PROGRAMAÇÃO ORIENTADA A OBJETO EM C++ UTILIZANDO ALLEGRO: PRODUÇÃO DO JOGO "BRASIL INDÍGENA"

Waine Barbosa de Oliveira Junior, João Felipe Sarggin Machado

waine@alunos.utfpr.edu.br, joaomachado@alunos.utfpr.edu.br

Resumo – A disciplina de Técnicas de Programação exige o desenvolvimento de um software de plataforma em formato jogo, com o objetivo do aprendizado de engenharia de software e implementação da programação orientada a objetos em C++. Foi criado, então, o jogo "Brasil Indígena", que consiste em um jogo de plataforma em 2D, dividido em duas fases e com dois protagonistas. Para o desenvolvimento do jogo, foi feita a modelagem daquele com o uso de Diagrama de Classes e Atividades em UML (Unified Modeling Language). Após a etapa de análise, realizou-se o desenvolvimento do jogo em linguagem C++, utilizando-se a biblioteca Allegro e o programa Microsoft Visual Studio 2017 para tal. Foram contemplados os conceitos usuais de Orientação a Objeto, como classe abstrata, sobrecarga de operadores, métodos virtuais, polimorfismo, classes aninhadas, dentre outros. Após a implementação, foram realizados testes, os quais demonstraram a funcionalidade do jogo e o sucesso do projeto. Por fim, salienta-se que foi cumprido o objetivo de aprendizado visado pelo projeto.

Palavras-chave Trabalho de Técnicas de Programação, Jogo de Plataforma, Programação Orientada a Objetos em C++, Biblioteca Gráfica, Allegro, Brasil Indígena, UML (Unified Modeling Language), UTFPR, Programação Orientada a Eventos.

Abstract – The Programming Technics subject requires the development of a platform software in a game format, with the objective of learning software engineering and the implementation of objected oriented programming in C++. So the game "*Brasil Indígena*" was created, which is a 2D platform game divided in two phases, with two protagonists. For the development of the game, the modeling in UML, using classes and activities diagrams, was made. After the modeling stage, the game was developed in C++ language using the Allegro Library and the software Microsoft Visual Studio 2017. Concepts of object oriented programming, such as abstract class, overload, virtual methods, polymorphism, nested classes, among others, were contemplated. After the implementation, tests were made, which proved the functionality of the game and the success of the project. Lastly, it is important to emphasize that the objective of learning objected oriented programming was achieved.

Key-words Programming Technics activity, Platform Game, Object Oriented Programming in C++, Graphic Library**,** Allegro**,** Brasil Indígena, UML (Unified Modeling Language), UTFPR, Event-driven Programming.

INTRODUÇÃO

Para a atividade prática supervisionada da matéria de Técnicas de Programação, pertencente à grade curricular do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR) no câmpus Curitiba, foi requerida a produção de um software de plataforma em forma de jogo, utilizando-se a programação orientada a objetos, a biblioteca gráfica Allegro e os conceitos vistos em sala de aula, com o objetivo de aprofundar esses e adquirir novos conhecimentos, tendo como exemplo a persistência de objetos e o conceito de evento.

O trabalho permite que os alunos tenham, na maioria das vezes, o primeiro contato com a produção de um projeto relativamente grande utilizando programação orientada a objetos, assim exigindo que conceitos previamente vistos em salas sejam utilizados, além do aprendizado de outros imprescindíveis na área da programação, como persistência de objetos e diagrama de atividades.

Então foram levantados os pré requisitos e então feita uma modelagem em UML, visando a organização do projeto. O jogo, então, foi desenvolvido adequadamente em C++ utilizando a biblioteca gráfica Allegro para tal, com os conceitos adequados de programação orientada a objetos e o uso do aplicativo GraphicsGale [1] para a manutenção e produção de sprites. Após isso o jogo foi testado e otimizado visando melhor performance e experiência para o jogador, além da correção de bugs detectados.

Em sequência são apresentados: a explicação acerca do jogo; o desenvolvimentdo desse em C++; os conceitos utilizados e não utilizados, além do porquê; a conclusão sobre o trabalho; as considerações pessoais; a divisão do projeto e então os agradecimentos**.**

EXPLICAÇÃO DO JOGO

A base para história e desenvolvimento do jogo é a colonização portuguesa e espanhola na América em meados do século XVI, visando expor os conflitos e massacres sofridos pelos nativos. Então foram criados dois protagonistas, Teçá e Raoni, que tiveram suas aldeias massacradas pelos homens brancos e, movidos pela revolta, agora buscam vingar seu povo e expulsar os invasores dali.

Figura 1: Teçá.

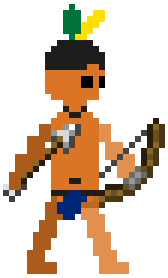

Figura 2: Raoni.

O jogo “Brasil Indígena” apresenta duas fases, ambas se passando na floresta e tendo como inimigos homens brancos. O jogador tem como objetivo matar todos inimigos da fase para não ser colonizado, possuindo três chances para isso. A tela inicial consiste em um fundo com uma floresta e o mar ao fundo, um escrito com o nome do jogo e um menu que apresenta as opções “Novo Jogo”, “Pontuações” e “Sair”. Aquela leva à escolha dos jogadores, essa mostra as pontuações dos jogos e esta fecha o jogo.

Figura 3: Menu do Jogo.

Figura 4: Exemplo de Pontuações.

Ao passar para a escolha de jogadores escrito “um jogador” e “dois jogadores”, cada qual possui logo abaixo duas opções de personagens para jogar. “Teçá” e “Raoni” para jogador único e as combinações “Teçá - Raoni” e “Raoni - Teçá” para dois jogadores, com o primeiro nome representando qual personagem será controlado pelo jogador um e o segundo qual será pelo jogador dois. Também há um botão “voltar”, que retorna ao menu do jogo.

Então são apresentadas as opções de escolha de modo de jogo, sendo elas “Campanha”, “Fase 1” e “Fase 2”. Aquela consiste em jogar sequencialmente as fases um e dois e as outras em jogar a fase referida. Assim como anteriormente, também é apresentado o botão voltar, o qual volta para a escolha de jogadores.

Figura 5: Escolha de jogadores.

Figura 6: Escolha de fases.

Após a escolha do modo de jogo, a fase é iniciada. Cada jogador tem três chances para eliminar todos inimigos, caso não consiga nelas, uma mensagem é escrita na tela e se retorna ao menu. Caso o jogador morra e ainda possua chances, a fase é reiniciada (não necessariamente como anteriormente pois certos inimigos e obstáculos tem probabilidade de não serem adicionados).

Figura 7: Tela de derrota.

Os inimigos são: Espadachim, Cavaleiro e Mosqueteiro. Aquele possui uma espada e tenta chegar perto o bastante do jogador para atacá-lo. Esse é um homem montado num cavalo carregando uma lança, ele corre rapidamente atrás do jogador tentando atravessá-lo com sua lança. Este é um homem com um mosquete, atirando no jogador quando o vê em sua frente. Foi escolhido representar os inimigos por meio de primitivas por motivos que serão explicados posteriormente no artigo.

Os obstáculos presentes são: a rede, o espinho e a armadilha. A rede é ligada a uma corda que quando o jogador colide, aciona a rede que começa a cair. Caso o jogador seja atingido por ela, fica preso por certo tempo e toma uma grande quantidade de dano. Inimigos não acionam ou tomam dano da rede. Os espinhos estão no chão, dão dano e knockback tanto em jogadores quanto inimigos. Armadilhas estão presentes em plataformas e são acionadas quando o jogador pisa em cima delas, prendendo o jogador por um tempo e ferindo-o. Após ser acionada, fica um tempo nesse estado e então desaparece.

Quanto às plataformas, há dois tipos: aquelas que é possível atravessar debaixo para cima e as que não. Usualmente apenas o chão não é possível atravessar. Outro recurso para movimentação é a corda na qual é possível subir e descer, presentes ao lado de altos montes.


Figura 8: Armadilha, espinho e rede, respectivamente.

O jogador fere inimigos atacando-os ou fazendo com que eles batam em espinhos. Já o jogador é ferido ao ser atacado, encostar em um inimigo ou colidir com um obstáculo (espinho, armadilha ou rede). Tanto inimigo quanto jogador tomam um knockback após serem atacados (com exceção do cavaleiro) e ficam um curto tempo invulneráveis após o dano.

Durante a fase é possível pausar o jogo. Então é apresentado um fundo marrom escrito pause com as opções “Continuar” e “Menu”, a primeira continua o jogo e a segunda retorna ao menu.

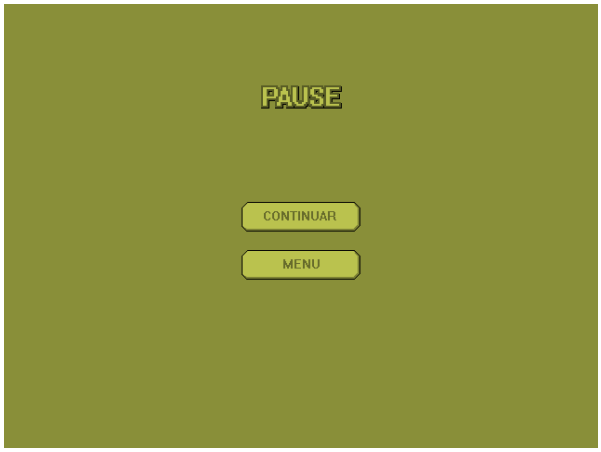

Figura 9: Pause durante a fase

Caso o jogador mate todos os inimigos, a mensagem “Fase Completa” é impressa na tela, a pontuação será salva e o jogo retornará ao menu. Mas caso o modo de jogo campanha tenha sido escolhido e ainda haja fases, a próxima fase será iniciada.

DESENVOLVIMENTO DO JOGO

Nesta seção seguem, em tabela, as funcionalidades para o jogo, a situação desses com relação ao progresso e então a explicação sobre o desenvolvimento do software.

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

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	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.	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.	Requisito previsto inicialmente e realizado
6	Ter inimigo “Chefão” na última fase	Requisito previsto inicialmente e não realizado
7	Ter três tipos de obstáculos.	Requisito previsto inicialmente e realizado
8	Ter em cada fase entre um e dez tipos de obstáculos com número aleatório de obstáculos.	Requisito previsto inicialmente e realizado
9	Ter representação gráfica de instâncias.	Requisito previsto inicialmente e realizado
10	Ter em cada fase um cenário de jogo com obstáculos.	Requisito previsto inicialmente e realizado
11	Gerenciar colisões entre jogador e inimigos.	Requisito previsto inicialmente e realizado
12	Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (*ranking*).	Requisito previsto inicialmente e parcialmente realizado
13	Permitir Pausar o Jogo	Requisito previsto inicialmente e realizado
14	Permitir Salvar Jogada.	Requisito previsto inicialmente e não realizado

Visto a complexidade que o software teria, houve uma modelagem de projeto em UML para facilitar a compreensão e também para servir de roteiro na programação. O “Jogo” sendo classe principal, teve seu loop contemplado com um Diagrama de Atividade (Figura 11) para explanar seu funcionamento e a ordem de acontecimentos. Também foi desenvolvido um Diagrama de Classes (Figura 10) aonde encontram-se 40 classes, dentre elas “Personagem”, “Entidade” e “Obstáculo”, todas abstratas que são implementadas por outras classes como “Jogador” para aquela, “Projétil” para essa e “Espinho” para esta. Além de outras classes de gerenciamento como “gerenciaAnim”, “gerenciaPontuações” e “gerenciadorBotões” as quais gerenciam, respectivamente, “Animação”, “Pontuação” e “Botão”.

Cronologia do desenvolvimento

O primeiro passo foi implementar algumas funções e relações básicas entre entidades, como a movimentação e a colisão com plataformas. Com a gradativa evolução, foram implementadas funcionalidades como ataque, colisão de projéteis, colisão com obstáculos, invulnerabilidade, *knockback*, dentre outras. Após a implementação de todas relações e funcionalidades levantadas para as entidades no início do projeto, foi iniciada a sessão de testes e correção de erros. Nessa parte do projeto foram utilizados conceitos de física mecânica (aceleração, velocidade, gravidade e suas relações), geometria analítica (plano cartesiano e coordenadas relativas) e matemática (condições para um retângulo colidir com outro).

Então se deu início ao desenvolvimento do gerenciamento de fases do jogo a partir da criação da classe "ListaFases". Essa tinha como objetivo gerenciar o número de jogadores das fases, quais eles seriam, qual fase seria jogada e se essa estaria em modo campanha (modo sequencial). Para tal foi necessária a criação de um menu primitivo, o qual por si exigiu a criação da classe botão.

Após a implementação do menu primitivo e da classe botão, em uma conversa com alguns colegas de sala de aula em estágio de desenvolvimento mais avançado, foi feita a adição da classe "Mapa" para gerenciamento das relações entre entidades na fase, com a migração dessas, que previamente eram feitas na classe "Fase".

Depois que todas funcionalidades e relações entre fases e entidades foram feitas, foi iniciada a implementação das classes de gerenciamento gráfico do jogo. Para tal foram criadas três novas classes, "Animação", "ListaAnimação" e "gerenciaAnim". A primeira possui as características da animação, como número de frames, largura, altura, ID e taxa de atualização. A segunda consiste num agrupado de animações que cada entidade agregaria. A terceira gerencia a criação de protótipos de animação e a inclusão dessas na "ListaAnimação" de cada entidade. Para tal foi utilizado o padrão de projeto *Prototype*, com o objetivo de carregar apenas uma vez as imagens de cada "Animação".

Após alguns dias, o UML atualizado do jogo foi apresentado ao professor doutor Jean M. Simão. Foram apontados vários erros no projeto, principalmente de coesão e desacoplamento, além de dicas para mudanças na ListaFases e a criação de uma nova classe para unir os atributos que Entidade e Botão tinham em comum. No mesmo dia os principais erros foram corrigidos e o UML foi reapresentado no dia seguinte. Foi apontado pelo professor que estávamos no caminho certo e que os principais erros haviam sido corrigidos.

Depois do encontro ainda foi implementado o conceito de pontuação, a qual consistiria no tempo que o jogador leva para terminar uma fase. Para tal foi adicionada a classe "gerenciaPontuações" que possui a classe aninhada "Pontuação". Essa tem como atributos a pontuação e um ID que indica a qual fase ou modo aquela pertence. Aquela gerencia a ordenação e o número de pontuações que são gravados, faz a manutenção via arquivos de texto da pontuação e possui função para imprimi-las.

Com os principais fatores do código caminhados, foi iniciado o desenvolvimento das sprites. Para tal foi utilizado o *software* "*GraphicsGale*" [1]. Os botões, o fundo do menu, as frases e logos do jogo foram os primeiros a serem produzidos. Então foi iniciada a produção das sprites das entidades, a qual tomou consideravelmente mais tempo que aquelas.

Foi escolhido desenhar os inimigos apenas com primitivas numa tentativa de "inversão de papéis" de índios e europeus. Desde o início da Era Moderna índios e nativos americanos tiveram suas vozes e culturas silenciadas, criando-se no imaginário popular um estereótipo aculturado do indígena, como é possível perceber em comentários de alguns políticos com relação a reservas indígenas. Por meio da representação gráfica detalhada do indígena e a genérica do homem branco, se busca fazer uma metáfora com a profundidade e diveresidade da cultura indígena, e fazer com que europeus "provem do próprio veneno".

Com o objetivo de testar erros e afins que eventualmente não foram detectados, o software foi enviado para amigos e familiares para testes, os quais relataram alguns erros que ocorreram e esses foram prontamente corrigidos. Finalmente foi possível afirmar que o jogo estava pronto.

Todo código e suas atualizações podem ser encontrado num repositório no *GitHub* [2].

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Nesta seção seguem duas tabelas, a dois consistindo nos conceitos da programação orientada a objetos que foram ou não utilizados e a três na justificativa para a segunda.

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

N.	Conceitos	Uso	Onde
1	**Elementares:**		
	- Classes, objetos,	Sim	Todos .h e .cpp
	- Atributos (privados), variáveis e constantes	Sim	Todos .h e .cpp
	- Métodos (com e sem retorno).	Sim	Todos .h e .cpp
	- Métodos (com retorno *const* e parâmetro *const*).	Sim	Todos .h e .cpp
	- Construtores (sem/com parâmetros) e destrutores	Sim	Todos .h e .cpp
	- Classe Principal.	Sim	Jogo.h e .cpp
	- Divisão em .h e .cpp.	Sim	No projeto.
2	**Relações de:**		
	- Associação	Sim	Mapa e Entidade, dentre outras

	- Agregação via associação	Não	
	- Agregação propriamente dita.	Sim	Jogo.h
	- Herança elementar.	Sim	Fase1.h, Fase2.h e Botao.h
	- Herança em diversos níveis.	Sim	Todas classes que herdam de Entidade.h
	- Herança múltipla.	Não	
3	**Ponteiros, generalizações e exceções**		
	- Operador *this*	Sim	Classes Animação, ListaAnimação, Jogador, Personagem
	- Alocação de memória (*new* & *delete*)	Sim	Classes Fase, Personagem, gerenciaAnim, gerenciaPontuacoes, Jogador e Mosqueteiro
	- Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via Templates)	Sim	Lista.h
	- Uso de Tratamento de Exceções	Não	
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Entidade.h, protoEntidade.h, Personagem.h, Inimigo.h, Arma.h, Projetil.h, Obstaculo.h, Jogador.h e Animacao.h
	- Operadores (2 tipos de operadores pelo menos)	Sim	Jogador.h e Personagem.h
	Persistência de Objetos		
	- Texto via Arquivos de Fluxo	Sim	gerenciaPontuacoes.h
	- Binário	Não	
5	**Virtualidade:**		
	- Métodos Virtuais.	Sim	Entidade.h, ProtoEntidade.h, Fase.h, Plataforma.h
	- Polimorfismo	Sim	Entidade.h
	- Métodos Virtuais Puros / Classes Abstratas	Sim	Entidade.h, ProtoEntidade.h, Personagem.h, Inimigo.h e Fase.h
	- Coesão e Desacoplamento	Sim	Todo projeto
6	**Engenharia de Software**		
	- Levantamento de Requisitos Textualmente e Tabelado (Ou por meio equivalente como Diagrama de Requisitos da SysML)	Sim	Tabela 1
	- Diagrama de Classes em *UML*	Sim	Figura 10
	- Diagrama de Atividades em *UML*	Sim	Figura 11
	- Outros diagramas em *UML*, Diag. de Estados, Diag. de Seqüência, Diag. de Pacotes etc. - E/ou Levantamento de Casos de Uso e sua expressão por meio de Diagrama de Casos de Uso em *UML* - E/ou outros diagramas estabelecidos, como Diag. de Fluxo de Dados (DFD) ou Diagrama em SysML (Diag. de Requisitos. de Blocos etc).	Não	
7	**Biblioteca Gráfica**		
	- Funcionalidades Elementares.	Sim	Jogo.h, Fase.h, derivados de Entidade.h, Botao.h e Animacao.h
	- Funcionalidades Avançadas como: • tratamento de colisões • duplo *buffer* • *especificar aqui outras*	Não	
	Obs.: especificar quais funcionalidades.		
8	**Interdisciplinaridades por meio da utilização de Conceitos de Matemática, Física etc**		

	- Ensino Médio (especificar quais Conceitos aqui)	Sim	Cinética na movimentação dos personagens (Física), colonização da América (História)
	- Ensino Superior (especificar quais Conceitos aqui)	Sim	Coordenadas relativas e movimentação no plano X, Y (Geometria Analítica)
	Organizadores e Estáticos		
	Espaço de Nomes (*Namespace*) criada pelos autores. E/ou Classes aninhadas.	Sim	gerenciaPontuacoes.h e ListaFases.h
	Atributos estáticos e chamadas estáticas de métodos.	Sim	Jogo.h, Fase.h e gerenciaAnim.h
9	**Standard Template Library *(STL)* e String OO**		
	Vector da STL (p/ objetos ou ponteiros de objetos de classes definidos pelos autores) *e/ou* *List da STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores). e/ou Pilhas, Filas, Bifilas, Filas de Prioridade, Conjuntos, Multi-Conjuntos, Mapas **ou** Multi-Mapas*.	Sim	*vector da STL* na Lista.h e gerenciaPontuacoes.h
	A classe Pré-definida *String* ou equivalente.	Sim	Classe gerenciaPontuacoes
	*Obs.: Listar apenas os utilizados		
10	**Uso de Conceito Avançado no tocante a Orientação a Objetos.**		
	Ou Padrões de Projeto: *GOF* ***Ou Programação orientada a eventos e visual:*** *Objetos gráficos como formulários, botões etc (*Listar apenas os utilizados) ***Ou Programação concorrente:*** *Threads (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando* Posix, C-Run-Time ***ou*** Win32API ou afins (com ou sem uso de *Mutex, Semáforos,* ***ou*** *Troca de mensagens*). ***Ou API de Comunicação em Rede:*** *Cliente Servidor.*	Sim	GOF: Prototype e Flyweight (gerenciaAnim.h) Eventos: Fase.h, Jogo.h

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

No.	Conceitos	Justificativa
1	**Elementares**	Classe e Objetos foram utilizado porque são imprescindíveis para o desenvolvimento de qualquer programa que utiliza oriantação a objetos. Atributos foram utilizados porque é necessário guardar, conhecer e modificar estados e características de classe; variáveis foram utilizadas por serem necessárias para o funcionamente de praticamente todo programa orientado a objeto; variáveis constantes foram utilizadas principalmente no retorno de funções por serem um bom costume da orientação a objetos. Métodos com e sem retorno foram utilizados para modificação e retorno de variáveis de classe, além de coisas como a manutenção da fase, do jogo e afins. Classe principal foi utilizada para agregar os principais objetos de todo o jogo, como as fases, os jogadores, o gerenciador de pontuações, dentre outros. Divisão em .h e .cpp foi utilizada para facilitar e organizar o desenvolvimento do projeto.
2	**Relações**	Associação foi utilizado pela constante necessidade de acesso de uma classe a informações que estavam em outra, como no caso do mapa ou da fase. Agregação foi utilizada principalmente pela classe principal (Jogo) para instanciar objetos que estariam presentes durante a execução de todo o programa.

		Herança elementar foi utilizada para especificar ou adicionar métodos e atributos de uma certa classe a outra. Herança em diversos níveis foi utilizada por permitir que atibutos sejam adicionados e especificados conforme a complexidade da classe aumentava.
3	**Generalizações**	Operador this foi utilizado para manutenções que o objeto precisa fazer em si mesmo ou em outro objeto (adicionar um ponteiro a para si mesmo em outro objeto). Operadores new e delete foram utilizados para alocação de memória, possibilitando que instâncias de objetos fossem criadas em um escopo e utilzadas em outro Template foi utilizado para possibilitar a generalização do tratamento de uma lista de classes. Tratamento de exceções não foi utilizado pela dificuldade de implementação do conceito.
4	**Sobrecarga e Persistência de Objetos**	Sobrecarga tanto de construtoras quanto métodos foi utilizado para facilitar a mudança e manutenção de certas características da classe. Sobrecarga de operador foi utilizada para facilitar a checagem e a mudança de atributos da classe Personagem e Jogador, como checar se aquele está vivo ou se esse perdeu todas as chances. A persistência de objetos via arquivos texto foi utilizada para manter as pontuações (tempos) dos jogadores mesmo após o programa ser reiniciado.
5	**Virtualidade**	Métodos virtuais, polimorfismo e classes abstratas foram utilizados por serem um dos pilares da programação orientada a objetos e facilitarem muito a programação do jogo, além de tornarem o código muito mais coeso. Coesão e desacoplamento foram utilizados para tornar possível o reaproveitamento do código em outros projetos.
6	**Engenharia de Software**	Tanto o levantamento de pré-requisitos quanto o diagrama de classes e de atividades foram utilizados por facilitarem na organização e visualização de todo o jogo, permitindo que seja possível uma visão “macro” de todo projeto e suas relações.
7	**Biblioteca Gráfica**	Funcionalidades elementares foram utilizadas por serem necessárias para a representação gráfica e a manutenção de um jogo, como desenhar as sprites e atualizar os estados do jogo. Funcionalidades avançadas não foram utilizadas pela complexidade de aplicação, além de possíveis consequências que o mínimo erro nessas fossem trazer ao programa.
8	**Interdiscplinaridade e Organizadores**	Conceitos de física foram utilizados pois a movimentação de todos personagens seguem padrões reais da física. A relação histórica foi um ponto chave do projeto, pois norteou o enredo, a história e a arte do jogo, visando com que esses transmitam o ar de século XVI e colonização das Américas. Conceitos de geometria analítica foram utilizados para movimentação do mapa e de todas as entidades com relação a esse, também para tornar possível saber quando uma entidade estava na tela ou não. Classes aninhadas foram utilizadas em classes que necessitam de um encadeamento, como na ListaFases, e também em classes que fazem manutenção de um objeto que nenhuma outra classe tem acesso, como no gerenciaPontuações. Atributos estáticos foram utilizados para evitar redundância, como no caso do gerenciaAnim, para informar eventos que ocorrem em uma classe para outra classe, como no caso de Fase e Jogo e para compartilhamento de objetos que são comum a todos objetos dàquela classe, como ocorre na fase com os jogadores e a campanha.
9	**STL**	O vector da STL foi utilizado por ser um template sólido e não

		permitir evntuais erros de adição e remoção de objetos que um template criado para o projeto poderia ter. A string da STL foi utilizada para ler e escrever dados em arquivos de texto.
10	**Uso de conceitos avançados.**	Os padrões de projetos citados foram utilizados por serem soluções sólidas para problemas ou otimizações que ocorreram durante o desenvolvimento. No caso do *Prototype* o fato de todos objetos de certa classe possuírem o mesmo conjunto de animações e no caso do *FlyWeight* carregar as imagens apenas uma vez. A orientação a eventos foi utilizada para fazer a manutenção de todo o jogo, com base em cliques no mouse ou o pressionamento de teclas.

COMPARAÇÃO ENTRE DESENVOLVIMENTOS

Desde o início do trabalho foi fácil perceber que, para um código reutilizável, é imprescindível o uso da programação orientada a objetos, visto que nele é possível usufruir de conceitos como associação, agregação, herança e afins, recursos que não estão disponíveis numa linguagem procedimental. Também o modo de produção do código orientado a objetos difere do procedimental pelo planejamento prévio que deve ser feito para já levar em conta fatores como coesão e desacoplamento, exigindo a produção de um diagrama de classe UML ou semelhante para uma visão macro do programa e organização desse.

A experiência de produzir um grande código, comparado aos produzidos em Fundamentos de Programação, foi muito interessante por permitir um lampejo de como códigos "de verdade" (aqueles utilizados em jogos, *photoshop*, redes sociais, entre outros *softwares*) funcionam e são produzidos. A necessidade de aprender conceitos e bibliotecas sozinho também é um fator que diferencia Técnicas de Programação de Fundamentos de Programação, visto que nessa era possível produzir códigos e fazer a prova utilizando apenas o aprendido em sala de aula. Já para Técnicas de Programação é necessário que o aluno aprenda alguns conceitos sozinho, devido ao grande número desses, e então tire dúvidas com o professor. Uma espécie de "treinamento" para o que ocorrerá no mercado de trabalho.

DISCUSSÃO E CONCLUSÃO

Os vários erros cometidos durante a produção do jogo permitiram entender a importância de um código coeso e desacoplado, visto que em um código acoplado um mínimo erro pode fazer com que várias funções e classes tenham que ser alteradas, enquanto que naquele uma mínima mudança deve resolver o problema.

A instrução por parte do professor doutor Jean Simão também foi de extrema importância, assim como o que foi ensinado em sala de aula, pois isso possibilitou saber o que estava errado e quais conceitos deveriam ser utilizados para desenvolver certas partes do código. Também a importância da organização do código para o desenvolvimento do projeto, por permitir "dividir e conquistar" a produção do código.

DIVISÃO DO TRABALHO

Tabela 4. Lista de Atividades e Responsáveis.

Atividades.	Responsáveis
Levantamento de Requisitos	Waine e João
Diagramas de Classes	Mais Waine que João
Programação em C++	Waine
Implementação de *Template*	Waine
Implementação de classes aninhadas	Waine
Produção e manutenção de sprites	Waine e João
Escrita do Trabalho	Waine
Revisão do Trabalho	Mais Waine que João

AGRADECIMENTOS

A todos os colegas de salas de aula que contribuiram para o desenvolvimento do trabalho, especialmente Moisés e Ian.

Ao professor Jean Simão que, com as várias correções e dicas para desenvolvimento, permitiu que apredessemos vários conceitos novos e aplicassemos eles ao jogo.

Ao professor Rafael Barreto que ressaltava que o curso de Engenharia de Computação não era apenas "programação de jogos", lembrando-nos que algum dia iríamos acordar e não pensar o que fazer no jogo.

Aos amigos e amigas que ouviram nossos desabafos sobre a dificuldade de progressão no trabalho e ainda assim não nos abandonaram, especialmente Gabriella, Ingryd, Victor, Júlia, Maria, Vitória e todos os seguidores do Waine no *Twitter*.

REFERÊNCIAS CITADAS NO TEXTO

[1] Aplicativo GraphicsGale. Disponível em: <https://graphicsgale.com/us/>. Acesso em: 16 de out. de 2017.
[2] Repositório do GitHub. Disponível em: <https://github.com/jrwaine/BRAS_INDIG>. Acesso em: 25 de nov. De 2017.

REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

[A] Tutoriais Allegro. Disponível em: <https://wiki.allegro.cc/index.php?title=Allegro_5_API_Tutorials>. Acesso em: 16 de out. de 2017.
[B] Vídeo Aulas de Allegro. Disponível em: <https://www.youtube.com/watch?v=2UY0W76hIv8&list=PLCA525BA9EBA68264>. Acesso em: 16 de out. de 2017.
[C] Slides do professor Dr. Jean Simão. Disponível em: <http://www.dainf.ct.utfpr.edu.br/~jeansimao/> . Acesso em: 26 de out. de 2017.

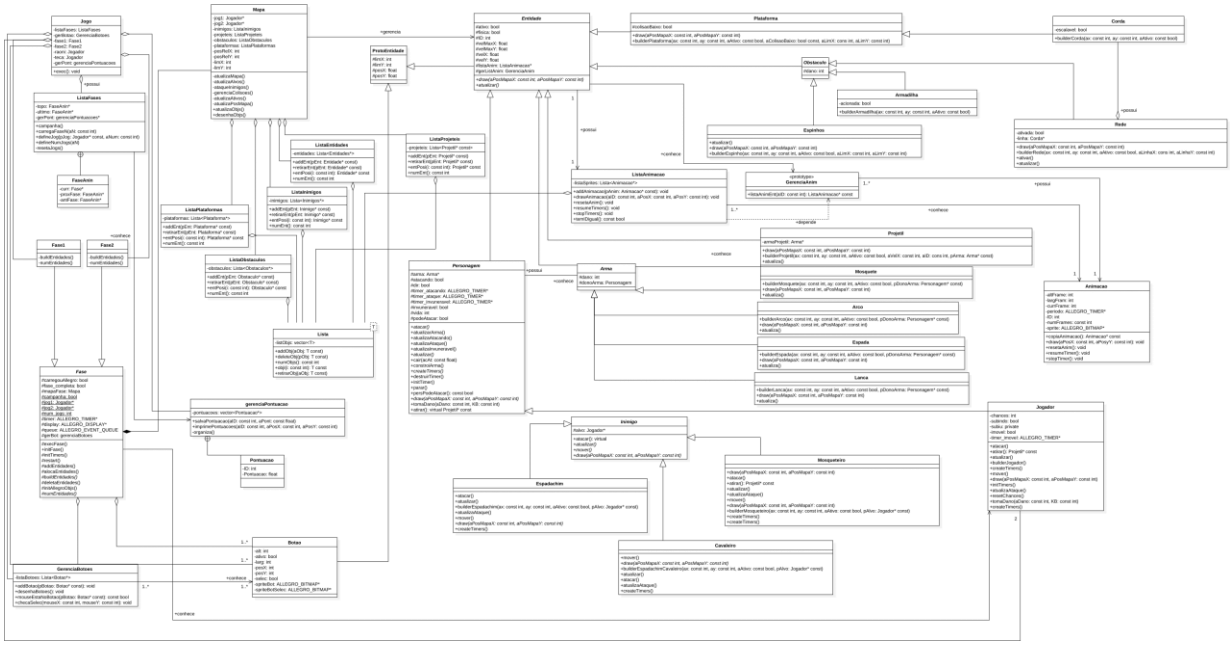

Figura 10: Diagrama de Classes

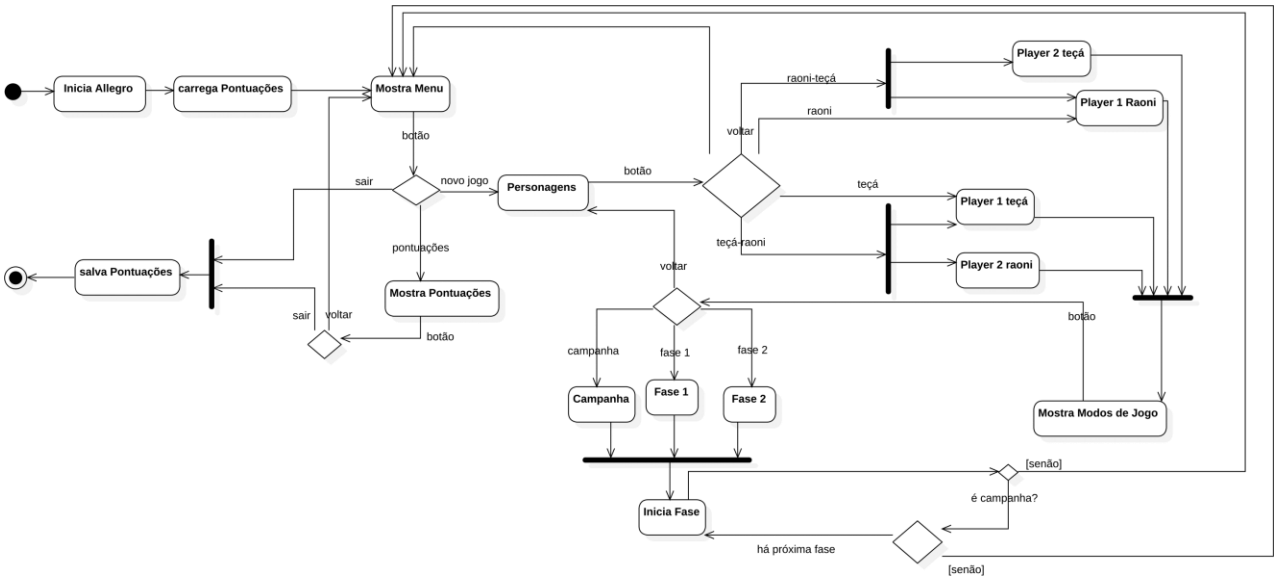

Figura 11: Diagrama de Atividades do Jogo