TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR: *CAVEESCAPE++ – MODELO & ESPECIFICAÇÃO DO TRABALHO*

Lucas Maciel Ferreira, Wesley dos Santos Leite
lucasmacielferreira@alunos.utfpr.edu.br, wesleyleite@alunos.utfpr.edu.br

Disciplina: **Técnicas de Programação – ICSE20** / 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 – Em software do estilo plataforma no formato de um jogo, elaborado como requisito para a disciplina de Técnicas de Programação, desenvolveu-se um projeto com a intenção de aprimorar conhecimentos relativos à programação orientada a objetos, de modo a ser realizado em C++. Por meio do cumprimento de requisitos relativos à proposta pedida, criou-se o jogo CaveEscape++, em que é possível controlar até dois jogadores e atacar inimigos em um cenário, que diverge em duas fases diferentes, com obstáculos também presentes na construção de cada fase. Os requisitos foram propostos textualmente e também por modelagem de análise e projeto, sendo que este último foi construído em um diagrama de classes por meio da Linguagem de Modelagem unificada (Unified Modeling Language - UML). Por meio do desenvolvimento do jogo até as etapas em que estivesse implementado e devidamente testado, foi possível cumprir as funcionalidades à rigor dos requisitos, obtendo-se êxito no ganho de conhecimento, principalmente no que diz respeito à formação acadêmica e profissional, processo devido à aplicação prática de conteúdos adquiridos durante o semestre.

Palavras-chave ou Expressões-chave (máximo quatro itens, não excedendo três linhas)**:** Relatório para o trabalho de Técnicas de Programação, Desenvolvimento de jogo de plataforma, Aprimoramento acadêmico relativo à programação orientada a objetos, Aplicação prática de conteúdos de Técnicas de Programação.

INTRODUÇÃO

Em trabalho realizado na disciplina de Técnicas de Programação, desenvolvido como requisito fundamental para aprovação na mesma, como alicerce de aprimoramento de formação acadêmica e profissional no âmbito universitário.

Teve-se como foco a proposta implementada na forma de um jogo de plataforma, requerido pelo professor e estabelecido no plano de ensino e no site da disciplina[1].

Por meio do desenvolvimento em C++ de caráter voltado à programação orientada a objetos, aplicou-se o método do ciclo simplificado de Engenharia de Software, executando requisitos relacionados por sua compreensão, além de modelagem(análise e projeto) em UML por modelo dado e testes funcionais no sistema.

A seguir apresentam-se as questões relevantes referentes ao trabalho, incluindo sua concepção e implementação prática, de modo a englobar todos os requisitos e conceitos solicitados e gerando conclusões a respeito dos resultados finais dos mesmos.

EXPLICAÇÃO DO JOGO EM SI

O jogo criado por nossa dupla, denominado de CaveEscape++, tem como temática um ambiente de caverna, como o próprio nome sugere, além de personagens e obstáculos baseados em elementos medievais. Ao entrar no software, o usuário se depara com uma interface de menu, como o visualizado na figura 1, que é responsável por direcionar o mesmo

para dentro do jogo, verificar pontuações antigas ou até sair da aplicação, tudo conforme a escolha do usuário.

Figura 1. Menu inicial do jogo.

Como um jogo tradicional de plataforma, o objetivo do mesmo é atingir o fim de determinada fase, de modo que é necessário não ser neutralizado no processo, seja por meio dos inimigos ou dos obstáculos. A aplicação suporta até dois jogadores ao mesmo tempo, sendo ambos controlados pelo teclado. O jogo possui duas fases, que podem ser escolhidas também por menu adequado como na figura 2, sendo que cada uma delas possuem tipos de obstáculos e inimigos que divergem em tipo, além da geração aleatória dentro da própria fase com relação a sua construção, o que entrega uma experiência única ao jogador em cada entrada.

Figura 2. Menu de escolha de fases.

Como citado anteriormente, o jogo possui entidades medievais, sendo o jogador um cavaleiro, como visto nas figuras 3 e 4 (o segundo jogador tem cor diferente).

Figuras 3 e 4. Imagens dos jogadores.

De modo que o mesmo possui a habilidade de danificar inimigos, aumentando sua pontuação a cada ataque e avançando pela fase ao se movimentar, ressaltando-se que existem indicadores de vida e pontuação nos cantos superiores da tela. A figura 5 ilustra a estrutura da fase, com a presença de jogadores, inimigos e obstáculos.

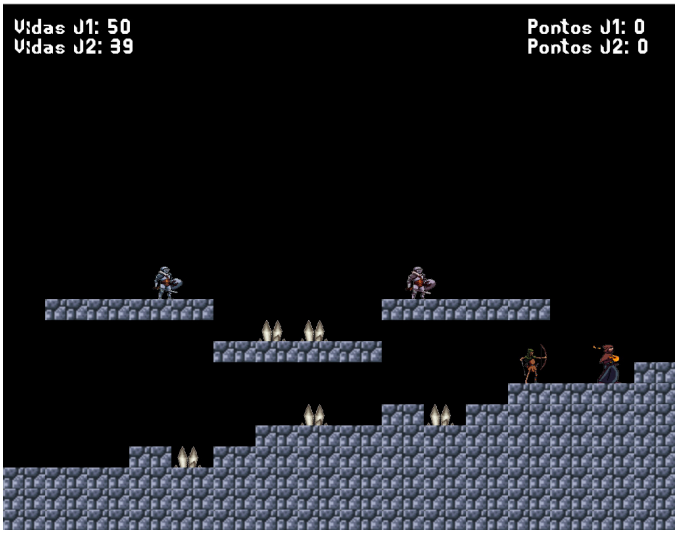

Figura 5. Exemplo de uma fase sendo jogada com diversas entidades.

Continuando a estrutura da fase, o jogo possui três tipos de inimigos diferentes. O lobisomem visto na figura 6 tem funções mais simples, de apenas danificar e pular ao colidir com um jogador, porém com a habilidade especial de aumentar a sua velocidade caso esteja próximo em contato com qualquer jogador.

Figura 6. Inimigo do tipo lobisomem.

Os outros dois inimigos, que são o esqueleto e o mago, possuem ataques de projéteis além do tradicional dano de corpo. O esqueleto da figura 7 fica parado ao se aproximar de algum jogador, atirando uma flecha, além de ser incapaz de pular em qualquer situação. Já o mago, como visto na figura 8, é descrito como o "chefão" do jogo, possuindo o poder de teletransportar, sendo que o faz depois de um tempo de repouso e com uma escolha aleatória de jogador. Além disso, o mago possui maior velocidade quando próximo de jogador, ademais atirando uma bola de fogo danificadora, de forma a combinar diversos elementos diferentes.

Figuras 7 e 8. O esqueleto na esquerda e o mago representado na direita

Com relação aos obstáculos (que são três também), existe a plataforma, que possui uma força que se contrapõe a gravidade, estabelecendo uma área de contato acima do solo. Mais diferenciados são os outros, como o caso do espinho ilustrado pela figura 9, que danifica jogadores e inimigos em contato (sendo que o subsequente de forma reduzida), além da gosma representada na figura 10, com a habilidade de reduzir a velocidade e desabilitar o pulo do jogador em contato, de forma que no inimigo apenas reduz a velocidade, porém de forma amena se comparada a do jogador.

Figura 9 e 10. Visualização do espinho na esquerda e da gosma na direita.

DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Como fundamentado na proposta de criação do jogo, existiram requisitos que precisavam ser cumpridos para a avaliação do trabalho, de modo que foram a prioridade na busca pela execução do mesmo, apresentando características intrínsecas ao processo de desenvolvimento orientado a objetos. A tabela 1 a seguir apresenta os dez requisitos exigidos para implementação do jogo, além da situação e especificações de sua implementação.

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 e realizado.	Requisito cumprido via namespace Menus, com as classes incluídas no pacote (incluindo a classe abstrata Menu) e seus respectivos objetos, além de criação de um Observer, tudo isso 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 cujos objetos são agregados dentro de cada fase (Pela classe Fase), podendo ser apenas um jogador ou dois, ficando a critério do usuário.
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.	Requisitos foram realizados pela construção das fases (presentes no namespace Fases), pela presença de menus (no namespace Menus) , pelas classes jogador e inimigo (além de suas derivações) e pelo GerenciadorEventos que permite as ações dos jogadores.
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	Tal requisito foi realizado pela criação das classes Lobisomem, Esqueleto e Mago (todos derivados de Inimigo). De modo que o esqueleto e o mago lançam projéteis e esse último também atua como “chefão”. Além de representação gráfica pelo GerenciadorGrafico.
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 cumprido pela classe Fase (dentro do namespaces Fases) , que utiliza do método de utilizar um Tilemap, que gera os inimigos a partir de um número aleatório (respeitando os limites) estabelecido na construtora de cada fase.
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um	Requisito previsto inicialmente e realizado.	Requisito cumprido pela criação das classes Plataforma, Gosma e

	causa dano em jogador se colidirem.		Espinho (derivados de Obstaculo). A classe Espinho danifica o jogador em colisão. Além de representação gráfica pelo GerenciadorGrafico.
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 cumprido pela classe Fase (dentro do namespaces Fases) , que utiliza do método de utilizar um Tilemap, que gera os obstáculos a partir de um número aleatório (respeitando os limites) estabelecido na construtora de 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 pela classe Fase (dentro do namespaces Fases com suas derivadas FasePrimeira e FaseSegunda) , que utiliza do método de utilizar um Tilemap, que gera o cenário com jogadores, inimigos e obstáculos.
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.	Requisito cumprido pela classe GerenciadorColisoes(presente no namespace Gerenciadores). O método que implementa gravidade está dentro da classe Entidade .
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 inicialmente e parcialmente realizado.	Faltou implementar o salvar e recuperar jogada.
Total de requisitos funcionais apropriadamente realizados. *(Cada tópico vale 10%, sendo que para ser contabilizado deve estar realizado efetivamente e não parcialmente)*			**90%** (noventa por cento).

Pode-se continuar discorrendo sobre as implementações das classes dentro do programa. De modo que foi criado um diagrama de classes em UML para orientar e guiar os rumos do funcionamento do sistema, como o da figura 11(esse pode ser melhor visualizado no arquivo separado enviado por email). Para se gerir de forma mais desacoplada o sistema, o namespace Gerenciadores engloba diversas funções relacionados às colisões, eventos e visualização gráfica do programa (sendo essa feita pela biblioteca da SFML). De forma geral as Entidades do jogo seguem uma estrutura polimórfica, de modo que as classes abstratas Obstaculo e Inimigo derivam em diversos elementos do jogo. As fases são construídas pelo o método de

utilizar o tilemap, conseguindo estabelecer agregação com os objetos (gerando então jogadores, obstáculos, inimigos e etc). Alguns observadores foram criados para estabelecer padrões de projeto e melhorar o desacoplamento do sistema. Além disso, o namespace Menus representa a parte que será mostrada e terá interação com o usuário.

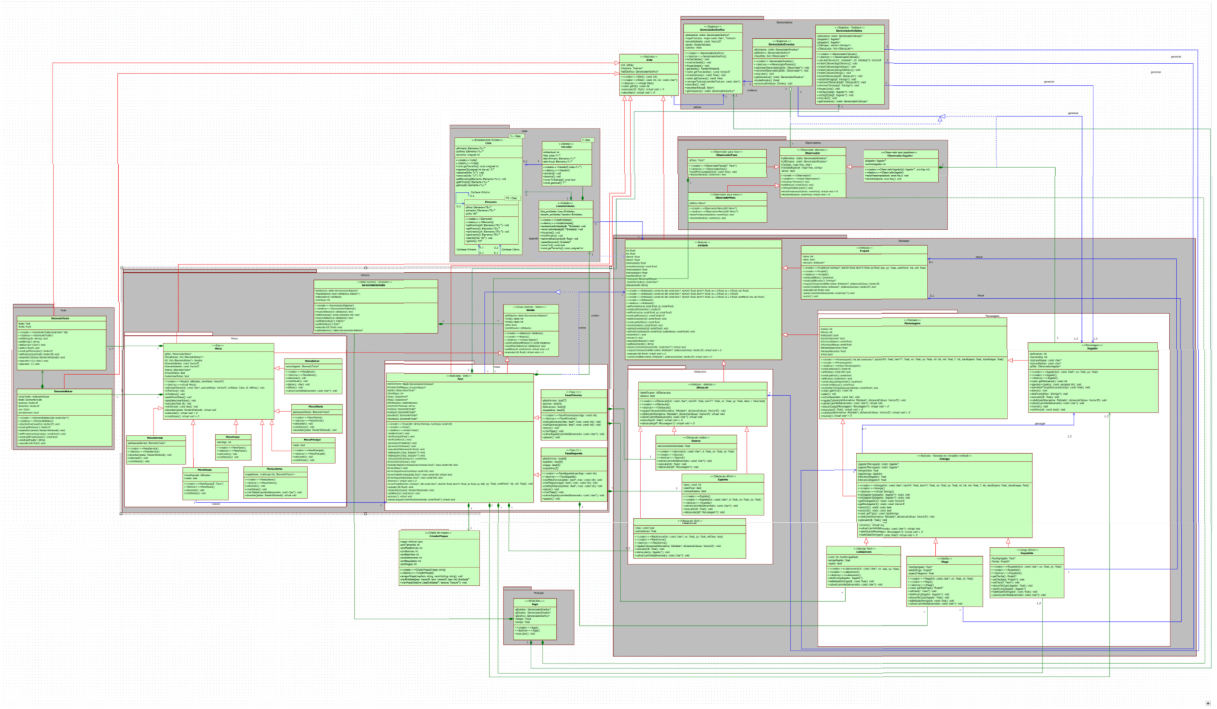

Figura 11. Diagrama de Classes de base em UML.

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

O desenvolvimento do trabalho exigiu também diversos conceitos aprendidos durante a disciplina, tanto em aula quanto por aprimoramento de conhecimento adquiridos durante a construção do jogo. Todas essas noções são pilares para a construção da formação acadêmica e profissional, de modo a envolver conteúdos do plano de ensino de toda disciplina de Técnicas de Programação. A tabela 2 apresenta esses conceitos, que são importantes para funcionalidade e organização do código, realçando a sua pertinência dentro do âmbito computacional.

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	Todos .h e .cpp, como exemplo nas classes nos *namespaces* Entidades e Gerenciadores .
1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Na maioria dos .h e .cpp, como exemplo nas classes nos *namespaces* Personagens e Fases.
1.3	- Classe Principal.	Sim	main.cpp & Jogo.h/.cpp

1.4	- Divisão em .h e .cpp.	Sim	No desenvolvimento como um todo. Todas as classes, exceto os templates “Lista” e “Iterador” foram divididos em arquivos .h e .cpp.
2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional.	Sim	Em vários dos .h e .cpp, como exemplo nas classes nos *namespaces* Gerenciadores e Fases.
2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	Em vários dos .h e .cpp, como exemplo nas classes nos *namespaces* Fases e Texto.
2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	Em alguns dos .h e .cpp, como exemplo nas classes nos *namespaces* Personagens e Menus.
2.4	- Herança múltipla.	Sim	Precisamente nas classes .h e .cpp, das classes Fase e Menu.
3	**Ponteiros, generalizações e exceções**		
3.1	- Operador *this* para fins de relacionamento bidirecional.	Sim	Precisamente em alguns dos .cpp, como nas classes Esqueleto, Mago, ElementoBotao, Observador, Menu, Fase e Jogador.
3.2	- Alocação de memória (*new* & *delete*).	Sim	Presente em diversos .cpp, como em CriadorMapas, Jogador, Fase e nos namespaces Gerenciadores e Menus.
3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).	Sim	Nas classes Lista.h e Iterador.h.
3.4	- Uso de Tratamento de Exceções (*try catch*).	Não	
4	**Sobrecarga de:**		
4.1	- Construtoras e Métodos.	Sim	Em vários .h e .cpp. Como exemplo nos namespaces Obstaculos, Personagens e Fases.
4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).	Sim	Operator= em Jogador.h e .cpp. Operator- - e operator += em ElementoTexto .h e .cpp.
---	**Persistência de Objetos (via arquivo de texto ou binário)**		
4.3	- Persistência de Objetos.	Não	
4.4	- Persistência de Relacionamento de Objetos.	Não	
5	**Virtualidade:**		. . .
5.1	- Métodos Virtuais Usuais.	Sim	Em alguns .h e .cpp. Como exemplo nos namespaces Obstaculos e Personagens.
5.2	- Polimorfismo.	Sim	Usado para classes em alguns .h e .cpp. Como nos namespaces Obstaculos e Personagens.
5.3	- Métodos Virtuais Puros / Classes Abstratas.	Sim	Em alguns .h e .cpp, como nas classes Entidades, Personagem, Menu e Observador.
5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.	Sim	Um exemplo extenso é o uso do Observer pela classe Observador. Com observers para Fase, Menu e Jogador.
6	**Organizadores e Estáticos**		

6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Vários namespaces foram criados. Como exemplos Gerenciadores, Entidades, Menus, Fases, Texto, Listas, Observadores, Estados, Principal, Obstáculos e Personagens.
6.2	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Na classe Elemento dentro de Lista.h
6.3	- Atributos estáticos e métodos estáticos.	Sim	Em algumas classes como Inimigo, Fase, Estado, GerenciadorEstado, Observador e no namespace Gerenciadores.
6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	Na maioria dos .h e .cpp. Como exemplo nos namespaces Gerenciadores, Listas, Texto, Personagens, Estados.
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	A classe String foi usada como exemplo no namespace Texto. As classes Vector e List foram usadas como exemplo no GerenciadorColisoes.
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	Uso da classe Mapa no namespace Estados e em Fases.
---	**Programação concorrente**		
7.3	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins.	Não	
7.4	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.	Não	
8	**Biblioteca Gráfica / Visual**		
8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: ● tratamento de colisões ● duplo *buffer*	Não	
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	Com o uso da SFML e a classe GerenciadorEventos. Além dos namespaces de Menus e Texto.
---	**Interdisciplinaridades via utilização de Conceitos de <u>Matemática Contínua e/ou Física</u>.**		

8.3	- Ensino Médio Efetivamente.	*Sim*	Como exemplo o cálculo da aceleração da gravidade.
8.4	- Ensino Superior Efetivamente.	*Não*	
9	**Engenharia de Software**		
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	Realizada por observar e discutir frequentemente os requisitos.
9.2	- Diagrama de Classes em *UML*.	Sim	O diagrama foi realizado e está presente neste relatório inclusive.
9.3	- Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões.	Sim	Os padrões de projeto utilizados foram o Singleton (namespace Gerenciadores), Template Method(namespace Personagens), Observer (namespace Observadores), Iterator (namespace Listas), Mediator (classe GerenciadorColisoes) e State (namespace Estados).
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	Os testes respeitaram os requisitos e o diagrama de classes.
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	Foi usado o GitHub. O link do mesmo é: https://github.com/LucasMacielFer/jogoSimao.git
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Foram realizadas 2 reuniões com o professor nos dias 20/08 e 27/08.
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Tanto o Wesley quanto o Lucas participaram de 4 reuniões na monitoria. As datas constam nos dias 14/08, 19/08, 22/08 e 26/08, envolvendo os monitores Nicky (2 vezes), Giovane e Gabrielle, com os devido relatos por email para o professor. Além disso, o Lucas também participou do curso do Peteco.
10.4	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Trabalho revisado da equipe do Leonardo Yuji e do Enzo Kira.
Total de conceitos apropriadamente utilizados.			**80%** (oitenta por cento).

DISCUSSÃO E CONCLUSÕES

O desenvolvimento do jogo se deu de forma satisfatória em relação à progressão do mesmo, de modo que apesar das dificuldades devido a motivos externos como a greve, ainda sim ocorreu de forma tranquila e contribuiu para o desenvolvimento profissional e acadêmico.

Com relação aos detalhes do desenvolvimento, é importante frisar a extensa cooperação entre a dupla, com comunicação quase diária, seja pessoalmente, por mensagem ou pelas

diversas reuniões por chamada de voz. Foi possível se construir um programa que se baseou em primeiro nos requisitos estabelecidos pelo professor, onde se teve implementação gradual de partes essenciais ao funcionamento do mesmo, desde os processos de criação de gráficos e colisão até as construções do menu e das fases. Foi possível praticar diversos conceitos aprendidos durante a disciplina, desde as bases da programação orientada a objetos até aplicação de padrões de projeto e o polimorfismo que é recorrente no sistema.

Também se teve auxílio do professor durante as aulas e reuniões, além das reuniões na monitoria, que foram fundamentais para resolução de problemas e discussões importantes sobre o andamento do projeto.

Como resultado, o jogo desenvolveu-se bem à luz dos requisitos, apesar de nem todos os conceitos conseguirem ser implementados e um requisito ficar faltando. Concomitantemente a esse processo mais técnico, também se teve resultados agradáveis na estética e tematização do software, apesar de um fato a ser lamentado advém da falta de tempo para implementar funcionalidades qualitativas como animações e controle sonoro, já que o ênfase foi dado aos requisitos principais (que no caso é o essencial dentro do projeto).

Diversos conceitos foram aprendidos de forma exterior com a intenção de realizar o objetivo, dando ênfase à utilização da biblioteca SFML para desenvolvimento dos gráficos. Além disso, cabe reforçar a compreensão dos padrões de projeto GoF, em que foi essencial o suporte da monitoria e o material disponibilizado, que descrevia bem os processos como nos vídeos do Giovane[2] e do Matheus Burda[3], que foram referência também para implementação de alguns algoritmos.

Por conseguinte, destaca-se o elevado nível de estudo e dedicação desenvolvido, de forma a ser um diferencial em qualquer formação e currículo, de modo a buscar atingir projetos mais ambiciosos e com escopo maior no futuro.

DIVISÃO DO TRABALHO

A tabela 4 a seguir representa as principais atividades de cada participante, destacando a colaboração da dupla em diversos módulos de fundamentação e constituição do programa.

Tabela 4. Lista de Atividades e Responsáveis.

Atividades.	**Responsáveis**
Compreensão de Requisitos	Lucas e Wesley
Diagramas de Classes	Mais Lucas que Wesley
Programação em C++	Lucas e Wesley
Implementação de *Template*	Mais Wesley que Lucas
Tentativa de implementação da Persistência dos Objetos.	Mais Lucas que Wesley
Implementação do Gerenciador de colisões	Mais Wesley que Lucas
Implementação dos Obstáculos	Lucas
Implementação dos Inimigos	Wesley
Implementação do Gerenciador de eventos	Mais Lucas que Wesley
Implementação do Gerenciador gráfico	Mais Lucas que Wesley
Implementação da Máquina de Estados	Lucas
Implementação das fases	Mais Lucas que Wesley
Implementação dos menus	Lucas e Wesley
Implementação dos observers	Lucas
Implementação de elementos de texto e botão	Lucas e Wesley
Criação apresentação do trabalho	Mais Wesley que Lucas
Escrita do Trabalho	Mais Wesley que Lucas
Revisão do Trabalho	Mais Wesley que Lucas

Em termos de realização e colaboração, tanto Lucas quanto Wesley estiveram envolvidos em 100% das atividades, de modo que se teve comunicação extensa e diária entre os integrantes da dupla, integrando todo o projeto.

Em termos de divisão para com o trabalho, de forma geral o Lucas executou cerca de 55% do processo, enquanto o Wesley executou os outros 45% restantes.

AGRADECIMENTOS PROFISSIONAIS

Agradecimentos são necessários para todos os monitores que auxiliaram em reunião com a nossa dupla, incluindo eles o Nicky, Giovane, Gabrielle e Ricardo. Além do auxílio pelo material do Matheus Burda. A cooperação de todos foi essencial para o desenvolvimento do projeto e gratidão também para os revisores do relatório a ser lido, cuja dupla citada é a de Leonardo Yuji e de Enzo Kira.

REFERÊNCIAS CITADAS NO TEXTO

[1] SIMÃO, J. M. Página de Internet do Prof. Simão., Curitiba – PR, Brasil, Acessado em 01/09/2024 - https://pessoal.dainf.ct.utfpr.edu.br/jeansimao/ .

[2]SALVI, Giovane Lucas. JogoPlataforma2D-Jungle, UTFPR, 2023. Disponível em: https://github.com/Giovanenero/JogoPlataforma2D-Jungle.git. Acesso em: 01/09/2024.

[3] BURDA, Matheus. Desert, UTFPR, 2022. Disponível em: https://github.com/MatheusBurda/Desert.git . Acesso em: 01/09/2024.

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A]SALVI, Giovane Lucas. JogoPlataforma2D-Jungle, UTFPR, 2023. Disponível em: https://github.com/Giovanenero/JogoPlataforma2D-Jungle.git. Acesso em: 01/09/2024.

[B]BURDA, Matheus. Desert, UTFPR, 2022. Disponível em: https://github.com/MatheusBurda/Desert.git . Acesso em: 01/09/2024.

[C]GAMMA, E. *et al.* Padrões de projeto: Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2007.

[D]CRAFTPIX. Free werewolf sprite sheets pixel art. Craftpix.net Disponível em: https://craftpix.net/freebies/free-werewolf-sprite-sheets-pixel-art/. Acesso em: 01/09/2024.

[E]CRAFTPIX. Free knight sprite sheets pixel art. Craftpix.net Disponível em: https://craftpix.net/freebies/free-knight-character-sprites-pixel-art/. Acesso em: 01/09/2024.

[F]CRAFTPIX. Free skeleton sprite sheets pixel art. Craftpix.net Disponível em: https://craftpix.net/freebies/free-skeleton-pixel-art-sprite-sheets/. Acesso em: 01/09/2024.

[G]CRAFTPIX. Free wizard sprite sheets pixel art. Craftpix.net Disponível em: https://craftpix.net/freebies/free-wizard-sprite-sheets-pixel-art/. Acesso em: 01/09/2024.