas Butschkau Vidal pela atenção e ajuda, a Pedro Foresti Leão e Carolina Fernandes Souza pela revisão do trabalho e a Maira Pires de Castro pela criação da arte visual do projeto.

REFERÊNCIAS CITADAS NO TEXTO

[1] 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.

[2] Endereço para o repositório online de D.A.P.C. na plataforma GitHub. Disponível em: <https://github.com/daniapc/Jogo>

[3] Endereço para o repositório online de F.C.B. na plataforma GitHub. Disponível em: <https://github.com/ChicoCB/Jogo>

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A] Tutoriais para o SFML 2.5. Disponível em: <https://www.sfml-dev.org/tutorials/2.5/>