Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

My Spacecraft: Um jogo para construção de satélites em 3D

Victor Eduardo F. Castor

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Marcelo Antonio Marotta

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

My Spacecraft: Um jogo para construção de satélites em 3D

Victor Eduardo F. Castor

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Marcelo Antonio Marotta (Orientador)
CIC/UnB

Prof.a Dr.a Carla Koike
CIC/UnB

Prof. Dr. Édison Ishikawa
CIC/UnB

Prof. Dr. João Luiz Azevedo De Carvalho
Coordenador do Curso de Engenharia da Computação

Brasília, 06 de julho de 2023

Dedicatória

Dedico este trabalho primeiramente a Deus, depois em memória da minha querida avó Zilda Fernandes, que demonstrou seu amor por meio do exemplo e da humildade durante os anos em que esteve ao meu lado e ao lado da família.

Agradecimentos

Agradeço ao meu orientador, Prof. Marcelo Marotta, pelo valioso apoio e orientação durante a conclusão deste trabalho de curso. Agradeço especialmente pela disponibilidade em compartilhar seu tempo e conhecimento, dedicando-se a esclarecer minhas dúvidas, oferecer feedback construtivo e direcionar meus esforços na direção certa. Também agradeço à minha família e amigos por terem me dado suporte quando mais precisei e por terem estado ao meu lado nos meus dias de luta e vitória. E que esta seja mais uma conquista!

Resumo

Este trabalho apresenta uma descrição detalhada do projeto *My Spacecraft*, um jogo que envolve a montagem de satélites em 3D e conta com elementos como níveis, missões, gerenciamento de recursos e pontuação, com foco principal na criatividade. Inicialmente, o jogo é voltado para alunos do ensino médio. Descrevem-se sua jogabilidade, diagramas de classe, casos de uso e de sequência, ferramentas utilizadas, resultados, entre outros. Além disso, avalia-se a aplicação do jogo no contexto educacional e seu potencial futuro de uso.

Palavras-chave: montagem de satélites, criatividade, contexto educacional, aprendizado baseado em jogos, informática na educação

Abstract

This work offers a detailed depiction of *My Spacecraft*, an engaging 3D satellite assembly game. With a strong focus on fostering creativity, the game encompasses elements like levels, missions, resource management, and scoring. Originally designed for high school students, it delves into the gameplay mechanics, class diagrams, use cases, and sequences, along with the tools utilized and the outcomes achieved. Moreover, it assesses the game's effectiveness in an educational setting and explores its potential for future applications.

Keywords: satellite assembly, creativity, educational context, game-based learning, informatics in education

Sumário

Lista de Figuras

Capítulo 1

Introdução

No ensino médio, uma das matérias mais difíceis de ser ministrada e, consequentemente, absorvida pelos alunos é a matéria de física, abrangendo cerca de 35% das dúvidas totais de acordo com o estudo realizado pelo site "Seu Professor", que oferece reforço escolar para mais de 30 mil estudantes [1]. Isso ocorre devido à dificuldade que os estudantes têm em encontrar explicações detalhadas e de fácil entendimento na internet para o conteúdo ensinado, além do grande distanciamento entre o que é lecionado dentro da sala de aula e o mundo exterior [2]. Isso pode resultar em uma aprendizagem superficial e pouco significativa, em que os alunos memorizam fórmulas e definições para serem reproduzidas nas provas e logo esquecidas depois [3]. Portanto, para tornar a aprendizagem nessa área mais significativa, o uso de jogos virtuais orientados ao ensino tem se tornado presente nos dias atuais.

Os jogos virtuais podem ser utilizados como ferramentas de ensino, uma vez que conseguem induzir o jogador, por meio da curiosidade e imersão, à solução de problemas que exigem o uso do raciocínio lógico [4]. Esses jogos são capazes de reproduzir qualquer cenário desejado, tornando possível que um jogador vivencie experiências que, no mundo real, seriam de difícil acesso. Entretanto, criar uma abordagem nesse nível, que consiga entreter o aluno e, ao mesmo tempo, ensinar conceitos complexos da física, pode ser um desafio. Assim, um passo em direção à facilitação desse ensino por meio dos jogos seria, por exemplo, o ensino da astronomia, pois envolve conceitos sobre satélites, lançamento em órbitas, etc., e é considerado um dos temas de maior interesse por alunos de diversos países [5].

Como será visto no capítulo posterior, existem vários estudos sobre jogos orientados ao ensino de ciências para estudantes de diversos níveis. Isso é válido, uma vez que trata-se de uma área que necessita de ferramentas adicionais para um bom ensino. Alguns dos jogos envolvem astronomia, outros não; alguns são especialmente direcionados para estudantes de nível médio, outros não. No entanto, nenhum deles aborda especificamente

conceitos sobre satélites, especialmente voltados ao ensino médio. A falta de literatura nessa área, portanto, instigou o desenvolvimento do estudo abordado neste documento.

Dentro desse contexto, foi desenvolvido o *My Spacecraft*, um jogo educativo voltado, inicialmente, para alunos do ensino médio. Seu principal objetivo é ensinar conceitos sobre satélites artificiais de forma mais interativa. Ao jogar *My Spacecraft*, os alunos têm a oportunidade de construir seus próprios satélites e gerenciar recursos à medida que avançam de nível. Assim, eles podem experimentar na prática como é projetar esses dispositivos e compreender os desafios que envolvem esse processo. O jogo fornece informações sobre os módulos do satélite, permitindo que os estudantes ampliem seu conhecimento teórico de forma contextualizada. Essa abordagem busca unir o aprendizado teórico e prático, proporcionando uma experiência educacional mais completa.

No capítulo seguinte deste documento, serão relatados alguns trabalhos relacionados encontrados, que tratam de estudos similares à proposta de *My Spacecraft*. Posteriormente, será feita uma visão geral e mais detalhada da proposta do jogo, apresentando a estruturação juntamente com diversos diagramas, como diagrama de caso de uso, de atividade, de sequência, etc. Em seguida, será mostrado o protótipo, revelando trechos-chave de código e lógicas internas do jogo. Então, será apresentado um exemplo de caso de uso, com uma gameplay completa do jogo, para, por fim, apresentar as conclusões e considerações finais no último capítulo.

Capítulo 2

Trabalhos Relacionados

A seguir, serão apresentados alguns estudos relacionados que abordam a criação e uso de jogos educativos voltados para as áreas STEM (Ciência, Tecnologia, Engenharia e Matemática em inglês), destinados a estudantes em geral, bem como o impacto produzido por essas iniciativas. Também serão analisadas a utilidade de cada um dos trabalhos relacionados para o documento apresentado e proposto.

2.1 Investigating the Impact of an Adventure-based 3D Solar System Game on Primary School Learning Process

O estudo *Investigating the Impact of an Adventure-based 3D Solar System Game on Primary School Learning Process* [6] aborda a criação e experimentação de um jogo de aprendizagem de aventura 3D em turmas escolares do ensino primário. No jogo *The Final Frontier*, o jogador deve explorar dois planetas: Mercúrio e Vênus, além do satélite natural da Terra, a Lua. No primeiro estágio, o jogador deve coletar cinco meteoros usando sua mochila a jato em Mercúrio. Após coletá-los, deve responder a um questionário antes de seguir em frente. Se responder corretamente, avança para a próxima fase em direção a Vênus. Lá, deve atravessar o ambiente, tomando cuidado com o timer de temperatura. Ao terminar, responde a outro questionário e avança para a Lua, onde deve coletar dez estrelas e assim por diante.

O jogo foi desenvolvido na plataforma Unreal Engine 4 (UE4) com a ajuda de desenvolvedores de jogos, um professor e um engenheiro pedagógico. Durante o desenvolvimento, foram levados em conta estudos anteriores que revelam boas práticas de desenvolvimento, como o uso de guia, diálogos em primeira pessoa, agentes animados, vozes humanas em vez de sintetizadas, instruções e objetivos claros, entre outros [7]. Além disso, foi utilizado

um questionário como recapitulação. Essa parte do estudo contribuiu para a formulação de missões claras para o jogo *My Spacecraft*.

Após o desenvolvimento, um experimento foi realizado com um total de 53 crianças entre 9 e 10 anos. Elas foram divididas em dois grupos: um que aprendeu o conteúdo de forma tradicional com o professor explicando tudo e outro que jogou o jogo *The Final Frontier* para obter o conhecimento equivalente. Por fim, um questionário foi aplicado para avaliar qual dos dois grupos melhor absorveu o conteúdo e como foi a experiência em geral. Os resultados foram satisfatórios, mostrando que o grupo que utilizou o jogo como método de aprendizado teve um desempenho 10,4% melhor do que aqueles que aprenderam de forma tradicional. Além disso, 100% dos estudantes gostaram da experiência de aprender por meio de jogos.

Em geral, o estudo apresentado foi muito bom para indicar boas estratégias na criação de jogos STEM voltados para o aprendizado de crianças, além de mostrar experimentalmente o impacto positivo do jogo no aprendizado delas. Fica em questão, entretanto, como jogos STEM impactariam estudantes do ensino médio.

2.2 Games-Based Learning for Systems Engineering Concepts

O trabalho *Games-Based Learning for Systems Engineering Concepts* [8] discute a criação de um jogo de cartas chamado *Space Tug Skirmish*, cujo objetivo seria ensinar os princípios básicos de engenharia de sistemas para estudantes da pós-graduação. Para isso, seria necessário dissolver toda a complexidade em conceitos chave fundamentais e de fácil entendimento, o que aconteceu no verão de 2011. Em uma conferência realizada por estudantes e professores da Iniciativa de Pesquisa em Avanço em Engenharia de Sistemas do MIT (SEAri), foram identificados quais seriam esses conceitos chave: 1) Benefício, 2) Escolha de design, 3) Recursos, 4) Incertezas, 5) Dependência de tempo e 6) Valores contingentes. Segundo eles, tais princípios são os pilares para a organização de qualquer sistema artificial complexo.

Dentre os princípios chave para a organização de sistemas complexos, benefícios seriam os impactos positivos gerados de acordo com as escolhas de design. Recursos são o que é gasto para se adquirir os benefícios, como esforço, dinheiro e tempo. Incertezas são os futuros possíveis a curto prazo. Dependência de tempo são as sequências ordenadas de ações de curto prazo que geram impacto a longo prazo, passíveis de estratégias. Valores contingentes representam a habilidade das escolhas de mudarem ao longo do tempo.

Considerando portanto os pilares, o jogo multi-jogador foi desenvolvido. Ele foca a construção e sobrevivência de um rebocador espacial para cada jogador. Ao longo de

turnos, os jogadores podem aprimorar seu rebocador ou atacar outros jogadores. Além disso, precisam tomar cuidado com cartas de efeitos especiais, já que elas podem surgir a qualquer momento, alterando o cenário do jogo e, assim, forçando os jogadores a rebuscar suas estratégias iniciais.

O jogo se mostrou um sucesso no aprendizado dos envolvidos, vindo a ser implementado mais tarde em um ambiente virtual Unity [9]. A coleta de dados possibilitou a identificação de padrões de estratégias em termos de tomadas de decisões, vindo a ser útil para pesquisas futuras e para o próprio *My Spacecraft*, na hora de realizar gerenciamento de recursos. Os únicos problemas apontados foram a falta de um design chamativo para o jogo e a ausência de animações cativantes. Além disso, o jogo é dirigido para estudantes de pós-graduação, os quais, por si só, já possuem uma gama de conhecimentos mais elevados do que aqueles de ensino médio.

2.3 At Play in The Cosmos

At Play in The Cosmos [10] é um jogo desenvolvido por astrônomos, programadores, professores, estudantes e designers, resultado da parceria entre o centro acadêmico universitário Wisconsin-Madison e Norton & Company. O objetivo principal do projeto foi a criação de um jogo multiplataforma voltado para estudantes de graduação, cuja proposta é ensinar conceitos de astronomia por meio de uma exploração intuitiva e divertida do espaço. Desenvolvido em Unity, e portado para iOS, PC e Android, o jogo contém parte dos dados armazenados na nuvem da corporação Norton (dados sensíveis), e outra parte (dados não sensíveis) no próprio dispositivo do usuário, com a ajuda da tecnologia HTML5. Quanto à jogabilidade, o jogador assume o papel de piloto de uma nave espacial que explora o espaço em busca de um planeta colonizável, seguindo uma história que o orienta por uma série de missões intergalácticas.

O desenvolvimento durou cerca de 15 meses, e uma das principais preocupações dos criadores foi como tornar o jogo o mais realista possível, usando poucos elementos de narrativas fictícias e focando em dados científicos reais. Além disso, foi pensada a melhor estratégia de aprendizagem intuitiva para que os alunos pudessem aprender o conteúdo de forma efetiva. Foi decidido que não seria conveniente forçá-los a fazer cálculos fora do jogo, visto que isso quebraria a imersão e afastaria estudantes não tão interessados em matemática. Uma das estratégias adotadas foi mostrar as equações nas telas e deixar que os próprios usuários arrastassem valores numéricos para dentro das fórmulas.

Após ser criado, o jogo foi disponibilizado diretamente para um total de 184 estudantes universitários, os quais avaliaram positivamente o jogo. Alguns sugeriram que ele fosse oferecido como crédito extra, enquanto outros recomendaram que ele fosse utilizado em

laboratórios para fins de análise e discussão. Um dos únicos problemas encontrados após o lançamento foi o fato do jogo ser curto e ter missões extremamente lineares, deixando a desejar uma maior liberdade para exploração e aplicação de fórmulas em situações mais diversas.

A revisão da literatura acima revelou boas práticas de programação, como a forma de armazenar dados sensíveis na nuvem, além de mostrar que é sim possível incluir cálculos matemáticos em um jogo, mas de forma que não quebre a imersão e não canse o jogador. A própria crítica do jogo, sobre ele ser curto, incentiva a criação de jogos não necessariamente longos, mas com possibilidade fácil de expansão, o que foi pensado durante a construção de *My Spacecraft*.

2.4 Game-Based Learning for Safety and Security Education

O artigo *Game-Based Learning for Safety and Security Education* [11] apresenta um estudo realizado na Universidade Purdue Northwest, que avaliou a eficácia do uso de jogos de computador no ensino de boas práticas de segurança no ambiente de trabalho. De acordo com os dados relatados no artigo, em 2017, cerca de 5.147 pessoas sofreram ferimentos graves, enquanto outras 2,8 milhões sofreram algum tipo de dano não letal em seus locais de trabalho [12]. Nesse contexto, o estudo apresentado mostra-se relevante, uma vez que o uso de jogos virtuais pode proporcionar um ambiente seguro para a aprendizagem das boas práticas de segurança e da utilização de ferramentas específicas, sem que haja risco de lesão real. Isso pode auxiliar na prevenção de acidentes de trabalho e na redução de danos e prejuízos para empresas e colaboradores.

Assim, foram desenvolvidos dois jogos para os testes. O primeiro simula em 3D a utilização de um esmeril de bancada, que nada mais é do que uma ferramenta elétrica com um disco giratório para polir peças metálicas. Lá, o jogador deve escolher os materiais de proteção adequados entre as várias opções possíveis, além de precisar selecionar corretamente as formas seguras de utilizar a ferramenta. Já o segundo jogo trata de um Role-playing game (RPG), em que o jogador caminha por dentro de uma empresa passando por todos os protocolos e áreas de segurança vitais, como a guarda de segurança externa, a vigilância por circuito fechado de televisão (CCTV surveillance), barreiras protetivas, entre outros.

O primeiro jogo foi desenvolvido utilizando a Unreal Engine, com a ajuda do software 3D Max para a criação dos modelos 3D. Já o segundo jogo foi desenvolvido utilizando a ferramenta Unity, e foram utilizados os softwares Autodesk Maya e Adobe Fuse para criar seus modelos 3D. O software Mixamo também foi utilizado para aplicar as animações

nos modelos criados. Os modelos foram criados visando uma semelhança o mais próxima possível da realidade, para contribuir com a imersão.

Ao final do desenvolvimento, o primeiro jogo foi aplicado em uma turma do curso de Segurança e Saúde para Tecnologias de Engenharia. Os estudantes foram divididos em três grupos. O primeiro receberia as instruções de segurança via aulas tradicionais. O segundo receberia as instruções por meio da prática em laboratórios e o terceiro receberia as instruções por meio do jogo. Assim, foi identificado que a aprendizagem por meio do jogo foi cerca de 6% melhor que a da experiência prática e cerca de 23% melhor que a aula teórica. Já o segundo jogo foi aplicado com estudantes do ensino médio. O feedback mostrou que 86% dos participantes se sentiram mais familiarizados com segurança física depois de jogar enquanto que 82% se sentiram confortáveis jogando e 84% gostaram da experiência do jogo.

Apesar de não estar diretamente relacionado a jogos voltados para a astronomia ou ciências mais específicas, o estudo mostrou-se relevante para evidenciar que outra utilização dos jogos é proporcionar uma segurança adicional. Provavelmente, o estudo teria sido ainda mais eficaz se os jogos desenvolvidos fossem criados utilizando a tecnologia de Realidade Virtual (VR), observação que faço após a leitura do documento.

2.5 Improvement in Teachers' Knowledge and Understanding of Basic Astronomy Concepts Through Didactic Games

A pesquisa financiada pela Universidade de Ljubljana, *Improvement in Teachers' Knowledge and Understanding of Basic Astronomy Concepts Through Didactic Games* [13], trata de um estudo realizado sobre o impacto de jogos didáticos no aprendizado dos próprios professores do ensino primário. Muitos autores consideram que o pensamento abstrato sobre astronomia, como por exemplo as alterações da iluminação da Terra durante o dia e a noite, e a conexão da rotação do planeta com as fases da Lua, são conceitos muitas vezes difíceis para até mesmo professores compreenderem adequadamente (Bisard et al., 1994; Kavanagh et al., 2005; Şahin et al., 2017; Sarıoğlan & Küçüközer, 2014; Trumper, 2001; Trumper, 2003). Considerando isso, dois jogos didáticos foram criados para que os professores pudessem aplicar o conhecimento em seus ambientes de aula.

O primeiro jogo foi nomeado de *Astrotheatre*. Nesse jogo, três estudantes são escolhidos para assumir respectivamente os papéis da Terra, da Lua e do Sol, enquanto os demais ficam em volta assumindo o papel de estrelas das constelações. Assim, os que ficam no meio realizam suas respectivas movimentações, com o estudante que representa o sol

segurando uma lanterna e iluminando a Terra que gira ao seu redor, e a Lua, que gira ao redor da Terra. O segundo jogo foi batizado de *Triplets*. Trata-se de um jogo de tabuleiro em que os estudantes precisam escolher, entre as 16 possíveis opções de cartas, 3 cartas que estejam correlacionadas entre si, por exemplo, a foto da Lua em um estado, o nome que se dá a isso e a posição em que ela se encontra ao redor da Terra.

Assim, foi aplicado um questionário com algumas perguntas básicas relacionadas à matéria aos professores antes e depois das aulas didáticas. Os resultados apontaram que houve um incremento do número de respostas corretas por parte deles. No geral, o estudo realizado revela-se importante ao mostrar mais uma vez que ensinar conceitos abstratos das ciências, como astronomia, é algo difícil, uma vez que não há meios experimentais de fácil acesso para se realizar na prática. O estudo também aponta que mesmo os professores podem se beneficiar com as brincadeiras realizadas em sala de aula. Apesar de se tratar de um jogo físico, e não virtual, a dinâmica, o entusiasmo e a adrenalina ainda se interconectam entre ambos.

Capítulo 3

Proposta

O objetivo de *My Spacecraft* é proporcionar um ambiente 3D onde os jogadores poderão explorar partes isoladas de um satélite, conhecidas como módulos, e interagir com elas para criar seus próprios satélites personalizados. Ao entrarem nesse ambiente, os jogadores encontrarão um quadro de missões que servirá como guia durante o processo de montagem. Ao completarem todas as missões com sucesso, poderão avançar para o próximo nível e desbloquear novas missões.

É importante ressaltar que os módulos terão diferentes níveis de compatibilidade entre si e influenciarão no desempenho geral do satélite montado. Por exemplo, se os jogadores não montarem o artefato corretamente, poderá haver falta de energia para alimentar todas as funções do satélite ou então uma insuficiência de poder de sinal para estabelecer uma comunicação eficaz com a Terra. Essas limitações serão exibidas no quadro de *status* e refletirão na pontuação do jogador. Portanto, para obter uma pontuação alta, os jogadores deverão planejar cuidadosamente a construção de seus satélites, levando em consideração os requisitos e as restrições das peças disponíveis.

3.1 Visão Geral

O jogo consistirá de três telas principais: a tela inicial, por meio da qual o jogador poderá iniciar o jogo; a tela do jogo em si, na qual ele poderá jogar e avançar de nível; e a tela de conclusão, na qual ele poderá ver o recorde alcançado. Essas e outras funcionalidades podem ser observadas no diagrama de casos de uso representado na Figura 3.1. Pelo diagrama, observa-se que na tela inicial, além da possibilidade de iniciar o jogo, há a opção de carregar algum jogo salvo previamente, excluir esse jogo ou seu recorde, visitar o painel de créditos ou sair do jogo. Se optar por iniciar, poderá ou não fazer o tutorial antes de jogar e avançar de nível. Durante a partida, ainda será possível ajustar o volume do jogo ou acessar o menu de opções, que oferecerá outras possibilidades como alterar

a sensibilidade do mouse, salvar o jogo ou sair. No cenário de conclusão do jogo, será possível reiniciar a partida, retornar à tela inicial ou sair do jogo.

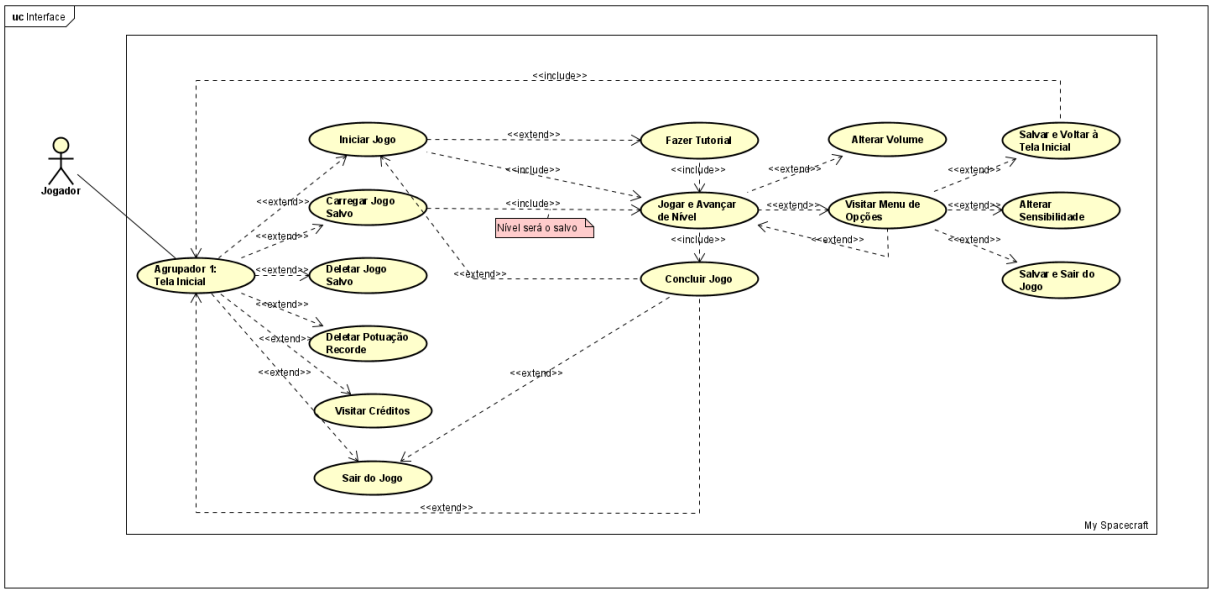

Figura 3.1: Diagrama de Caso de Uso - Interface (Fonte: [14]).

3.2 Jogabilidade

Uma vez visto o fluxo geral básico das telas do jogo e como elas interagem entre si, será importante mostrar como exatamente funciona a etapa "Jogar e Avançar de Nível". O fluxo dessa estapa pode ser observado por meio do diagrama de atividade modelado na Figura 3.2. Ao iniciar um nível, o jogador terá a capacidade de executar três atividades em paralelo: rotacionar o satélite, dar zoom no mesmo ou ativar os inputs de construção ou remoção de módulos. Para rotacionar, basta clicar com o botão direito do mouse e deslizar para a direção desejada. Para dar zoom, basta usar a roda do mouse.

Para acionar o input de construção, basta ao jogador selecionar um dos módulos disponíveis e escolher o local onde deseja acoplar ao satélite. Por outro lado, caso opte por remover um módulo, basta clicar no botão vermelho correspondente. Ao ser ativado, esse botão permitirá a remoção de módulos previamente construídos, bastando ao jogador clicar nas peças que deseja eliminar. É importante destacar que existem certas restrições para a remoção de módulos, como a impossibilidade de remover a peça base do satélite ou aquelas responsáveis por conectar outras.

Quando o jogador seleciona um módulo e posiciona o mouse na área onde deseja construir, o módulo poderá ser identificado por três cores distintas: verde indica que

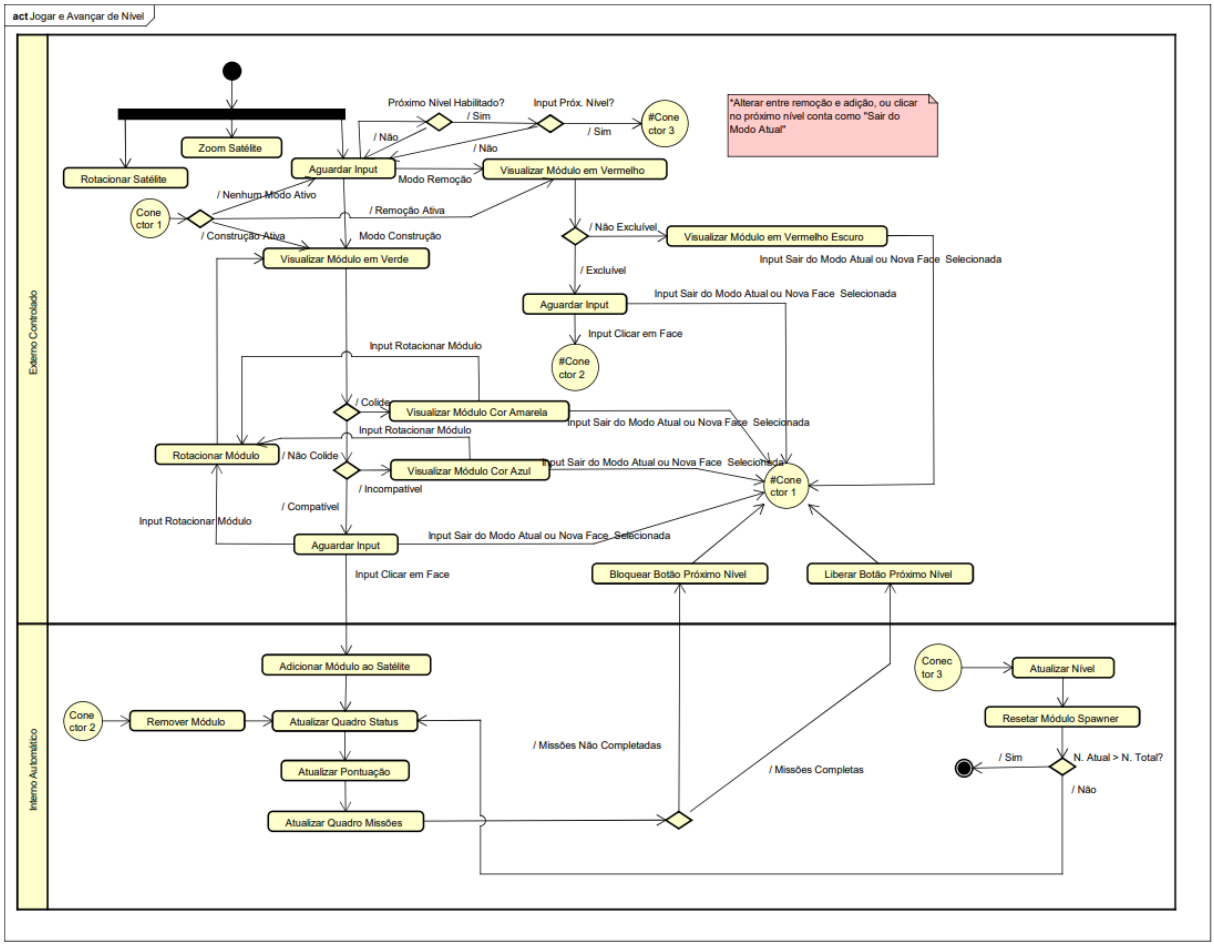

Figura 3.2: Diagrama de Atividade - Jogar e Avançar de Nível (Fonte: [14]).

a peça pode ser colocada sem problemas, amarelo indica uma colisão com outra peça e, portanto, não é possível ser colocada, e azul indica incompatibilidade com a peça adjacente, impossibilitando sua inserção. Além disso, o jogador pode realizar rotações no módulo conforme sua preferência ao clicar nas teclas W, A, S e D. Nos casos em que não for possível realizar a construção devido a colisões ou incompatibilidades, o jogador deverá selecionar outra área ou módulo para construção, ou escolher um input diferente. Caso a construção seja viável e o jogador clique para colocar o módulo, este será acoplado ao satélite na posição desejada, refletindo as atualizações no quadro de status, pontuação e quadro de missões. Neste momento, é verificado se todas as missões foram concluídas. Em caso afirmativo, o botão para avançar de nível é liberado; caso contrário, permanece bloqueado.

Se o jogador optar por entrar no modo de remoção durante a partida, ele poderá passar o mouse sobre as peças já construídas. As peças que podem ser removidas ficarão vermelhas, permitindo que o jogador as clique para eliminá-las. Já as peças que não

podem ser removidas serão exibidas em vermelho escuro. Se o jogador decidir remover uma peça, o fluxo de atualização do status, pontuação, etc., será repetido. Por fim, se o botão estiver liberado e o jogador decidir avançar para o próximo nível, verifica-se se o próximo nível é maior que a quantidade total de níveis do jogo. Se for, o jogo é encerrado; caso contrário, o quadro de status, pontuação e missões são atualizados, "resetando"a fase para o novo nível e repetindo o ciclo de aguardar o input inicial.

3.3 Requisitos de Software

Sabendo agora qual é a ideia central do jogo, será importante estabelecer alguns requisitos funcionais e não-funcionais necessários que o jogo deverá cumprir. Esses requisitos funcionarão como métrica para avaliar a qualidade final do produto, uma vez finalizado.

3.3.1 Requisitos Funcionais

Em termos de requisitos funcionais, o jogador, ao ver o tutorial, deve conseguir entender os comandos básicos e o que deve ser feito para avançar de nível. As missões devem ser claras e objetivas. O jogador deve ser capaz de salvar seu jogo e carregá-lo a qualquer momento. O sistema deve ser capaz de fornecer uma interface simples para que o usuário consiga montar as peças de seu satélite ou removê-las. O sistema deve fornecer ao jogador dicas sobre como montar seu satélite sempre que o jogador precisar.

3.3.2 Requisitos Não-Funcionais

Em relação aos requisitos não-funcionais, o sistema deve ter uma confiabilidade elevada, ou seja, durante sua execução, deve ter pouco ou nenhum erro, sendo que, caso haja erro, esse erro não deve gerar impactos negativos na experiência do usuário. Além disso, o sistema deve ser portátil para qualquer máquina com sistema operacional Windows 7 para cima, contendo ao menos 8 GB de RAM e um processador Intel ou AMD produzido nos anos 2010 para cima.

3.4 Estruturação

My Spacecraft foi criado seguindo o padrão de projeto por componentes. Isso significa que o jogo gira em torno de objetos que podem ter componentes acoplados a eles. Esses componentes são principalmente scripts, mas também podem incluir configurações fornecidas pela própria engine. Eles são responsáveis por adicionar funcionalidades aos objetos aos quais estão acoplados.

Levando em consideração a estruturação por componentes e o fato de que o jogo foi implementado utilizando a Unity Engine, que trabalha com hierarquia, serão apresentados agora os principais objetos do jogo, suas controladoras, quando aplicável, e suas principais tarefas em relação aos demais objetos. É importante mencionar que haverá uma simplificação do conteúdo apresentado aqui para facilitar a compreensão geral dos esquemas. Por exemplo, não será mostrada a classe MonoBehavior, a qual, por padrão as demais classes do Unity herdam. Ela provê métodos básicos como 'Start()', 'Update()' e outros.

Dentre os objetos que constituem o nível superior da hierarquia do jogo, está: *Game Sounds*, responsável por prover sons em geral; *Event Manager*, responsável por manusear os eventos de fluxo de tela e funcionalidades fora de jogo; *Module Spawner*, responsável por gerenciar a criação de módulos durante o jogo; *Main Camera*, responsável por prover a renderização do jogo em si, seja em pontos específicos ou não; e *Canvas*, responsável por prover os elementos de interface do usuário.

3.4.1 Game Sounds

Os principais componentes que compõem o objeto de jogo *Game Sounds* são a controladora *Sound Controller* e dois *Audio Sources*, que permitem que o objeto emita sons. A razão para ter dois desses componentes é porque um deles é responsável pela música do jogo, enquanto o outro trata dos sons de efeito do jogo. A função da controladora é fornecer os métodos de interface que, quando executados por outros objetos, produzem o som desejado, além de oferecer os métodos de controle de volume. Também inclui o método de embaralhamento de músicas para reproduzi-las em ordem aleatória. O diagrama desse componente pode ser visualizado na Figura 3.3. Nota-se que é mostrada apenas uma parte do diagrama. Caso queira visualizar a figura completa, veja o Apêndice A.

3.4.2 Event Manager

Event Manager é composto basicamente pela sua controladora, denominada *Event Manager Controller*, que é responsável por gerenciar quais objetos de jogo devem ou não aparecer em uma determinada cena, formando telas específicas, como a tela de título, tela de jogo, entre outras. Como se trata de uma função abrangente, ela se conecta a praticamente todos os objetos do jogo. É a controladora que interage com a *Save System Controller* para carregar ou salvar o progresso do jogo em execução. O diagrama correspondente pode ser visualizado na Figura 3.4.

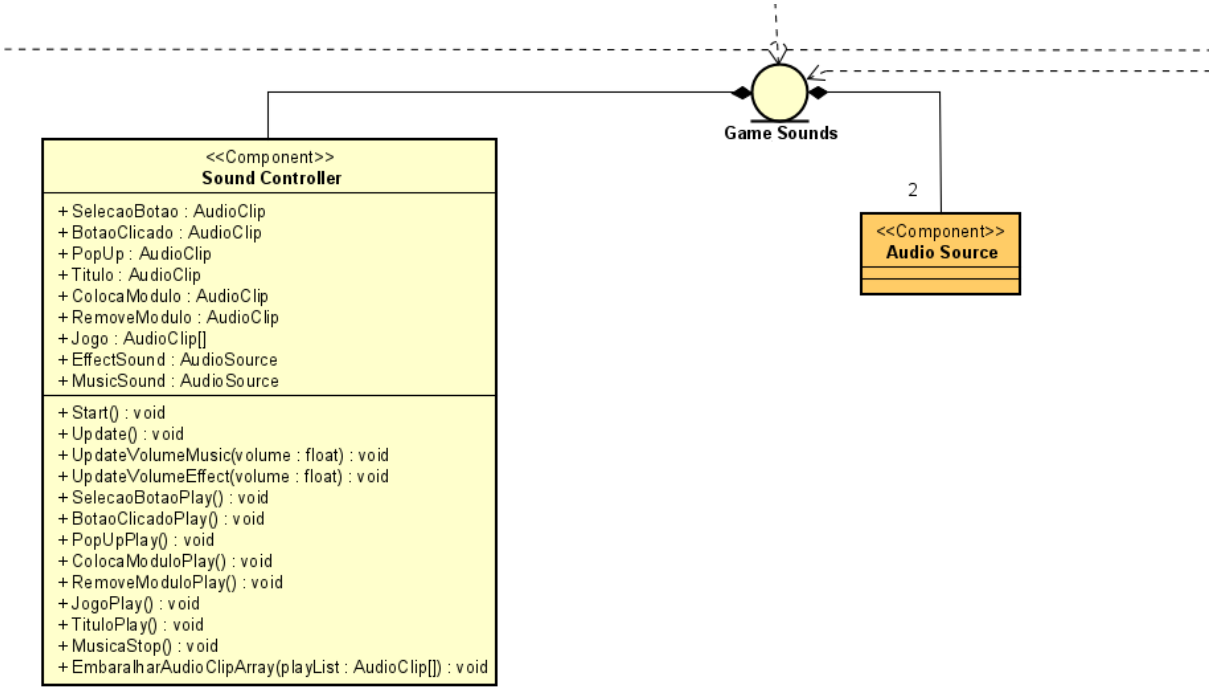

Figura 3.3: Diagrama de Classes - Game Sounds (Fonte: [14]).

3.4.3 Module Spawner

O objeto *Module Spawner* é um pouco mais complexo em sua funcionalidade. Como mencionado anteriormente, ele é responsável por gerenciar os módulos que formarão o satélite. Isso é realizado por meio do seu componente *Module Spawn Controller*, que determina quais módulos devem ser construídos, sua rotação e onde devem ser posicionados. O componente também verifica colisões, compatibilidade entre os módulos e se eles devem ser removidos, entre outras tarefas.

Devido a essa capacidade, cada módulo criado se torna automaticamente um objeto filho do *Module Spawner*. Esses objetos filhos possuem uma controladora personalizada chamada *Custom Module Controller*, que varia de acordo com o tipo de módulo existente, permitindo a alteração de métodos e atributos conforme necessário. Todas essas controladoras personalizadas dos objetos filhos herdam do *Module Generic Script*, que fornece métodos universais contendo regras que todos os objetos devem seguir. A controladora armazena informações sobre a identidade do módulo, seu estado de construção ou remoção, entre outros detalhes relevantes.

Cada um dos objetos de módulos possui seus próprios objetos filhos chamados *faces*. As faces têm a responsabilidade de identificar quando o mouse passa sobre elas, a fim de determinar se devem enviar informações de cliques, como o clique para criar um novo

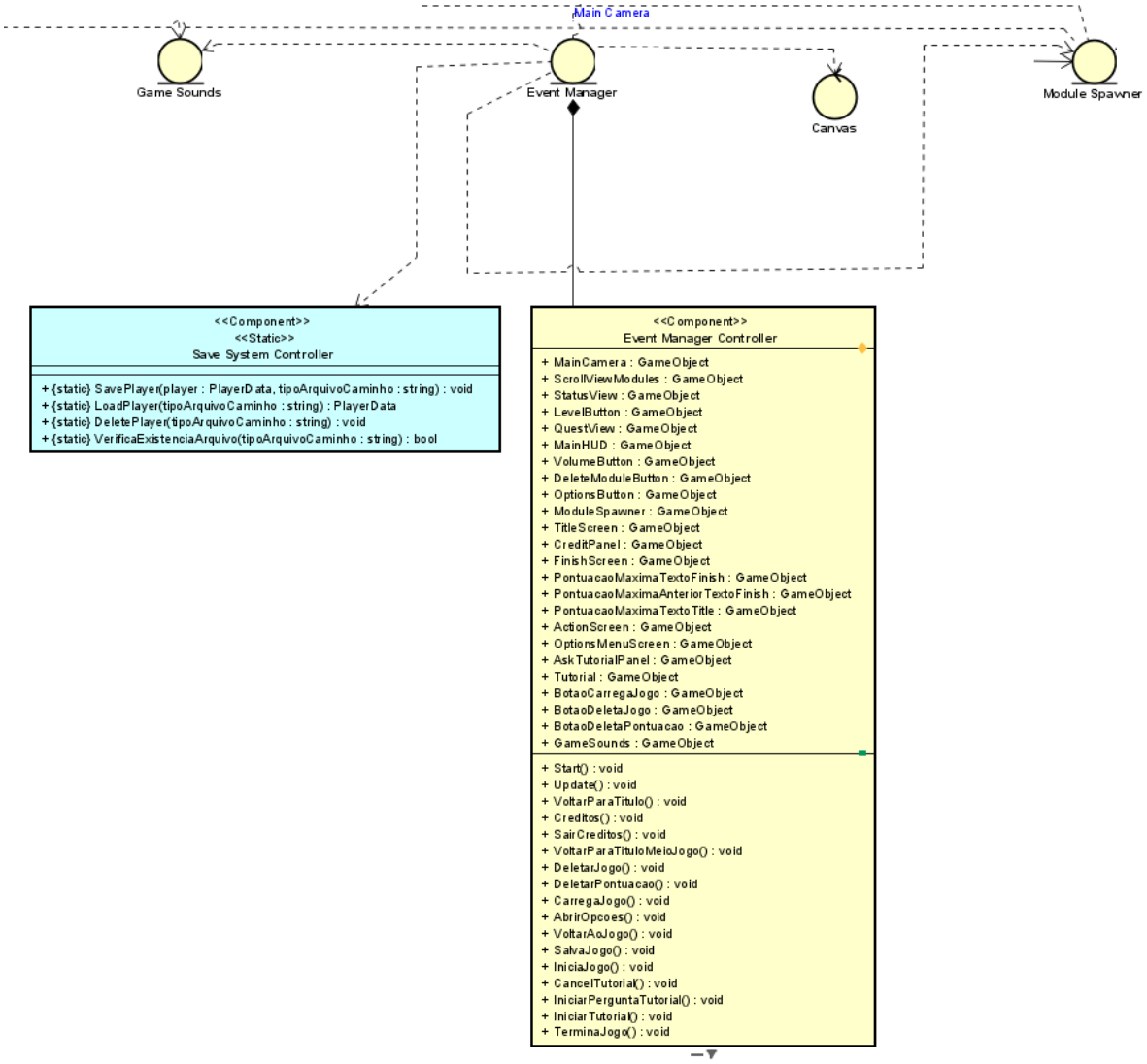

Figura 3.4: Diagrama de Classes - Event Manager (Fonte: [14]).

módulo. É por esse motivo que essas faces possuem o componente *Mesh Collider*. A controladora responsável por essas faces é a *Select Face Controller*. O diagrama de *Module Spawner* está representado na Figura 3.5.

3.4.4 Main Camera

O objeto de jogo *Main Camera* é composto por diversos componentes essenciais para o seu funcionamento. O componente *Transform* é responsável por gerir a posição da câmera na cena. O componente *Audio Listener* é responsável por renderizar o som ambiente. O componente *Camera* contém as funções genéricas para uma câmera no Unity. Por fim, o componente *Camera Movement Controller* possui os métodos e atributos necessários

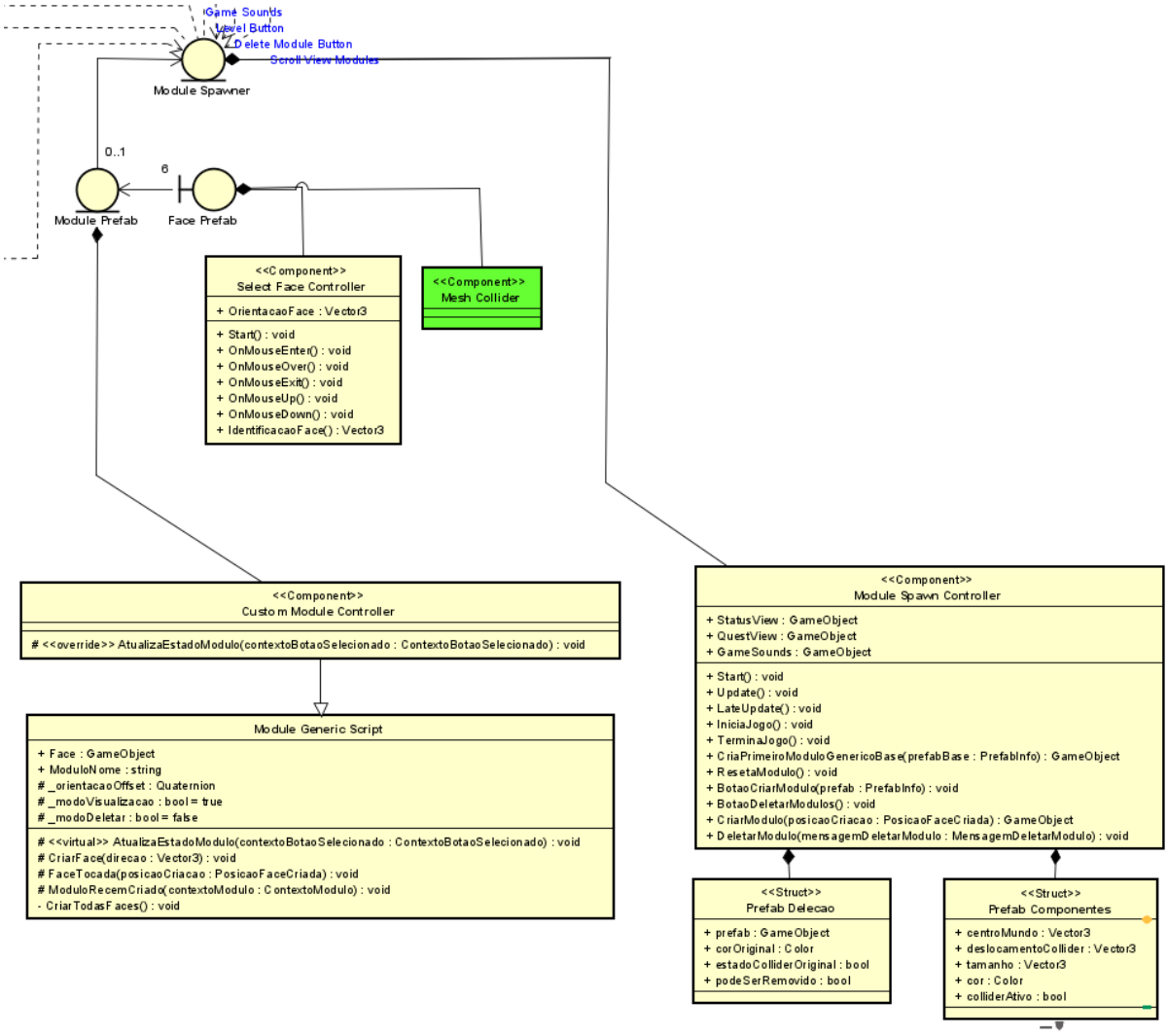

Figura 3.5: Diagrama de Classes - Module Spawner (Fonte: [14]).

para que a câmera possa seguir as instruções de interação do jogador, como realizar zoom, rotacionar ao redor do satélite e trocar o cenário de fundo do jogo, quando necessário. O diagrama correspondente pode ser visualizado na Figura 3.6.

3.4.5 Canvas

O objeto Canvas fornece os elementos da interface do usuário, que estão organizados em uma subcategoria específica no formato de hierarquia. Essa subcategoria é dividida principalmente em objetos relacionados à tela de título, *Title Screen*, à tela de jogo em execução, *On Game*, e à tela de fim do jogo, *Finish Game Screen*. Cada um dos três será agora apresentado a seguir. Também serão mostradas subseções para cada um, caso haja.

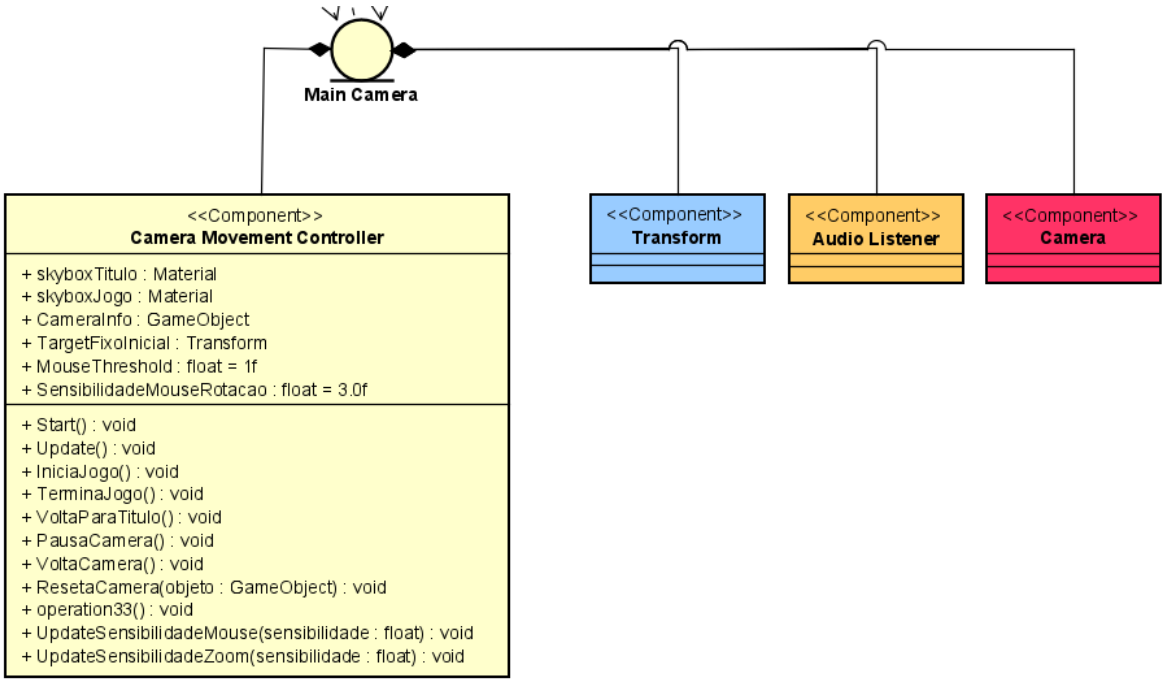

Figura 3.6: Diagrama de Classes - Main Camera (Fonte: [14]).

Title Screen

Na tela de título, encontra-se tanto o título principal quanto o painel de créditos, conforme pode ser visto na Figura 3.7. O título principal contém objetos de texto que representam o nome do jogo, a versão e a pontuação máxima alcançada. Além disso, exibe botões para iniciar, carregar e excluir o jogo, bem como os botões para excluir a pontuação, ativar o painel de créditos e sair. Todos esses botões são pré-fabricados a partir de um objeto de botão de jogo existente chamado *Game Button Prefab*, que contém funcionalidades principais para interações com o mouse. Além disso, cada botão possui o componente *Basic Sound Controller*, que se comunica com o objeto *Game Sounds* para reproduzir sons de clique. O painel de créditos consiste principalmente em objetos de título, corpo de texto e um botão de voltar, que também é pré-fabricado a partir do "Game Button Prefab".

On Game - Options Panel

Durante a execução do jogo, é possível que em algum momento o painel de opções seja ativado. O diagrama desse painel é representado na Figura 3.8 e, como pode ser observado, ele é composto por um título em forma de texto, um painel de sensibilidade que permite ajustar a sensibilidade do mouse, e os botões para retomar o jogo, voltar ao título, rever

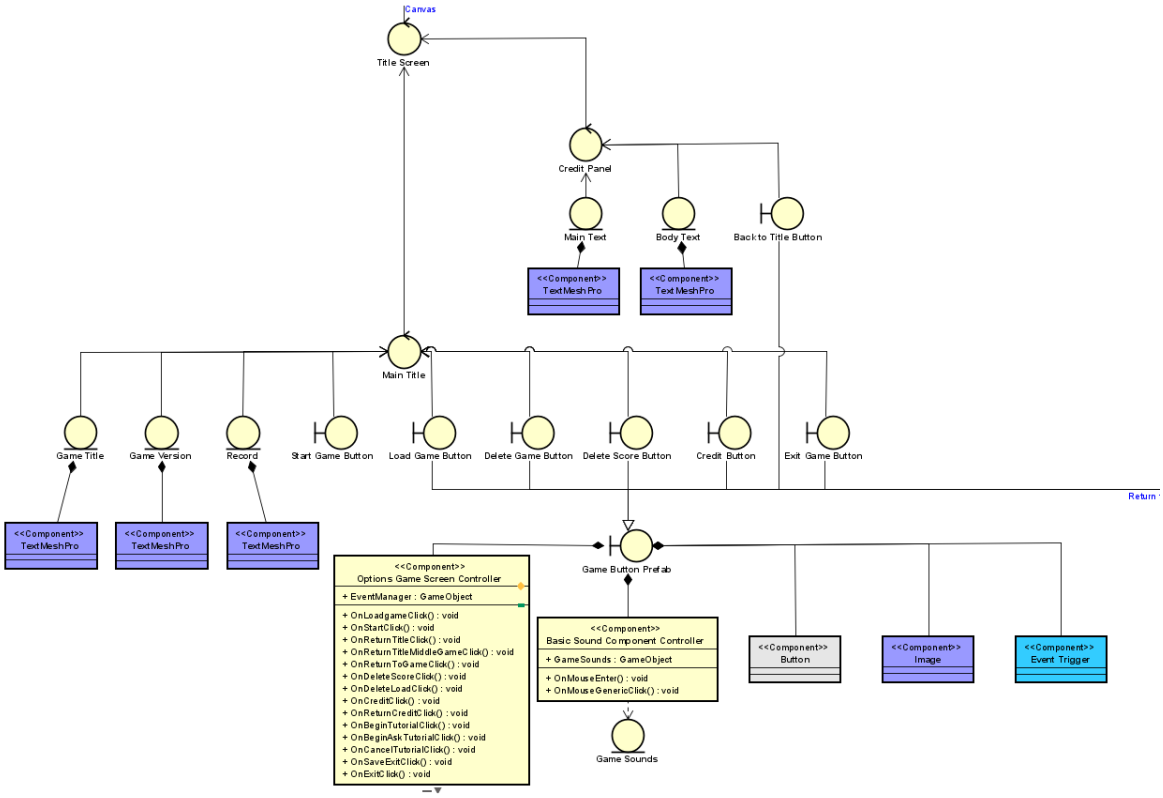

Figura 3.7: Diagrama de Classes - Title Screen (Fonte: [14]).

o tutorial, salvar e sair. Novamente, todos esses botões são pré-fabricados a partir do objeto *Game Button Prefab*. O painel de sensibilidade é controlado por meio de sliders, que permitem ajustar a sensibilidade do mouse tanto para movimento quanto para zoom.

On Game - Action

A tela *Action* representa a tela principal da partida, na qual o jogador passará a maior parte do tempo. Nessa tela, há diversos objetos importantes, incluindo o *Delete Module Button*, que, quando clicado, ativa o modo de remoção de módulos. O objeto *Camera Info* mostra informações na tela sobre o estado da câmera, como se ela está travada ou desbloqueada. Além disso, o objeto *Quest View* é responsável por exibir as missões da partida atual. O diagrama correspondente a esses três objetos pode ser visualizado na Figura 3.9.

É importante destacar que o *Delete Module Button* possui tanto um *Basic Sound Controller* quanto um componente personalizado chamado *Button Delete Controller*. O *Quest View* também possui sua própria controladora, que recebe informações sobre a atualização para o próximo cenário de missões, entre outras funcionalidades. Além disso, o *View Port* e o *Scroll Rect* são componentes internos do Unity, que permitem que o

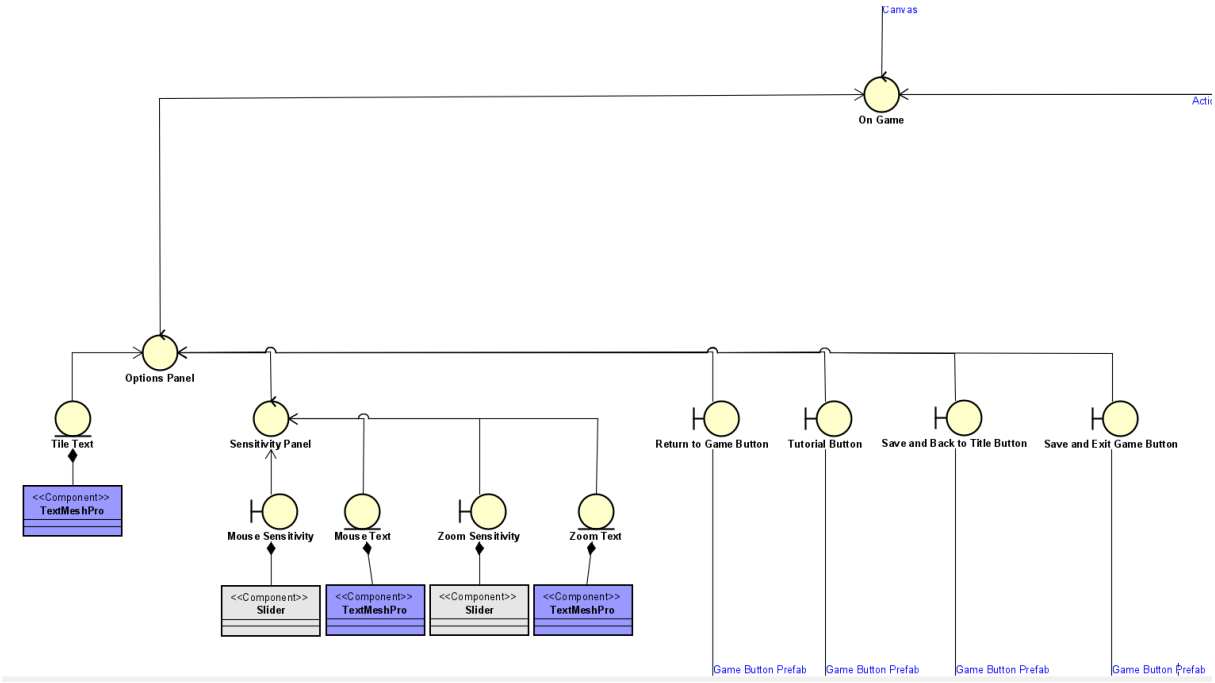

Figura 3.8: Diagrama de Classes - Game Options (Fonte: [14]).

Quest View seja exibido corretamente na tela do jogo. O componente de imagem fornece a parte gráfica do que é exibido na tela para o jogador.

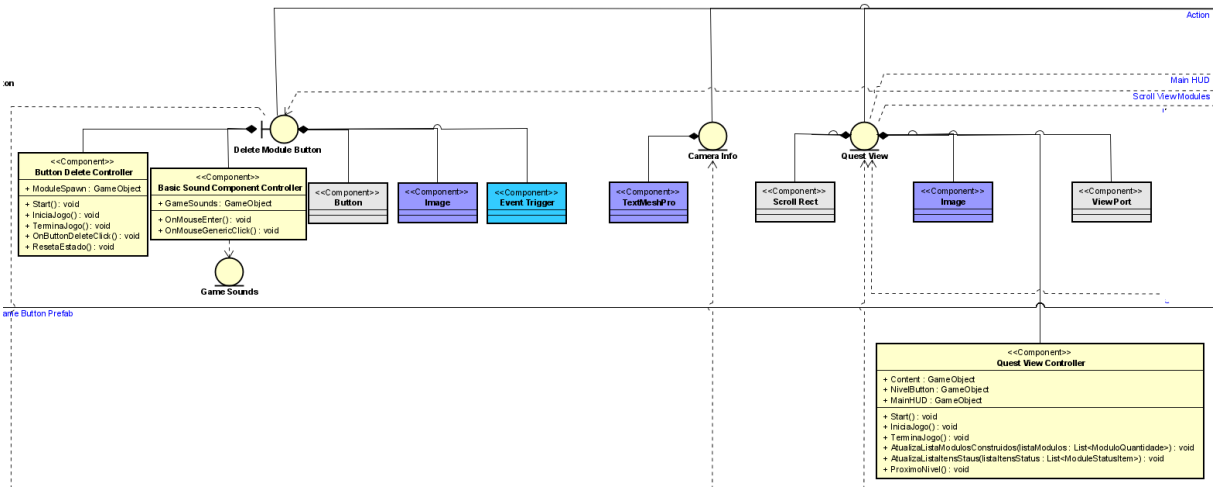

Figura 3.9: Diagrama de Classes - Action - Delete Module, Camera Info e Quest View (Fonte: [14]).

Na mesma tela de jogo, encontram-se também os objetos *Level Button* e *Status View*, como mostra a Figura 3.10. O *Level Button* é o botão responsável por levar o jogador para o próximo nível quando habilitado e clicado. Além dos componentes discutidos ante-

riormente relacionados a um botão, ele possui sua própria controladora, que é responsável por fornecer essa funcionalidade específica. Essa controladora também determina quando o botão deve estar habilitado ou não, de acordo com a lógica do jogo. O módulo *Status View* contém o texto atualizado com informações sobre o status do satélite. Sua controladora determina o que deve ser exibido na tela, de acordo com as regras e informações relevantes ao jogo.

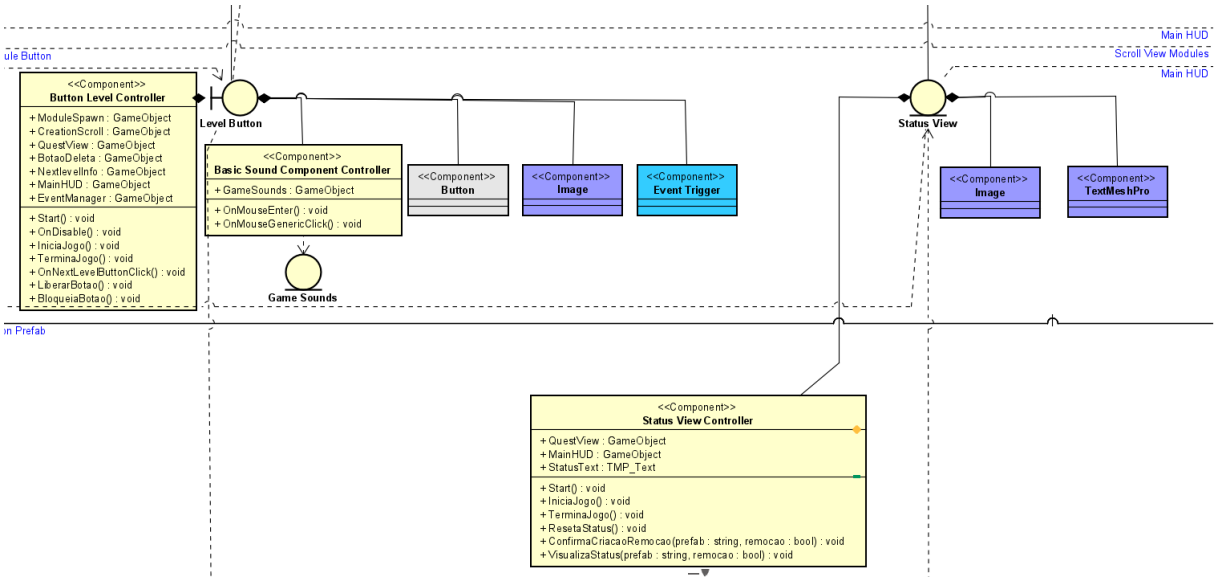

Figura 3.10: Diagrama de Classes - Action - Level Button e Status View (Fonte: [14]).

Ainda na mesma seção há o *Scroll View Modules*, que consiste de ser basicamente uma janela contendo vários botões para escolha e seleção de módulos. Como mostrado no esquemático da Figura 3.11, a controladora responsável por criar os botões de seleção e enviar as informações de seleção para o *Module Spawner* é a *Creation Scroll Controller*. Os botões de seleção são criados utilizando o pré-fabricado chamado *Module Buttons*, que inclui componentes como o *Basic Sound Controller* e o *Button Create Module Controller*, que determinam as ações a serem executadas quando seus objetos o qual fazem parte são pressionados. *Module Info Tooltip* é uma janela que é ativa quando o mouse passa sobre um dos botões de seleção, apresentando informações específicas sobre esse módulo.

O objeto *Options Button* é exatamente o botão que, quando clicado, ativa o painel de opções. Sua estrutura é semelhante aos demais botões, com a diferença de possuir uma controladora específica para ativar ou desativar o painel correspondente. Já o objeto *Next Level Info*, também presente na tela, consiste em um texto de parabenizações que é mostrado sempre que o jogador completa todas as quests e libera o botão de próximo nível. O diagrama correspondente a ambos os objetos pode ser encontrado na Figura 3.12.

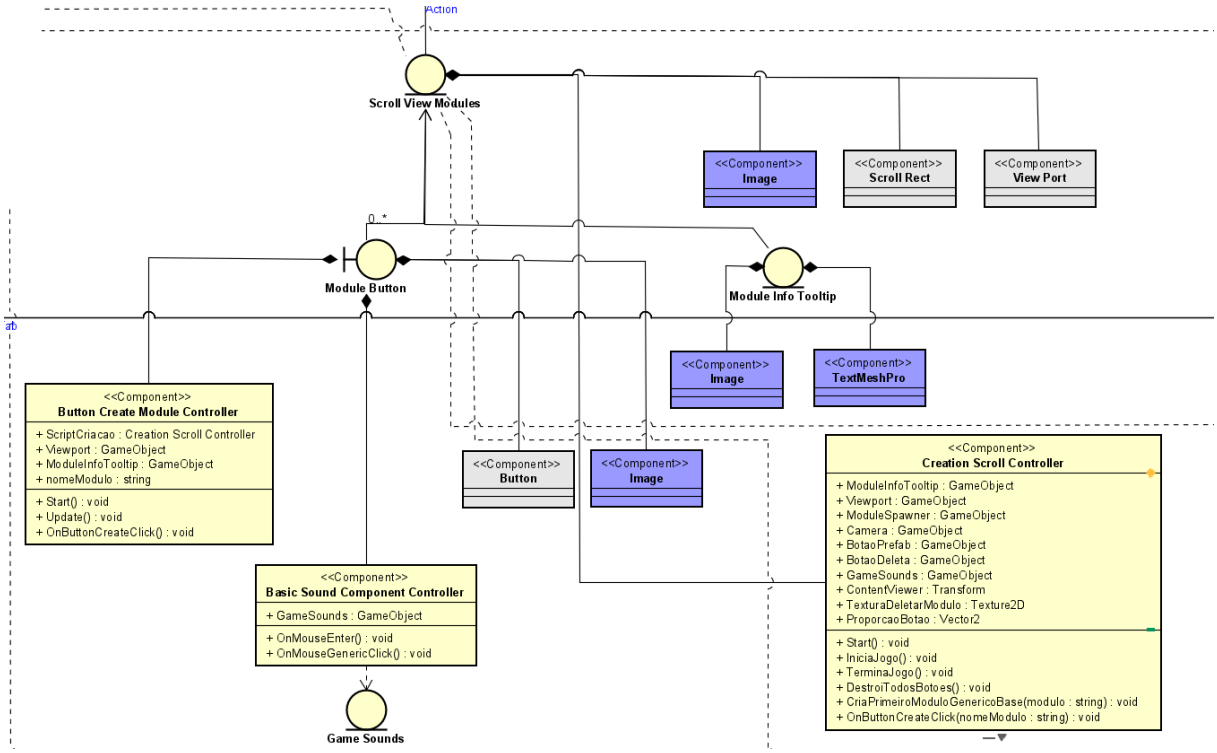

Figura 3.11: Diagrama de Classes - Action - Scroll View Modules (Fonte: [14]).

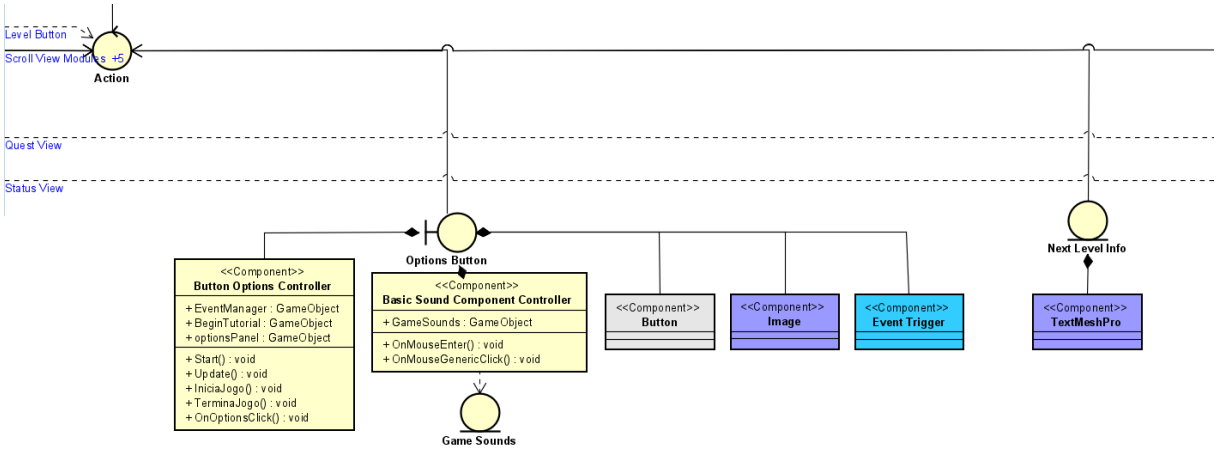

Figura 3.12: Diagrama de Classes - Action - Options Button e Next Level Info (Fonte: [14]).

Por fim, concluindo a seção *Action* de *On Game*, tem-se os objetos *Main HUD* e *Volume Button*. O *Main HUD* é o objeto que exibe informações sobre o nível e a pontuação atual do jogador. Seu componente *Main HUD Controller* monitora essas informações e é responsável por notificar outros módulos quando ocorre uma ação importante, como

alcançar o nível máximo. O *Volume Button* é um objeto de botão semelhante aos outros mencionados anteriormente. Quando clicado, ele exibe um painel com dois sliders para ajustar o volume e o som dos efeitos do jogo. Sua controladora, especificada na Figura 3.13, é responsável por gerenciar essas funcionalidades.

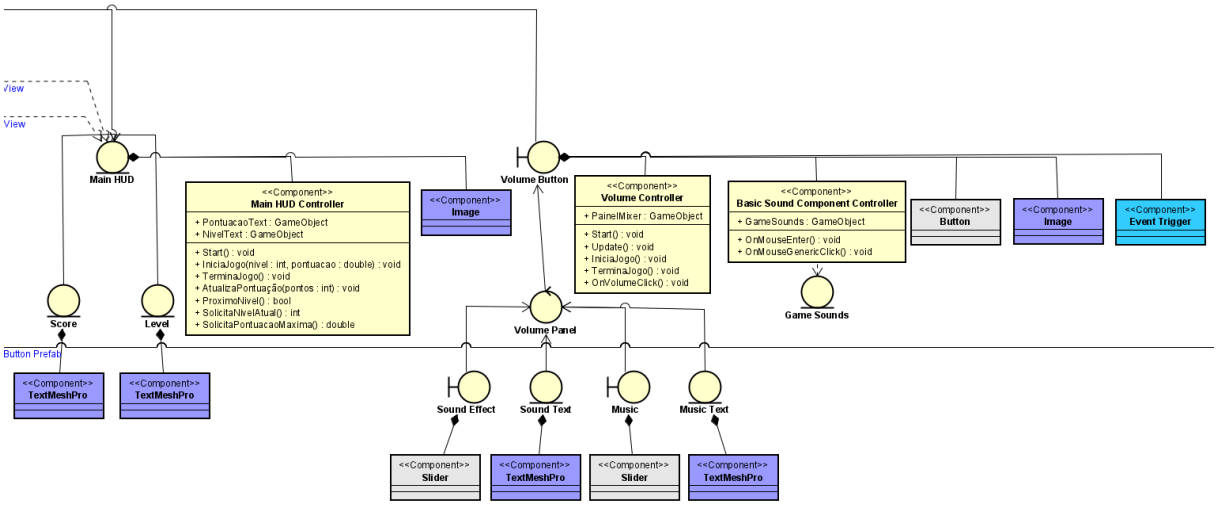

Figura 3.13: Diagrama de Classes - Action - Main HUD e Volume Button (Fonte: [14]).

On Game - Tutorial

Conforme pode ser observado no diagrama da Figura 3.14, o objeto *Tutorial* do jogo possui um painel de pergunta que, quando ativado, questiona o jogador se ele deseja ou não iniciar o tutorial. O jogador pode interagir com os dois botões "sim"e "não"que são exibidos em sequência. Ambos herdam do *Buttons Prefab*. Se o jogador optar por iniciar o tutorial, o objeto *Begin Tutorial* é ativado e sua controladora entra em ação. Essa controladora será responsável, essencialmente, por gerenciar os textos do tutorial e mostrar ou ocultar elementos de interface à medida que o tutorial avança.

Finish Screen

A tela de finalização do jogo será composta pelos textos de parabéns, pontuação máxima alcançada, pontuação anterior alcançada, e pelos botões de reiniciar o nível, voltar para a tela de título ou sair do jogo. Todos os botões, mais uma vez, herdam do *Buttons Prefab*. O diagrama correspondente pode ser encontrado na Figura 3.15. Como se trata de uma tela que contém apenas botões e texto, o único componente apresentado é o *TextMeshPro*, referente a texto.

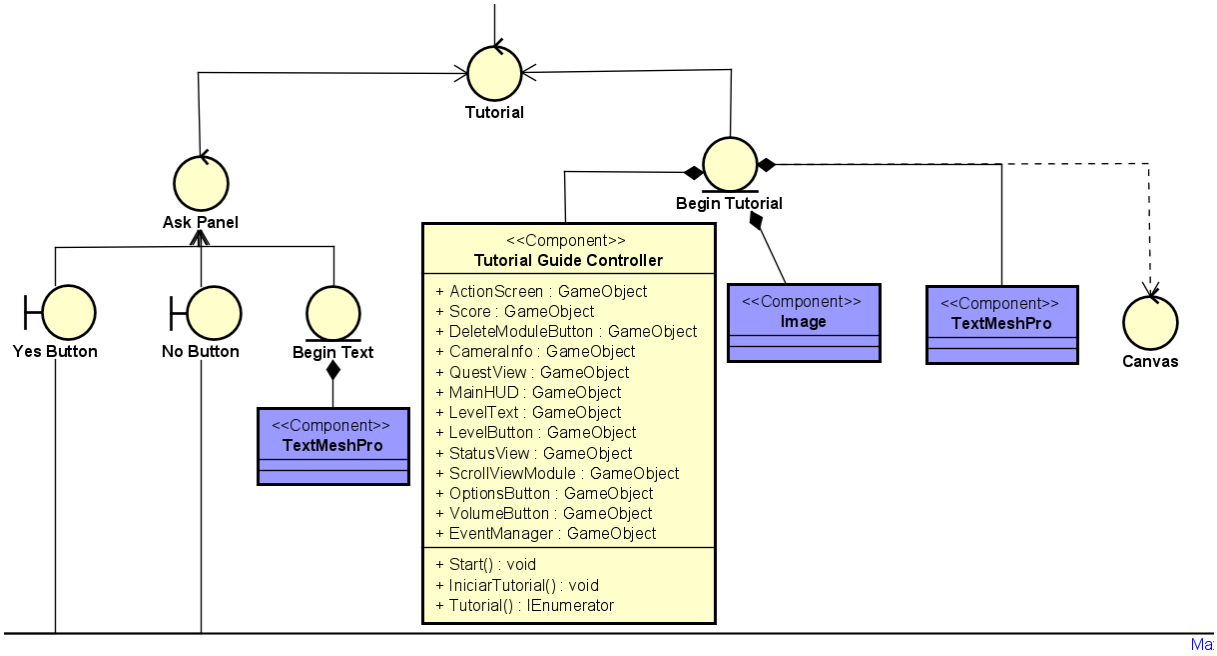

Figura 3.14: Diagrama de Classes - Tutorial (Fonte: [14]).

3.4.6 Imports

Em suma, foram criadas três bibliotecas a serem utilizadas pelo jogo: a biblioteca principal chamada de *Biblioteca*, *Modules Archive*, e *Quests Archive*. Além dessas três há também mais uma, que na verdade seria um plugin, que foi importado de fora da Unity *Quik Graph*, que é responsável por prover métodos de operação com grafos. Todas esses import podem ser visualizados nas Figura 3.16 e Figura 3.17. A biblioteca principal provê métodos genéricos e estruturas de comum uso entre os demais módulos. *Modules Archive* provê uma estruturação dos módulos do satélite, como seus respectivos nomes, funções e peculiaridade a partir de um dicionário. *Quests Archive* provê o aglomerado de quests que haverá em cada nível.

3.5 Funcionamento

Agora que se tem um melhor entendimento da estrutura do projeto, é importante destacar o funcionamento interno do jogo, no que se refere à criação e remoção de módulos do satélite do jogador. O diagrama de sequência apresentado na Figura 3.18 ilustra claramente como esse processo ocorre.

Ao receber o comando do componente *Module Spawn Controller*, uma instância de um objeto de módulo é criada. Se o jogo estiver iniciando, o primeiro módulo instanciado,

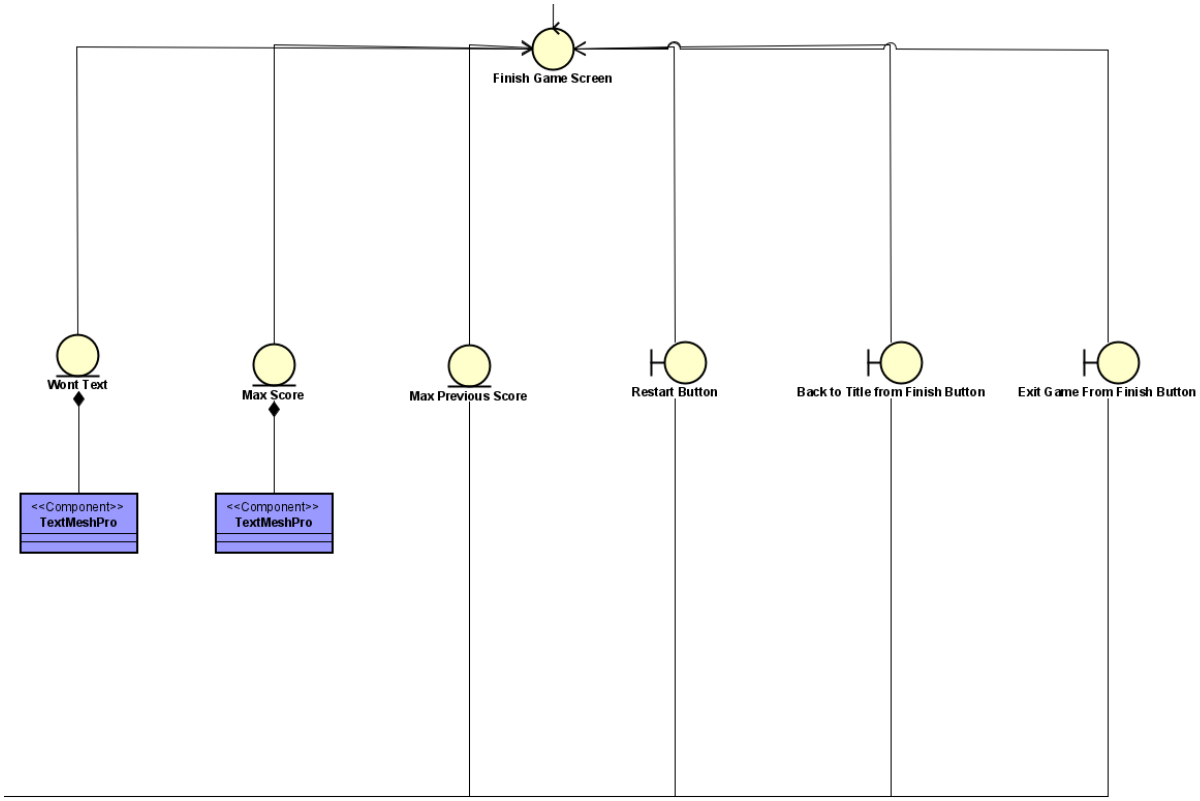

Figura 3.15: Diagrama de Classes - Finish Game Screen (Fonte: [14]).

chamado *Module 1*, será o módulo base pré-definido do jogo. Assim que o módulo é criado, o *Module Spawn Controller* envia o contexto para o componente *Custom Module Controller* do *Module 1*, que, a partir daí, consegue identificar suas características e criar as seis faces correspondentes. Como mencionado anteriormente, essas faces são responsáveis por receber as interações do mouse do jogador, como cliques, por exemplo. Cada face criada recebe o método de identificação, indicando se é uma face lateral, superior, etc.

Com todas as faces do objeto instanciadas, é possível o jogador clicar em uma delas, indicando que deseja construir um novo módulo na posição normal dessa face. Quando esse comando é recebido, a face envia a mensagem "FaceTocada()"para o componente do *Module 1*, contendo todas as informações necessárias para a criação do próximo módulo. Ao receber essa mensagem, o *Module 1* verifica o contexto atual do jogo, se é de criação ou remoção. No caso de criação, o módulo envia a mensagem para o pai, o *Module Spawn Controller*, a fim de criar um novo módulo. O *Module Spawn Controller* se certifica de que todas as condições estão adequadas para a construção e, caso estejam, instancia o *Module 2*, conforme pré-determinado pelo jogador. Esse processo se repete para os demais módulos criados.

Caso o contexto seja remoção, o *Module 1* envia a mensagem "DeletarModulo()"para

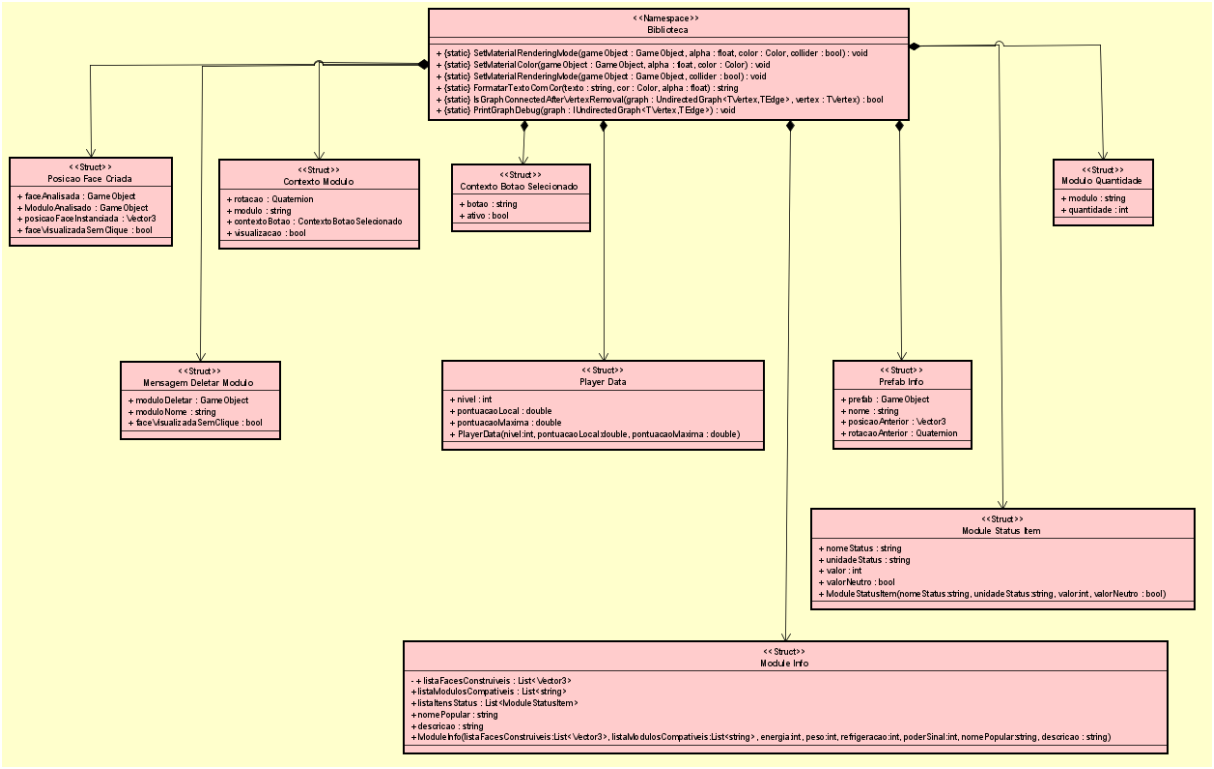

Figura 3.16: Diagrama de Classes - Biblioteca (Fonte: [14]).

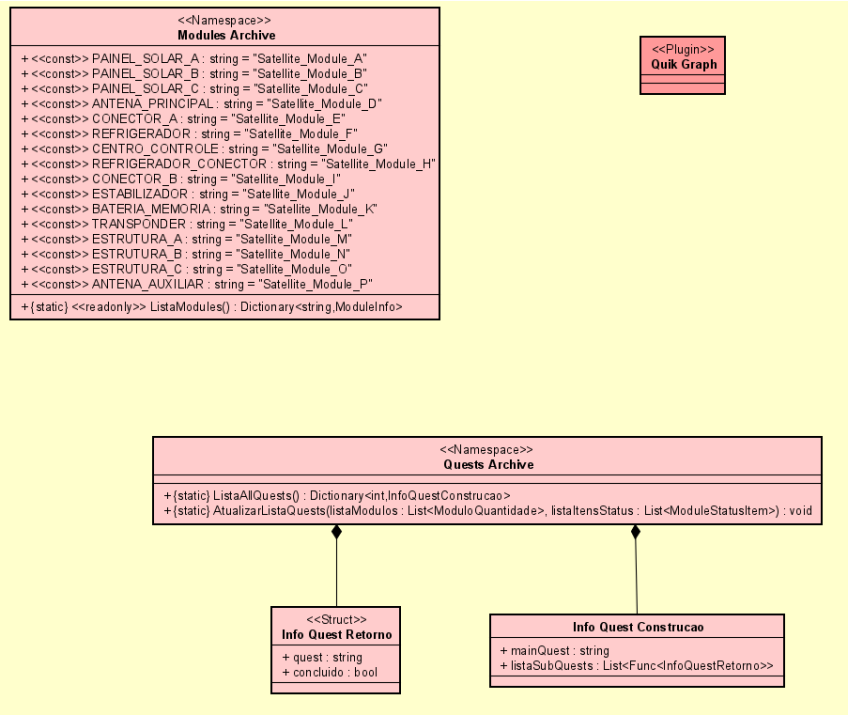

Figura 3.17: Diagrama de Classes - Modules and Quests Archive (Fonte: [14]).

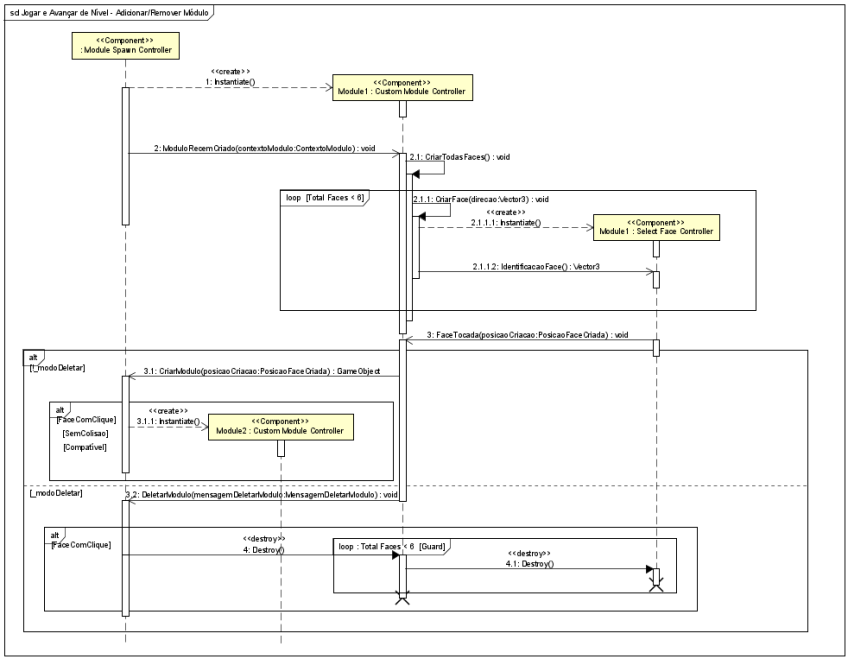

Figura 3.18: Diagrama de Sequência - Adicionar e Remover Módulos (Fonte: [14]).

o pai, o *Module Spawn Controller*. Quando esse pai recebe, confirma se as condições são satisfeitas, enviando o comando para o módulo se destruir. O módulo então recebe o comando e o reenvia para suas faces filhas também se destruírem. Após as faces se destruírem, ele também se destrói.

Capítulo 4

Protótipo

Para o desenvolvimento do projeto, foi utilizada a Unity Engine na versão 2021.3.20f1, juntamente com o Visual Studio Code na versão 1.78.2. A linguagem utilizada foi o C#. Sabendo disso, a seguir serão apresentados alguns trechos-chave e resultados da programação considerados importantes e essenciais para o funcionamento do jogo.

4.1 Estruturação dos Módulos

Os módulos são as partes ou componentes do satélite, cada um com suas particularidades individuais. Para garantir um desenvolvimento limpo e uma manutenção futura fácil, eles foram implementados utilizando estruturas e dicionários. O primeiro bloco de código a ser apresentado é o *ModuleStatusItem*. Essa estrutura determina como o quadro de status será afetado após a construção ou remoção de um determinado módulo. O trecho de código pode ser visto a seguir:

```
1  public struct ModuleStatusItem
2      {
3          public string nomeStatus;
4          public string unidadeStatus;
5          public int valor;
6          public bool valorNeutro;
7
8          public ModuleStatusItem(string nomeStatus, string unidadeStatus,
               int valor, bool valorNeutro)
9          {
10             this.nomeStatus = nomeStatus;
11             this.unidadeStatus = unidadeStatus;
12             this.valor = valor;
13             this.valorNeutro = valorNeutro;
14         }
```

```
    }
```

Listing 4.1: Estrutura ModuleStatusItem

O campo *nomeStatus* corresponde ao nome do status que será afetado. O campo *unidadeStatus* indica a unidade de medida desse status, como por exemplo Watts ou decibéis. O campo *valor* representa o valor atual desse status, enquanto o booleano *valorNeutro* indica se o impacto desse status será positivo, negativo ou neutro para o satélite. Por exemplo, foi decidido que o status *peso* com o valor x para um módulo não terá influência positiva ou negativa, tornando-o um tipo de status *neutro*. A próxima estrutura que será apresentada é aquela que realmente representa uma unidade de módulo:

```
public struct ModuleInfo
    {
        public List<Vector3> listaFacesConstruiveis;
        public List<string> listaModulosCompativeis;
        public List<ModuleStatusItem> listaItensStatus;
        public string nomePopular;
        public string descricao;

        public ModuleInfo(List<Vector3> listaFacesConstruiveis, List<
            string> listaModulosCompativeis = null, int energia = 0, int
            peso = 0, int refrigeracao = 0, int poderSinal = 0, string
            nomePopular = "", string descricao = "")
        {
            string UnidadeEnergia = "Wats";
            string unidadePeso = "Kg";
            string unidadeRefrigeracao = "Wats";
            string unidadeSinal = "dB";

            this.listaFacesConstruiveis = listaFacesConstruiveis;
            this.nomePopular = nomePopular;
            this.descricao = descricao;
            this.listaModulosCompativeis = listaModulosCompativeis;
            this.listaItensStatus = new List<ModuleStatusItem>
            {
                new ModuleStatusItem(Biblioteca.Constantes.
                    ENERGIA_DISPONIVEL, UnidadeEnergia, energia, false),
                new ModuleStatusItem(Biblioteca.Constantes.PESO,
                    unidadePeso, peso, true),
                new ModuleStatusItem(Biblioteca.Constantes.REFRIGERACAO,
                     unidadeRefrigeracao, refrigeracao, false),
                new ModuleStatusItem(Biblioteca.Constantes.PODER_SINAL,
                    unidadeSinal, poderSinal, false),
```

```
            };
        }
    }
```

Listing 4.2: Estrutura ModuleInfo

Como pode ser observado, essa estrutura contém a estrutura anteriormente apresentada, *ModuleStatusItem*, juntamente com outras novas definições. A lista *listaFacesConstruíveis* consiste nas direções normais das faces de um módulo, representando as faces em que é possível realizar construções subsequentes. Por exemplo, uma antena pode permitir a construção apenas em sua face de base, enquanto as outras faces são inacessíveis. A face de base, nesse contexto, seria representada por um vetor *Vector3.down*.

A lista *listaMódulosCompatíveis* representa os módulos que são compatíveis com a peça em questão, permitindo que apenas esses módulos sejam construídos de forma subsequente e adjacente. A lista *listaItensStatus* é composta por todos os status que são impactados pela presença ou ausência do módulo em questão. O campo *nomePopular* é o nome simplificado do módulo que será exibido ao jogador. Já a string *descrição* contém informações detalhadas sobre o módulo, fornecendo dicas sobre sua funcionalidade e uso.

Por fim, a estrutura é finalizada com a apresentação do construtor genérico base, comum a todos os módulos. É importante observar que é nesse ponto que todos os status que serão exibidos ao jogador são criados, juntamente com suas respectivas unidades de medida. Agora, será apresentado o trecho de código do dicionário utilizado para acessar um determinado módulo dentro do jogo:

```
public static readonly Dictionary<string, ModuleInfo> ListaModules = new
    Dictionary<string, ModuleInfo>()
        {
            { ModulesArchive.Constantes.PAINEL_SOLAR_A, new ModuleInfo(
               new List<Vector3>()
                {
                    Vector3.back
                },
                new List<string>()
                {
                    ModulesArchive.Constantes.CONECTOR_B
                },
                energia: 200, peso: 50, refrigeracao: -50,
                nomePopular: "Painel Solar Tipo A",
                descricao: "Modulo responsavel por prover media
                   quantidade de energia ao satelite. Necessita de
                   conector tipo B.")
            },
```

```
15              { ModulesArchive.Constantes.CONECTOR_A, new ModuleInfo(new
                    List<Vector3>()
16                  {
17                      Vector3.up, Vector3.down
18                  },
19                  new List<string>()
20                  {
21                      ModulesArchive.Constantes.CONECTOR_A, ModulesArchive
                            .Constantes.BATERIA_MEMORIA, ModulesArchive.
                            Constantes.TRANSPONDER
22                  },
23                  energia: -5, peso: 5, refrigeracao: -5,
24                  nomePopular: "Conector Tipo A",
25                  descricao: "Modulo necessario para acoplar transponders
                        e baterias.")
26              }
27          };
```

Listing 4.3: Dicionário de Modulos

Como pôde ser observado, para simplificação, o dicionário contém apenas dois módulos: *Painel Solar Tipo A* e *Conector Tipo A*. Por exemplo, o primeiro módulo mostra que apenas a face normal *Vector3.back* pode ser utilizada para realizar construções. Além disso, ele se conecta apenas com o módulo *Conector de Tipo B* e possui uma sequência de status que são impactados, juntamente com seu nome popular e descrição. O *Conector Tipo A* segue a mesma lógica, assim como todos os outros módulos que não foram apresentados aqui. A Figura 4.1 ilustra as direções padrão da Unity para lidar com vetores, sendo o eixo Z positivo o sentido para dentro da tela.

4.2 Manipulação em Grafo

Quando um jogador começa a montar seu satélite, encaixando um módulo no outro, e deseja remover um módulo posteriormente, é importante garantir que ele não consiga remover módulos que deixem duas partes do satélite desconectadas. Para implementar essa lógica, é essencial utilizar grafos que, nesse contexto, sejam não direcionados e sem peso. A ideia da implementação é que cada módulo criado e adicionado se torne um novo vértice no grafo, enquanto as arestas representam a adjacência entre dois módulos. Se o jogador deseja realizar uma remoção, o código simula a remoção do vértice em questão e verifica se o grafo continua conectado após essa ação. O resultado encontrado é retornado como *true* ou *false*. O código a seguir exemplifica essa abordagem:

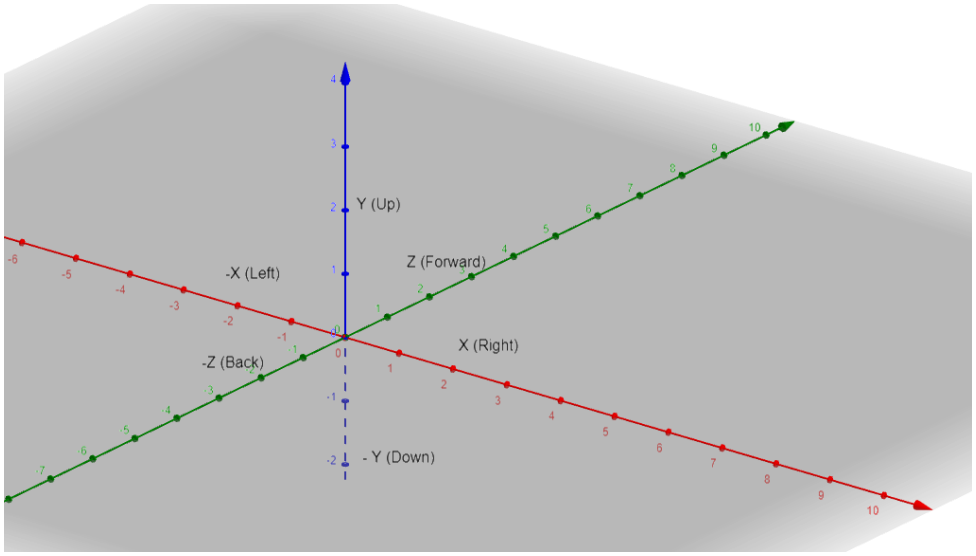

Figura 4.1: Vetores do Unity (Fonte: [15]).

```
public static bool IsGraphConnectedAfterVertexRemoval<TVertex, TEdge>(
    UndirectedGraph<TVertex, TEdge> graph, TVertex vertex)
        where TEdge : IUndirectedEdge<TVertex>
        {
            if (graph.VertexCount <= 1 || graph.Vertices.First().Equals(
                vertex)) // nao permite a remocao do primeiro elemento ou
                 quando grafo tem apenas 1 elemento.
                return false;

            // Cria uma copia do grafo sem o vertice que sera removido
            var graphWithoutVertex = graph.Clone();
            graphWithoutVertex.RemoveVertex(vertex);

            // Realiza uma busca em largura
            var visitedVertices = new HashSet<TVertex>();
            var queue = new Queue<TVertex>();
            queue.Enqueue(graphWithoutVertex.Vertices.First());
            visitedVertices.Add(queue.Peek());
            while (queue.Count > 0)
            {
                var currentVertex = queue.Dequeue();
                foreach (var neighbor in graphWithoutVertex.
                    AdjacentVertices(currentVertex))
                {
                    if (!visitedVertices.Contains(neighbor))
                    {
                        visitedVertices.Add(neighbor);
```

```
                        queue.Enqueue(neighbor);
                    }
                }
            }

            // verifica se todos os vertices foram visitados
            return visitedVertices.Count == graphWithoutVertex.
                VertexCount;
        }
```

Listing 4.4: Algoritmo para Viabilidade de Remoção

É importante destacar que é realizada uma busca em largura para verificar se todos os vértices do grafo foram visitados. Se todos os vértices forem visitados, o grafo é considerado conectado; caso contrário, é considerado desconectado. Além disso, o código também verifica se o jogador está tentando remover o módulo base do satélite, evitando que isso aconteça. Aqui está o trecho de código que conecta um vértice aos demais vértices existentes:

```
private void ConectaModuloAoGrafo(GameObject modulo)
    {
        PrefabComponentes prefabComponentes = ObtemComponentes(modulo);

        Vector3 tamanhoCaixa = (prefabComponentes.tamanho / 2) + new
            Vector3(MARGEM_CONEXAO, MARGEM_CONEXAO, MARGEM_CONEXAO);
        tamanhoCaixa *= FATOR_CONEXAO;
        //Pega elementos no eixo X
        tamanhoCaixa.x /= FATOR_CONEXAO;
        Collider[] colliders = Physics.OverlapBox(prefabComponentes.
            centroMundo, tamanhoCaixa);
        //Pega elementos no eixo Y
        tamanhoCaixa.x *= FATOR_CONEXAO;
        tamanhoCaixa.y /= FATOR_CONEXAO;
        colliders = colliders.Concat(Physics.OverlapBox(
            prefabComponentes.centroMundo, tamanhoCaixa)).ToArray();
        //Pega elementos no eixo Z
        tamanhoCaixa.y *= FATOR_CONEXAO;
        tamanhoCaixa.z /= FATOR_CONEXAO;
        colliders = colliders.Concat(Physics.OverlapBox(
            prefabComponentes.centroMundo, tamanhoCaixa)).ToArray();

        int moduloID = modulo.GetInstanceID();
        if (!_grafoConectividadeModulos.ContainsVertex(moduloID))
            _grafoConectividadeModulos.AddVertex(moduloID);
```

```
22          foreach (Collider col in colliders)
23          {
24              GameObject obj = col.gameObject;
25              if (obj.tag == Biblioteca.Constantes.TAG_MODULE) //ignore
                    LODs
26              {
27                  int colId = col.gameObject.GetInstanceID();
28                  if (colId != moduloID)
29                  {
30                      if (!_grafoConectividadeModulos.ContainsVertex(colId
                            ))
31                          _grafoConectividadeModulos.AddVertex(colId);
32
33                      int source = Mathf.Min(colId, moduloID);
34                      int target = Mathf.Max(colId, moduloID);
35                      var edge = new UndirectedEdge<int>(source, target);
36                      if (!_grafoConectividadeModulos.ContainsEdge(edge))
37                          _grafoConectividadeModulos.AddEdge(edge);
38                  }
39              }
40          }
41      }
```

Listing 4.5: Algoritmo para Adicionar o Módulo ao Grafo

É notável o uso de caixas invisíveis de colisão para verificar se um módulo está adjacente a outro, conforme ilustra a Figura 4.2. Quando um módulo é adicionado ao satélite, uma caixa invisível, do mesmo tamanho do módulo acrescido da *MARGEM_CONEXAO*, é criada. Essa caixa invisível cresce individualmente ao longo dos três eixos tridimensionais, conforme ilustrado pelos cubos vermelhos na mesma figura. À medida que a caixa invisível cresce, ela verifica se existem outros módulos adjacentes ao longo desse eixo, que, se houver, são adicionados à matriz de colisores (colliders). Após esse processo ser realizado para cada um dos três eixos, verifica-se se os vértices (módulos) detectados já existem. Em caso afirmativo, é verificado se a conexão entre eles já existe. Se a conexão não existir, uma nova aresta é adicionada ao grafo.

4.3 Encaixe e Rotacionamento Automático

Para que um módulo seja posicionado corretamente no satélite, é essencial calcular sua posição precisa no espaço. Inicialmente, sabe-se apenas que ele estará localizado na posição diretamente normal à face clicada pelo jogador. Para realizar esse cálculo, é necessário utilizar o componente de colisão do módulo, pois é a partir desse componente que é possí-

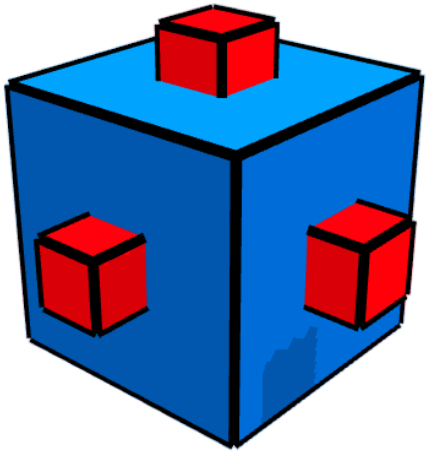

Figura 4.2: Conector de Módulos (Fonte: [16]).

vel obter informações, como suas dimensões. O trecho de código a seguir demonstra como a nova posição é calculada de forma precisa:

```
1  private Vector3 CalcularNovaPosicaoPelaFace(GameObject modulo)
2      {
3
4          PrefabComponentes prefabComponentes = ObtemComponentes(modulo);
5
6          SelectFaceScript scriptFace = _posicaoCriacao.faceAnalisada.
               GetComponent<SelectFaceScript>();
7          Vector3 offSetProxModulo = Vector3.zero;
8          if (scriptFace != null)
9              offSetProxModulo = Vector3.Scale(scriptFace.OrientacaoFace,
                   prefabComponentes.tamanho / 2);
10
11         Vector3 novaPosicao = _posicaoCriacao.posicaoFaceInstanciada +
               offSetProxModulo + _memoriaRotacao;
12
13         // Ajusta possivel diferenca entre modulos
14         if (_posicaoCriacao.ModuloAnalisado != null)
15         {
16             PrefabComponentes prefabComponentesModuloAnterior =
                   ObtemComponentes(_posicaoCriacao.ModuloAnalisado);
17             novaPosicao -= prefabComponentes.deslocamentoCollider -
                   prefabComponentesModuloAnterior.deslocamentoCollider;
18         }
19
20         return novaPosicao;
```

```
21      }
```

Listing 4.6: Algoritmo para Calcular a Posição de Criação do Módulo

Observa-se que, inicialmente, são obtidos os componentes do módulo que será construído. Em seguida, é necessário obter a orientação, ou seja, o vetor normal da face na qual o jogador clicou para realizar a construção. Ao ter acesso às dimensões do módulo, dividindo-as por dois e realizando o produto vetorial com a orientação mencionada, obtém-se um deslocamento (offset) a ser somado ao centro do colisor do objeto clicado através da face. Essa soma, juntamente com as informações de rotação (caso o módulo tenha sido rotacionado), define a nova posição em que o módulo deve ser colocado. Se houver uma diferença de deslocamento entre o colisor do objeto clicado e o colisor do novo objeto, é realizada a subtração dessa diferença da nova posição calculada. A Figura 4.3 ilustra essa nova posição de forma visual.

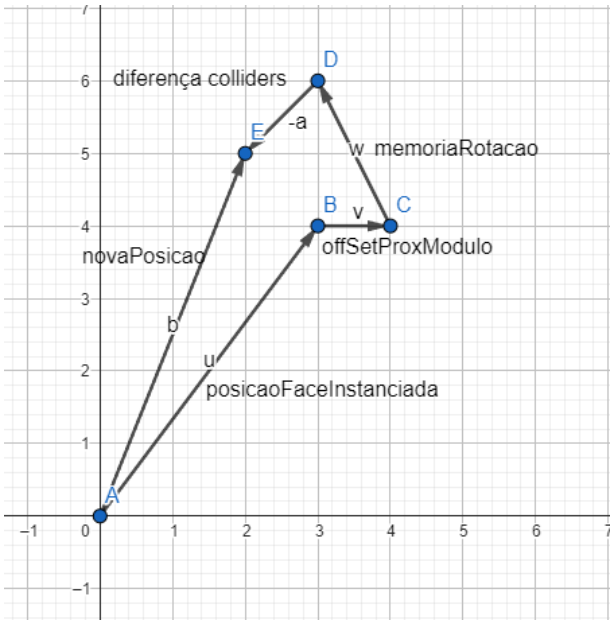

Figura 4.3: Exemplo da Nova Posição (Fonte: [15]).

A seguir, será abordado a rotação automática do módulo. Essa funcionalidade ocorre quando o jogador seleciona um módulo e deseja colocá-lo em uma face específica. Nesse caso, o módulo é rotacionado automaticamente de forma a posicionar a face construível no sentido oposto à face construível do módulo já existente. O método responsável por essa rotação automática é o seguinte:

```
1  private void AjustaRotacaoPelaFaceSelecionada()
2      {
```

```
        if (_prefabSimulado.prefab == null)
            return;

        Vector3 tamanhoCaixa = new Vector3(MARGEM_CONEXAO,
            MARGEM_CONEXAO, MARGEM_CONEXAO);

        BoxCollider collider = _prefabSimulado.prefab.GetComponent<
            BoxCollider>();
        bool estadoColliderOriginal = collider.enabled;

        Biblioteca.Metodos.SetMaterialRenderingMode(_prefabSimulado.
            prefab, true);

        Collider[] colliders = Physics.OverlapBox(_posicaoCriacao.
            faceAnalisada.transform.position, 2 * tamanhoCaixa);

        foreach (Collider col in colliders)
        {
            GameObject obj = col.gameObject;
            if ((obj.tag == Biblioteca.Constantes.TAG_FACE) && (obj !=
                _posicaoCriacao.faceAnalisada))
            {
                SelectFaceScript scriptFace = obj.GetComponent<
                    SelectFaceScript>();
                if (scriptFace != null)
                {
                    Vector3 OrientacaoFace = scriptFace.OrientacaoFace;
                    List<Vector3> listaFaces = ModulesArchive.Metodos.
                        ListaModules[_prefabSimulado.nome].
                        listaFacesConstruiveis;
                    if (listaFaces != null && !listaFaces.Contains(
                        OrientacaoFace))
                    {
                        Quaternion rotacao = Quaternion.FromToRotation(
                            listaFaces[0], OrientacaoFace);
                        _pendenciaRotacaoSimulado = rotacao; //
                            necessario para dar tempo do collider ser
                            gerado corretamente antes
                    }
                }
            }
        }

        Biblioteca.Metodos.SetMaterialRenderingMode(_prefabSimulado.
            prefab, estadoColliderOriginal);
```

```
35      }
```

Listing 4.7: Algoritmo para Ajustar a Rotação do Módulo Automaticamente

Como pode ser observado, também é utilizado o conceito de caixas invisíveis. Essas caixas são criadas na face em que se deseja construir o módulo. Essa caixa captura a normal da outra face do módulo já construído, obtendo assim sua orientação. Em seguida, o método acessa o dicionário de módulos e obtém a lista de normais que o módulo a ser construído permite conectar. Se o vetor normal obtido anteriormente não estiver presente nessa lista, significa que o módulo precisa ser rotacionado para que a face construível fique na posição correta. Isso é realizado utilizando um método específico do Unity chamado *FromToRotation*, que calcula a rotação necessária para posicionar o módulo corretamente. A nova rotação calculada é então armazenada em uma variável de rotação pendente, para ser aplicada no próximo quadro.

4.4 Funcionamento das Missões

Foram criados dois tipos de missões base para serem concluídas: *QuestTipoA* e *QuestTipoB*. A primeira determina qual módulo e quantos desses módulos devem estar presentes no satélite para que a missão seja concluída. A segunda indica qual status deve ser alcançado ou qual limite máximo esse status pode atingir para que a missão seja concluída. Por exemplo, pode ser necessário que a energia total do satélite seja igual ou superior a 200 Watts, ou então que o peso total do satélite seja inferior a 2000 kg. O tipo de limite a ser considerado será determinado pelo booleano *valorMaximo*. O trecho de código correspondente a ambos os tipos de missões está apresentado abaixo:

```
1  private static InfoQuestRetorno QuestTipoA(List<ModuloQuantidade>
       listaModulos, string modulo, int quantidade)
2          {
3              InfoQuestRetorno infoQuest;
4              infoQuest.concluido = false;
5              ModuloQuantidade moduloQuantidade = listaModulos.Find(x => x
                   .modulo.Equals(modulo));
6              int quantidadeAtual = 0;
7              if (moduloQuantidade.modulo != null)
8                  quantidadeAtual = moduloQuantidade.quantidade;
9              infoQuest.quest = quantidadeAtual + "/" + quantidade + "
                   modulo \"" + ModulesArchive.Metodos.ListaModules[modulo].
                   nomePopular + "\"\n";
10             if (quantidadeAtual >= quantidade)
11             {
```

```
                infoQuest.quest = Biblioteca.Metodos.FormatarTextoComCor
                    (infoQuest.quest, Color.green, 1f);
                infoQuest.concluido = true;
            }
            return infoQuest;
        }
```

Listing 4.8: Estrutura da Quest Tipo A

```
private static InfoQuestRetorno QuestTipoB(List<ModuleStatusItem>
    listaItensStatus, string status, int valor, bool valorMaximo)
        {
            InfoQuestRetorno infoQuest;
            infoQuest.quest = "";
            infoQuest.concluido = false;
            ModuleStatusItem moduleStatusItem = listaItensStatus.Find(x
                => x.nomeStatus.Equals(status));
            int valorAtual = 0;
            if (moduleStatusItem.nomeStatus != null)
                valorAtual = moduleStatusItem.valor;
            if (valorMaximo)
            {
                infoQuest.quest = valorAtual + " <= " + valor + " status
                    \"" + status + "\"\n";
                if (valorAtual <= valor)
                {
                    infoQuest.quest = Biblioteca.Metodos.
                        FormatarTextoComCor(infoQuest.quest, Color.green,
                         1f);
                    infoQuest.concluido = true;
                }
                else
                    infoQuest.quest = Biblioteca.Metodos.
                        FormatarTextoComCor(infoQuest.quest, Color.red, 1
                        f);
            }
            else
            {
                infoQuest.quest = valorAtual + " > " + valor + " status
                    \"" + status + "\"\n";
                if (valorAtual > valor)
                {
```

```
                    infoQuest.quest = Biblioteca.Metodos.
                        FormatarTextoComCor(infoQuest.quest, Color.green,
                         1f);
                    infoQuest.concluido = true;
                }
            }
            return infoQuest;
        }
```

Listing 4.9: Estrutura da Quest Tipo B

O retorno de cada tipo de missão é uma estrutura, conforme exemplificado abaixo. Nela, é indicado o texto da missão visível ao jogador e se ela está concluída ou não. Essas missões compõem o dicionário de missões de acordo com o nível em que o jogador se encontra. O método que contém o dicionário é sempre executado quando um módulo é adicionado ou removido, atualizando-o conforme necessário.

```
 public struct InfoQuestRetorno
    {
        public string quest;
        public bool concluido;
    }
```

Listing 4.10: Estrutura de Retorno de Quest

```
public static Dictionary<int, InfoQuestConstrucao> ListaAllQuests;

        public static void AtualizarListaQuests(List<ModuloQuantidade>
            listaModulos, List<ModuleStatusItem> listaItensStatus)
        {
            ListaAllQuests = new Dictionary<int, InfoQuestConstrucao>();

            ListaAllQuests.Add(1, new InfoQuestConstrucao
            {
                mainQuest = "Crie um satelite basico essencial que
                    contenha as seguintes pecas:",
                listaSubQuests = new List<Func<InfoQuestRetorno>>()
                {
                    () => QuestTipoA(listaModulos, ModulesArchive.
                        Constantes.CENTRO_CONTROLE, 1),
                    () => QuestTipoA(listaModulos, ModulesArchive.
                        Constantes.PAINEL_SOLAR_A, 2),
                    () => QuestTipoA(listaModulos, ModulesArchive.
                        Constantes.ANTENA_PRINCIPAL, 1),
```

```
15                      () => QuestTipoA(listaModulos, ModulesArchive.
                            Constantes.ESTABILIZADOR, 1),
16                  }
17              });
18
19              ListaAllQuests.Add(4, new InfoQuestConstrucao
20              {
21                  mainQuest = "Agora use a criatividade para montar um
                        satelite com mais de 2000 Kg de peso:",
22                  listaSubQuests = new List<Func<InfoQuestRetorno>>()
23                  {
24                      () => QuestTipoA(listaModulos, ModulesArchive.
                            Constantes.CENTRO_CONTROLE, 1),
25                      () => QuestTipoA(listaModulos, ModulesArchive.
                            Constantes.ESTABILIZADOR, 1),
26                      () => QuestTipoB(listaItensStatus, Biblioteca.
                            Constantes.ENERGIA_DISPONIVEL, 0, false),
27                      () => QuestTipoB(listaItensStatus, Biblioteca.
                            Constantes.REFRIGERACAO, 0, false),
28                      () => QuestTipoB(listaItensStatus, Biblioteca.
                            Constantes.PODER_SINAL, 0, false),
29                      () => QuestTipoB(listaItensStatus, Biblioteca.
                            Constantes.PESO, 2000, false)
30                  }
31              });
32          }
```

Listing 4.11: Dicionário para Manipulação de Quests

Acima está um exemplo das missões que compõem os níveis 1 e 4. Observa-se o uso da estrutura *InfoQuestConstrucao*, que está representada abaixo. Trata-se de uma estrutura do painel de missões para cada nível, mostrando a missão principal e, em seguida, as missões ou submissões adicionais. Como pode ser observado, as missões foram projetadas de forma a facilitar sua manutenção. É relativamente simples aumentar ou diminuir o número de níveis e suas variações de missões.

```
1  public struct InfoQuestConstrucao
2      {
3          public string mainQuest;
4          public List<Func<InfoQuestRetorno>> listaSubQuests;
5      }
```

Listing 4.12: Estrutura do Painel de Missões

4.5 Salvamento do Jogo

Existem três informações que podem ser salvas no jogo: o recorde do jogador, o nível em que ele se encontra e a pontuação local alcançada até o momento desse determinado nível. A estrutura que representa essas informações se encontra logo abaixo. Nota-se a inicialização dos atributos com o valor -1, indicando que a informação ainda não foi gravada até o momento atual. Os métodos que utilizam a estrutura mencionada também se encontram abaixo.

```
1  public struct PlayerData
2      {
3          public int nivel;
4          public double pontuacaoLocal;
5          public double pontuacaoMaxima;
6
7          public PlayerData(int nivel = -1, double pontuacaoLocal = -1,
               double pontuacaoMaxima = -1)
8          {
9              this.nivel = nivel;
10             this.pontuacaoLocal = pontuacaoLocal;
11             this.pontuacaoMaxima = pontuacaoMaxima;
12         }
13     }
```

Listing 4.13: Estrutura de Armazenamento do Jogo

```
1  public static void SavePlayer(PlayerData player, string
       tipoArquivoCaminho)
2      {
3          string caminho = Application.persistentDataPath +
               tipoArquivoCaminho;
4
5          string json = JsonUtility.ToJson(player);
6          File.WriteAllText(caminho, json);
7      }
```

Listing 4.14: Algoritmo para Salvar o Jogo

```
1  public static PlayerData LoadPlayer(string tipoArquivoCaminho)
2      {
3          string caminho = Application.persistentDataPath +
               tipoArquivoCaminho;
```

```
        PlayerData player = new PlayerData();

        if (File.Exists(caminho))
        {
            string json = File.ReadAllText(caminho);
            player = JsonUtility.FromJson<PlayerData>(json);
        }
        else
        {
            Debug.LogError("Dado nao encontrado em " + caminho);
        }

        return player;
    }
```

Listing 4.15: Algoritmo para Carregar um Jogo Salvo

```
public static void DeletePlayer(string tipoArquivoCaminho)
    {
        string caminho = Application.persistentDataPath +
            tipoArquivoCaminho;

        if (File.Exists(caminho))
            File.Delete(caminho);
    }
```

Listing 4.16: Algoritmo para Deletar um Jogo Salvo

Observa-se o uso de um arquivo simples no formato JSON para salvar as informações. Para realizar a operação de salvar, são fornecidos como argumentos o *PlayerData* e o caminho de destino, que pode ter uma extensão de /score.data ou /player.data. A primeira indica que apenas a pontuação recorde será salva, enquanto a segunda indica que apenas o nível juntamente com a pontuação alcançada até o momento será salva. Para carregar os dados, o arquivo JSON correspondente é buscado e aberto. Foi utilizada a função *Application.persistentDataPath* do Unity para formar o caminho de salvamento do jogo. Esse caminho varia de acordo com a máquina em que o jogo está sendo executado.

Capítulo 5

Caso de Uso

Será simulada agora uma partida completa do jogo, passando pelas principais funcionalidades mencionadas até então. Imagens das telas do jogo acompanharão os cenários para um melhor entendimento.

5.1 Início de Jogo

Ao iniciar o jogo, o jogador se deparará com a tela apresentada na Figura 5.1. Como observado, considerando que esta é sua primeira partida, sua pontuação recorde estará zerada e ele não terá acesso a funcionalidades como carregar um jogo salvo ou deletar a pontuação. No entanto, ele poderá ver os créditos, conforme mostrado na Figura 5.2, sair do jogo ou iniciar a partida clicando no primeiro botão. Ele escolherá iniciar a partida. Antes de prosseguir, nota-se na Figura 5.2 que o jogo utiliza algumas músicas, todas elas são de uso gratuito, assim como as artes utilizadas.

5.1.1 Tutorial

Ao iniciar a partida, o jogador encontrará uma mensagem perguntando se ele gostaria de fazer o tutorial, como mostra a Figura 5.3. Nessa simulação, ele aceitará fazer o tutorial, o que o levará à tela apresentada na Figura 5.4. Mensagens serão exibidas explicando ao jogador como o jogo funciona. Neste momento, ele não poderá fazer nada além de prosseguir com o tutorial, clicando em qualquer tecla ou com o mouse para avançar.

5.1.2 Interface

O tutorial terminará, fornecendo ao jogador a liberação da interface da partida, conforme mostrado na Figura 5.5. É possível observar a lista dos módulos na lateral esquerda, o botão do modo de remoção na lateral inferior esquerda, o quadro de status à direita,

Figura 5.1: Tela Inicial.

Figura 5.2: Painel de Créditos.

o quadro de missões no meio à direita e o botão de avançar de nível na parte inferior direita. Percebe-se que esse botão está bloqueado inicialmente, pois ainda há missões a serem concluídas. Na parte superior da tela, é exibido o nível em que o jogador se

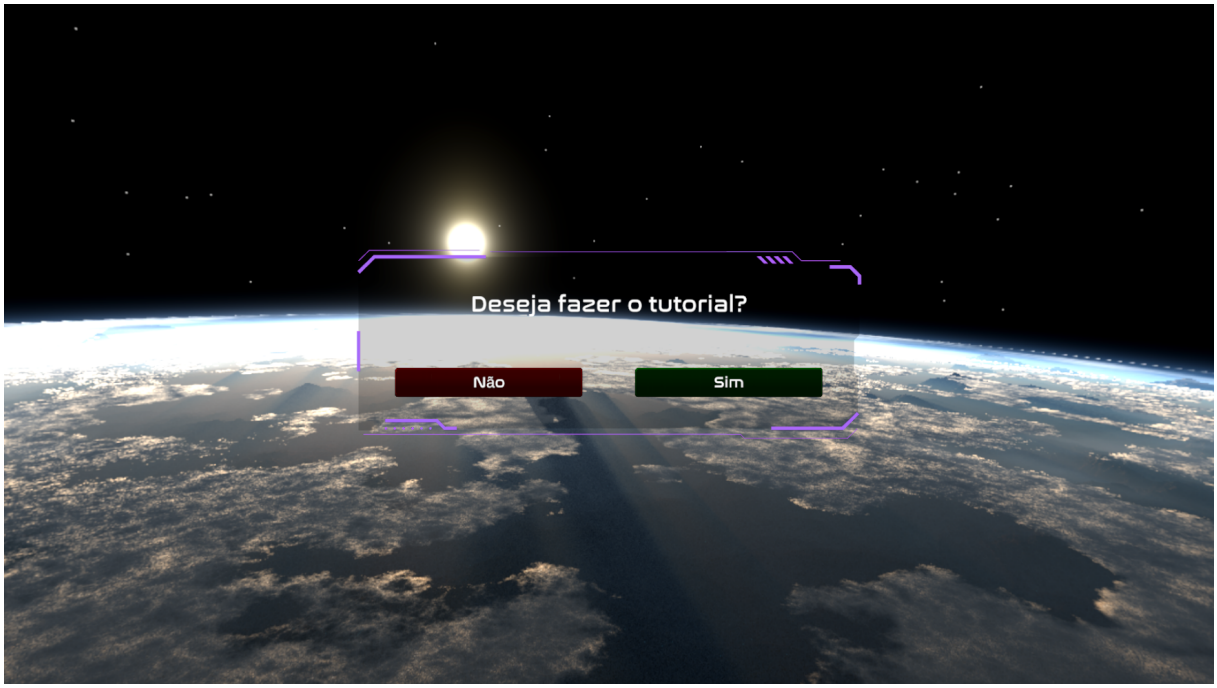

Figura 5.3: Painel de Tutorial.

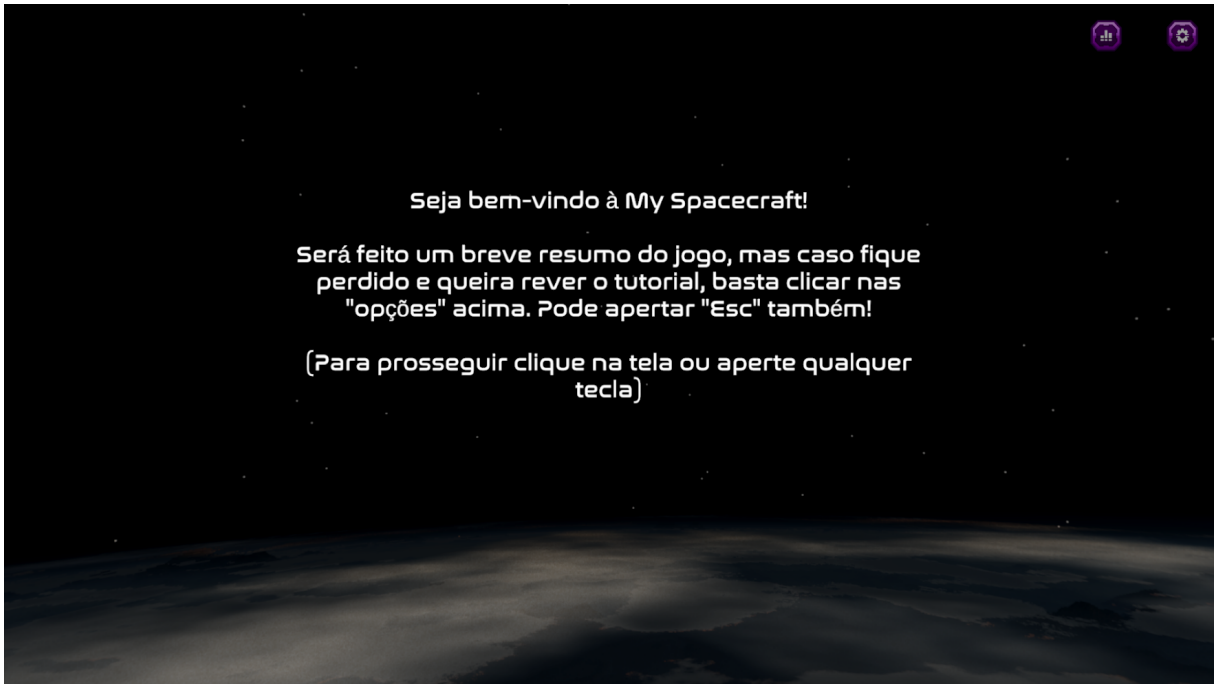

Figura 5.4: Tutorial.

encontra, sua pontuação atual, os botões para alterar o volume do jogo e o botão de engrenagem para acessar o menu de opções.

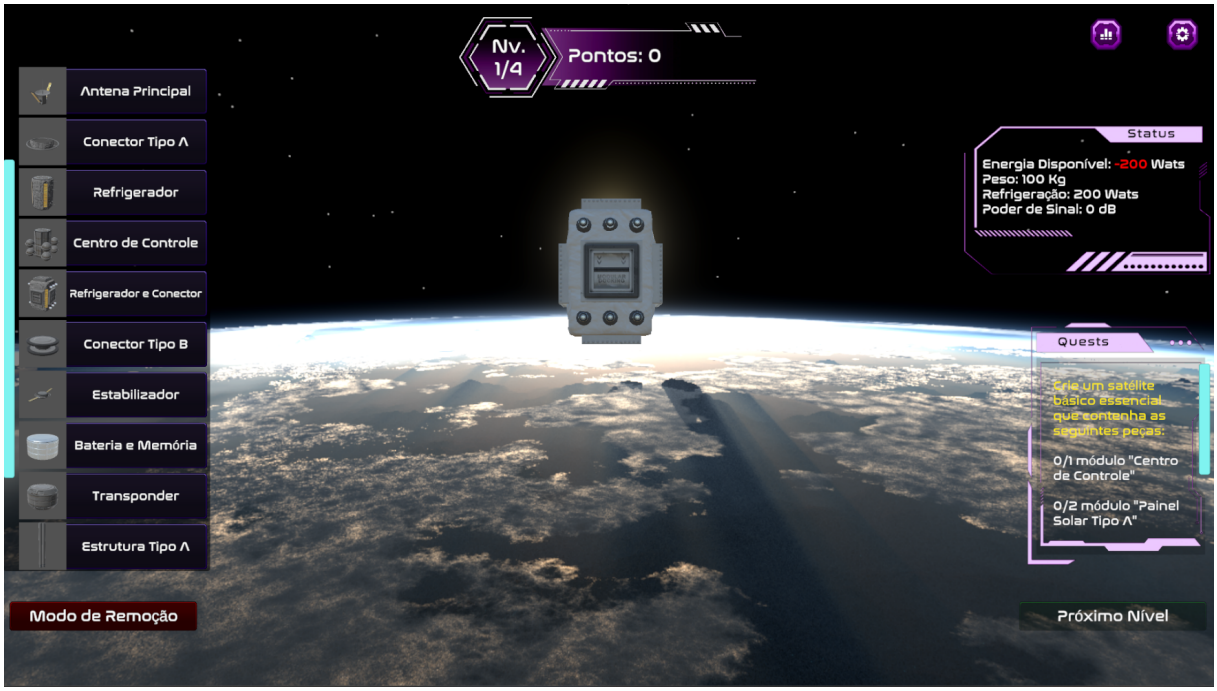

Figura 5.5: Início da Partida.

5.2 Meio de Jogo

Nessa simulação de partida, o jogador, eventualmente, achará o volume do jogo muito alto. Então, ele acessará o botão de volume para ajustar o som do jogo de acordo com sua preferência. A tela da Figura 5.6 será apresentada a ele. Após realizar essa configuração, ele lerá as instruções das missões e iniciará o processo de montagem.

5.2.1 Montagem

O jogador decidirá, a princípio, colocar um painel solar em seu satélite e logo notará que, conforme explicado no tutorial, ao passar o mouse sobre o módulo desejado, uma mensagem com mais informações sobre a peça aparecerá, como mostra a Figura 5.7. No entanto, por estar com pressa, ele resolverá ignorar essa caixa de informações, clicando diretamente no módulo para adicioná-lo ao satélite, o que o levará a notar o aparecimento da cor azul de incompatibilidade, como mostra a Figura 5.8. Ao perceber que nada acontece ao clicar com o mouse, ele compreenderá que algo está errado. Desse modo, ele terá um pouco mais de paciência para identificar o problema, e, em seguida, verá pela mensagem na caixa que o painel solar só se encaixa com um conector do tipo B. Ele procurará o conector na lista e o colocará em seu satélite, como mostra a Figura 5.9. Dessa forma, ele conseguirá adicionar o painel solar, conforme ilustrado na Figura 5.10.

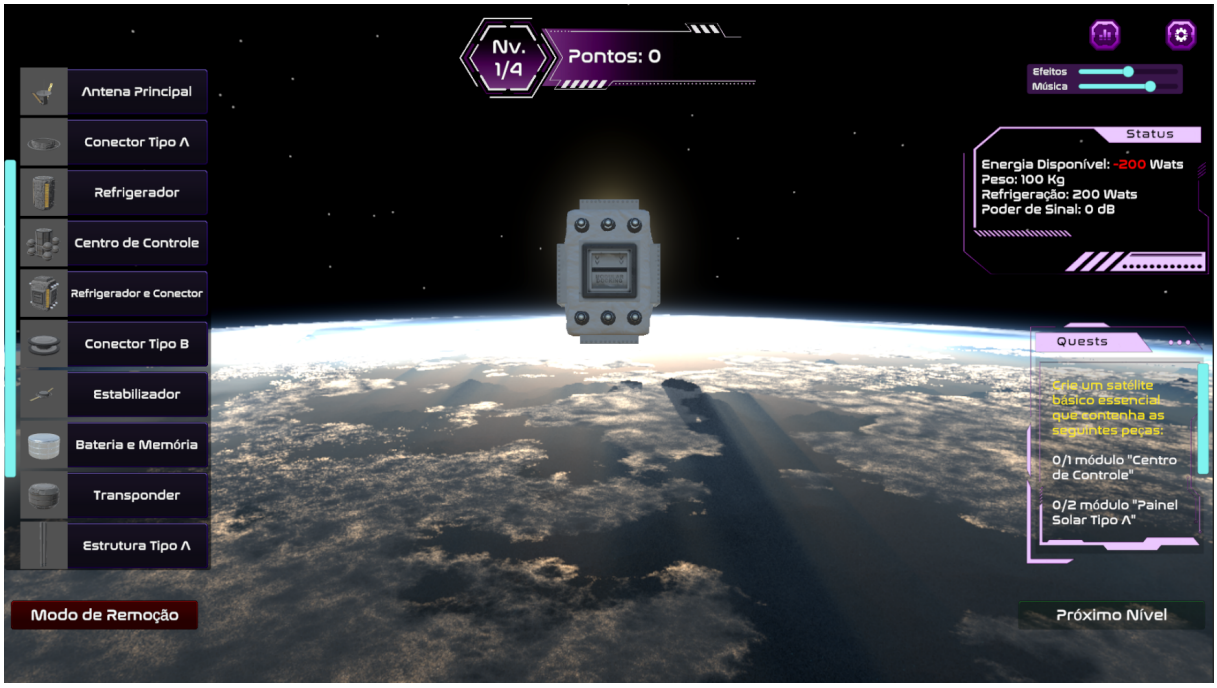

Figura 5.6: Painel de Volume no Canto Superior.

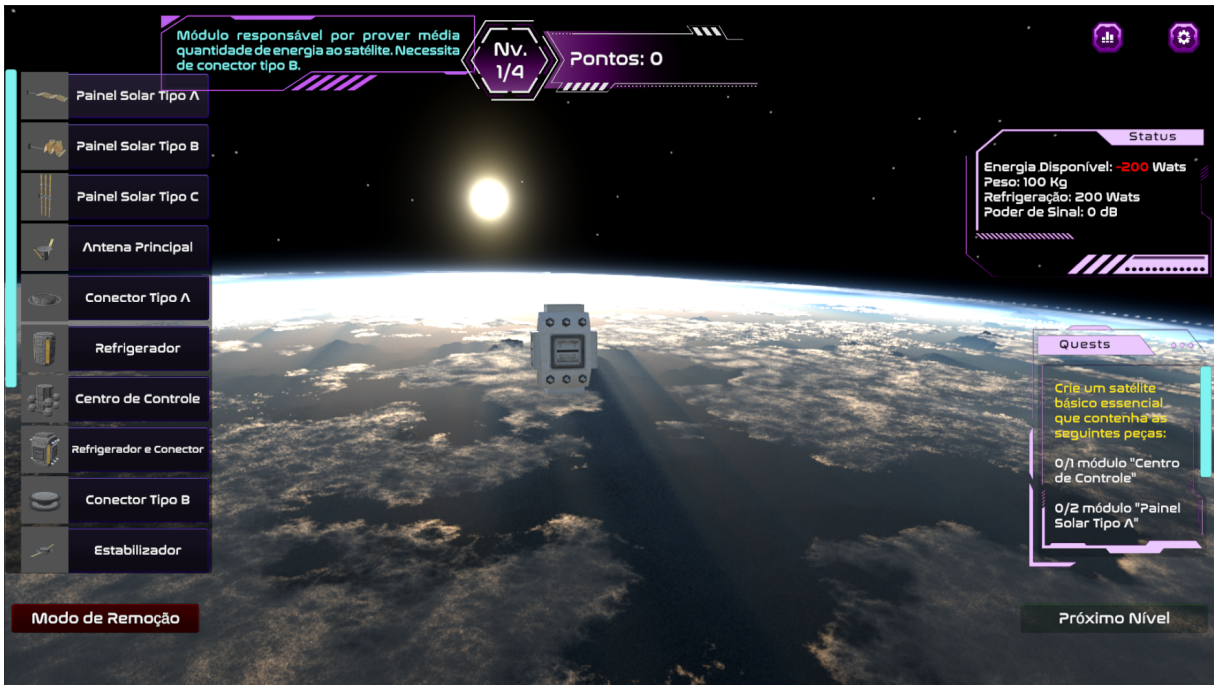

Figura 5.7: Caixa de Informações.

Durante o processo de montagem, ele percebe que adicionou mais refrigeradores do que o necessário, o que faz com que o atributo *Energia Disponível* em sua caixa de status fique em vermelho. Assim, ele decidirá remover alguns refrigeradores, clicando no botão

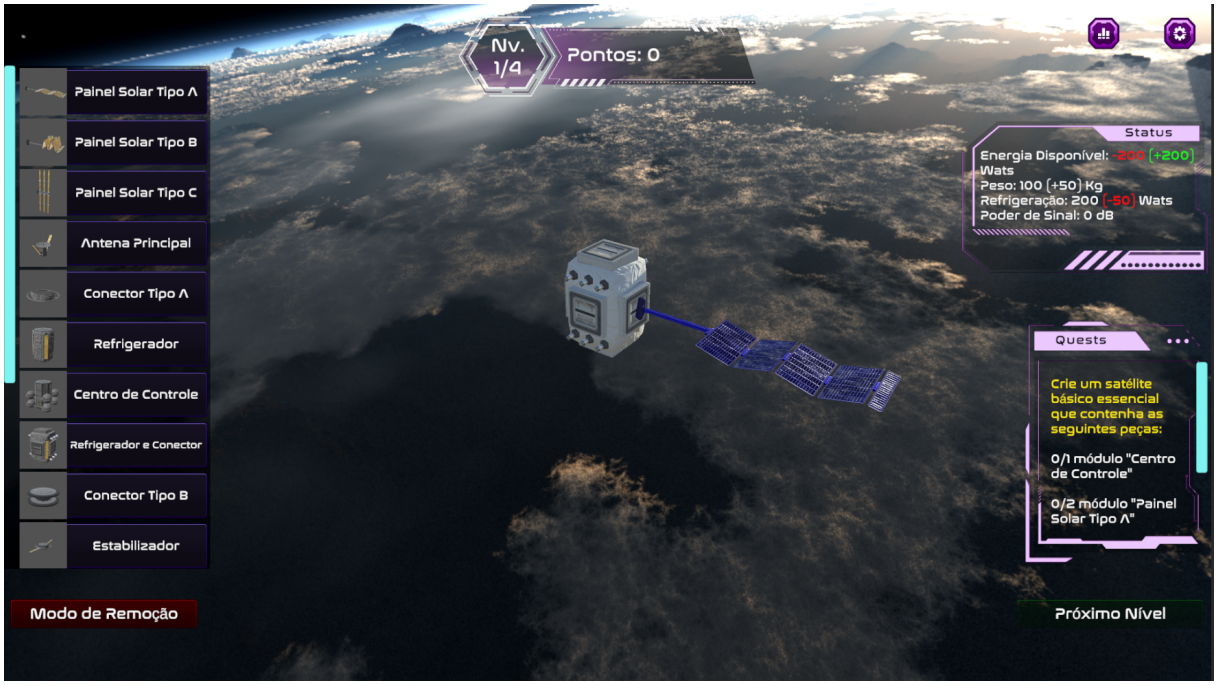

Figura 5.8: Incompatibilidade entre Módulos.

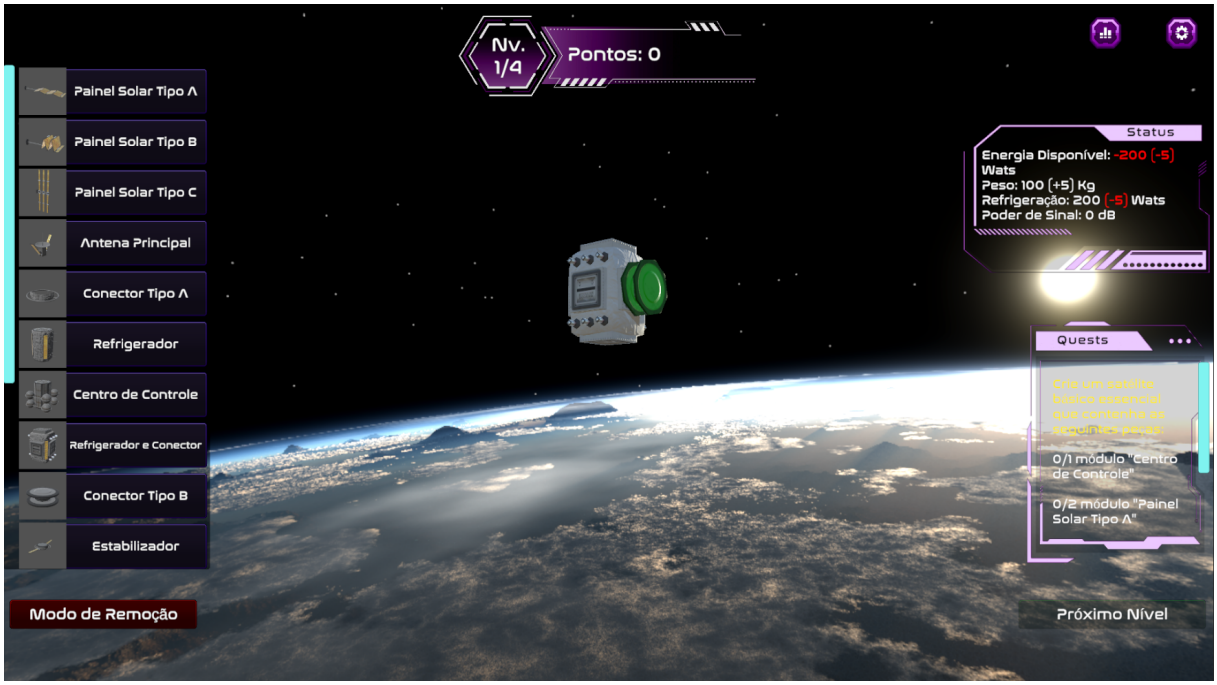

Figura 5.9: Compatibilidade entre Módulos.

de remoção e, em seguida, selecionando os módulos a serem removidos, como mostrado na Figura 5.11. Por fim, o jogador conseguirá concluir com sucesso o primeiro nível. Uma mensagem de parabéns aparecerá em sua tela e o botão de próximo nível será

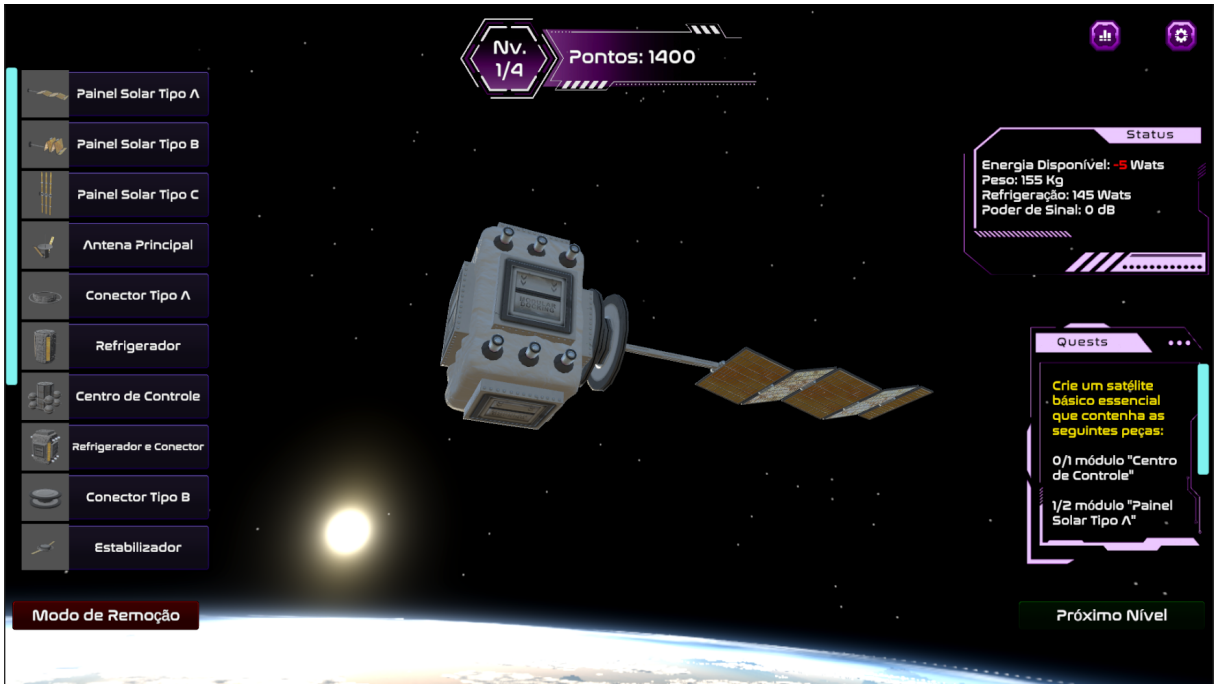

Figura 5.10: Exemplo de Módulo Encaixado.

desbloqueado, como mostra a Figura 5.12. Ele avançará para o próximo nível clicando nesse botão e perceberá que as missões e o status serão atualizados, e todos os blocos criados anteriormente desaparecerão, deixando-o novamente apenas com o módulo base compondo seu satélite.

5.2.2 Opções

Ao avançar para o próximo nível, o jogador perceberá que algo o incomodava desde o início: a sensibilidade usada para rotacionar o satélite. Ele identificará a existência de um menu de opções e clicará no botão correspondente, levando-o à tela apresentada na Figura 5.13. Ele ajustará a sensibilidade de acordo com sua preferência, mas descobrirá que, na verdade, o problema estava em seu mouse o tempo todo. Portanto, ele precisará sair do jogo para comprar um novo mouse e continuar jogando *My Spacecraft*. No entanto, antes de fazer isso, ele salvará sua partida atual, clicando no último botão para salvar e sair.

Ao resolver seus problemas e retornar para iniciar o jogo novamente, ele notará que, desta vez, o botão de carregar o jogo salvo estará desbloqueado, como mostra a Figura 5.14. Clicando nesse botão, ele será redirecionado para seu jogo salvo no nível dois, onde estava anteriormente.

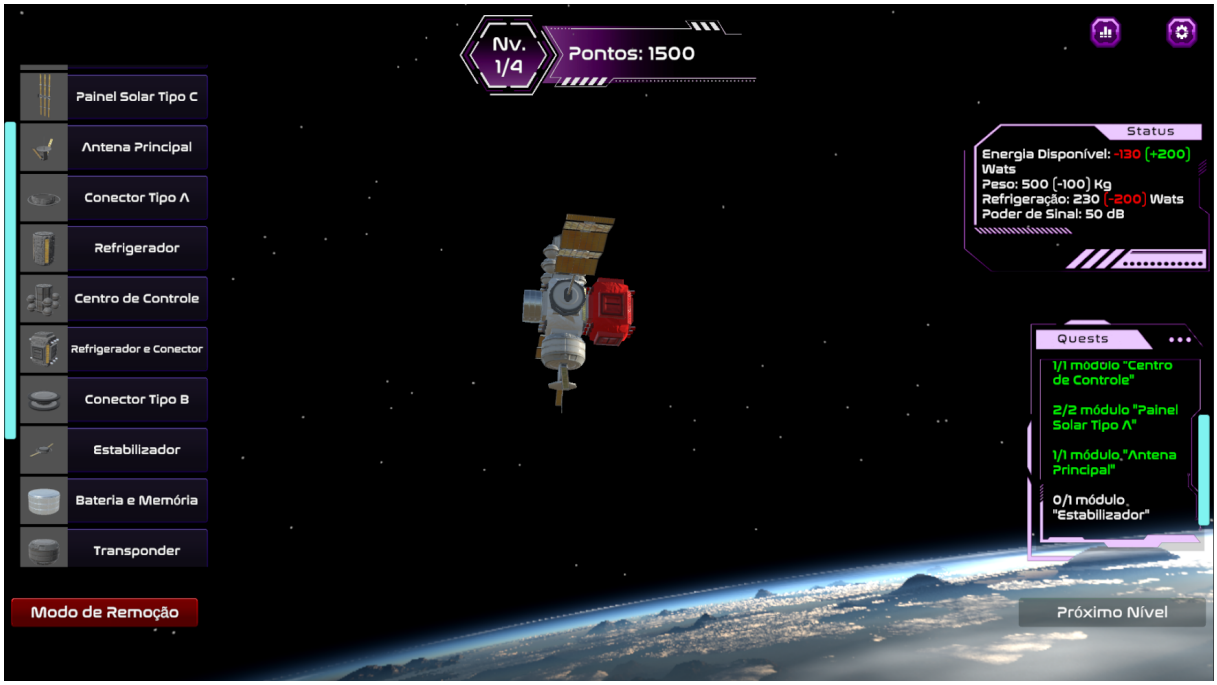

Figura 5.11: Remoção de Módulos.

Figura 5.12: Próximo Nível Desbloqueado.

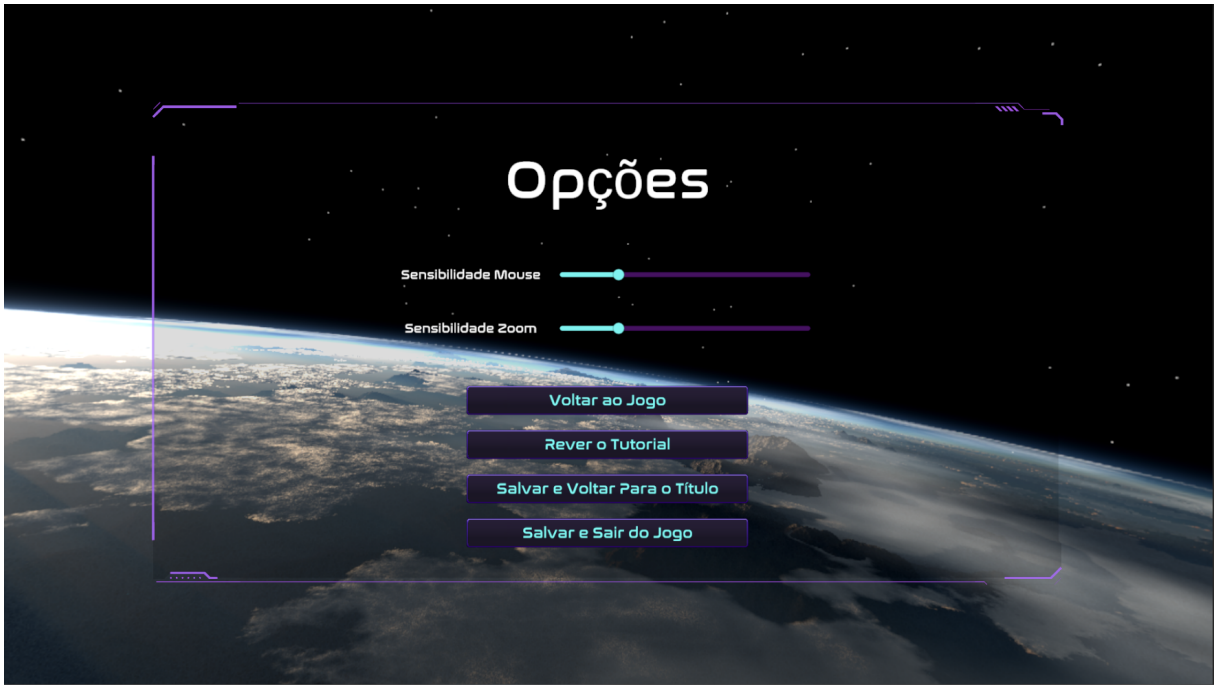

Figura 5.13: Painel de Opções.

Figura 5.14: Botão de Carregar Jogo Disponível.

5.3 Fim de Jogo

O jogador, extremamente habilidoso, conseguirá, por fim, concluir todos os níveis e será direcionado para a tela de conclusão, mostrada na Figura 5.15. Lá, ele perceberá que

alcançou um novo recorde. Ele achará a experiência muito positiva e retornará à tela inicial do jogo. Lá, ele verá que sua pontuação atual está sendo exibida e que, se desejar, pode excluí-la clicando no botão correspondente, *Deletar pontuação*, que estará desbloqueado, como mostra a Figura 5.16. No entanto, ele optará por não fazer isso e iniciará uma nova partida, pois gostou da experiência e busca alcançar uma pontuação ainda melhor.

Figura 5.15: Tela de Conclusão.

Figura 5.16: Botão de Deletar a Pontuação Disponível.

Capítulo 6

Observações Finais

O projeto tinha como objetivo ir um pouco além da criação de satélites. Permitir que o jogador pudesse testá-lo, programar equações de lançamento e vê-lo orbitar ao redor da Terra fazia parte do plano inicial. Entretanto, por se tratar de um trabalho de conclusão de curso, havia pouco tempo disponível para programar, aprender, testar e documentar, então essas partes foram retiradas. Nada impede, porém, que no futuro o projeto passe por novas revisões e melhorias, permitindo a implementação dessas e outras funcionalidades.

Como pode ser visto, já é possível que um aluno ou qualquer pessoa interessada tenha uma experiência sólida ao jogar o atual *My Spacecraft*. O jogo estimula a criatividade, ensina sobre módulos de satélite e gerenciamento de recursos e desperta o interesse pela área da astronomia. No geral, a experiência foi desafiadora, mas proveitosa. Muito do conhecimento adquirido ao longo do curso se mostrou útil em algum aspecto, como, por exemplo, cálculo, física e álgebra, para conseguir estruturar a lógica interna no Unity; estrutura de dados e linguagens, métodos e técnicas de programação para tornar tudo possível. Foi o primeiro contato com a Unity e com um projeto de "médio porte". Várias das funcionalidades da engine foram aprendidas durante o desenvolvimento.

Verifica-se que o jogo cumpre os requisitos funcionais e não funcionais estabelecidos. Pessoas que nunca jogaram *My Spacecraft*, sem qualquer dica, conseguiriam entender seu funcionamento através do tutorial. O botão de opções oferece a possibilidade de salvar o jogo a qualquer instante. A interface com o usuário é simples, apresentando apenas uma lista com as fotos e nomes dos módulos para que o jogador clique e o construa, ou então o remova por meio do botão vermelho de remoção logo abaixo dessa lista. O jogo, durante suas diversas execuções, não apresentou erros visíveis e, além disso, conseguiu ser executado em máquinas diferentes que cumpriam os requisitos mínimos estipulados. É importante ainda destacar que a metodologia *Scrum* foi utilizada durante todo o desenvolvimento, com reuniões breves (*sprints*) todas as semanas com o *Scrum Master*, no caso, o orientador, Prof. Marcelo.

Por fim, espera-se agora que, como mencionado, o projeto não fique parado e possa ser melhorado e desenvolvido ao longo do tempo. Espera-se também que uma equipe maior se forme para contribuir com o andamento e conhecimento, e que alunos se voluntariem para testar e fornecer feedback. Da forma como o jogo se encontra atualmente, talvez não seja muito vantajosa a sua aplicação no nível de ensino médio, visto que os alunos estão em um período em que o tempo é extremamente corrido e escasso. O jogo ainda não permite uma aplicação real da matemática e física aprendidas por eles nesse nível de ensino. No entanto, ele pode ser utilizado como uma atividade extra e bônus dentro de um ambiente de sala de aula ou laboratório, visto que, ainda assim, é possível que consigam projetar satélites personalizados, similares à realidade.

Referências

[1] Gonçalves, Ana Paula Yabiku: *Disciplinas são consideradas as mais difíceis.* Available at `https://www2.jornalcruzeiro.com.br/materia/375267/disciplinas-sao-consideradas-as-mais-dificeis`, 2012. Accessed on Juny 26, 2023. 1

[2] Silva, Marco Aurélio da: *O ensino de física para alunos do ensino médio.* Available at `https://educador.brasilescola.uol.com.br/estrategias-ensino/o-ensino-fisica-para-alunos-ensino-medio.htm`. Accessed on Juny 26, 2023. 1

[3] Moreira, Marco Antonio: *Desafios no ensino da física.* Rev. Bras. Ensino Fís., 43, 2021. 1

[4] Marketing: *Jogos eletrônicos na educação: porque incluí-los no plano de aula?* Available at `https://educadordofuturo.com.br/tecnologia-na-educacao/jogos-eletronicos-educacao/`, 2021. Accessed on Juny 26, 2023. 1

[5] Fróes, André Luís Delvas: *Astronomia, astrofísica e cosmologia para o ensino médio.* Rev. Bras. Ensino Fís., 36, 2014. 1

[6] Mawas *et al.*: *Investigating the impact of an adventure-based 3d solar system game on primary school learning process.* KM&EL, 12:165–190, 2020. 3

[7] Tobias *et al.*: *Handbook of research on educational communication and technology.* NY: Springer, páginas 485–503, 2014. 3

[8] Ross *et al.*: *Games-based learning for systems engineering concepts.* Elsevier B. V., páginas 430–440, 2014. 4

[9] Initiative, Systems Engineering Advancement Research: *Digital space tug skirmish.* Available at `http://seari.mit.edu/dsts.php`, 2015. Accessed on February 15, 2023. 5

[10] Squire *et al.*: *At play in the cosmos.* International Journal of Designs for Learning, 12:1–15, 2021. 5

[11] Jin *et al.*: *Game-based learning for safety and security education.* EduLearn, 14:114–122, 2020. 6

[12] Labor Statistics, Bureau of: *Injuries, illnesses, and fatalities.* Available at `http://www.bls.gov/iif/.com`, 2019. Accessed on February 16, 2023. 6

[13] Susman, Pavlin: *Improvement in teachers' knowledge and understanding of basic astronomy concepts through didactic games.* Baltic Science Education, 19:1020–1033, 2020. 7

[14] *Diagrama criado com o auxílio do software astah uml.* `https://astah.net/`, acesso em 2023-05-05. 10, 11, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26

[15] *Imagem criada com o auxílio de geogebra.* `https://www.geogebra.org/`, acesso em 2023-07-06. 31, 35

[16] *Imagem criada com o auxílio de toytheater.* `https://toytheater.com/cube/`, acesso em 2023-07-06. 34

Apêndice A

Diagrama de Classe Completo