SPACE ROYALE: DESENVOLVIMENTO DE JOGO BATTLE ROYALE COM UNREAL ENGINE

Sara Helena Régis Theiss da Silva, Luciana Pereira de Araújo Kohler – Orientadora
Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

shrtheiss@furb.br, lpa@furb.br

Resumo: *Este trabalho consiste na especificação e desenvolvimento de um jogo no estilo Battle Royale utilizando o motor de jogos Unreal Engine 4. O jogo consiste em um simulador de naves espaciais, sendo a nave modelada na ferramenta Blender, no qual os jogadores se enfrentam com dois tipos de armas e vão eliminando uns aos outros até que reste apenas um. Para que os jogadores possam se reunir em uma partida, foi realizado um tratamento para criação e participação em sessões. O resultado alcançado foi de um jogo funcional no qual se tem um jogador host, que se apresenta para os demais como um servidor, e os jogadores restantes que são participantes da sessão e, como clientes, dependem do servidor para fazer suas requisições. Em testes ainda foram identificados alguns problemas com relação à performance, contudo o desenvolvimento na plataforma do Unreal Engine 4 foi bem-sucedido.*

Palavras-chave*: Jogos multiplayer. Battle Royale. Simulador de naves. Unreal Engine 4. Blender.*

1 INTRODUÇÃO

Os primeiros jogos multiplayer já existiam nos anos 1980 e a metodologia utilizada para permitir que dois ou mais jogadores participassem dele ao mesmo tempo era por meio de uma divisão na tela (PANTEL; WOLF, 2002). Cada divisão mostrava a visão de cada jogador que, normalmente, eram 2 ou 4, e eles rodavam em apenas uma máquina. Esses jogos evoluíram para o funcionamento em mais de uma máquina. Contudo, antes dos jogos multiplayer como são conhecidos hoje, o mecanismo utilizado para conectar os jogadores em diferentes máquinas era a Rede de Área Local (LAN). Assim, os jogadores passaram a usar seus próprios consoles, mas esses deveriam estar localizados na mesma rede para que pudessem se conectar (PANTEL; WOLF, 2002, p. 23).

A ideia de uma vertente dos jogos multiplayer surgiu nos anos 1999, com a novela Battle Royale de Koushun Takami, mas ficou conhecida quando foi feita a adaptação para os cinemas já no ano seguinte (LIMA; SILVA, 2018, p. 81.). O contexto apresentado por Takami possibilitou a criação de um novo gênero de jogos, que acabou recebendo o mesmo nome de sua obra, no qual os jogadores se enfrentariam diretamente com os recursos oferecidos para coroar um vencedor (CHOI; KIM, 2018, p. 5).

Em vista da popularidade dos jogos Battle Royale (ROSENBUSCH; RÖTTGER; ROSENBUSCH, 2020, p. 462), foi considerada a possibilidade de criação de um jogo desse tipo utilizando o motor de jogos Unreal Engine 4, fornecido pela Epic Games Inc. Assim surgiu a ideia para o seguinte trabalho, no qual foi desenvolvido o jogo Space Royale.

O presente jogo se categoriza como um Battle Royale e se caracteriza pelo confronto entre naves espaciais, possibilitando até cinco jogadores. A nave do jogador possui dois tipos de armas: básica e especial. A arma básica oferece uma base sólida com um tempo de recarga menor, podendo ser disparada com maior frequência. Em contrapartida, são necessários vários disparos para causar dano suficiente. Em conjunto a ela, está a arma especial na qual o jogador se apoia para gerar um dano mais intenso ou para auxiliá-lo com uma habilidade, porém, com uma cadência maior de tiros. As armas especiais, podem ser do tipo Ataque, que apenas geram uma determinada quantidade de dano; tipo Bomba, que ao atingir um inimigo causam dano que continua sendo descontado da vida inimiga por determinado tempo, ou seja, deixa a vida do inimigo queimando; tipo Lentidão, que deixam a nave inimiga mais lenta por certo tempo; e tipo Escudo, que diferente dos outros tipos, é uma arma defensiva, que seguram determinada quantidade de dano que o jogador receberia.

Diante deste cenário, o objetivo deste trabalho é disponibilizar um jogo do gênero Battle Royale que permita a interação entre jogadores de diferentes lugares utilizando o motor de jogos Unreal Engine. Os objetivos específicos deste trabalho são: (I) disponibilizar um jogo de batalha com naves em que os usuários lutarão entre si para que apenas um seja o vencedor; (II) criar armas categorizadas entre armas de ataque, bombas, armas de lentidão e escudos; (III) fornecer um jogo com boa jogabilidade.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção está dividida da seguinte forma: a seção 2.1 aborda o tema Battle Royale; a seção 2.2 descreve brevemente sobre jogos multiplayer; por fim, a seção 2.3 aborda sobre os trabalhos correlatos.

2.1 BATTLE ROYALE

Segundo Choi e Kim (2018, p. 5), o termo Battle Royale se tornou comum com o lançamento, nos anos 2000, do filme japonês de mesmo nome baseado na novela de Koushun Takami. Ele passou a ser utilizado para descrever o contexto apresentado no filme quando aplicado a jogos digitais. Jogos desse gênero são definidos por um mapa em que todos os jogadores são invocados em um ponto de início diferente e devem se enfrentar até a morte, restando apenas um campeão, envolvendo elementos de combate e sobrevivência (ZHOU et al, 2021, p. 280).

A popularidade do gênero Battle Royale vem crescendo nos últimos tempos no mundo dos games, com grandes referências como o Fortnite e o PUBG. Na literatura e no cinema também podem ser encontrados trabalhos que se inspiraram na obra de Takami, como a obra de Suzanne Collins que ficou mundialmente conhecida como Jogos Vorazes (ROSENBUSCH; RÖTTGER; ROSENBUSCH, 2020, p. 462).

Por conseguinte, os jogos de Battle Royale têm como objetivo final a sobrevivência. Em um jogo em que os jogadores são postos lado a lado em uma batalha de armas, habilidade e estratégia (CHOI; KIM, 2018, p. 9), a característica principal esperada é a imprevisibilidade. Para Rosenbusch, Röttger e Rosenbusch (2017 apud AHN, 2020, p. 464), essa incerteza desperta emoção e faz com que mais jogadores sejam atraídos por jogos do gênero. Alguns jogos ainda, ao invés de serem definidos em apenas uma batalha, evocam batalhas individuais para que os jogadores possam reafirmar sua vitória e comprovar suas habilidades. Ao final dos torneios, todos os resultados são reunidos para gerar o verdadeiro vencedor.

2.2 JOGOS MULTIPLAYER

Os jogos multiplayer são aqueles que permitem que vários jogadores possam interagir em tempo real dentro do jogo. Eles se reúnem em uma sessão, que comumente é chamada de *lobby* e podem se comunicar com o propósito de colaboração ou competitividade (MANNINEN, 2003, p. 2). Em jogos que exploram a competitividade, a proposta multiplayer é muito procurada, pois oferece a possiblidade de jogar contra outras pessoas ao invés de enfrentar um computador. Isso acontece pois a inteligência, espontaneidade e intuição de inimigos humanos oferecem um maior desafio (GAMMARANO; COSTA, 2020, p. 1).

Alguns jogos oferecem opções de comunicação entre os jogadores, como chats textuais, figurinhas, ou até mesmo chamadas de voz. Mesmo em jogos que não possuem esses recursos, jogadores que jogam com amigos tendem a se reunir em salas de bate-papo por meio de outros programas para facilitar a comunicação em jogos colaborativos ou em equipe (MANNINEN, 2003, p. 2).

2.3 TRABALHOS CORRELATOS

Nessa seção são apresentados jogos desenvolvidos em trabalhos acadêmicos com similaridades em relação ao desenvolvido nesse trabalho, sendo que dois dos mencionados possuem o gênero multiplayer. O primeiro trata-se de jogo de batalha naval multiplayer para celular desenvolvido na plataforma Java 2 Micro Edition (J2ME) utilizando Perfil de Dispositivo de Informação Móvel (MIDP) que é detalhado no

Quadro 1 (TRUPEL, 2008). Planeta X, o segundo jogo apresentado no Quadro 2, foi desenvolvido em Virtual Reality Modeling Language (VRML) com temática espacial e gênero Role-Playing Game (RPG) sendo disponibilizado na internet embutido em HyperText Markup Language (HTML) (ARAUJO; CORDENONSI, 2003). Por fim, Fawe, do gênero Massively Multiplayer Online Role-Playing Game (MMORPG) foi desenvolvido com HTML5, Canvas e Websocket e pode ser jogado em navegadores compatíveis através de celulares e computadores, conforme apresentado no Quadro 3 (JANSEN, 2020).

Quadro 1 - Batalha Naval

Referência	Trupel (2008).
Objetivos	Tentar destruir a frota de navios do outro jogador.
Principais funcionalidades	O jogo permite ao jogador informar seu nome e sua preferência pelo jogo individual (contra outro jogador) ou em duplas; o servidor do jogo monitora uma porta específica para o registro de novos jogadores; o servidor do jogo monitora uma porta para cada jogador que já esteja registrado; o servidor do jogo associa os jogadores aos adversários conforme a preferência escolhida por cada um; o servidor não implementa o conceito de ranking; o jogo permite uma interação entre o jogador e seu avatar.
Ferramentas de desenvolvimento	Plataforma J2ME utilizando MIDP.
Resultados e conclusões	O autor atesta que as funcionalidades do jogo foram cumpridas, possibilitando aos jogadores jogarem de maneira individual ao se confrontar com outro jogador ou em dupla em confrontos com dois jogadores simultaneamente no servidor. Também constatou problemas de latência em casos de máquinas únicas, porém resolvidos facilmente esses problemas quando suportados por redes de computadores. No entanto, não foram realizados testes com celulares, apenas em simuladores, porém não interferiu no resultado.

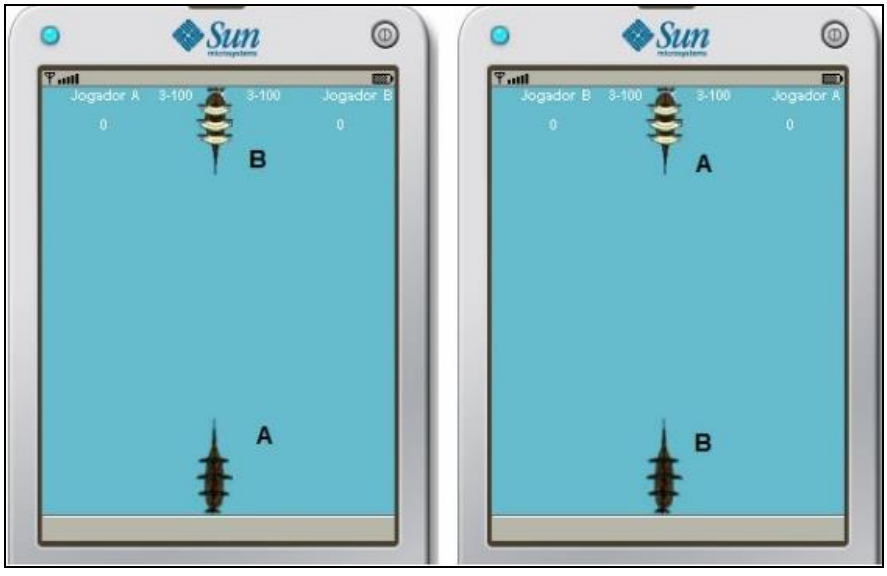
Fonte: elaborado pelo autor.

Ainda, segundo Trupel (2008, p. 27), Batalha Naval é um jogo para dois jogadores em que cada jogador tenta destruir a frota de navios do outro. O jogo pode ser dividido em três modos (selecionáveis ou não pelo jogador): “clássico”, “tiro extra ao acertar” e “atire tantas vezes enquanto tiver navios”. No modo clássico cada jogador dispara um tiro por vez. Já no modo “tiro extra ao certar”, toda vez que o jogador acertar um navio inimigo, ganha outra jogada, mas se erra, passa a vez para o adversário. E no último modo “atire tantas vezes enquanto tiver navios”, os jogadores têm permissão para continuar atirando enquanto houver navios em suas frotas.

Uma vez que o modo de jogo é definido, devem-se colocar os navios no mapa. Cada jogador começa com cinco navios dos quais, podem ser: porta-aviões: necessita de cinco tiros para ser abatido; *destroyer*: necessita de quatro tiros para ser abatido; cruzador: necessita de quatro tiros para ser abatido; submarino: necessita de três tiros para ser abatido; barco patrulha: necessita de dois tiros para ser abatido.

A disposição de cada navio no tabuleiro pode ser na horizontal ou vertical. Depois que o jogador posicionar todos os cinco navios, poderá ser iniciado o jogo. A escolha do primeiro jogador a jogar é randômica. Uma jogada consiste em selecionar uma região do mapa sendo que se a região selecionada contiver um pedaço de um navio, o jogador poderá jogar novamente até errar e a vez é dada ao adversário. O primeiro jogador a afundar os cinco navios do adversário será o campeão. A Figura 1 demonstra a interface gráfica do jogo.

Figura 1 - Batalha Naval



Fonte: Trupel (2008).

Quadro 2 - Planeta X

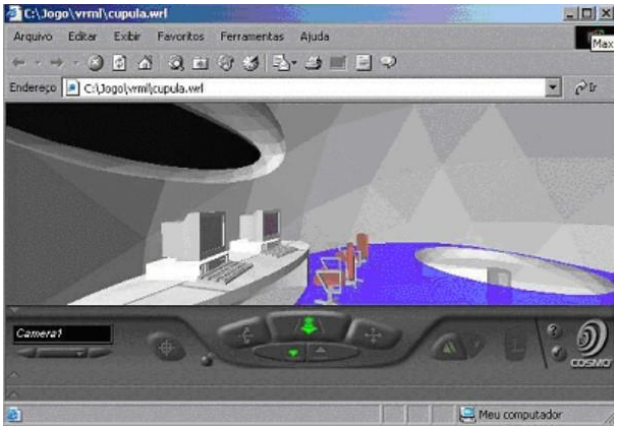
Referência	Araujo e Cordenonsi (2003).
Objetivos	Analisar as pistas e descobrir o que aconteceu com os colonizadores do Planeta X.
Principais funcionalidades	O jogo permite ao jogador a exploração do mapa, oferecendo recursos para que o jogador, por meio de raciocínio, encontre as pistas e descubra o que aconteceu com os colonizadores do Planeta X. A tática do jogo é reunir o maior número de pistas e juntá-las em um quebra cabeça, levando-o a desvendar o mistério do desaparecimento dos colonizadores.
Ferramentas de desenvolvimento	Ambientes virtuais utilizando VMRL e HTML.
Resultados e conclusões	O autor afirma que a utilização da linguagem VRML pode ser utilizada para tornar o mundo mais real com a geração de realidade virtual, mas aponta que para uma melhor performance é necessária uma taxa de transferência muito mais alta e recursos de hardwares mais potentes também, pois o resultado ainda ficou distante de um usuário comum. Além da performance, um ponto negativo encontrado foi que ela requere um plug-in para a visualização e, por esse motivo, não é sempre aceita na internet. Apesar desses dois apontamentos, garante que a linguagem oferece muitos recursos vantajosos como: a animação, visualização de objetos 3D, a interação que o usuário pode ter com o ambiente e, principalmente, a fácil integração com a linguagem Java.

Fonte: elaborado pelo autor.

Segundo Araujo e Cordenonsi (2003), as pistas do jogo podem ser objetos que o usuário achar. O usuário deve checar o objeto e verificar se pode existir alguma pista que o leve a algum lugar ou a alguma conclusão. Outra opção é um texto que pode ser encontrado dentro do jogo e o usuário acaba por ler o texto, tendo de associar com outras pistas para montar o quebra-cabeça e chegar à conclusão final (ARAUJO; CORDENONSI, 2003).

A aplicação é composta por duas fases, em que o usuário tem de passar por elas para chegar ao fim do jogo. A primeira fase se passa em uma nave (Figura 2), em que o usuário deve descobrir como fazer com que ela funcione. Após ligar a nave, ela leva o usuário até o planeta que é o ambiente da segunda fase (Figura 3) em que o jogador deverá encontrar pistas até o final do jogo (ARAUJO; CORDENONSI, 2003).

Figura 2 - Primeira fase do jogo Planeta X



Fonte: Araujo e Cordenonsi (2003).

Figura 3 - Segunda fase do jogo Planeta X

Fonte: Araujo e Cordenonsi (2003).

Quadro 3 - Fawe

Referência	Jansen (2020).
Objetivos	Salvar os remanescentes e seguir para as maiores cidades dos 3 continentes mais seguros e desenvolvidos.
Principais funcionalidades	Dentro do jogo é possível realizar ações como andar, se comunicar ou atacar inimigos em tempo real e em um mundo virtual. O jogador cria seu próprio personagem e pode seguir seu caminho próprio dentro do mundo virtual com outros jogadores.
Ferramentas de desenvolvimento	HTML 5 e WebSockets.
Resultados e conclusões	O autor aponta que os maiores desafios encontrados foi trabalhar com conceitos como multithreading, alto consumo de rede e processamento e que esses ainda podem ser melhorados dentro do jogo. Com a aplicação de questionários foi possível identificar que o jogo teve uma resposta positiva e agradou a maioria dos participantes, porém enfrentou problemas com performance, que afetou a movimentação do personagem. Apesar de gostarem do jogo, muitos identificaram lentidão ou travamento em algum ponto do jogo.

Fonte: elaborado pelo autor.

Fawe é um protótipo de jogo do gênero MMORPG em que os jogadores conseguem interagir entre si conectados em diferentes dispositivos (JANSEN, 2020). O nome Fawe surgiu a partir das iniciais dos quatros elementos em inglês organizados de maneira que tenha um sentido fonético. Os elementos são ordenadamente *fire* (fogo), *air* (ar), *water* (água) e *earth* (terra). Estes quatro elementos são utilizados como base para toda a história do jogo (JANSEN, 2020).

A história de Fawe se passa em um mundo em constante destruição causado pela raça humana. Para evitar a destruição completa, os Deuses criaram um mecanismo. De todos os 12 continentes existentes no mundo, há sempre 3 ocultos. De tempos em tempos, os 3 continentes mais destruídos pela raça humana são engolidos pela terra (ou pelo mar, ou pelo fogo) e três novos continentes emergem, renovados e livres da influência humana (JANSEN, 2020). Os jogadores começam o jogo num ambiente destruído, prestes a ser engolido chamado de Finraza. As primeiras missões visam salvar os remanescentes e seguir para as maiores cidades dos 3 continentes mais seguros e desenvolvidos (JANSEN, 2020).

A mecânica do jogo é simples. Todas as ações são realizadas apenas com mouse ou *touchscreen*. Assim que o personagem entra no mundo é possível andar apenas com um toque ou clique na localização desejada e automaticamente o personagem deve ir até o ponto desejado. Os gráficos são estilo de visão *top down* (visto de cima), em que o jogador vê o seu personagem sempre no centro da tela (Figura 4). Alguns menus para visualizar chat de conversa, mochila e missões aparecerem pelos cantos da tela. Também aparece um mini-mapa na parte superior da tela para que o jogador possa se localizar no mundo.

Figura 4 - Fawe

Fonte: Jansen (2020).

3 DESCRIÇÃO DO JOGO SPACE ROYALE

Nesta seção é detalhado como foi feito o desenvolvimento do jogo Space Royale, considerando seus requisitos, sua especificação demonstrada por meio de diagramas e a implementação das principais funcionalidades.

3.1 ESPECIFICAÇÃO

O Quadro 4 apresenta em ordem os Requisitos Funcionais (RF) descritos, enquanto o Quadro 5 corresponde aos Requisitos Não Funcionais (RNF) presentes.

Quadro 4 - Requisitos funcionais do jogo

RF01: O jogo deve permitir que o usuário crie um lobby e dê início à partida.
RF02: O jogo deve permitir que o usuário entre em um lobby existente.
RF03: O jogo deve apresentar os números de jogadores aguardando na sala antes de uma partida.
RF04: O jogo deve apresentar a quantidade de vida da nave durante a partida.
RF05: O jogo deve apresentar a disponibilidade das armas durante a partida.
RF06: O jogo deve apresentar o tempo da partida corrente.
RF07: O jogo deve permitir que o usuário movimente sua nave pelo mapa.
RF08: O jogo deve permitir que o usuário ative suas armas selecionadas quando estiverem disponíveis.
RF09: O jogo deve aplicar dano à nave do jogador quando atingido por um disparo, de acordo com o tipo do disparo.
RF10: O jogo deve apresentar os resultados da partida quando esta acaba para o jogador.

Fonte: elaborado pelo autor.

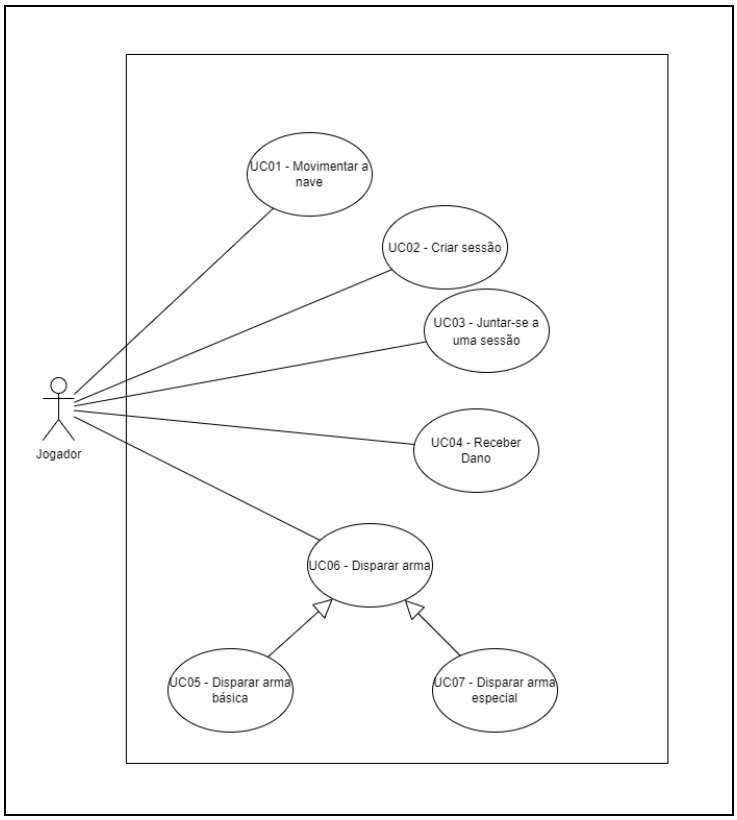
Quadro 5 - Requisitos não funcionais do jogo

RNF01: O jogo deve ser disponibilizado para plataforma desktop.
RNF02: O jogo deve ser desenvolvido no motor de jogos Unreal Engine 4.
RNF03: O tratamento dos atributos da Nave a das Armas deverá ser feito por meio da linguagem de programação C++.
RNF04: A relação dos atributos com a interface gráfica deverá ser feita por meio da linguagem de script visual Blueprints.
RNF05: A nave do jogo deve ser modelada pela ferramenta Blender.
RNF06: A nave deve possuir uma coloração neutra e possuir duas saídas para as armas básicas e especiais.

Fonte: elaborado pelo autor.

Apresentado na Figura 5, o diagrama de casos de uso elaborado demonstra o papel que o jogador exerce dentro do jogo. Neste caso, o único ator existente é o próprio jogador e é ele que irá interagir com a aplicação integralmente.

Figura 5 - Diagrama de casos de uso



Fonte: elaborado pelo autor.

Conforme apresentado na Figura 5, segue a descrição de cada caso de uso:

a) `UC01`: o objetivo deste caso de uso é permitir que o usuário possa movimentar sua nave pelo mapa. Ao iniciar uma partida, o jogador é invocado em um dos pontos de início espalhados pelo mapa e a partir dali ele pode utilizar os seguintes atalhos do teclado para controlar a nave: Shift para acelerar, no qual a nave avança um pouco à frente na direção para onde o usuário está voltado; seta para direita para rotacionar a nave em seu eixo em sentido horário, sem mudar sua localização, apenas sua direção; seta para esquerda para rotacionar a nave em seu eixo no sentido anti-horário, também sem mudar sua localização, apenas sua direção;
b) `UC02`: o objetivo deste caso é permitir que o usuário possa criar uma sessão de jogo na qual será o host. No menu inicial do jogo o usuário clica na opção de `Criar Sessão`, o sistema cria uma sessão e o jogador é enviado para a tela de lobby, na qual ele pode visualizar a quantidade de pessoas conforme elas entram em sua sessão e tem a opção de iniciar a partida, levando todos para o mapa do jogo;
c) `UC03`: o objetivo deste caso é permitir que o usuário possa entrar em sessões criadas por outros jogadores. No menu inicial o usuário seleciona a opção de `Entrar em Sessão`, o sistema apresenta uma tela de carregamento enquanto procura por sessões existentes. Encontrando sessões, o sistema tenta entrar na primeira, não conseguindo ele tenta a próxima e assim por diante. Se foi possível entrar em uma sessão, o usuário é automaticamente levado para onde todos os outros estão, na tela de lobby. Caso contrário, ele volta para o menu inicial para que possa tentar novamente ou crie sua própria sessão. Vale ressaltar que o usuário não consegue entrar em sessões que já contém o número máximo de jogadores, sendo cinco o máximo;
d) `UC04`: o objetivo deste caso é permitir que o usuário possa receber dano de outros jogadores. Ao iniciar uma partida, quando o usuário é atingido por um tiro, o sistema deve aplicar o dano equivalente ao tiro recebido na vida ou em outro atributo da nave do jogador, de acordo com o tipo de tiro. Tiros do tipo Lentidão, devem afetar sua velocidade, enquanto tiros do tipo Ataque ou Bomba devem afetar sua vida;
e) `UC05`: o objetivo deste caso é permitir que o usuário possa disparar sua arma básica. Estando em uma partida, ao clicar o atalho do teclado Q, o sistema verifica a disponibilidade da sua arma básica, ou seja, se a arma não está em tempo de recarga. Quando a arma está disponível, será invocado um tiro da nave com os atributos equivalentes à arma básica. Após, será atualizado o componente da tela que mostra que a arma está em recarga e ele ficará desabilitado pelo tempo de recarga que a arma possui. Caso a arma já estava em recarga, o atalho não resultará em nenhuma ação;

f) UC06: o objetivo deste caso é permitir que o usuário possa disparar uma arma. Em uma partida, se o usuário ativar uma arma, o sistema irá invocar um tiro da nave com os atributos da arma disparada;
g) UC07: o objetivo deste caso é permitir que o usuário possa disparar sua arma especial. Estando em uma partida, ao clicar o atalho do teclado W, o sistema verifica a disponibilidade da sua arma especial, ou seja, se a arma não está em tempo de recarga. Quando a arma está disponível, será invocado um tiro da nave com os atributos equivalentes à arma especial. Após, será atualizado o componente da tela que mostra que a arma está em recarga e ele ficará desabilitado pelo tempo de recarga que a arma possui. Caso a arma já estava em recarga, o atalho não resultará em nenhuma ação.

Os principais casos de uso estão descritos no Apêndice A nos Quadro 11, Quadro 12, Quadro 13 e Quadro 14, nos quais é tratado o fluxo seguido dentro do jogo para execução dos casos e as condições que o permeiam. Em complemento, o Quadro 6 ainda faz a relação entre os casos de uso e os requisitos funcionais do sistema apresentando a matriz de rastreabilidade.

Quadro 6 - Matriz de rastreabilidade dos casos de uso com os requisitos funcionais

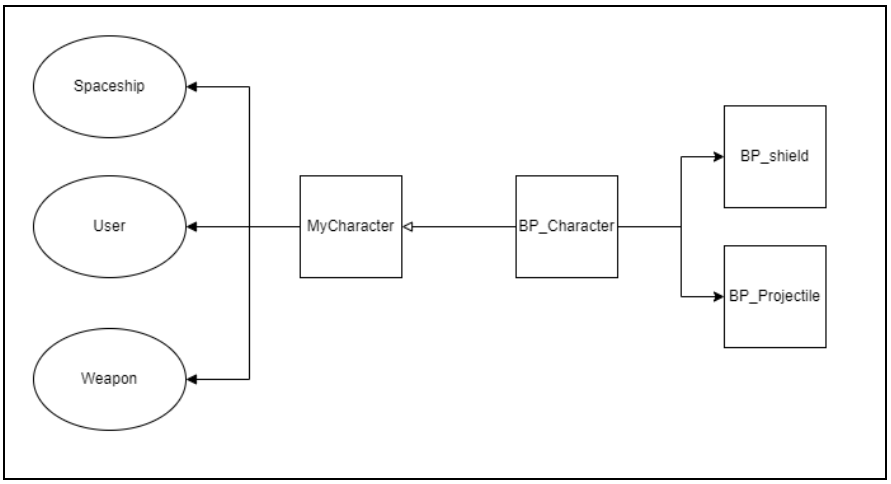
	RF01	RF02	RF07	RF08	RF09
UC01 – Movimentar a nave			X		
UC02 – Criar sessão	X				
UC03 – Juntar-se a uma sessão		X			
UC04 – Receber dano					X
UC05 – Disparar arma básica				X	
UC06 – Disparar arma				X	
UC07 – Disparar arma especial				X	

Fonte: elaborado pela autora.

Retratando a estruturação do projeto, foi elaborado um diagrama estrutural dos objetos de cena que demonstra a arquitetura fundamental para o funcionamento do jogo. Conforme a Figura 6, a parte chave da implementação são os elementos do tipo Character que formam uma herança, no qual o componente BP_Character herda os atributos do MyCharacter. Esses componentes representam o jogador dentro do jogo e, por isso, se conectam aos outros elementos da seguinte forma:

a) Spaceship: a relação se dá a fim de controlar a nave que o jogador utilizará na batalha, utilizando seus atributos como vida e velocidade para determinar como ele será apresentado para os outros participantes da partida;
b) Weapon: aqui tem-se a representação das armas e essa relação garante que elas sejam disparadas de acordo com seu tempo de recarga e desfiram um dano aos inimigos equivalente ao tipo de armamento;
c) User: ele oferece ao Character a possibilidade de tratar os resultados do jogador por meio de estatísticas, como a quantidade de abates que o jogador costuma ter por partida, média de dano desferido e a colocação média em que ele termina;
d) BP_Shield: a associação deste componente se dá para que o jogador possa ativar armas do tipo escudo, porém ele não trata os atributos relativos a uma arma desse tipo, apenas controla o resultado visual da ativação que seria o *spawn* de um escudo ao redor da nave;
e) BP_Projectile: similarmente ao BP_Shield, este elemento faz a representação visual do tiro, permitindo o *spawn* para os outros tipos de arma que lançam o disparo como um projétil. Em se tratando de um tiro que se projeta contra um inimigo, este elemento sai do âmbito gráfico e intermedia a chamada da função que aplica dano aos jogadores.

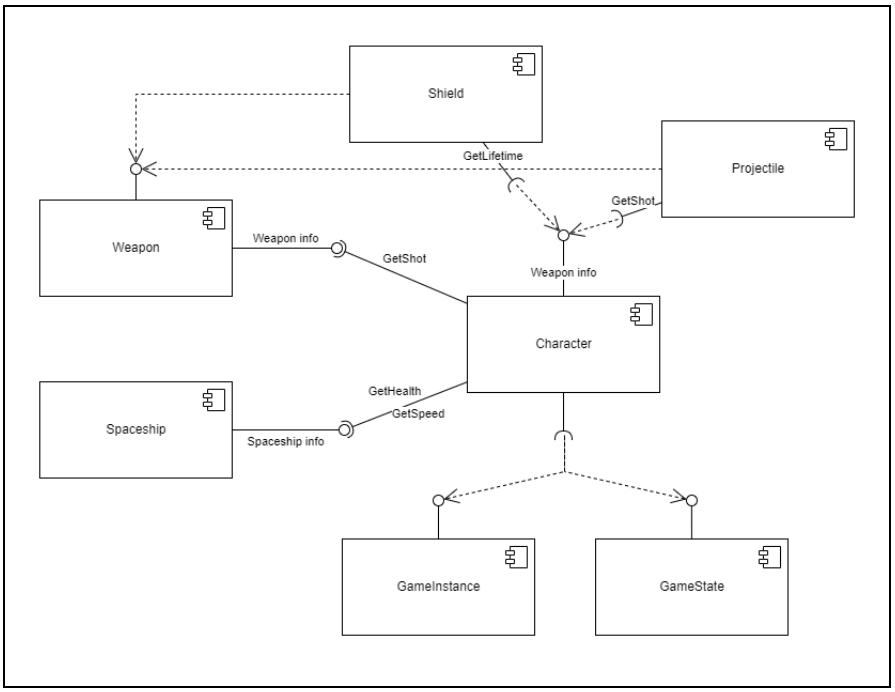
Figura 6 – Diagrama estrutural dos objetos de cena



Fonte: elaborado pela autora.

A representação do vínculo entre os componentes do projeto é ilustrada no diagrama de componentes na Figura 7. Aqui também se tem o componente `Character`, equivalente ao jogador, como ponto central do elo.

Figura 7 - Diagrama de componentes



Fonte: elaborado pela autora.

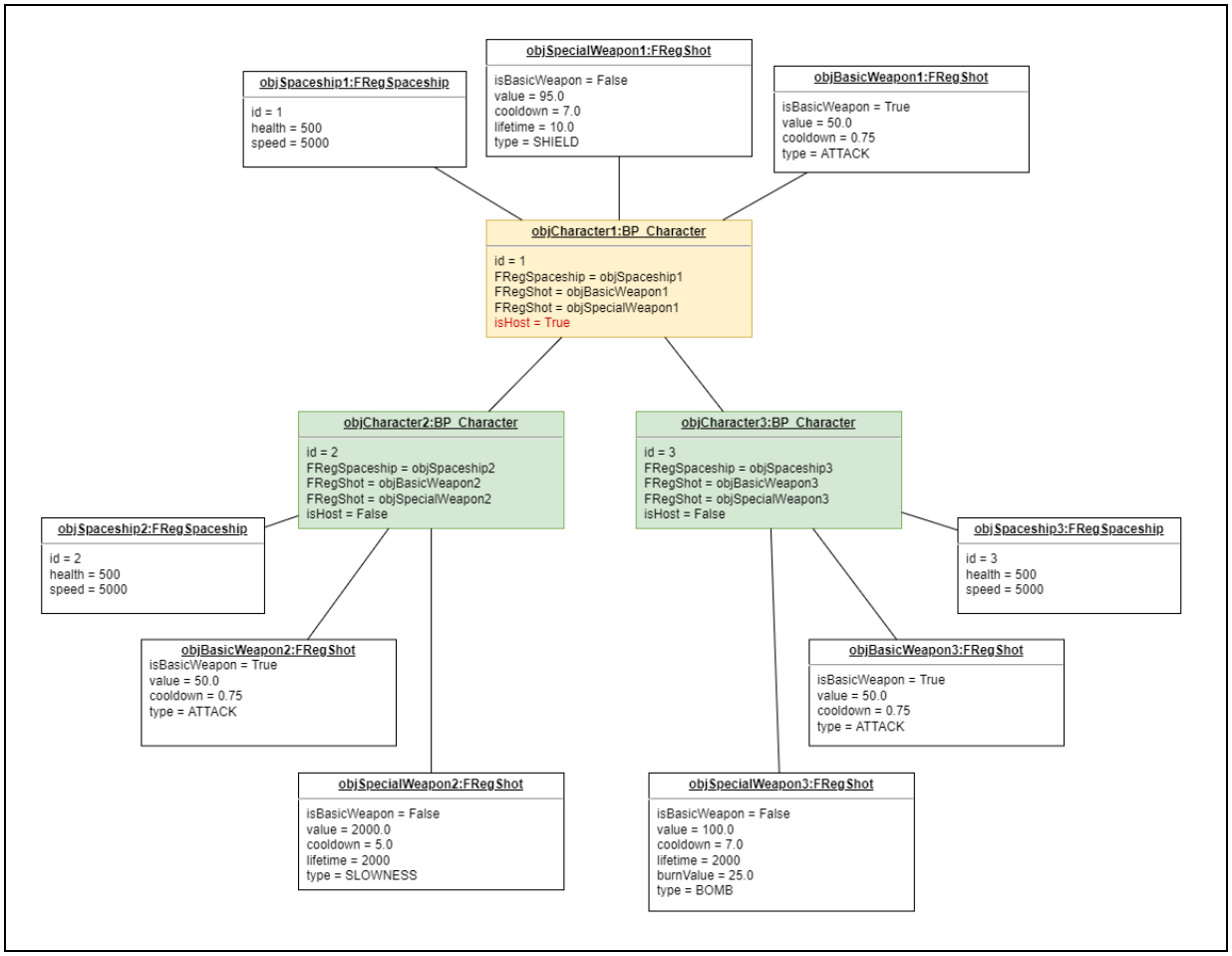
Em associação direta com os componentes `Weapon` e `Spaceship`, o `Character` solicita informações necessárias para a correta execução de suas funções. Para o `Weapon`, ele solicita informações do `Shot` que contém os valores relativos ao disparo da arma, como dano, tempo de recarga, tempo de efeito e tipo. Já ao `Spaceship` requisita as informações de vida e velocidade da nave. Esse vínculo direto é consolidado pois, para cada partida, ter-se-á uma nave, uma arma básica e uma arma especial previamente selecionadas. Assim, para que não precisem ser consultados seus valores a cada demanda, eles já são referenciados no componente `Character` para um acesso facilitado e performático.

Não obstante, a relação apresentada com o `Shield` e o `Projectile` ocorre de maneira oposta, pois são esses componentes que requererão informações. Conforme explanado anteriormente, os elementos `BP_Shield` e `BP_Projectile`, correspondentes aos componentes, trazem a representação gráfica para a ativação das armas. Para que essa ativação reflita corretamente a arma em questão, o `Projectile` precisa dos valores do `Shot` para aplicar a qualquer nave inimiga atingida, enquanto o `Shield` precisa apenas do tempo de vida da arma de escudo, para tratar por quanto tempo deve estar visualmente aparente.

Por fim, os componentes `GameState` e `GameInstance` são padronizados do Unreal Engine e podem ser consultados a qualquer momento no aspecto universal do projeto, pois abrigam informações gerais das condições do jogo. À vista disso, o `Character` possui dependência neles, mesmo sem uma conexão direta, pois as informações que eles entregam são necessárias para a tomada de decisões, ou seja, elas influenciam o comportamento do jogador.

Demonstrando a relação entre os objetos durante uma partida, a Figura 8 apresenta um diagrama de objetos. Nele, pode-se notar a relação de três objetos do tipo `Character`, representando três jogadores. O `objCharacter1` é o *host* da partida e, portanto, os outros dois objetos são conectados a ele para que possam repassar as requisições como ativar suas armas. Além disso, cada jogador terá uma relação com objetos do tipo `FRegSpaceship` e `FRegShot` que contém os valores de sua nave, arma básica e arma especial selecionada.

Figura 8 - Diagrama de objetos



Fonte: elaborado pela autora.

3.2 IMPLEMENTAÇÃO

O motor de jogos Unreal Engine 4 (UE4) foi utilizado para a criação do projeto da aplicação, sendo escrito na linguagem C++ em conjunto com a linguagem de script visual Blueprint. A criação das classes C++ foi feita por meio do Visual Studio 2019, enquanto os Blueprints são de uso próprio do sistema Unreal Engine e manipulados dentro do próprio programa. Com UE4 foi criado o projeto do jogo utilizando um recurso disponibilizado pela plataforma, entre os modelos preparados para a construção de determinados estilos de jogos diferentes. Dessa forma, o modelo escolhido foi o modelo "*Flying*", composto por um mapa de exemplo, uma nave para ser controlada pelo usuário e os comandos básicos de navegação.

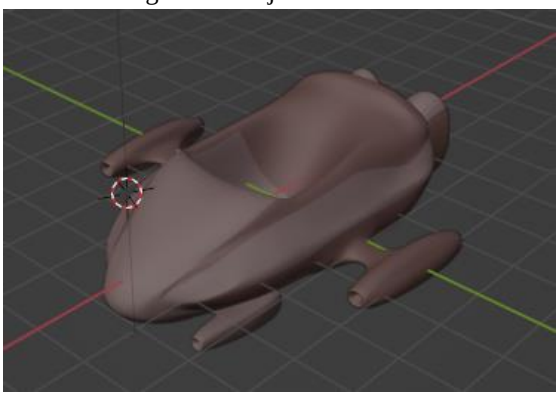
O mapa inicial de exemplo foi descartado pelo seu tamanho limitado e, em seu lugar, um mapa novo foi desenvolvido com um tamanho maior, permitindo uma capacidade maior de naves na tela e obstáculos, adquiridos na loja da Epic Games, com o objetivo de melhorar a experiência do usuário durante a navegação do jogo. Além disso, foram criadas as seguintes telas para a interface do jogo: menu inicial, lobby, tela da partida e resultado da partida.

O Blender foi utilizado para a criação do recurso visual do objeto, no que se refere à modelagem 3D da nave e por meio de plugins se tornou possível enviar o modelo direto para a aplicação do projeto no UE4. Na criação de um novo projeto no programa Blender é trazido um objeto com formatação padrão de cubo, o qual foi selecionado como

base para a modelagem da nave. No processo de modelagem, o modificador de propriedade espelhar foi utilizado para refletir em ambos os lados do objeto as mudanças no intuito de formá-lo simetricamente. Além desse modificador de propriedade, foi também adicionado o modificador de subdivisão com o propósito de adicionar pontos à malha existente, fornecendo mais detalhes à imagem do objeto e atribuindo uma forma visualmente mais agradável ao objeto.

Foram adicionadas propriedades de material para definir pinturas na nave, os materiais que incorporaram o objeto são de duas colorações diferentes para o corpo da nave e para a raja que a nave possui nas laterais. Após isso, no modo de escultura foram aplicados diferentes pinceis para esculpir a lataria da nave, moldar a superfície e criar uma percepção de profundidade em alguns pontos. Podem ser observadas na Figura 9 as alterações listadas na respectiva seção.

Figura 9 - Objeto 3D da nave



Fonte: elaborado pela autora.

Para realizar a exportação do objeto modelado no Blender foi utilizado o plugin "*Pipeline: Send to Unreal*" disponibilizado pelo UE que facilita a comunicação entre as aplicações, possibilitando assim o envio direto do Blender para o UE4 dos componentes na formatação padrão que serão utilizados no desenvolvimento do jogo.

Para o desenvolvimento de jogos por meio do UE, é recomendada a utilização das classes C++ e Blueprints em conjunto. No que se refere às classes C++ são feitos os tratamentos de negócio, enquanto os Blueprints manejam a união com os componentes das telas. Nesse tratamento, foram criadas as classes C++ `GameMode`, `GameInstance`, `Pawn`, `HUD` e `Character`. No entanto, algumas dessas classes implementadas na linguagem C++ se tornaram dispensáveis, de modo que poderiam ser realizadas de maneira mais simples por meio de Blueprints. Por fim, a única classe C++ remanescente foi a `Character`, sendo necessária para o tratamento dos atributos da nave. Simultaneamente, foi criado o Blueprint `BP_Character` que possui a classe C++ como pai. Essa união permite que o Blueprint possua os mesmos atributos da classe C++ com o adicional do tratamento de seu modelo 3D, de movimentação e de ativação das armas.

Acerca da criação e busca das sessões, é utilizado o componente `GameInstance`. Foi criada a classe C++ para esse componente e implementados os métodos relatados no Quadro 7**Erro! Fonte de referência não encontrada.**.

Quadro 7 - Métodos criados para gerir as sessões do jogo

Métodos criados para gerir as sessões do jogo	
`Createsession`	Responsável por criar uma sessão, no qual é definido a quantidade de jogadores suportados pela sessão e se a conexão deve ser feita em uma rede LAN ou WAN. O jogador que inicia a sessão, torna-se o seu host e age como um servidor para os demais jogadores que ingressarem na sessão posteriormente.
`Findsessions`	Através desse método são retornadas as sessões ativas, no qual apresenta o número de jogadores presentes na sessão e o número máximo suportado por ela.
`Joinsession`	Após a utilização do método `Findsessions`, o `Joinsessions` é responsável pela introdução dos jogadores em alguma das sessões que foi retornada anteriormente.
`Destroysession`	Nesse método é finalizada uma sessão removendo todos os integrantes. Assim como o `Joinsession`, é necessário utilizar o método `Findsessions` para passar a sessão a ser destruída.

Fonte: elaborado pela autora.

Apesar de ter sido concluída a implementação desses métodos, o teste dos mesmos não trouxera resultados funcionais. O jogador *host* sucedia em criar uma sessão e entrar no *lobby* do jogo. Contudo, outros jogadores que

buscavam por sessões ativas, não conseguiam encontrar a sessão criada. Dessa forma, cada jogador que buscava entrar em um dos *lobbys* existentes, terminava por criar um outro pelo fato de não conseguirem identificar os demais já criados.

Buscando por soluções para este problema, foram encontradas outras formas de implementar essas funcionalidades, ainda recorrendo a uma classe C++, como por exemplo por meio de sessões do programa Steam. Não obstante, essas diferentes soluções retornavam o mesmo problema e, por fim, a conclusão encontrada foi de substituir a classe C++ por um Blueprint. A implementação dos métodos por meio de Blueprints permitiu os jogadores de encontrar sessões ativas e se unir a elas de acordo com a sua capacidade, criando sessões novas apenas quando não se encontram mais sessões com vagas. Vale ressaltar que as características definidas na criação de uma sessão foram de cinco jogadores máximo e utilização de rede WAN.

Após integrar uma sessão, os jogadores são enviados para um *lobby* que apresenta a quantidade de jogadores presentes e, para o host, também apresenta a opção de iniciar a partida. Assim que a partida é iniciada, os jogadores são transportados para o mapa da partida. A quantidade de jogadores é tratada e validada por meio do componente `GameState`, implementado por meio de Blueprints.

A criação das estruturas que foram utilizadas para tratar os atributos do objeto nave e do objeto arma foi feita em C++, mas precisou ser ajustada para a utilização dentro dos Blueprints por meio da adição de propriedades definidas pelo UE. A seguir, no Quadro 8 e Quadro 9 são apresentados exemplos do resultado das estruturas de arma que é complementada com a estrutura própria para os valores de disparo. Em ambos os exemplos, na primeira linha é adicionado um atributo para indicar que a estrutura criada na segunda linha é uma `USTRUCT`, podendo ser usada nos Blueprints. Na `linha 4` é indicado ao UE que o corpo da estrutura vai ser definido a seguir. Qualquer variável criada antes desse comando não será reconhecida em Blueprints. Nas `linhas` como a número `6`, indica-se ao UE que será criada uma variável, que ele chama de propriedade. Nas linhas restantes é feita a definição dos atributos que compões o disparo ou a arma.

Quadro 8 - Estrutura FRegShot contém os valores do disparo da arma

```
1  USTRUCT(BlueprintType)
2  struct FRegShot
3  {
4      GENERATED_BODY()
5
6      UPROPERTY(EditAnywhere, BlueprintReadWrite)
7          float range;
8      UPROPERTY(EditAnywhere, BlueprintReadWrite)
9          float value; // damage, slowness
10     UPROPERTY(EditAnywhere, BlueprintReadWrite)
11         float cooldown;
12     UPROPERTY(EditAnywhere, BlueprintReadWrite)
13         float lifetime;
14     UPROPERTY(EditAnywhere, BlueprintReadWrite)
15         float burnValue;
16     UPROPERTY(EditAnywhere, BlueprintReadWrite)
17         TEnumAsByte<EWeaponType> type;
18 };
```

Fonte: elaborado pela autora.

Quadro 9 - Estrutura FRegWeapon contém os valores da arma

```
1  USTRUCT(BlueprintType)
2  struct FRegWeapon
3  {
4      GENERATED_BODY()
5
6      UPROPERTY(EditAnywhere, BlueprintReadWrite)
7          int id;
8      UPROPERTY(EditAnywhere, BlueprintReadWrite)
9          bool isBasicWeapon;
10     UPROPERTY(EditAnywhere, BlueprintReadWrite)
11         FString name;
12     UPROPERTY(EditAnywhere, BlueprintReadWrite)
13         FString description;
14     UPROPERTY(EditAnywhere, BlueprintReadWrite)
15         FRegShot shot;
16 };
```

Fonte: elaborado pela autora.

Com as estruturas prontas, pode ser criado um componente para a tela do jogo que mostra a quantidade de vida, quando as armas estão disponíveis e o tempo decorrido da partida até então. Após criar esse componente da tela, pode

ser criada uma função para tratar a vida da nave, a qual será atualizada a todo momento para indicar a quantidade de vida da nave salva na estrutura. Dessa forma, sempre que a nave tiver a vida alterada, refletirá na tela do jogo.

Além da vida da nave, também foi feito o tratamento para alterar a velocidade da nave conforme a velocidade salva na estrutura. A quantidade de tempo na partida foi tratada da mesma maneira que o *timer* no lobby, tendo como única diferença um tratamento para que ao invés de decrementar o valor, ele aumente, a partir do início da partida. Com isso, os outros atributos que devem ser tratados são os atributos dos tiros.

Quando o jogador dispara a arma da nave é buscado o tempo de recarga do tiro e é tratado para que o componente da tela que mostra o atalho para a arma que disparou fique indisponível, impedindo o jogador de ativar as armas durante esse período de indisponibilidade e se tornando intuitivo para o jogador saber sempre quando a arma dele possui disparos ou não. Portanto, para o disparo da nave foi criado um objeto do tipo `BP_Projectile` que envia os valores do disparo para ser gerado e chama o método `Hit` quando atingir uma nave inimiga para que a vida dela seja tratada de acordo com o disparo. Além disso, é gerado um som de explosão quando o tiro acerta uma nave especificamente.

Caso o disparo seja do tipo ataque, ele vai descontar o valor do disparo da vida atual da nave. Disparos do tipo bomba descontam um valor menor da vida da nave, mas aplicam o efeito de queima na nave que causa danos durante o tempo aplicado. As armas do tipo lentidão não causam danos às naves inimigas, mas diminuem a velocidade da movimentação das naves por um determinado período após a colisão dos disparos. Por fim, o último tipo de arma sendo do tipo escudo que não é gerado o *spawn* do tipo `BP_Projectile`, mas sim um componente do tipo `BP_Shield`, responsável por criar um escudo temporário e visível aos jogadores que diminui o dano recebido dos disparos das naves inimigas.

Devido à utilização de sessões, os tiros e o escudo não podem ser gerados normalmente, porque senão apareceria somente ao jogador que ativou a arma correspondente. Para que os disparos sejam visíveis para todos, foi necessário criar uma função que roda no servidor, uma opção dos Blueprints que utiliza o servidor do Unreal Engine, e essa função chama outra função que roda em `Multicast`. Esse método `Multicast` faz o *spawn* do componente, seja ele o disparo ou o escudo, para que os demais jogadores possam visualizar os mesmos. O componente do escudo recebeu um tratamento diferente dos disparos, pois o método anexa esse componente ao objeto da nave do jogador que o ativou para que o escudo se movimente junto da nave e cubra o seu entorno em todo o momento que estiver ativado.

Com o projeto utilizado do UE, a nave já possuía o tratamento da movimentação, podendo se mover para os lados, para frente, para cima e para baixo. Contudo, o componente do tipo `Pawn`, utilizado para a nave, é um componente que o seu movimento não pode ser refletido para os outros, fazendo assim com que os jogadores não conseguissem ver o movimento uns dos outros durante as sessões.

Foi identificado então a existência de outro componente derivado do `Pawn`, chamado `Character`, que é capaz de replicar os movimentos das naves para todos os jogadores presentes na sessão. Contudo, isso resultou na mudança da movimentação, pois as funções que a compunham eram específicas do `Pawn` e não poderiam ser aplicadas no `Character`, fazendo assim que sejam alteradas as funções da movimentação além do seu componente.

Em virtude dessas mudanças, a fluidez da movimentação do projeto exemplo foi perdida, concedendo lugar à uma movimentação mais rígida e com certas limitações. Essas alterações também influenciaram o movimento da nave, impactando em algumas de suas limitações como a impossibilidade dela de permanecer parada e a inclinação que ela fazia para baixo conforme ela se virava. Com esse tratamento da movimentação, foi habilitada à nave a possibilidade de permanecer parada e a capacidade de se virar para os lados sem alterar a sua orientação vertical.

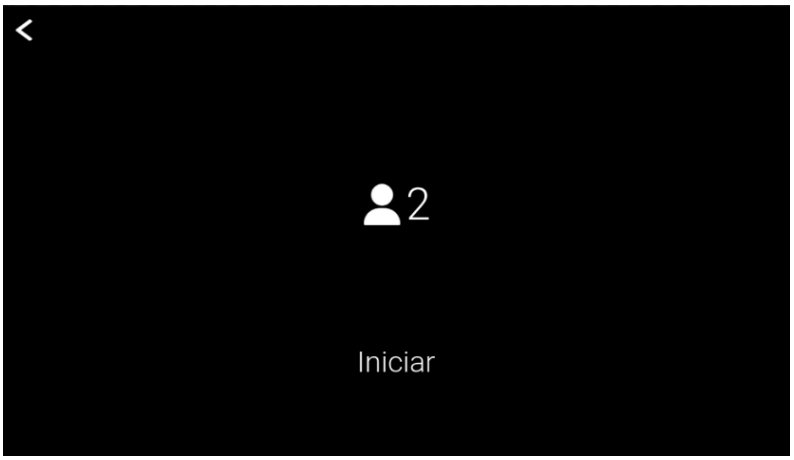
A verticalidade do jogo foi retirada no processo com o propósito de simplificar a jogabilidade, corrigir erros relacionados à orientação da nave e limitar a mira dos jogadores. Sendo assim, os tiros sempre são disparados na mesma altura das naves e retos em frente delas, aumentando a taxa de acerto dos disparos dos jogadores e tornando os jogos mais rápidos e objetivos.

Ao iniciar o jogo, o menu é apresentado com as opções para criar uma sessão ou se juntar a uma já existente, conforme Figura 10. Quando o usuário cria uma sessão ou prefere entrar em uma, ele é direcionado diretamente para a tela de *lobby* mostrada na Figura 11 que possui a quantidade de jogadores presentes e a opção para iniciar a partida.

Figura 10 - Menu inicial do jogo Space Royale

Fonte: elaborado pela autora.

Figura 11 - Tela de lobby do jogo Space Royale



Fonte: elaborado pela autora.

Quando o *host* da sessão inicia a partida são todos levados para o mapa do jogo, iniciando em pontos de *spawn* pré-definidos de forma que todos sejam invocados virados uns para os outros em uma área aleatória do mapa. Conforme Figura 12, durante a partida o jogador pode conferir sua barra de vida na parte superior da tela, a quantidade de pessoas restantes, o tempo de jogo e seus atalhos das armas, que fica desabilitado quando em tempo de recarga. Na Figura 13 e Figura 14 são apresentados exemplos de ativação da arma básica juntamente com a arma especial do tipo Ataque e ativação da arma especial do tipo escudo, respectivamente. Os tipos de arma Bomba e Lentidão são semelhantes ao tipo Ataque, somente tendo uma coloração diferente para identificação, sendo dourado para o primeiro e azul para o segundo. Quanto ao cenário do jogo, foram adicionados obstáculos espalhados pelo mapa para limitar a movimentação e permitir a localização dentro dele, como pode ser observado na Figura 15. Os obstáculos utilizados são objetos disponibilizados pelo Unreal de forma gratuita. Por fim, quando um jogador é eliminado ou quando ele vence a partida, é levado para uma tela com os resultados, na qual é apresentada sua colocação e o tempo em que ficou na partida, segundo Figura 16.

Figura 12 - Tela da partida no jogo Space Royale

Fonte: elaborado pela autora.

Figura 13 - Exemplo de tiro de arma básica e arma especial do tipo ataque

Fonte: elaborado pela autora.

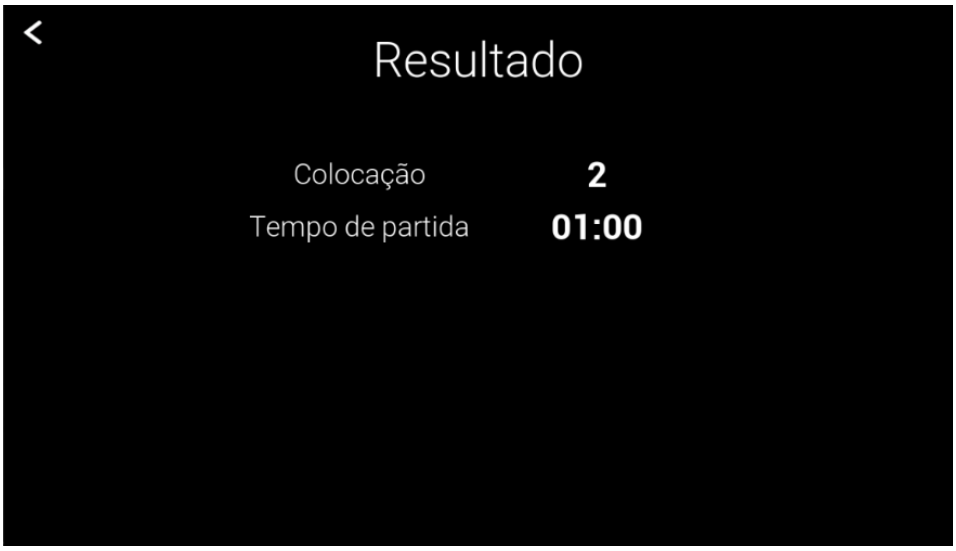
Figura 14 - Exemplo da ativação de arma especial do tipo escudo

Fonte: elaborado pela autora.

Figura 15 - Exemplo do cenário do mapa

Fonte: elaborado pela autora.

Figura 16 - Tela de resultados da partida no jogo Space Royale



Fonte: elaborado pela autora.

4 RESULTADOS

Neste trabalho foi desenvolvido um jogo multiplayer de Battle Royale que permite ao usuário batalhar contra outros jogadores com suas naves e com o auxílio de duas armas. Nesta batalha, apenas um jogador sai vencedor, então conforme os jogadores são eliminados, eles recebem a colocação em que terminaram. O Quadro 10 faz um comparativo entre este jogo e os apresentados nos trabalhos correlatos na seção 2.3. Com exceção do trabalho de Araujo e Cordenonsi (2003), todos os trabalhos são multiplayer, possibilitando que os jogadores possam se encontrar em uma partida ou em um mapa.

Três dos trabalhos apresentam um tipo de simulador: o trabalho de Trupel (2008) é um simulador de batalha naval e, tanto o trabalho de Araujo e Cordenonsi (2003), como este trabalho, são simuladores de nave. Com relação à perspectiva do usuário no jogo, Trupel (2008) e Jansen (2020) desenvolveram uma perspectiva *top-down*, na qual o mapa é apresentado de cima; Araujo e Cordenonsi (2003) trouxeram uma perspectiva em primeira pessoa, na qual o usuário se encontra dentro da nave para explorar o planeta; por sua vez, este trabalho foi feito em uma perspectiva de terceira pessoa, na qual o usuário segue a nave. Os trabalhos correlatos foram selecionados por possuírem similaridades com este no quesito de serem multiplayer ou por se tratar também de uma simulação de naves, no caso do trabalho de Araujo e Cordenonsi (2003).

Quadro 10 - Comparativo de todos os trabalhos correlatos

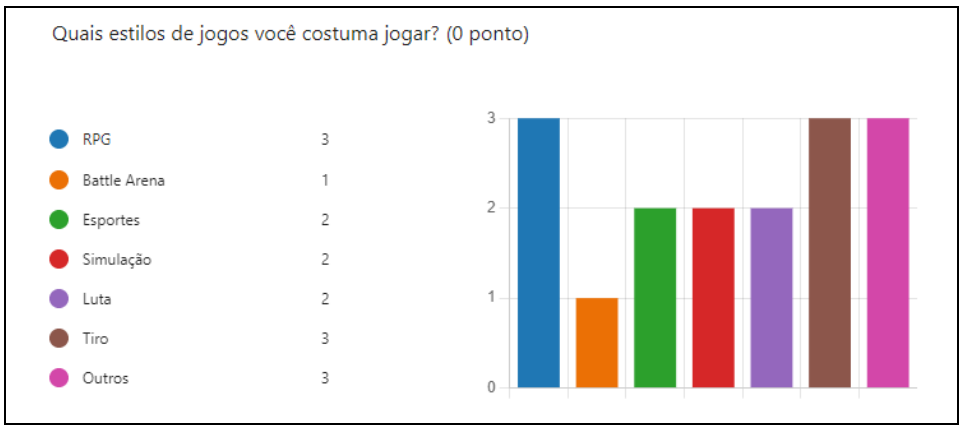
Características \ Trabalhos Correlatos	Trupel (2008)	Araujo e Cordenonsi (2003)	Jansen (2020)	SpaceRoyale
Ferramenta de desenvolvimento	Plataforma J2ME utilizando MIDP	Ambientes virtuais utilizando VRML e HTML	HTML 5 e WebSockets	C++, Unreal Engine e Blueprints
Multiplayer	X		X	X
Simulador	X	X		X
Perspectiva	Top-down	Primeira pessoa	Top-down	Terceira pessoa

Fonte: elaborado pelo autor.

Para validar o alcance deste trabalho com relação ao tipo de jogo apresentado e com relação aos resultados de jogabilidade foi realizado um questionário com 12 perguntas, incluindo temas como o perfil da pessoa com relação a jogos e outras diretamente sobre a jogabilidade. O questionário feito por meio do Office Forms foi respondido após 3 usuários testarem o jogo, disponibilizado por meio de um executável gerado pelo Unreal que contém o jogo e algumas dependências necessárias. Caso a máquina não possua tais dependências, um instalador é aberto para que elas possam ser resolvidas.

Dentre os 3 usuários, todos jogam jogos digitais, sendo que um deles joga com frequência e os outros dois costumam jogar diariamente. Referente aos tipos de jogos, a Figura 17 apresenta uma inclinação a jogos de RPG, tiros e outros estilos, enquanto jogos de Battle Arena são costumeiros para apenas um deles. Respondendo sobre a preferência entre jogos *Single Player* e jogos *Multiplayer*, dois deles escolheram a segunda opção com preferida.

Figura 17 - Gráfico ilustrando os tipos de jogos mais costumeiros



Fonte: elaborado pela autora.

Continuando o questionário, os usuários responderam a uma avaliação com relação aos seguintes aspectos do jogo: jogabilidade, animação, som, interação entre jogadores, regras do jogo e pontuações. As avaliações foram registradas com números de 1 a 5 e ficaram com média variando entre 3,0 e 3,6. Explicando a visão geral que tiveram sobre a jogabilidade, apontaram um problema no qual a nave de um dos jogadores parou de se movimentar, as naves ficaram invisíveis depois que restarem apenas dois jogadores e continuou mostrando somente os tiros uma da outra. Apesar dos problemas, um deles relatou que a jogabilidade foi fácil e ainda deu a sugestão de apresentar o atalho para acelerar a nave na tela, assim como são apresentados os atalhos para disparo das armas. Ao final, complementaram com o que acharam do jogo, declarando que gostaram do Battle Royale e do design do jogo, ou então que gostaram da dinâmica, mas não dos erros que foram relatados, pois impedia que os controles fossem usados como queriam, como no caso do movimento da nave que ficou travada.

5 CONCLUSÕES

O principal objetivo deste trabalho era o desenvolvimento de um jogo multiplayer no estilo Battle Royale de naves com o motor de jogos Unreal Engine. Para tal, foi utilizada a linguagem C++ em conjunto com Blueprints para aproveitar ao máximo o que o motor oferece. Foi realizada a modelagem da nave do jogo por meio da ferramenta Blender e pode ser alcançado um resultado positivo, conforme relatado na seção 4, que descreve que os usuários gostaram do design final. No tocante aos objetivos específicos, foi alcançado o primeiro objetivo com a entrega de um jogo funcional, no qual os jogadores se enfrentam e conseguem lutar até que reste apenas um. O segundo objetivo que

previa a categorização das armas também foi alcançado, mas parcialmente. Em alguns testes foi identificado que o escudo as vezes não bloqueia corretamente o dano recebido, apesar de ser aplicado à nave corretamente e permanecer nela somente pelo tempo previsto. Por fim, o último objetivo que era fornecer um jogo com boa jogabilidade também foi alcançado parcialmente. Conforme visto nos questionários realizados, apesar de os usuários terem achado fácil a jogabilidade do jogo, o problema que ocorre com a movimentação é um empecilho e, por isso, afeta a experiência que os jogadores podem ter.

O principal desafio encontrado foi a elaboração das sessões, pois inicialmente foi feita uma implementação em C++ que resultou em problemas para encontrar a sessão criada. Mesmo após testar outros métodos na linguagem, o problema persistia. Por esse motivo, foi optado por utilizar os métodos disponibilizados para Blueprints no Unreal Engine. A solução foi muito positiva, pois ofereceu uma saída funcional e com maior facilidade. Contudo, apesar de ter se resolvido o problema com as sessões, o fato de o usuário entrar em uma sessão requer tratamentos diferentes para que, tudo que um usuário realiza em sua máquina, possa ser visto por seu oponente no outro lado.

Assim, foi alterado o componente utilizado para tratar a nave do jogador e foi refeita a movimentação dela e os *spawn* dos tiros e do escudo. Para que tudo seja refletido entre os jogadores foi necessário utilizar funções com a opção de rodar no servidor, disponibilizado para Blueprints, de modo que ele se torna o responsável por aplicar a ação em todos os clientes.

De acordo com os resultados apresentados na seção 4, o resultado não foi o esperado, mas foi adequado. O jogo está funcionando como deveria com relação à gestão de sessões, disparo das armas, tratamento de dano, recarga das armas e resultados com a colocação do jogador. Contudo, está pecando com relação à movimentação da nave, que apresentou travamento para um dos usuários e que também ficou invisível para os inimigos em algum momento da partida. Por se tratar da movimentação do usuário pelo mapa, esse erro acaba afetando bastante a jogabilidade, pois atrapalha a *gameplay* dele.

Sendo assim, o objetivo de demonstrar o desenvolvimento de um Battle Royale utilizando o Unreal Engine pode ser alcançado, apesar de ainda haver melhorias a serem feitas no resultado geral do jogo. O que se acredita ser um grande motivo da dificuldade encontrada com o motor, foi as tentativas de implementar algumas funções sem o conhecimento de que ele não permite alguns comandos em C++. Por exemplo, todas as variáveis criadas, precisam ser inicializadas em sua criação, pois inicializar a variável na linha seguinte à sua criação já leva o motor a estourar e fechar com erros. Por conta disso, também acabou sendo usado os Blueprints para, pelo menos, 80% da implementação. Mesmo os tratamentos que foram deixados em C++ por conta da facilidade poderiam ter sido implementados por Blueprints sem problemas, mas ao final, a implementação mista foi a ideal para o projeto.

Deste modo, as possíveis extensões deste trabalho são:

a) melhor a performance durante a partida;
b) corrigir erros com a movimentação da nave;
c) implementar a possibilidade de listar as sessões encontradas, para que o usuário possa escolher em qual quer entrar;
d) aumentar a capacidade de jogadores por sessão;
e) implementar um perfil para o usuário, no qual possam ser registradas as suas conquistas;
f) implementar uma lista de amigos para que os jogadores possam manter contato após as partidas;
g) implementar a delimitação de zona ao mapa com uma nuvem tóxica que vai suprimindo as áreas das pontas até o centro e queima jogadores que a tocarem;
h) implementar uma opção de rever a partida, no qual os jogadores poderão retornar à partida para assisti-la.
i) implementar uma lista de amigos para que os jogadores possam manter contato após as partidas.

REFERÊNCIAS

AHN, Jin Kyeong. A study on game dynamics of Battle Royale genre. **Journal of Korea Game Society**. [S.L.], v. 17, n. 5, p. 27–38, 2017. Disponível em: https://doi.org/10.7583/JKGS.2017.17.5.27. Acesso em: 26 jun. 2022.

ARAUJO, Fabrício Viero de; CORDENONSI, Andre Zanki. Planeta X: desenvolvimento de um jogo virtual utilizando a tecnologia vrml. In: 8VO TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO, 8., 2003, Santiago. **Proceedings [...]**. Santiago: Universidade de Chile, 2003. p. 1-15. Disponível em: http://www.tise.cl/vol2003/TISE2003/Planeta%20X.pdf. Acesso em: 26 jun. 2022.

CHOI, Gyuhyeok; KIM, Mijin. Battle Royale Game: In Search of a New Game Genre. **International Journal of Culture Technology**, Busan, v.2, n.2, p. 5-11, Jun. 2018. Disponível em: http://203.241.190.58/iacst/Journals/ijct-2-2.pdf. Acesso em: 21 set. 2021.

JANSEN, Paulo Ricardo. **Fawe**: Jogo Multiplayer Online Usando Html5 e Websocket. 2020. 82 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade do Sul de Santa Catarina, Florianópolis, 2020. Disponível em: https://repositorio.animaeducacao.com.br/bitstream/ANIMA/16808/1/TCC%20-%20Paulo%20Ricardo%20Jansen%20%2815%29.pdf. Acesso em: 26 jun. 2022.

GAMMARANO, Igor de Jesus Lobato Pompeu; COSTA, Everaldo Marcelo Souza da. O futuro é multiplayer: interação social como elemento de preferência dos usuários de jogos online. **Pretexto**. [S.L.], v. 21, n. 3, p. 120-136. 11 dez. 2020. Disponível em: http://revista.fumec.br/index.php/pretexto/article/view/6098. Acesso em: 26 jun. 2022.

LIMA, Luiz Fernando Martins de; SILVA, Thais Maria Gonçalves da. Fascismo em Battle Royale, de Koushun Takami: fascismo em battle royale, de koushun takami. **Revista da Universidade Federal de Minas Gerais**, Belo Horizonte, v. 24, n. 12, p. 78-97, maio 2018. Disponível em: https://doi.org/10.35699/2316-770X.2017.12602. Acesso em: 27 jun. 2022.

MANNINEN, Tony. Interaction Forms and Communicative Actions in Multi-player Games. **Game Studies**. [S.L.], v. 3, n. 1, p. 1-10. maio 2003. Disponível em: https://www.researchgate.net/profile/Tony-Manninen/publication/220200727_Interaction_Forms_and_Communicative_Actions_in_Multiplayer_Games/links/5c6e7761a6fdcc404ec21869/Interaction-Forms-and-Communicative-Actions-in-Multiplayer-Games.pdf. Acesso em: 26 jun. 2022.

PANTEL, Lothar; WOLF, Lars C.. On the Impact of Delay on Real-Time Multiplayer Games. In: 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video, 12., 2002, Miami. **Proceedings [...]**. New York: [S.N.], 2002. p. 23-29. Disponível em: https://dl.acm.org/doi/10.1145/507670.507674. Acesso em: 26 jun. 2022.

ROSENBUSCH, Hannes; RÖTTGER, Jonas; ROSENBUSCH, David. Would Chuck Norris Certainly Win the Hunger Games? Simulating the result reliability of battle royale games through agent-based models. **Simulation & Gaming**. [S.L.], v. 51, n. 4, p. 461-476. ago. 2020. Disponível em: https://journals.sagepub.com/doi/metrics/10.1177/1046878120914336. Acesso em: 26 jun. 2022.

TRUPEL, Vilson. **Desenvolvimento de um Jogo Multiplayer para Celular**. 2008. 68 f. TCC (Graduação) - Curso de Ciências da Computação, Universidade Regional de Blumenau, Blumenau, 2008. Disponível em: http://dsc.inf.furb.br/arquivos/tccs/monografias/2008-2-27-vf-vilsontrupel.pdf. Acesso em: 26 jun. 2022.

ZHOU, Nina; RYAN, Lisa; DOERNER, Matt; PIERSE, Christopher. AI Coach for Battle Royale Games. In: Chi Play '21: 2021 Annual Symposium on Computer-Human Interaction in Play, 1., 2021, [S.L.]. **Proceedings [...]**. [S.L.]: Association for Computing Machinery, 2021. p. 280-286. Disponível em: https://dl.acm.org/doi/abs/10.1145/3450337.3483456. Acesso em: 26 jun. 2022.

APÊNDICE A – DESCRIÇÃO DOS CASOS DE USO

Conforme diagrama da Figura 5 este apêndice apresenta a descrição dos principais casos de uso especificados, sendo eles: UC03 no Quadro 11, UC04 no Quadro 12, UC01 no Quadro 13 e UC06 no Quadro 14.

Quadro 11 - Descrição do UC03 – Juntar-se a uma sessão

Casos de Uso	UC03 – Juntar-se a uma sessão.
Descrição	Este caso demonstra o procedimento para a entrada em um lobby.
Ator	Jogador.
Pré-condição	Nenhuma.
Fluxo principal	1. O jogador clica o menu para entrar em uma sessão. 2. O sistema apresenta uma tela de carregamento enquanto busca por sessões disponíveis e adiciona o usuário à primeira sessão.
Fluxo alternativo	A1 – O sistema não encontra nenhuma sessão: 1. O sistema volta o jogador ao menu inicial. A2 – Ocorreu um erro ao tentar juntar-se a uma sessão ou ela já está cheia: 1. O sistema volta o jogador ao menu inicial.
Pós-condição	O jogador visualiza a tela de lobby dentro de uma sessão.

Fonte: elaborado pela autora.

Quadro 12 - Descrição do UC04 – Receber dano

Casos de Uso	UC04 – Receber dano.
Descrição	Este caso demonstra o recebimento de dano ao ser atingido por um tiro inimigo.
Ator	Jogador.
Pré-condição	O jogador deve estar em uma partida.
Fluxo principal	1. O jogador é atingido por um tiro inimigo. 2. O sistema aplica o dano equivalente ao tiro à nave do jogador de acordo com o tipo de disparo, podendo afetar a sua vida ou a velocidade.
Fluxo alternativo	A1 – A vida do jogador chega a 0: 1. O sistema remove o jogador da sessão e apresenta a tela com os resultados da partida.
Pós-condição	A vida ou a velocidade da nave do jogador é diminuída.

Fonte: elaborado pela autora.

Quadro 13 - Descrição do UC01 – Movimentar a nave

Casos de Uso	UC01 – Movimentar a nave.
Descrição	Este caso demonstra o procedimento para a movimentação da nave.
Ator	Jogador.
Pré-condição	O jogador deve estar em uma partida.
Fluxo principal	1. O jogador clica em um dos atalhos: Shift, Seta para a direita ou Seta para a esquerda. 2. O sistema recebe o atalho e move a nave na direção correspondente à opção atrelada a ele.
Fluxo alternativo	A1 – A nave está colidindo com algum objeto: 1. O sistema move a nave na direção mais próxima da esperada sem passar por cima do objeto, mas tentando circulá-lo.
Pós-condição	A nave é movida para uma nova posição.

Fonte: elaborado pela autora.

Quadro 14 - Descrição do UC06 – Disparar arma

Casos de Uso	UC06 – Disparar arma.
Descrição	Este caso demonstra o procedimento para o disparo das armas.
Ator	Jogador.
Pré-condição	O jogador deve estar em uma partida.
Fluxo principal	1. O jogador clica em um dos dois atalhos: Q ou W. 2. O sistema verifica se a arma está em tempo de recarga e, caso não esteja, faz o disparo do tiro ou ativação do escudo, dependendo do tipo de arma.
Fluxo alternativo	A1 – Caso a arma esteja em tempo de recarga: 1. O sistema não realiza o disparo e apenas acaba este caso de uso.
Pós-condição	A arma equivalente ao atalho é ativada e entre em tempo de recarga.

Fonte: elaborado pela autora.