TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR: DESENVOLVIMENTO DE UM JOGO DE PLATAFORMA

Guilherme Gomes Barboza, Luan Carlos Klein
barbozaguilherme717@gmail.com, luanklein@utfpr.edu.br

Disciplina: **Técnicas de Programação – 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 - O jogo de plataforma Orc's Injustice foi desenvolvido para a disciplina de Técnicas de Programação da UTFPR, ministrada pelo professor Dr. Jean Simão. Têm por fim o aprendizado de técnicas de engenharia de software, particularmente de programação orientada a objetos em C++. O jogo consiste em 2 fases, nas quais um personagem controlado pelo jogador enfrenta inimigos em determinado cenário, e ao final enfrenta um boss, o inimigo final do jogo. Para o desenvolvimento do mesmo, utilizou-se o levantamento textual dos requisitos, elaborando assim a modelagem do sistema, utilizando como recurso o Diagrama de Classes e Linguagem de Modelagem Unificada (Unified Modeling Language - UML). Após isso, fez-se a implementação do código utilizando a linguagem C++, contemplando os conhecimentos doutrinados em aulas, dentre os quais se incluem os conceitos usuais de Orientação a Objetos como Classe, Objeto e Relacionamento, Polimorfismo, Gabaritos, Persistências de Objetos por Arquivos, Sobrecarga de Operadores e Biblioteca Padrão de Gabaritos (Standard Template Library - STL). Ao finalizar a implementação, os testes foram realizados e o jogo demonstrou sua funcionalidade dentro daquilo que foi proposto inicialmente. Com isso, ressalta-se que, com o desenvolvimento do jogo o objetivo de aprendizado visado foi concluído com êxito.

Palavras-chave ou Expressões-chave - Desenvolvimento de um jogo de plataforma em C++; Utilização de conceitos usuais em Orientação a Objetos; Programação orientada a eventos.

Abstract- The Orc's Injustice platform game was developed for the Programming Techniques discipline of UTFPR, taught by Professor Dr. Jean Simão. They aim to learn software engineering techniques, particularly object-oriented programming in C ++. The game consists of 2 stages, in which a player controlled character faces enemies in a certain scenario, and in the end faces a boss, the final enemy of the game.
For the development of the same, the textual survey of the requirements was used, thus elaborating the modeling of the system, using as a resource the Unified Modeling Language (UML) and Class Diagram.After this, the code was implemented using the C ++ language, which includes the indoctrinated knowledge in classes, which include the usual concepts of Object Orientation such as Class, Object and Relationship, Polymorphism, Templates, Object Persistences by Files , Operator Overload, and Standard Template Library (STL). At the end of the implementation, the tests were performed and the game demonstrated its functionality within what was initially proposed. With this, it is emphasized that, with the development of the game, the target learning objective was successfully completed.

Key-words or Key-expressions - Development of a platform game in C ++; Using Common Concepts in Object Orientation; Event-oriented programming.

INTRODUÇÃO

O seguinte trabalho foi realizado para a disciplina de Técnicas de Programação, da UTFPR, ministrada pelo professor Dr. Jean Simão, no segundo semestre de 2018. O objetivo de tal é a aplicação e o aprendizado de conceitos básico e avançados de Engenharia de Software. Com o desenvolvimento dele, busca-se aplicar os conhecimentos adquiridos em sala de aula em um projeto que simula o mundo real do mercado de trabalho.

O trabalho consiste no desenvolvimento de um jogo de plataforma, o qual deve cumprir uma série de requisitos específicos. O jogo desenvolvido pela dupla foi o Orc's Injustice, intitulado assim pelos próprios membros. Neste, o jogador controla um personagem principal (orc), e enfrenta diversos inimigos comuns (elfo, guerreiro e guerreira), e atravessa inúmeros obstáculos (poços de ácido, espinhos e serras), e ao fim, combate o "chefão" do jogo (um mago).

Para desenvolvimento de tal, foi utilizado o ciclo clássico da engenharia de Software de forma simplificada. Primeiramente foi feito o levantamento de requisitos, e a modelagem do software se deu via o diagrama de classes em UML(Unified Modeling Language). Após, a implementação se deu na linguagem C++, utilizando a biblioteca gráfica SFML. Ao fim da implementação, ocorreu a realização de teste, utilizando o software desenvolvido em questão, e a cada erro encontrado, ou requisito não cumprido, o software foi re-implementado, e por vezes, remodelado a fim de atender tais exigências.

O seguinte relatório está dividido de forma que se apresente inicialmente a explicação do jogo em si, que consiste em explicar como funciona o jogo, qual seu funcionamento e como ele está organizado. Subsequentemente são apresentados os requisitos, e como se deu o desenvolvimento do projeto. Após, são apresentados os conceitos utilizados, e os motivos pelos quais foram usados. Por fim, há as conclusões e resultados obtidos pelo grupo, e a divisão das tarefas dos trabalhos dentro do grupo.

EXPLICAÇÃO DO JOGO EM SI

A história que dá sentido ao jogo é: "Um mago enfeitiçou toda a humanidade, e todos os entes estão sob o comando dele. Porém, os únicos não enfeitiçados foram os orcs. Cabe então, ao guerreiro mais bravo dos orcs, enfrentar esse mago, e salvar toda a sua raça. O mago enfeitiçou elfos e humanos, e fez deles seu exército. Para conseguir salvar toda sua nação orciana, o bravo guerreiro deve enfrentar os inimigos, e encarar diversos obstáculos colocados pelo mago para tentar parar o orc".

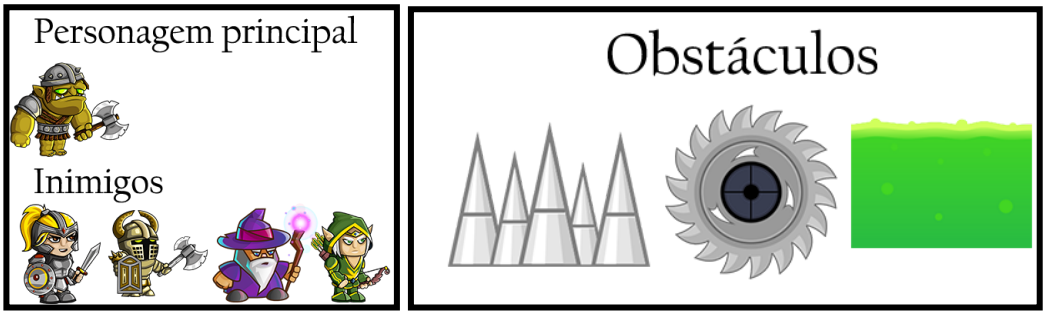

Imagem 1: Personagens e obstáculos

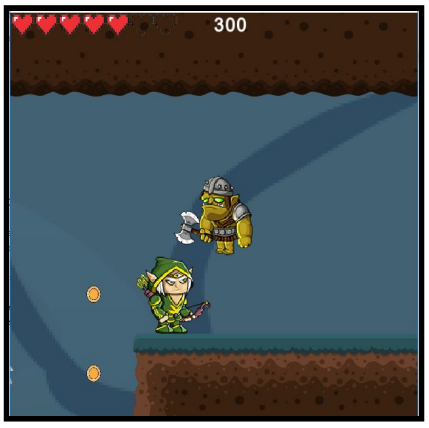

Imagem 2: Jogo em funcionamento

O jogo é constituído de 2 fases, ambas as quais são constituídas de diversos inimigos, que são, ou guerreiros, que dão dano através do contato, ou elfos, que são arqueiros e dão dano através de flechas. Os obstáculos são espinhos, poços de ácido (ao personagem ao tocar em algum desses, sofre dano total e morre), e serras, que dão um dano limitado. Ao final da segunda fase, o jogador enfrentará o chefão final, que é o mago. Ele tem diversos estágios de fúria, nos quais se alteram seu comportamento de movimentação, seus ataques e seu poder de dano.

Ao longo da fase, o usuário contará com alguns bônus, como moedas e vidas extras. Além de derrotar o boss o objetivo é somar a maior quantidade de pontos possíveis, para alcançar uma boa pontuação no ranking e ter seu nome na lista de top 10.

As teclas para movimentar o jogador e jogador 2 são, respectivamente, “a”,”w”,”d” e “←”,”↑”,”→”, para atacar são “x” e “k” e para pausar é “p”. Os botões do menu devem ser acionados via mouse.

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

1. Tabela requisitos

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, sendo que no último caso seria para que os dois joguem de maneira concomitante.	Requisito previsto inicialmente e realizado
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas.	Requisito previsto inicialmente e realizado
4	Ter três tipos distintos de inimigos (o que pode incluir ‘Chefão’, vide abaixo).	Requisito previsto inicialmente e realizado

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
6	Ter inimigo “Chefão” na última fase	Requisito previsto inicialmente e realizado
7	Ter três tipos de obstáculos	Requisito previsto inicialmente e realizado
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
9	Ter representação gráfica de cada instância.	Requisito previsto inicialmente e realizado
10	Ter em cada fase um cenário de jogo com os obstáculos.	Requisito previsto inicialmente e realizado
11	Gerenciar colisões entre jogador e inimigos.	Requisito previsto inicialmente e realizado
12	Gerenciar colisões entre jogador e obstáculos.	Requisito previsto inicialmente e realizado
13	Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (ranking).	Requisito previsto inicialmente e realizado
14	Permitir Pausar o Jogo	Requisito previsto inicialmente e realizado
15	Permitir Salvar Jogada.	Requisito previsto inicialmente e realizado
16	Ter bônus(moedas, vidas extras) ao longo do jogo.	Requisito previsto inicialmente e realizado

A biblioteca gráfica utilizada foi o SFML. O grupo optou por ela por ser tratar de uma biblioteca eficiente e de fácil uso. O material utilizado para o aprendizado desta foram videos provenientes da internet, em canais do youtube como por exemplo o “SFML 2.4 For Beginners”, de Hilze Vonck, sites de discussões, como o StackOverFlow e a própria página oficial de documentação do SFML.

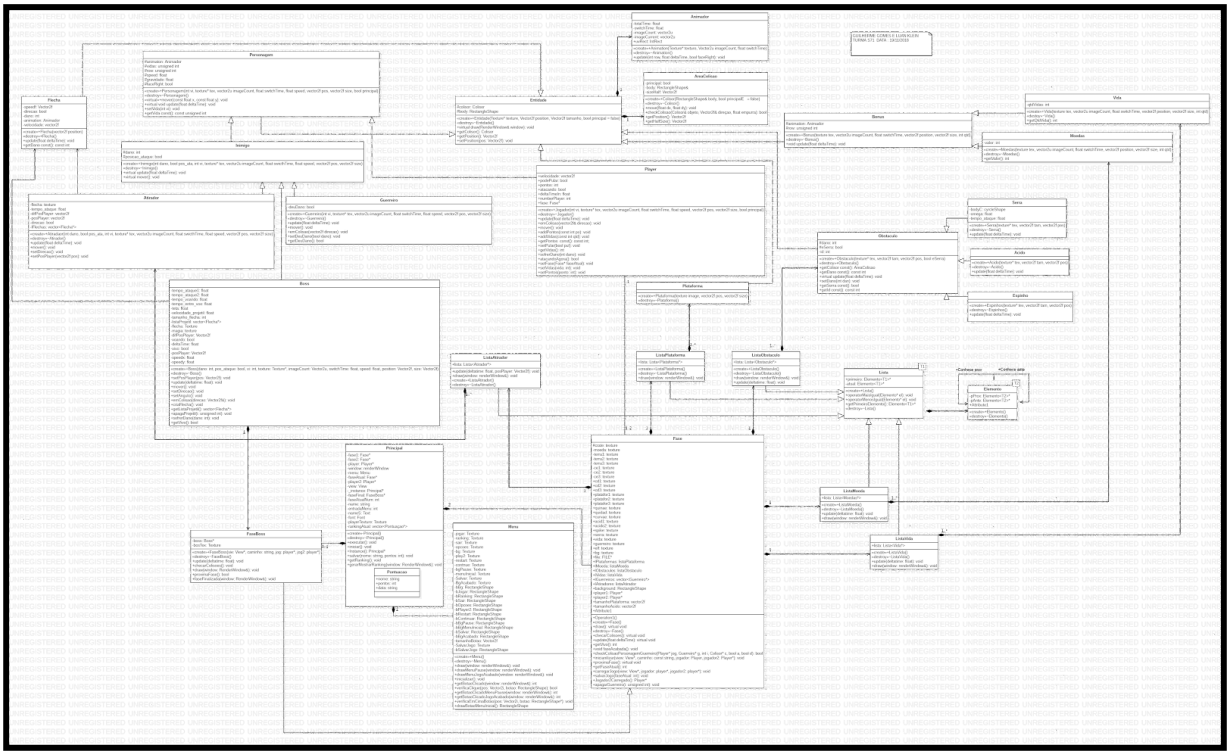

Imagem 3: Diagrama de classes referente ao jogo

Primeiramente foi desenvolvido uma classe AreaDeColisão, que será “o corpo” de cada objeto, e um animador, que é utilizado para atualizar a imagem do corpo. Ambas estão agregadas a uma classe Entidade, que contém os elementos básicos para todo corpo que será visível em tela para o usuário. A classe personagem, posteriormente criada, contempla tudo aquilo que os objetos móveis (Inimigos e player) têm em comum. Os inimigos se especializam em classes derivadas da classe Inimigo, pois apresentam métodos e atributos distintos.

Ao finalizar essa parte de personagens do jogo, foi desenvolvido as partes que iriam compor o cenário gráfico, que são o Bonus (posteriormente especializados em Moeda e Vidas, por apresentarem funcionalidades distintas), e Plataforma, ambas derivadas de Entidade. A classe template Lista, e as especializações (em listaDeGuerreiros, listaDePlataformas, listaDeMoeda etc) foram implementadas visando facilitar a organização e o desacoplamento dentro da classe Fase, cuja qual agregará todos os elementos que pertencem a parte gráfica do jogo (isso inclui lista de objetos, como por exemplo a listaDeMoeda). A classe Fase foi projetada para ser o mais genérica possível, permitindo assim, em momento futuro, a expansão de novas fases para o jogo, sem necessariamente fazer grandes alterações no código. A classe FaseBoss é uma especialização de Fase, que contém o boss, e ela a última fase do jogo.

A classe Menu é utilizada enquanto o usuário não está jogando, mas está com o jogo aberto, (isto é, está em pause, no menu inicial, acabou o jogo, etc). Ela foi pensada e planejada visando contemplar toda essa parte em que o usuário não está “usando” seu personagem. Foi desenvolvido a classe Principal, para realizar a “união” do Menu e da Fase, e para criar a lógica de execução por detrás do jogo.

Para a criação do Ranking, os dados já são salvos de maneira organizada, para que na hora da geração dele, em tempo de execução, menos processamento seja gasto. Para realizar a persistência de objetos relativa ao requisito Permitir Salvar Jogada, é salvo em um arquivo.dat todas as informações de posição sobre os inimigos, moedas, vidas e obstáculos ainda

existentes na fase atual e também as informações como pontos, vida e posição atual tanto do jogador 1 quando do jogador 2 (se este existir).

Os conceitos interdisciplinares que foram utilizadas são, de física, fórmulas de queda livre, movimento parabólico e MRU, e de matemática, como fórmulas de vetores, todas elas aprendidas no ensino médio, exceto, vetores que foram estudadas na graduação.

Para realizar a verificação da colisão, foi utilizado o método aprendido pelos alunos na disciplina de Geometria Analitica, que é o cálculo da distância Euclediana de dois pontos. Todos os personagens conhecem, por meio de um atributo estático, a gravidade do jogo. Por meio desta, é possível fazer com que todos os personagens tenham o mesmo tempo de queda nos pulos, e sofram a mesma influência da gravidade, semelhante a vida real.

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	Principal.h/.cpp
	- Divisão em .h e .cpp.	Sim	Em todas as classes do projeto exceto na template Lista
2	**Relações de:**		
	- Associação direcional. - Associação bidirecional.	Sim	Player conhecer Fase, e Fase conhecer Player
	- Agregação via associação - Agregação propriamente dita.	Sim	AreaColisao agregada a classe Entidade
	- Herança elementar. - Herança em diversos níveis.	Sim	Inimigo (e seus derivados), Player, Bonus (e seus derivados), Obstaculo (e seus derivados), Plataforma, para com Entidade
	- Herança múltipla.	Não	
3	**Ponteiros, generalizações e exceções**		

	- Operador this.	Sim	Fase.cpp e Player.cpp
	- Alocação de memória (new & delete).	Sim	Principal.cpp e Fase.cpp
	- Gabaritos/Templates criada/adaptados pelos autores (e.g. Listas Encadeadas via Templates).	Sim	Lista.h
	- Uso de Tratamento de Exceções (try catch).	Sim	Menu.cpp
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Todas as classes do jogo
	- Operadores (2 tipos de operadores pelo menos).	Sim	Lista.h (operadores += e -=)
	Persistência de Objetos (via arquivo de texto ou binário)	Sim	Principal.cpp. Persistência via arquivo de texto
	- Persistência de Objetos.	Sim	Fase.cpp, em salvar jogada
	-Persistência de Relacionamento de Objetos.	Sim	Fase.cpp, em salvar jogada
5	**Virtualidade:**		
	- Métodos Virtuais.	Sim	Entidade.cpp, destrutoras das classes base
	- Polimorfismo	Sim	Obstaculo.h e todas as suas classes filhas, usado em Fase.cpp
	- Métodos Virtuais Puros / Classes Abstratas	Sim	Personagem.h, Obstáculo.h, Inimigo.h
	- Coesão e Desacoplamento	Sim	Todos objetos
6	**Organizadores e Estáticos**		
	- Espaço de Nomes (Namespace) criada pelos autores.	Sim	Principal.h
	- Classes aninhadas (Nested) criada pelos autores.	Sim	Classe Pontuação aninhada a Principal.cpp
	- Atributos estáticos e métodos estáticos.	Sim	Personagem.cpp e Principal.cpp
	- Uso extensivo de constante (const) parâmetro, retorno, método..	Sim	Todos os getters e setters dos atributos das classes

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	Fase.h/.cpp, Principal.h/.cpp,
	-Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, MultiConjunto, Mapa ou Multi-Mapa	Não	
	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	Funcionalidades de gerar quadrados, settar posições e texturas, desenhar objetos
	- Programação orientada e evento em algum ambiente gráfico. OU - RAD – Rapid Application Development (Objetos gráficos como formulários, botões etc).	Sim	Principal.cpp, verificação de qual botão foi clicado
	Interdisciplinaridades por meio da utilização de Conceitos de Matemática, Física etc		
	- Ensino Médio.	Sim	Tempo de queda, velocidade de pulo (Física)
	- Ensino Superior.	Sim	Conceitos de distância de pontos. Uso de vetores (Geometria analitica)
9	**Engenharia de Software**		
	- Levantamento e rastreabilidade de cumprimento de requisitos.	Sim	Todo o código
	- Diagrama de Classes em UML.	Sim	Todo o código

	- Uso intensivo de padrões de projeto (GOF).	sim	Principal.cpp, uso do singleton
	- Testes a luz da Tabela de Requisitos	Sim	Todo o código
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 backups manuais, armazenamento físico (em 2 computadores), e em nuvem
	- Reuniões com o professor para acompanhamento do andamento do projeto.	Sim	3 (três) Dias: 19/10, 07/11 e 14/11,
	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto	Sim	3 (três): Dias: 05/11 e 12/11 com Ernesto Filho Dia: 05/11 com Victor Correa
	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Dupla César Vial e Eduardo Ribas

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

N°	**Conceito**	**Listar apenas os utilizados Situação**
1	**Elementares**	Utilizados para que o código se torne orientado a objetos, aumentando a coesão e desacoplamento.
2	**Relações de:**	A relação bidirecional foi usada para que o jogador conhecesse a fase em que ele está no momento do jogo e vice-versa. Heranças foram utilizadas para que classes que apresentam métodos e atributos iguais, herdarem de uma mesma classes esses, e apenas acrescentassem aquilo que era específico de cada uma.
3	**Ponteiros, generalizaçõe s e exceções**	O operador this foi utilizado pois no momento em que o usuário inicia uma nova fase, era necessário fazer com o que a Fase conhecesse o jogador, e o jogador a fase. Alocação de memória foi utilizado por que o tratamento do objeto sem necessariamente criar uma cópia, possibilitou que ele fosse tratado em diversos contextos diferentes, apenas referenciando seu local. Por exemplo, a Player dentro da classe principal; Gabaritos/Templates foram utilizados para a facilitação e generalização da classe lista, e das classes de diversos tipos diferentes. Uso de Tratamento de Exceções foi utilizado para evitar problemas em algumas partes dos códigos, que supostamente poderiam ocorrer erros inesperados.

4	**Sobrecarga de:**	Operador += para substituir a função incluir e operador -= para substituir a função apagar da template lista. A persistência de objeto foi utilizada tanto para salvar todas as posições dos inimigos, obstáculos, moedas, vidas e os players do jogo, quanto para salvar a pontuação de um jogador e seu nome.
5	**Virtualidade:**	Métodos Virtuais foram utilizados para que as classes derivadas umas das outras pudessem sobrescrever métodos com a mesma assinatura, ou, utilizar o método padrão. Polimorfismo foi utilizado para generalizar o funcionamento dos obstáculos, fazendo com que obstáculos diferentes, com métodos diferentes, mas assinaturas iguais, fossem considerados de maneira mais genérica. Métodos Virtuais Puros/Classes Abstratas foram utilizadas para obrigar as classes derivadas a sobrescrever métodos que seriam específicos de cada uma, para a possibilitar a utilização do polimorfismo. Coesão e Desacoplamento foi utilizado para facilitar o entendimento e a estrutura do código, além de melhorar a eficiência do mesmo.
6	**Organizadores e Estáticos**	Classes aninhadas (Nested) foram utilizadas para que apenas a classe que a tem possa utilizar-la. Por exemplo, apenas a Principal pode utilizar a Pontuação. Atributos e métodos estáticos foram utilizados porque todos os objetos de uma mesma classe, conhecessem o mesmo valor. No caso, todo personagem conheceria o valor da gravidade como 979. Uso extensivo de constante (const) como parâmetro, retorno e método foi utilizado para que evitar que alguns atributos apenas para leitura sejam modificados em lugares indevidos.
7	**Standard Template Library (STL) e String OO**	STL foi utilizada porque a lista guerreiro possuía problemas e substituimos para vector para um manuseio mais fácil e que desse menos problemas futuros.
8	**Biblioteca Gráfica / Visual**	Biblioteca gráfica foi utilizada porque era necessário haver representação gráfica dos objetos. A programação orientada a eventos foi utilizada porque era necessário tratar as diferentes opções de botões que se apresentam no menu. Conhecimentos de ensino médio e ensino superior foram necessários porque precisava tornar o jogo mais próximo da vida real, e realizar cálculos das colisões.
9	**Engenharia**	Levantamento e rastreabilidade de cumprimento de requisitos foi

	de Software	necessário porque era necessário saber o que deveria ser feito no software. Diagrama de Classes em UML foi utilizado porque era necessário estruturar o sistema, e entender como seriam as relações entre cada classe. Padrões de software foram utilizados pois apresentam facilidades na implementação e na compreensão/organização do código. Não foi realizado o uso intensivo, pois várias partes do código já estavam desenvolvidas quando o assunto foi estudado em aula, e por questões de tempo, a dupla optou por implementar padrões a partir daquele ponto do projeto, e voltar a implementar onde era possível, apenas se houvesse a disponibilidade. Testes a luz da Tabela de Requisitos foi utilizado para saber se aquilo já implementado está de acordo com o que foi planejado inicialmente.
10	**Execução de Projeto**	Controle de versão de modelos foi utilizado porque era necessário manter uma cópia de segurança caso ocorresse algum imprevisto no decorrer do desenvolvimento. Reuniões com o professor foram realizadas para auxiliar na construção do software e realizar correções. Reuniões com monitor da disciplina não foram realizadas as 10 solicitadas porque o grupo resolveu através de pesquisas e buscas na internet a solução de diversos problemas. As que foram realizadas foram úteis para receber uma opinião diferente sobre o jogo e seu desenvolvimento.

REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS

A programação orientada a objetos facilita muito a organização, o entendimento, e o controle do crescimento do software, se comparada o método procedimental. Caso o jogo fosse feito desta forma, seria muito complicado entender como as coisas funcionavam e se relacionam, além de que diversos trechos de código exatamente iguais teriam que ser reescritos, pela falta de um modo adequado e prático de reutiliza-lo. O método procedimental é muito útil para pequenos projetos, onde há poucas linhas de código, e uma lógica relativamente simples pode ser aplicada. Porém, quando um projeto é demasiadamente grande, apresentando vários requisitos e funcionalidades, e também a possibilidade de expansão, a orientação a objetos se faz fundamental, principalmente em projetos em equipe, onde cada um escreverá partes diferentes, mas que formarão um só ao final.

DISCUSSÃO E CONCLUSÕES

Levando-se em consideração os aspectos que foram englobados no projeto, que consistiu no desenvolvimento de um software de complexidade razoável, cujo desenvolvimento simulou uma situação real, pode-se afirmar que os resultados obtidos são satisfatórios ao comparar com os objetivos iniciais. Por intermédio desse projeto, os desenvolvedores aperfeiçoaram os conceitos aprendidos em sala de aula, aplicaram o

conhecimentos e desenvolveram a criatividade. Todos esses elementos foram fundamentais para a análise e desenvolvimento da solução, e serão de inestimável valia para projetos futuros, sejam eles acadêmicos ou profissionais, voltados para fins comerciais.

Cabe ressaltar que, além de aplicar os conhecimentos de sala de aula, também foi necessário pesquisar e buscar as informações, em livros e em sites da internet, como páginas de desenvolvedores, fóruns de discussões e documentações de ferramentas. Com isso, o senso de independência se eleva, fazendo com que o professor não seja a única fonte de informação, abrindo dessa forma, um grande leque de possibilidades.

Pela observação dos aspectos analisados, o grupo considera que os objetivos iniciais foram cumpridos, porém, estando ciente de que há muito ainda a aprender e que o software poderia ser melhorado em diversos aspectos. A orientação a objetos é um método muito poderoso e que é usado na maioria absoluta das grandes aplicações. Leva-se tempo até se adaptar e começar a utilizar as devidas formas nos lugares certos. Contudo, após essa prática, o grupo adquiriu certa experiência, e dessa forma, fez progressos relevantes na formação acadêmica e no entendimento de construção de softwares.

DIVISÃO DO TRABALHO

Atividade	**Responsável**
Levantamento de requisitos	Guilherme e Luan
Diagrama de classes	Guilherme e Luan (mais Luan)
Programação em C++	Guilherme e Luan
Implementação de template	Guilherme
Implementação de menu	Luan
Implementação fase	Guilherme e Luan(mais Guilherme)
Implementação principal	Guilherme e Luan
Implementação personagens	Guilherme e Luan
Implementação obstáculos	Guilherme e Luan (mais Guilherme)
Implementação persistência de objetos	Guilherme e Luan
Criação da fase(tilemap)	Guilherme
Implementação Ranking	Luan
Testes	Guilherme e Luan
Edição de imagens	Luan
Escrita do trabalho	Guilherme e Luan(mais Luan)
Revisão do trabalho escrito	César Vial e Eduardo Ribas

AGRADECIMENTOS

Agradecimento ao professor Doutor Jean Simão, pela ajuda na construção do software como um todo. Ao monitor Ernesto filho, pelos conselhos e dicas dados. Ao grupo de César Vial e Eduardo Ribas pela revisão do trabalho.

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

GOMILA, Laurent. **SFML and Visual studio.** Disponivel em: <https://www.sfml-dev.org/tutorials/2.5/start-vc.php>. Acesso em 19 de out de 2018.

SIMÃO, J. M. **Site das Disciplina de Fundamentos de Programação 2.** Dísponivel em: <http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/Fundamentos2.htm>. Acesso em 08 de set. de 2018.

VONCK, Hilze. **SFML 2.4 For Beginners**. Disponivel em: <https://www.youtube.com/playlist?list=PL21OsoBLPpMOO6zyVlxZ4S4hwkY_SLRW9>. Acesso em : 1 de out de 2018.