TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA UTFPR:
CANDLEBRIGHT: OMINOUS WHIP

José Henrique Ivanchechen, Raian Moretti
joseivanchechen@alunos.utfpr.edu.br, raian.moretti.eletro@gmail.com

Disciplina: Técnicas de Programação – CSE20 / S73 – Prof. Dr. Jean M. Simão Departamento Acadêmico de Informática – DAINF - Campus de Curitiba Curso Bacharelado em: Sistemas de Informação Universidade Tecnológica Federal do Paraná - UTFPR Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

Resumo. *Assim como exigido pela disciplina de Técnicas de Programação, desenvolvemos um jogo de plataforma com o intuito de aprimorar técnicas para programação orientada a objetos em C++ e também as devidas técnicas para Engenharia de Software. Para que isso fosse possível desenvolvemos um jogo de plataforma chamado Candlebright: Ominous Whip. Este jogo possui duas fases que podem ser jogadas por um ou dois jogadores, onde o(s) jogador(es) deve(m) combater tanto os inimigos quanto os obstáculos que aparecem. Para o desenvolvimento do mesmo, foram considerados os requisitos propostos e elaborado modelagem (análise e projeto), utilizando Diagrama de Classes em Linguagem de Modelagem Unificada (UML) usando como base um diagrama prévio proposto. Subsequentemente, em linguagem de programação C++, realizou-se o desenvolvimento que contemplou os conceitos usuais de Orientação a Objetos como Classes, Objetos e seus Relacionamentos, bem como alguns conceitos avançados como Classes Abstratas, Polimorfismo, Gabaritos, Persistências de Objetos por Arquivos, Sobrecarga de Operadores e Biblioteca Padrão de Gabaritos (STL). Depois da implementação, os testes e uso do jogo feitos pelos desenvolvedores demonstraram sua funcionalidade conforme os requisitos e o projeto elaborado. Por fim, salienta-se que o desenvolvimento em questão permitiu cumprir o objetivo de aprendizado visado.*

Palavras-chave*: Trabalho Acadêmico utilizando C++, Desenvolvimento de Jogo em C++ utilizando SFML, Programação Orientada a Objetos.*

Abstract. *As required by the Programming Techniques discipline, we developed a platform game with the aim of improving techniques for object oriented programming in C++ and also the proper techniques for Software Engineering. For that to be possible we developed a platform game called Candlebright: Ominous Whip. This game has two phases that can be played by one or two players, where the player(s) must fight both the enemies and the obstacles that appear. For the development of this game, the proposed requirements were considered and elaborated modeling (analysis and design) using Unified Modeling Language Class Diagram (UML) based on a previous proposed diagram. Subsequently, in C++ programming language, the development was made, which included the usual concepts of Object Orientation as Classes, Objects and their Relationships, as well*

as some advanced concepts such as Abstract Classes, Polymorphism, Templates, Object Persistence for Files, Overload of Operators and Standard Template Library (STL). After the implementation, the game developer's tests and use of the game demonstrated its functionality according to the requirements and the elaborated project. Lastly, it should be noted that the development in question has made it possible to meet the target learning objective.

***Key-words**: Academic Work using C++, Game Development in C++ using SFML, Object Oriented Programming.*

INTRODUÇÃO

O trabalho da disciplina de Técnicas de Programação resume-se em um software de plataforma, em forma de jogo, previamente selecionado com consentimento do professor e requisitos bem definidos.

Utilizamos de uma abordagem muito comum para desenvolvimento de softwares tendo início na modelagem do diagrama de classes em UML, posteriormente desenvolvendo em C++ orientada a objetos o que havíamos modelado, reiniciando o ciclo através da modelagem conforme se mostrou necessário e, por fim, executando os testes necessários no software.

Também neste documento, será apresentado do que se trata o jogo e como funciona, seu desenvolvimento, requisitos e conceitos aplicados, e algumas reflexões sobre o trabalho.

EXPLICAÇÃO DO JOGO

Ao iniciar o jogo é apresentado ao usuário uma tela de dimensões 600x400 onde são oferecidas as opções de Jogar ou Sair do jogo.

Figura 1. Menu Principal

Após selecionada a opção de Jogar são apresentadas ao usuário, novas opções para selecionar o mundo desejado(Figura 2). E logo após a seleção de jogadores (Figura 3).

Figura 2. Menu de Fases

Figura 3. Menu de Jogadores

Abaixo estão apresentadas ambas as fases do jogo:

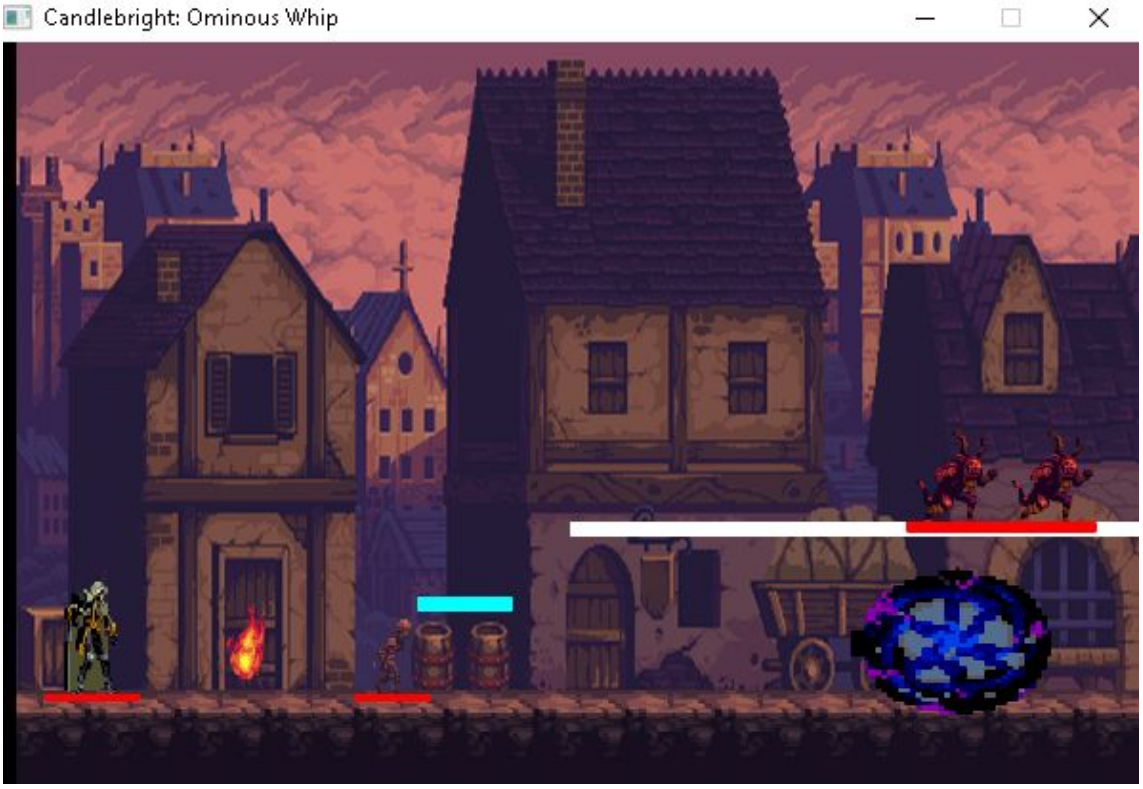

Figura 4. Cidade (Fase 1)

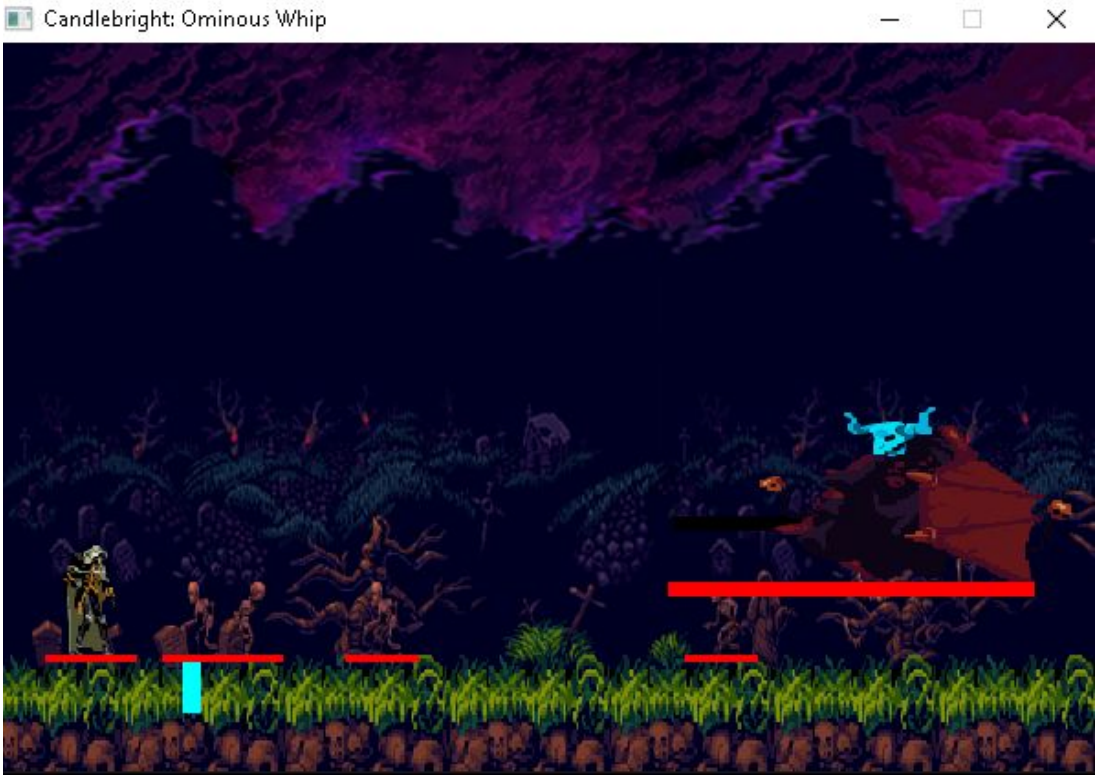

Figura 5. Cemitério (Fase 2)

DIAGRAMA DE CLASSES SIMPLIFICADO

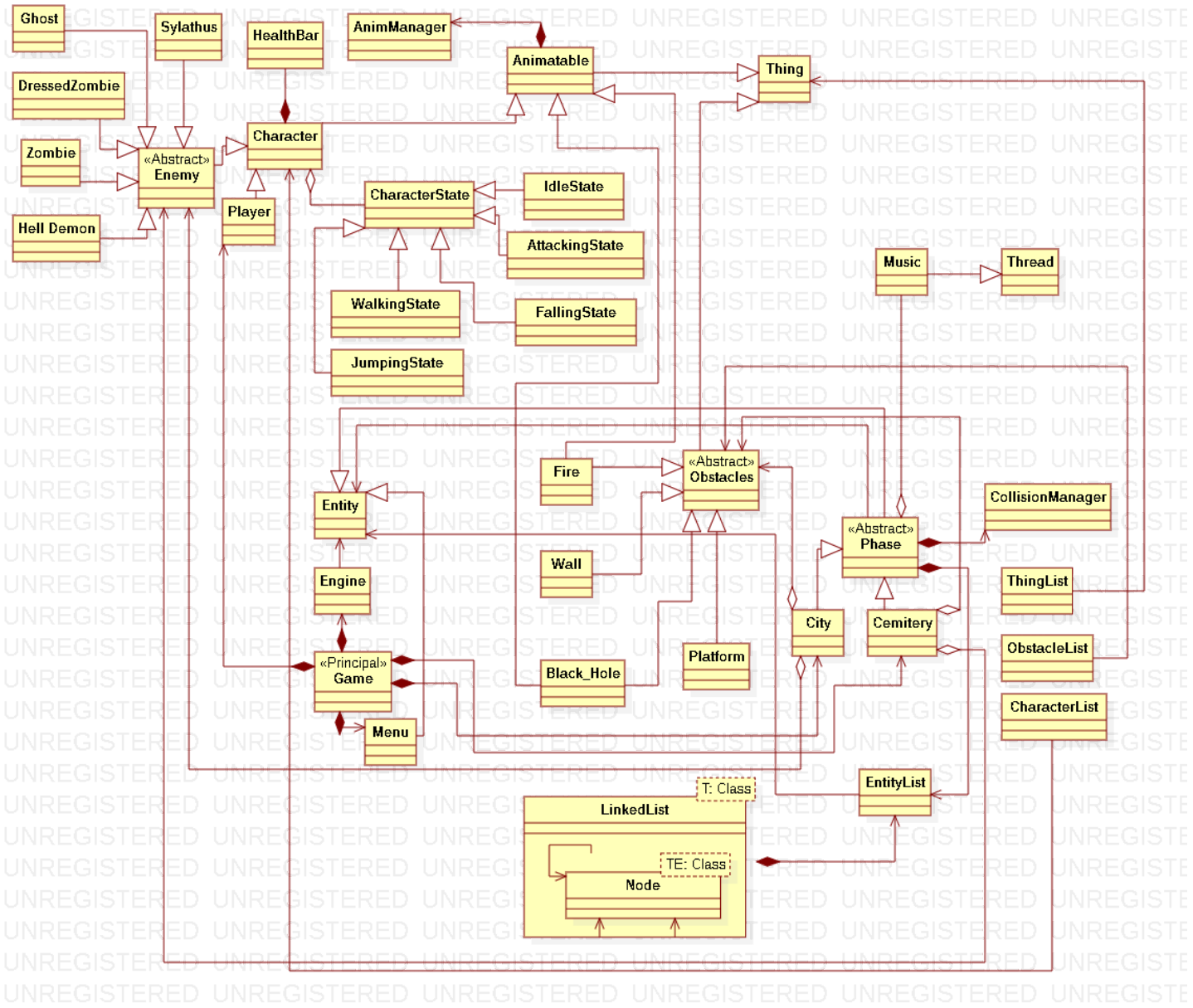

Figura 6. Diagrama de Classes Simplificado

DESENVOLVIMENTO DO JOGO ORIENTADO A OBJETOS

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

Nº	Requisito	Situação	Implementação
1	Apresentar menu de opções aos usuários do Jogo.	Requisito previsto inicialmente e realizado.	Requisito cumprido via classe Menu e seu respectivo objeto.
2	Permitir um ou dois jogadores aos	Requisito previsto	Requisito cumprido via classe Jogador,

	usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante.	**inicialmente e realizado.**	**cujos objetos são agregados em Game, podendo ser um ou dois efetivamente.**
3	**Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através das classes City e Cemitery, cujos objetos são agregados em Game, podendo ser um ou dois efetivamente.**
4	**Ter três tipos distintos de inimigos (o que pode incluir 'Chefão').**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através das classes Clothed_Zombie, Ghost, Hell_Demon, Sylathus e Zombie, com seus respectivos objetos sendo instanciados em Game e em cada Fase.**
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 através das construtoras de City e Cemitery, que instanciam os inimigos apenas uma vez. As instâncias aleatórias foram feitas através da função update dentro da classe Game.**
6	**Ter inimigo Chefão na última fase.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido via classe Sylathus, instanciada em Cemitery (segunda fase).**
7	**Ter três tipos de obstáculos.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através das classes Black_Hole, Fire e Wall.**
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 através das classes Fire, Wall, Platform e Black_Hole, as quais são instanciadas em City e Cemitery, também sendo instanciadas de forma aleatória na função update em Game.**
9	**Ter representação gráfica de cada instância.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido via classe abstrata Entity, que possui um método virtual puro chamado draw, responsável por desenhar a entidade na tela.**
10	**Ter em cada fase um cenário de jogo com os obstáculos.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através da criação de obstáculos dentro da construtora das classes City e Cemitery.**
11	**Gerenciar colisões entre jogador e inimigos.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através das classes CollisionManager e Phase.**

12	**Gerenciar colisões entre jogador e obstáculos.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através das classes CollisionManager e Phase.**
13	**Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (ranking).**	**Requisito previsto inicialmente e realizado parcialmente.**	**Tal requisito foi cumprido parcialmente, visto que não há representação gráfica do mesmo, tampouco cadastramento de dados do usuário e ordenação.**
14	**Permitir pausar o jogo.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido via função update dentro da classe Game.**
15	**Permitir salvar jogada.**	**Requisito previsto inicialmente e realizado.**	**Requisito cumprido através da função saveGame dentro da classe Game.**

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Tabela 2. Lista de Conceitos Utilizados e Não Utilizados

Nº	Conceito	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	Todos .h e .cpp
	- Divisão em .h e .cpp.	Sim	No desenvolvimento como um todo.
2	**Relações de:**		
	- Associação direcional. & - Associação bidirecional.	Sim	*Engine* com *Entity*. *Character* com *CharacterState*

	- Agregação via associação. & - Agregação propriamente dita.	Sim	*Fire* associado com *City*. *AnimManager* agregado em *Animatable*.
	- Herança elementar. & - Herança em diversos níveis.	Sim	*Thing* herda de *Entity*. *Obstacle* herda de *Thing*.
	- Herança múltipla.	Sim	Classe *Fire* e *Black_Hole*.
3	**Ponteiros, generalizações e exceções**		
	- Operador *this*.	Sim	Classe LinkedList
	- Alocação de memória (*new* & *delete*).	Sim	Classe LinkedList
	- Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via *Templates*).	Sim	Classe LinkedList
	- Uso de Tratamento de Exceções (*try catch*).	Sim	Classe AnimManager
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Classe *Platform* e *Menu*.
	- Operadores (2 tipos de operadores pelo menos).	Sim	Classe *EntityList*.
	Persistência de Objetos (via arquivo de texto ou binário)		
	- Persistência de Objetos.	Sim	Funções de *save* e *load* dentro da classe *Game*.
	- Persistência de Relacionamento de Objetos.	Não	
5	**Virtualidade:**		
	- Métodos Virtuais.	Sim	Classe Character
	- Polimorfismo.	Sim	Classe Entity

	- Métodos Virtuais Puros / Classes Abstratas.	Sim	Classe Entity
	- Coesão e Desacoplamento.	Sim	No desenvolvimento como um todo.
6	**Organizadores e Estáticos:**		
	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Namespaces correspondentes aos pacotes presentes no diagrama de classes.
	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Classe *LinkedList* contém a classe *Node*.
	- Atributos estáticos e métodos estáticos.	Sim	Classe *Player*.
	- 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 *AnimManager*. Classe *ThingsList*.
	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, MultiConjunto, Mapa OU Multi-Mapa.	Sim	CharacterList.hpp
	Programação concorrente		
	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins.	Sim	Classe *Thread* e *Music*.
	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **OU**	Não	

	Troca de mensagens.		
8	**Biblioteca Gráfica / Visual**		
	- Funcionalidades Elementares. & - Funcionalidades Avançadas como: • tratamento de colisões • duplo *buffer*	Sim	Desenhar uma entidade na tela e tratamento de colisão.
	- Programação orientada a evento em algum ambiente gráfico. **OU** - *RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).	Sim	Eventos no SFML.
	Interdisciplinaridades por meio da utilização de Conceitos de Matemática e/ou Física.		
	- Ensino Médio.	Sim	Distância entre pontos.
	- Ensino Superior.	Sim	Distância entre pontos.
9	**Engenharia de Software**		
	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	No desenvolvimento como um todo.
	- Diagrama de Classes em *UML*.	Sim	No desenvolvimento como um todo.
	- Uso efetivo (quiçá) intensivo de padrões de projeto (particularmente GOF).	Sim	Padrão de projeto *State* em *CharacterState*.
	- Testes a 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) **OU** manual (via cópias	Sim	(Qual?) git

	manuais). & - Uso de alguma forma de cópia de segurança (backup).		
	- Reuniões com o professor para acompanhamento do andamento do projeto.	Sim	(Quantas e onde?) Total de 4 reuniões, na sala do mesmo.
	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Não	
	- Revisão do trabalho escrito de outra equipe e vice-versa	Sim	(Qual?) Equipe do Bruno e Heitor.

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

Nº	**Conceito**	**Situação**
1	Elementares	Classe, Objetos e Atributos foram utilizados justamente por se tratarem do objetivo do trabalho, que é programação orientada a objetos.
2	Relações	Associações fortes e fracas foram utilizadas para gerenciar elementos dentro das classes. Heranças foram utilizadas por causa da alta quantidade de métodos e atributos repetidos.
3	Ponteiros, generalizações, exceções	Alterar valores de fora da objeto. Exceções para garantir que não haja problemas na execução do jogo. Gabaritos para escrever código genérico.
4	Sobrecarga	Deixar o código melhor de ser escrito.
5	Virtualidade	Polimorfismo (e todo o resto) necessário devido as várias heranças.
6	Organizadores e Estáticos	Necessários para evitar problemas durante o desenvolvimento
7	Standard Template Library (STL) e String OO.	Foram utilizadas vários elementos da STL para criar listas e vetores. Threads foram utilizadas para manter o jogo funcionando sem parar.
8	Biblioteca Gráfica / Visual	Utilizados para fazer o jogo funcionar.
9	Engenharia de Software	Todos conceitos foram utilizados para manter o planejamento do desenvolvimento do jogo.
10	Execução de Projeto	Todos conceitos foram utilizados para fazer com que não haja problemas adversos no desenvolvimento.

REFLEXÃO COMPARATIVA

É possível perceber a dificuldade que seria encontrada, caso o projeto fosse desenvolvido de forma procedural. A existência e utilização da orientação a objetos traz consigo a possibilidade de modularizar e desacoplar o código de tal forma que não seria possível atingir os mesmos resultados de forma procedimental. Portanto, percebemos que a organização e reaproveitamento do código é muito superior quando se é utilizada a orientação a objetos.

CONCLUSÃO

Conclui-se, portanto, que o desenvolvimento do software reforça nosso aprendizado no decorrer do curso e apresenta uma enorme diversidade de projetos que podem ser desenvolvidos através da linguagem C++ orientada a objetos, mostrando-se uma forte ferramenta para resolução de problemas.

DIVISÃO DO TRABALHO

Atividades	Responsáveis
Levantamento de Requisitos	José e Raian
Diagramas de Classes	Mais Raian que José
Programação em C++	Mais José que Raian
Escrita do Trabalho	Mais Raian que José
Revisão do Trabalho	Mais José que Raian

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO E BIBLIOGRAFIAS

SIMÃO, J. M. Site de Técnicas de Programação, Curitiba –PR, Brasil, Acessado em 23 de junho de 2019. http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2

BELISLE, P. Site de Músicas, Curitiba –PR, Brasil, Acessado em 23 de junho de 2019. https://soundcloud.com/pascalbelisle

ZUNO, L. Site de Pixel Art, Curitiba –PR, Brasil, Acessado em 23 de junho de 2019. https://ansimuz.itch.io/