TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR: *DINOSAUR'S ADVENTURE*

Otávio Pepe, Letícia Forte
otaviopepe@alunos.utfpr.edu.br, leticilialety@gmail.com

Disciplina: **Técnicas de Programação – CSE20** / S73 – 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 "DINOSAUR'S ADVENTURE" é um jogo de plataforma desenvolvido como parte da disciplina de Técnicas de Programação, com o objetivo de aplicar técnicas de engenharia de software e programação orientada a objetos em C++. O protagonista do jogo é um dinossauro corajoso que deve enfrentar inimigos e superar obstáculos em duas fases que se diferenciam por dificuldades para o jogador. Os inimigos incluem um pintinho, uma galinha que atira projéteis e um galo muito forte. Além disso, há três obstáculos no jogo: paredes a serem escaladas para poder atravessar o mapa, meteoros que causam dano e petróleo derramado que deixa o jogador lento e impede ele de pular. Através da modelagem por meio de um Diagrama de Classes em UML, os requisitos foram considerados e o desenvolvimento foi implementado em C++, incorporando conceitos avançados de orientação a objetos, como polimorfismo e sobrecarga de operadores. Testes e validações foram realizados para garantir o funcionamento adequado do jogo. O desenvolvimento do jogo "DINOSAUR'S ADVENTURE" atingiu com sucesso os objetivos de aprendizado propostos, proporcionando aos alunos uma experiência prática na aplicação das técnicas estudadas.

Palavras-chave ou Expressões-chave: Relatório para o Trabalho em Técnicas de Programação, Trabalho Acadêmico Voltado a Implementação em C++, Normas Internas para Elaboração de Trabalho, Jogo de plataforma 2D.

INTRODUÇÃO

Este trabalho é desenvolvido no contexto da disciplina de Técnicas de Programação, com o objetivo de aplicar os conhecimentos adquiridos sobre programação orientada a objetos em C++. O foco deste projeto é a implementação de um jogo de plataforma, intitulado "DINOSAUR'S ADVENTURE", que foi previamente acordado com o professor como objeto de estudo.

O método utilizado para o desenvolvimento do jogo segue o ciclo de Engenharia de Software, que compreende a compreensão dos requisitos, a modelagem do software por meio de diagramas de classes em UML, a implementação em C++ orientado a objetos e a realização de testes para garantir a qualidade do software.

Nas seções subsequentes deste documento, serão apresentados detalhes sobre o jogo em si, como sua temática, mecânicas e objetivos. Também serão abordados os requisitos funcionais, com uma tabela que indica o nível de cumprimento de cada um. Além disso, será apresentado o diagrama de classes UML que representa a estrutura do jogo, juntamente com uma tabela que mostra o cumprimento dos requisitos de conceitos aprendidos durante a disciplina, explicando a escolha e aplicação de cada conceito.

As seções subsequentes deste documento detalharão cada um desses aspectos, proporcionando uma visão abrangente do desenvolvimento do jogo "DINOSAUR'S ADVENTURE" no âmbito da disciplina de Técnicas de Programação.

EXPLICAÇÃO DO JOGO EM SI

"DINOSAUR'S ADVENTURE" é um jogo de plataforma 2D no qual Mamench e seu amigo Rex estão lutando contra a evolução para não serem extintos. Para isso eles podem se mover para todas as direções e saltar, assim como neutralizar seus inimigos jogando ovos neles, porém seus inimigos tentarão neutralizá-los também. Ao ser atingido por um ovo ou ao baterem em um inimigo os amigos perdem um pouco de vida, se a vida de um deles chegar a zero, o jogo acaba em derrota. O jogo inicia em um menu no qual o jogador pode escolher jogar com 1 ou 2 jogadores e qual nível deseja jogar.

Figura 1. Menu.

No nível 1 é possível encontrar 2 tipos de obstáculos, poças de petróleo que reduz a velocidade(proporcionalmente de maneira aleatória para cada poça) do respectivo jogador, assim como o impede de pular, e paredes a qual o jogador deve escalar(sendo a dificuldade de escalar aleatória para cada parede). Também é possível encontrar 2 tipos de inimigos, pintinhos que correm de um lado para o outro(sendo sua velocidade aleatória para cada) até que batam no player morrendo mas causando dano a ele. Assim como também existem galinhas que ficam paradas jogando ovos que causam dano ao entrarem em contato com o player. Esses 4 objetos são criados aleatoriamente pelo mapa. Mapa este que o jogador deve atravessar para chegar ao nível 2, podendo matar qualquer um dos inimigos ao se acertar um ovo.

Figura 2. Nível 1.

No nível 2 é possível encontrar 2 tipos de obstáculos, as paredes que já estavam presentes no nível 1 e meteoros que quando em contado com o player causam dano aleatório(dentro de um intervalo) à ele. Também é possível encontrar 2 tipos de inimigos, os pintinhos que já estavam no primeiro nível. Assim como também os galos, que são “Chefões”, os quais ao encostarem no player dão dano a ele, ao passarem por um pintinho ficam mais rápidos e seu dano aumenta por um intervalo de tempo. Esses 4 objetos são criados aleatoriamente pelo mapa. Mapa este que o jogador deve atravessar para ganhar o jogo, podendo matar qualquer um dos pintinhos ao se acertar um ovo e os galos ao se acertar dez.

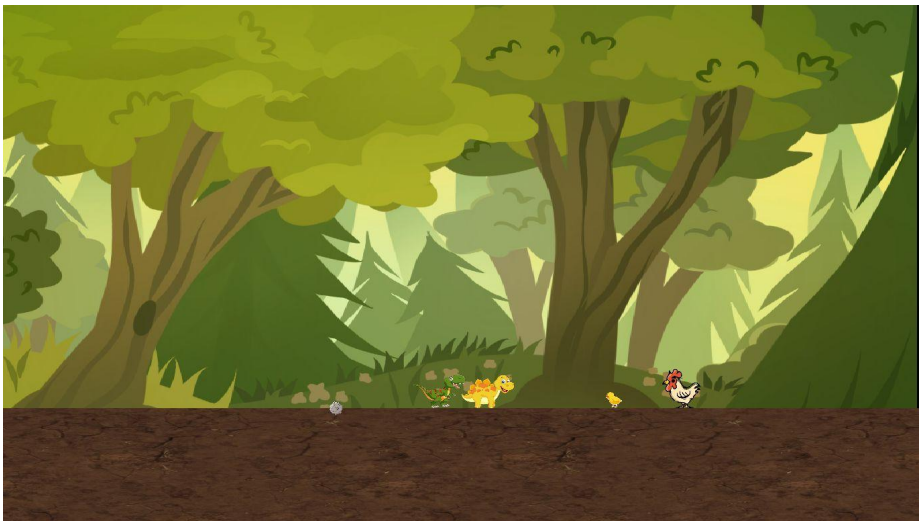

Figura 3. Nível 2.

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Para o desenvolvimento do jogo em c++ foram usados os conceitos de orientação a objetos, engenharia de software e modelagem(UML). Os quais seus requisitos funcionais, situação e implementação dos requisitos se seguem na tabela 1.

Tabela 1. Lista de Requisitos do Jogo e exemplos de 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 (previstas nos demais requisitos).	Requisito previsto inicialmente, mas realizado apenas **PARCIALMENTE** – faltou ainda a visualização de um ranking.	Requisito cumprido via classe StartMenu e seu respectivo objeto, 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 via classe Player, no qual através do menu pode-se escolher a quantidade de jogadores .
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 cumprido via classes LevelOne e LevelTwo, no qual o player pode neutralizar os inimigos com projéteis e através do menu pode-se escolher a fase, assim como ao se terminar a fase um passa-se automaticamente para a dois.
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 projetil contra o(s) jogador(es) e um dos inimigos dever ser um 'Chefão'.	Requisito previsto inicialmente e realizado.	Requisito realizado através do pacote Creature, sendo que na hierarquia de personagens há três tipos de inimigos: Chick, Chicken(o qual lança projétil) e Rooster('Chefão').
5	Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias (definindo um máximo) e sendo pelo menos 3 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito realizado através das classes Level(com Chick), LevelOne(com Chicken) e LevelTwo(com Rooster), no qual a geração de todos eles é aleatória, sendo o máximo e mínimo(sempre maior que 3) variáveis para cada .
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 realizado através do pacote Obstacles, sendo que na hierarquia de personagens há três tipos de obstáculos: Ground, Petroleum e Meteor(causador de dano).
7	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório (definindo um máximo) de instâncias (*i.e.*, objetos), sendo pelo menos 3 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito realizado através das classes Level(com Ground), LevelOne(com Petroleum) e LevelTwo(com Meteor), no qual a geração de todos eles é aleatória, sendo o máximo e mínimo(sempre

			maior que 3) variáveis para cada.
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.	Tal requisito foi realizado totalmente como se observa no pacote Levels, nos tipos LeveOne e LevelTwo a partir do Ground.h e Petroleum.h.
9	Gerenciar colisões entre jogador para com inimigos e seus projeteis, bem como entre jogador para com obstáculos. Ainda, todos eles devem sofrer o efeito de alguma ´gravidade´ no âmbito deste jogo de plataforma vertical e 2D.	Requisito previsto inicialmente e realizado.	Tal requisito foi realizado totalmente como se observa no pacote Managers, a partir do CollisionManager.h e do método move no pacote Entities, chamado por polimorfismo, que aplica o efeito de gravidade.
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/Recuperar** Jogada.	Requisito previsto e **NÃO** realizado.	Requisito NÃO realizado.
Total de requisitos funcionais apropriadamente realizados.			**80%** (oitenta por cento).

A partir de um diagrama previamente proposto pelo professor foi possível se inicializar o projeto, uma vez que já se tinha uma ideia geral de como iria funcionar. Depois esse diagrama foi modificado com o decorrer do tempo, sendo possível nele se observar as classes e suas relações, estando entre as principais a classe Game.

Nela, estão os objetos das classes níveis(classes essas que criam as estruturas e inimigos), no qual através do padrão de projeto State, Implementado pelas classes StateMachine(uma classe Singleton) e State, é possível navegar pelos níveis e menu. Sendo este responsável por além de acessar os níveis, escolher a quantidade de jogadores.

Outras classes importantes são as classes que fazem parte do pacote Managers. EventsManager(classe Singleton), GraphicsManager(classe Singleton) e CollisionManager, as quais servem para, respectivamente.

Gerenciar os eventos da janela (para fechá-la por exemplo) e do teclado (com ajuda do padrão de projeto Observer, implementado com as classes Input, Observer e suas classes derivadas) para dentre outras coisas, mover os jogadores e selecionar as opções do menu.

Criar a janela e nela mostrar as representações gráficas das classes(que possuírem representação) com ajuda na biblioteca gráfica SFML.

Gerenciar colisões entre os objetos do níveis assim como notificá-los de colisão e o responsável pela colisão para que esses objetos possam sofrer dano, não “afundar” sobre outros objetos e entre outras coisas.

Dentre os objetos dos níveis, temos o Player, Chicken, Chick e Roster. Objetos que sofrem a ação da gravidade implementada pelo método move com ajuda de polimorfismo. Método esse que considera a resistência do ar, seu cálculo é feito utilizando o conceito de força de arrasto, integral e derivada vistos na graduação e de MRUV visto no ensino médio. O método consiste simplificadamente em dividir o deslocamento em intervalos de tempo pequenos(derivada) e para esses intervalos aproximar o deslocamento como um MRUV,

sendo para cada intervalo de tempo recalculadas a aceleração(com base na gravidade e arrasto) e velocidade inicial, sendo no final feito uma "integral" para se ter o deslocamento.

Assim como também temos o Meteor, Ground e Petroleum, objetos que têm sua gravidade "anulada" por uma força normal fornecida pelo anteparo da superfície do planeta no qual o jogo se passa.

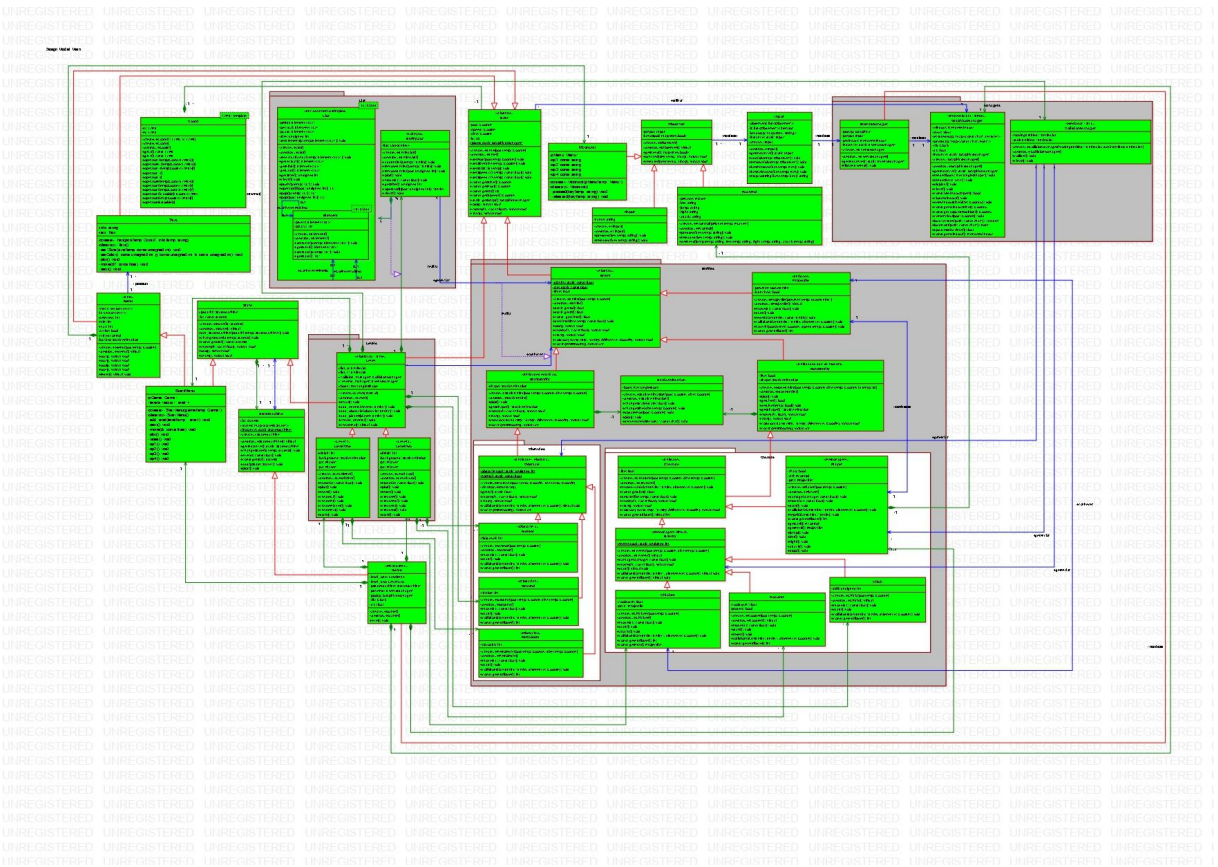

Figura 4. Diagrama de Classes de base em UML.

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

A seguir está a tabela 2, que mostra os conceitos vistos na matéria, assim como o status da sua realização e onde é possível os encontrá-los no projeto.

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

N.	Conceitos	Uso	Onde / O quê / Justificativa em uma linha
1	**Elementares:**		
1.1	- Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno).	Sim	Na maioria dos .h e .cpp.
1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Na maioria dos .h e .cpp, como nas classes nos *namespaces* Entities, Level e na classe State.
1.3	- Classe Principal.	Sim	Game.h/.cpp

1.4	- Divisão em .h e .cpp.	Sim	No desenvolvimento como um todo, como nas classes nos *namespaces* Creature e Obstacles.
2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional.	Sim	Em vários dos .h e .cpp, como nas classes nos *namespaces* Entities e Managers.
2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	Em vários dos .h e .cpp, como nas classes nos *namespaces* Entities e Levels.
2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	Em alguns dos .h e .cpp, como nas classes nos *namespaces* Entities e Creature.
2.4	- Herança múltipla.	Sim	Precisamente nos .h e .cpp, da classe Level e StartMenu.
3	**Ponteiros, generalizações e exceções**		
3.1	- Operador *this* para fins de relacionamento bidirecional.	Sim	Precisamente nos .h e .cpp, das classes State, Chicken e Player.
3.2	- Alocação de memória (*new* & *delete*).	Sim	Em alguns dos .h e .cpp, como nas classes List, LevelOne e Level.
3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).	Sim	Precisamente no .h das classes List e Coord.
3.4	- Uso de Tratamento de Exceções (*try catch*).	Não	Conceito não utilizado.
4	**Sobrecarga de:**		
4.1	- Construtoras e Métodos.	Sim	Precisamente nos .h , das classes Coord, List e EntityList.
4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).	Sim	Alguns, como o *operator*+ e o *operator*+= no Coord.h.
---	**Persistência de Objetos (via arquivo de texto ou binário)**		
4.3	- Persistência de Objetos.	Não	Conceito não utilizado.
4.4	- Persistência de Relacionamento de Objetos.	Não	Conceito não utilizado.
5	**Virtualidade:**		. . .
5.1	- Métodos Virtuais Usuais.	Sim	. . .
5.2	- Polimorfismo.	Sim	Em vários dos .h e .cpp, como nas classes nos *namespaces* Obstacles e Creature.
5.3	- Métodos Virtuais Puros / Classes Abstratas.	Sim	Precisamente nos .h e .cpp, das classes Ente e Entity.
5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.	Sim	Em alguns dos .h e .cpp, como nas classes nos *namespaces* Managers.
6	**Organizadores e Estáticos**		
6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Em alguns *namespaces*, como o namespace Entities, List e Managers.
6.2	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Precisamente na classe List.
6.3	- Atributos estáticos e métodos estáticos.	Sim	Em alguns dos .h e .cpp, como nas classes Entity e Ente.

6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	Em alguns dos .h e .cpp, como nas classes no namespace Creature e na classe State.
7	**Standard Template Library *(STL)* e String OO**		
7.1	- 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	Em alguns dos .h e .cpp, como nas classes Pcontrol, Input e StartMenu.
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	Em alguns dos .h e .cpp, como a utilização de mapa nas classes StateMachine e Observer.
---	**Programação concorrente**		
7.3	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins.	Não	Conceito não utilizado.
7.4	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.	Não	Conceito não utilizado.
8	**Biblioteca Gráfica / Visual**		
8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: ● tratamento de colisões ● duplo *buffer*	Sim	Precisamente nos .h e .cpp, das classes GraphicsManager e CollisionManager.
8.2	- Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive) em algum ambiente gráfico. **OU** - *RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).	Sim	Principalmente nos .h e .cpp, das classes EventsManager, Input, Observe, Pcontrol e Mcontrol .
---	**Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.**		
8.3	- Ensino Médio Efetivamente.	*Sim*	Vetores, sistema de coordenadas cartesiano, MRUV e entre outros.
8.4	- Ensino Superior Efetivamente.	*Sim*	Cálculo da força de arrasto, conceito de integral e derivada e entre outros.
9	**Engenharia de Software**		
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. **&**	*Sim*	Através da atualização contínua do UML e relatório, assim como através de reuniões/conversas com o professor e monitores.

9.2	- Diagrama de Classes em *UML*.	Sim	Utilizado durante todo o projeto, como apresentado ao professor nas reuniões.
9.3	- Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões.	Não	Uso somente de state, observer e singleton.
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	Durante a execução do projeto foram realizados diversos testes utilizando a tabela de requisitos e o UML.
10	**Execução de Projeto**		
10.1	- Controle de versão de modelos e códigos automatizado (via github e/ou afins). & - Uso de alguma forma de cópia de segurança (*i.e.*, *backup*).	Sim	Através do github, link: https://github.com/oaldsp/dinosaurs_adventure.
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	2 reuniões feitas nos dias 14/11 e 22/11.
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Otávio: 4 reuniões com os monitores nos dias 06/11(Murilo), 08/11(Ariel), 11/11(Ariel) e 13/11(Daniel). Letícia: 3 dias de cursos com o PETECO nos dias 10/10, 17/10, 07/11.
10.4	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Enzo Tacla e Daniel Sommer - S73;
Total de conceitos apropriadamente utilizados.			**85%** (oitenta e cinco por cento).

DISCUSSÃO E CONCLUSÕES

Princípios de programação orientada a objetos foram aplicados no desenvolvimento do projeto, resultando em um software mais organizado, modular e reutilizável. O uso adequado de classes, objetos e seus relacionamentos torna o código mais fácil de entender e manter. A modelagem em UML nos deu uma visão geral clara da estrutura do jogo e facilitou sua implementação em C++.

Analisando a quantidade e qualidade dos requisitos funcionais atendidos e a correta aplicação dos conceitos de orientação a objetos, foi possível verificar o bom desempenho do projeto. A coesão e a separabilidade foram respeitadas, resultando em classes bem definidas e bem encapsuladas. Além disso, a reutilização de código e as hierarquias de herança resultaram em um sistema mais flexível e extensível.Os encontros com o professor e os monitores proporcionaram troca de conhecimentos e conselhos, o que ajudou a aprimorar o trabalho. O desenvolvimento e a coordenação que resultaram dessas interações demonstraram um compromisso com a qualidade do projeto e a melhoria contínua.

DIVISÃO DO TRABALHO

A seguir a tabela 3, que mostra as atividades realizadas durante o projeto e seus respectivos responsáveis.

Tabela 3. Lista de Atividades e Responsáveis.

Atividades.	**Responsáveis**
Compreensão de Requisitos	Otávio e Letícia
Diagramas de Classes	Otávio e Letícia
Programação em C++	Otávio e Letícia
Implementação de *Template*	mais Otávio que Letícia
Implementação da Persistência dos Objetos...	Não realizado
Gerenciamento de eventos	mais Otávio que Letícia
Gerenciamento de colisões	Otávio e Letícia
Gerenciamento de gráfico	Otávio e Letícia
Menu	mais Otávio que Letícia
Níveis	mais Letícia que Otávio
Entidades	mais Otávio que Letícia
Escrita do Trabalho	Otávio e Letícia
Revisão do Trabalho	Otávio, Letícia, Enzo e Daniel.

- Otávio trabalhou em 100% das atividades ou as realizando ou colaborando nelas efetivamente.
- Letícia trabalhou em 60% das atividades ou as realizando ou colaborando nelas efetivamente.

AGRADECIMENTOS PROFISSIONAIS

Agradecimentos, ao professor Dr. Jean M. Simão pelo conteúdo ensinado e resolução de dúvidas sobre o projeto. Ao PETECO e seus membros pelo curso ofertado. Aos monitores e colegas que já realizaram a matéria pelas sugestoẽs de melhorias e resolução de dúvidas. Assim como Enzo e Daniel pela revisão do trabalho escrito.

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A] SIMÃO, J. M. Página de Internet do Prof. Simão., Curitiba – PR, Brasil, https://pessoal.dainf.ct.utfpr.edu.br/jeansimao/.

[B] BURDA. Tutorial Jogo SFML, Curitiba – PR, Brasil, https://youtube.com/playlist?list=PLSPev71NbUEBIQlT2QCd-gN6l_mNVw1cJ&feature=shared