INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Jogo sério para dispositivos móveis para educação sobre riscos naturais

Carlos Daniel Da Silva Rosa

Licenciado

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e Multimédia

Orientador : Prof. Doutor Rui Jesus

Júri:

Presidente: Prof. Doutor Pedro Viçoso Fazenda

Vogais: Prof. Doutor Pedro Albuquerque Santos
Prof. Doutor Rui Manuel Feliciano De Jesus

Setembro, 2023

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Jogo sério para dispositivos móveis para educação sobre riscos naturais

Carlos Daniel Da Silva Rosa

Licenciado

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e Multimédia

Orientador : Prof. Doutor Rui Jesus

Júri:

Presidente: Prof. Doutor Pedro Viçoso Fazenda

Vogais: Prof. Doutor Pedro Albuquerque Santos
Prof. Doutor Rui Manuel Feliciano De Jesus

Setembro, 2023

Agradecimentos

O desenvolvimento desta tese foi apenas possível com a ajuda e apoio de várias pessoas. Por isso, gostaria de expressar a minha sincera gratidão:

- Aos meus amigos e familiares que me apoiaram ao longo desta jornada e me deram motivação para continuar.
- A todos os voluntários que colaboraram para testar a aplicação desenvolvida nesta tese e forneceram as suas opiniões sinceras sobre a mesma.
- Ao Instituto Superior de Engenharia de Lisboa (ISEL) e os seus respetivos docentes e funcionários que me ensinaram e ajudaram ao longo do meu percurso académico.
- Ao Professor Doutor Rui Jesus, Professor Doutor Arnaldo Abrantes, Professora Maria Ana Baptista e ao Steven Hawthorn por me guiarem e me ajudarem ao longo do desenvolvimento desta tese.

Obrigado pela vossa ajuda, sem a qual não teria conseguido realizar este trabalho.

Resumo

Os jogos sérios têm a finalidade de ajudar a transmitir competências aos jogadores, sendo que essas competências tendem a ser mais duradouras em comparação com os métodos de ensino convencionais devido à interação que proporcionam aos jogadores. Nesse contexto, foi desenvolvido um jogo sério que procura transmitir informação aos jogadores sobre como se preparar e reagir em situações de tsunami, ao colocá-los num ambiente 3D onde ocorre um tsunami.

A dimensão séria do jogo é incorporada no ciclo de evolução do jogador, que engloba os seguintes elementos: o próprio jogo, pontos e a loja. O jogo é constituído por três níveis, cada um transmitindo competências distintas. Os pontos são obtidos à medida que o jogador sobrevive ou morre, com maior pontuação sendo atribuída às decisões mais acertadas. Na loja, os jogadores podem gastar os pontos conquistados para adquirir melhorias que poderão ser usadas em futuras partidas. Seguindo esse ciclo, o jogador progride gradualmente, até eventualmente aprender a maneira adequada de sobreviver a um tsunami.

O jogo foi avaliado com 10 utilizadores relativamente à usabilidade e experiência do utilizador. A avaliação revelou que o sistema de evolução implementado não é recomendado para utilizadores sem conhecimento prévio em jogos. No entanto, o jogo conseguiu oferecer uma experiência positiva e transmitir conhecimentos aos jogadores sobre como reagir em situações de tsunami. Esses resultados indicam que o jogo, embora apresente desafios para jogadores sem experiência prévia, desempenha de forma aceitável o papel de transmitir informações enquanto proporciona uma experiência agradável.

Palavras-chave: Jogo Sério, Desastres Naturais, Tsunami.

Abstract

Serious games are designed to teach players skills, skills that tend to be more enduring than conventional teaching methods due to the interaction they allow the player. With this in mind, a serious game has been developed that teaches players how to prepare for and react to a tsunami, placing them in a 3D environment where a tsunami is occurring.

The serious aspect of the game is present in the player's evolution cycle, which encompasses the following elements: the game itself, points, and the store. The game consists of three levels, each teaching different skills. The points are earned in the game when the player survives or dies, and players earn more points by making better decisions. In the store, players can spend earned points to obtain improvements that can be used in future plays. Following this cycle, the player gradually progresses and eventually learns what to do to survive a tsunami.

The game was evaluated by 10 users regarding the usability and the user experience. The evaluation demonstrated that the evolution system implemented is not suitable for users without prior gaming knowledge. However, it was able to provide a positive experience and impart knowledge to players about how to react in a tsunami situation. These results indicate that, although the game presents challenges for players without prior experience, it performs its role well in conveying information while providing an enjoyable experience.

Keywords: Serious Game, Natural disasters, Tsunami.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

JS Jogo Sério. 5, 6, 7, 54, 56, 58, 62, 63, 64, 65, 66, 67

NPC Non-player character. 20, 28, 44

SUS System Usability Scale. 56, 57, 59, 70

UI User Interface. 47

URP Universal Render Pipeline. 29

Glossário

CollectibleObject Componente presente nos recursos que são coletáveis. 27

Cutscene Vídeo não interativo que ocorre entre os segmentos de um jogo eletrônico e retrata parte do plano de fundo ou enredo do jogo. 18

Scene Um ambiente que contém objetos e componentes que constituem o jogo ou uma parte específica do jogo.. 50, 51

Scriptable Object Contentores de dados que guardam atributos e métodos. 48, 49

Tile Uma peça, geralmente em formato quadrático, que quando está adjacente a outras peças forma uma estrutura como um mapa.. xvi, xvii, 28, 29, 42, 43, 44

1

Introdução

Os tsunamis representam uma das forças naturais mais destrutivas e imprevisíveis que podem acontecer no mundo. Essas ondas gigantes, muitas vezes desencadeadas por terremotos submarinos, vulcões ou deslizamentos de terra no oceano, têm o potencial de causar devastação em larga escala em todas as áreas costeiras do planeta. Em Portugal, utilizando a Base de Dados Históricos Globais de Tsunamis do National Geophysical Data Center, NOAA e World Data Service (NGDC/WDS) [7], foram registadas várias ocorrências de tsunamis a atingirem locais como Lisboa, os Açores e até mesmo a Ilha da Madeira.

Uma das características mais perigosas dos tsunamis é sua capacidade de viajar a grandes velocidades em mar aberto e, quando se aproximam da costa, aumentam de altura dramaticamente. Isso significa que, mesmo que a onda seja relativamente pequena em alto mar, ela pode crescer até dezenas de metros de altura quando atinge a costa, inundando e destruíndo tudo em seu caminho. Assim como os terremotos, os cientistas não conseguem prever quando ocorrerá o próximo tsunami. No entanto, os Centros de Alerta de Tsunamis podem detetar tsunamis antes de atingirem a costa, fornecendo informações como altura da onda, tempo de chegada, localização e duração. Por outro lado, quando a fonte do tsunami está demasiado perto da costa, os Centros de Alerta de Tsunamis podem não adquirir essa informação a tempo [33].

Portanto, compreender os sinais naturais de um tsunami, ter planos de evacuação eficazes e educar as comunidades costeiras sobre os perigos dos tsunamis é fundamental para mitigar seu impacto mortal.

1.1 Motivação, Objetivos e Contribuições

No trabalho publicado em [3], foi conduzido um estudo abordando o conhecimento das crianças em Marrocos sobre o risco de tsunamis. Marrocos, um país do norte da África, compartilha uma localização costeira próxima de Portugal, tornando-se suscetível a riscos semelhantes de tsunamis. A pesquisa revelou que, embora as crianças compreendam a relação entre tsunamis e terremotos, enfrentam uma carência de informações essenciais. Por exemplo, têm dificuldade em identificar os sinais naturais de risco de tsunami e enfrentam incertezas relacionadas ao risco de tsunamis em Marrocos. Este cenário destaca um problema crítico, uma vez que as crianças constituem um grupo mais vulnerável do que os adultos durante um evento de tsunami, devido à sua locomoção limitada e dependência dos adultos [32]. Portanto, é de extrema importância educá-las eficazmente sobre os perigos dos tsunamis, capacitando-as para agir com autonomia em tais situações. As práticas atuais de simulação de tsunamis nas escolas não se baseiam em uma modelagem precisa das inundações causadas por tsunamis. Como resultado, é imperativo desenvolver ferramentas capazes de melhorar a educação sobre desastres e a capacidade de tomada de decisão em situações de emergência [33].

Para abordar esta questão, pretende-se desenvolver um jogo sério com o objetivo de instruir os utilizadores sobre como reagir em situações de tsunami. Embora os jogos sérios não sejam amplamente populares como método de ensino, eles existem há muitos anos, com vários jogos criados para ensinar habilidades aos jogadores. O jogo sério proposto visa simular um cenário de risco, aumentando a conscientização sobre os perigos associados a tsunamis e promovendo a tomada de decisões de forma inovadora, interativa e segura. Além de oferecer um método de ensino aprimorado, este jogo sério é focado na adaptabilidade para atender às necessidades individuais de cada jogador. Os utilizadores possuem diferentes níveis de facilidade de compreensão de certos conceitos, e é por isso que uma das principais motivações deste projeto é a implementação de algoritmos e métodos que permitam uma aprendizagem personalizada. O jogo se ajustará às lacunas de conhecimento do jogador, fornecendo feedback para aumentar a eficiência da aprendizagem. Os jogos sérios oferecem uma plataforma viável para a implementação destes algoritmos e métodos, e este trabalho visa explorar a eficácia

desse método educacional, registrando os resultados como referência para a eficácia geral dos jogos sérios. Assim, os objetivos especificos deste projeto são:

- Desenvolver um jogo sério para dispositivos móveis que transmita o conhecimento necessário para reagir a tsunamis;
- Utilizar técnicas de aprendizagem automática para treinar agentes que possam ajudar o utilizador;
- Avaliação sobre a efetividade do jogo sério em transmitir conhecimentos.

Este projeto oferece as seguintes contribuições científicas e técnicas:

- Uma aplicação móvel para Android capaz de ensinar os utilizadores como reagir numa situação de tsunami;
- Uma estudo sobre a eficácia dos jogos sérios na transmissão de informação aos utilizadores;
- Exploração de algoritmos e tecnologias para proporcionar uma aprendizagem mais eficiente.

O código do projeto desenvolvido está disponível no `https://github.com/deimered/Tendenko_Tsunami`.

Este trabalho foi desenvolvido no contexto do projeto MOBILIZE, "Jogo sério para dispositivos móveis para educação sobre riscos naturais", com a referência IPL/2022/MOBILIZE_ISEL, financiado pelo Instituto Politécnico de Lisboa.

1.2 Estrutura do documento

Este documento é composto por seis capítulos que abordam os seguintes tópicos:

- Capítulo 1 - Introdução a este trabalho, apresentando a motivação, os objetivos, as contribuições e a estrutura do documento;
- Capítulo 2 - Discussão sobre a definição de um jogo sério e exemplos de jogos sérios existentes relacionados com tsunamis;

- Capítulo 3 - Apresentação do método proposto para abordar o problema descrito neste documento;
- Capítulo 4 - Demonstração técnica de cada componente que compõe o jogo;
- Capítulo 5 - Apresentação da metodologia utilizada para avaliar o jogo e discussão dos resultados da avaliação;
- Capítulo 6 - Conclusão do trabalho e discussão sobre trabalhos futuros.

2

Trabalhos relacionados

2.1 Jogos sérios

Este trabalho baseia-se em criar um Jogo Sério (JS) com o intuito de transmitir uma mensagem de perigo e ensinar competências em como lidar com tsunamis. Nesta secção, vai ser discutido o conceito de um JS, como o mesmo se distingue de outras aplicações, se as características do mesmo conferem a necessidade de uma categorização num grupo distinto dos demais jogos e a efetividade dos mesmos.

O primeiro ponto a ser abordado é a definição de um JS. A primeira vez que essa designação foi utilizada foi no livro [26], escrito por Clark C. Abt. Nesse livro, o autor define jogos sérios como aqueles que possuem propósitos educacionais claros e explícitos, não sendo jogados primariamente para entretenimento. Este livro foi escrito durante o periodo da Guerra Fria e explora a possibilidade de utilizar jogos sérios para fins educacionais. Além disso, o autor menciona vários exemplos de jogos usados tanto dentro quanto fora das escolas, como jogos de mesa, cartas, de rua e computador, entre outros. Apesar do foco na educação, o livro reconhece que isso não significa que os jogos sérios não possam ser divertidos ou que não devam ser divertidos.

No trabalho publicado em [10], é apresentada uma abordagem para distinguir os jogos sérios de outras aplicações similares. Segundo o artigo, os jogos sérios são caracterizados por possuírem funções utilitárias que permitem a transmissão de conhecimento através da sua estrutura. Esses jogos operam fora do mercado de entretenimento convencional, sendo considerados artefatos digitais, analógicos ou híbridos. Uma função utilitária é uma característica que possui valor devido à sua utilidade na consecução de um objetivo específico. Nos jogos sérios, esses objetivos podem ser agrupados em três categorias principais: transmitir uma mensagem, oferecer treinamento ou fornecer transmissão de dados [10]. A transmissão de mensagem envolve a entrega de conteúdo educacional, informativo, persuasivo ou subjetivo, podendo um jogo conter múltiplos tipos de mensagem. O treinamento busca melhorar as habilidades cognitivas ou físicas do jogador. A transmissão de dados refere-se à troca, manipulação e coleta de informações dos jogadores. É importante destacar que essas funções utilitárias podem ser aplicadas de duas maneiras: interna ou externamente. A aplicação interna significa que a função utilitária está incorporada no jogo, enquanto a aplicação externa implica que a função está associada ao jogo por meio de atividades planeadas fora do contexto original do desenvolvedor. Por exemplo, pode-se utilizar um jogo de guerra para ensinar soldados sobre armas de fogo, mesmo que esse não tenha sido o propósito inicial do jogo. A partir destas características discutidas, é possível estabelecer uma definição concreta sobre o que são os jogos sérios.

O segundo tópico a ser abordado é a distinção entre os jogos sérios e outras aplicações existentes. Conforme definido anteriormente, a característica fundamental de um JS é sua capacidade de transmitir conhecimento por meio de funções utilitárias. No entanto, há jogos que não são considerados sérios, mas ainda assim podem transmitir conhecimentos aos jogadores, mesmo que não tenha sido intencionalmente planeado pelos criadores. Isso levanta a questão da diferença entre os jogos sérios e esses jogos com elementos sérios, que não está claramente definida. Para abordar essa questão, pode-se fazer referência ao artigo mencionado anteriormente [10], que introduz um conceito chamado "Serious Re-purposing" (reaproveitamento sério). O Serious Repurposing envolve a transformação de jogos em jogos sérios por meio de modificações externas. Essas modificações devem adicionar pelo menos uma nova função utilitária ao jogo. Essas alterações externas são o ponto crucial que diferencia os jogos sérios de outros tipos de jogos. Enquanto um JS é desenvolvido desde o início com as funções utilitárias em mente, outros jogos precisam passar por modificações para serem aplicados da mesma forma que os jogos sérios.

Os jogos sérios frequentemente são comparados a dois outros conceitos: simulação e gamificação. Uma simulação é uma representação de um modelo de comportamento ou processo baseado no mundo real, permitindo verificar sua evolução por meio da execução e interações propostas. Por outro lado, um JS tem como objetivo principal transmitir conhecimento aos jogadores, e, portanto, não precisa ser realista em seu ambiente. No entanto, alguns jogos sérios utilizam o realismo para aprimorar a aprendizagem dos utilizadores. Por exemplo, este projeto faz uso de elementos realistas para imergir os jogadores em uma situação de tsunami. Nesses casos, a distinção entre um JS e um simulador pode ser difícil de determinar [10][18]. No entanto, assim como um JS difere de outros tipos de jogos em seu propósito, um JS também difere de uma simulação no conceito de que os jogos sérios são desenvolvidos com funções utilitárias em mente, enquanto um simulador é focado em reproduzir um fenômeno real [18]. Além disso, usando a definição encontrada em [10], um jogo possui um objetivo e termina quando o jogador alcança esse objetivo. Por outro lado, um simulador não possui um objetivo final ou um fim que o jogador possa alcançar, portanto, não é considerado um jogo.

Em relação à gamificação, que consiste em adicionar componentes de jogos a atividades existentes, e.g. pontuação ou conquistas, para torná-las mais divertidas e interessantes, tanto os jogos sérios quanto a gamificação incorporam elementos de jogos e buscam tornar suas atividades mais atraentes. No entanto, a diferença fundamental está no uso desses elementos. A gamificação envolve adicionar elementos a uma tarefa já existente, enquanto um JS é criado desde o início com as funções utilitárias planeadas [18]. Após estas comparações, é possível concluir que os jogos sérios apresentam diferenças significativas em relação a outras aplicações e merecem uma categorização própria.

Por fim, vamos discutir a efetividade dos jogos sérios. Métodos tradicionais de educação baseados na relação professor-aluno frequentemente estão associados à memorização de fatos em vez de promover o entendimento e a reflexão sobre os mesmos [23][13]. Como resultado, esses métodos de ensino são muitas vezes considerados ineficientes ou passivos em comparação a outras abordagens[22]. Por outro lado, os jogos sérios oferecem ambientes experimentais que tornam a aprendizagem mais ativa para os jogadores. Ao colocar os utilizadores nesses ambientes, é promovido o uso de lógica e pensamento crítico para resolver os problemas apresentados. Desta forma, os jogadores desenvolvem um entendimento dos sistemas complexos que estão sendo ensinados e estabelecem associações desses sistemas com situações reais [23][22]. Com base neste

conhecimento, os jogos sérios podem ser considerados efetivos.

Para comprovar essa efetividade, foram realizadas pesquisas sobre estudos que examinam a eficácia dos jogos sérios. Dois artigos [23] e [24] discutem os jogos sérios neste contexto. Esses artigos foram selecionados porque investigaram a efetividade de diversos jogos sérios, abordando também o tema da comunicação de risco, que é também um tema neste projeto. Os resultados desses estudos indicam que os jogos sérios são excelentes ferramentas para disseminar conhecimento sobre riscos, identificar perigos, promover ações preventivas e incentivar a perspetiva do jogador em relação aos riscos. No entanto, existem poucos estudos que avaliam quantitativa ou qualitativamente a efetividade dos jogos sérios [23]. Além disso, é difícil comparar a eficácia de dois jogos sérios diferentes, uma vez que não existe um formato padrão seguido por todos os jogos, o que dificulta a comparação [24]. Adicionalmente, os jogos sérios não podem ser vistos como uma ferramenta capaz de disseminar conhecimento sobre desastres sozinha, pois muitos aspetos e detalhes, como distância da fonte ou variáveis de impacto, são simplificados para não comprometer a jogabilidade [23]. No entanto, como mencionado anteriormente, os jogos sérios são capazes de transmitir conhecimento sobre riscos, perigos e medidas preventivas para desastres. Esse conhecimento melhora a tomada de decisões em situações de risco, e quando compartilhado com crianças, também é disseminado para indivíduos não informados (os pais das crianças) [24]. Portanto, embora seja difícil determinar concretamente a efetividade dos jogos sérios, existem várias evidências de que eles influenciam positivamente a aquisição de conhecimento.

Em conclusão, nesta discussão, foi possível estabelecer que os jogos sérios se distinguem de outras aplicações pela intenção de transmitir conhecimento e pelo uso de funções utilitárias. A efetividade destes jogos é difícil de ser comprovada devido à falta de estudos sobre o assunto, no entanto, podem ser considerados ferramentas eficazes para a transmissão de conhecimento.

2.2 Jogos relacionados

Na fase de planeamento do jogo sério, foram efetuadas pesquisas sobre outros jogos já existentes com objetivos semelhantes aos objetivos do jogo proposto neste documento. Deste forma, é possivel determinar uma ideia base do que um jogo deste género deve possuir e permite esclarecer o que já foi feito e o que o jogo a ser desenvolvido pode implementar para se destacar dos jogos já existentes.

Começamos por abordar o jogo Treme-Treme, desenvolvido por uma colaboração entre o Departamento de Engenharia Civil (grupo de Engenharia Sísmica) e o Departamento de Engenharia Informática do Instituto Superior Técnico do IST, a empresa de jogos Dreamstudios e a empresa de design Flaidisaine, no âmbito do projeto europeu UPStrat-MAFA ("Urban disaster Prevention Strategies using MAcroseismic Fields and FAult Sources") [31]. O Treme-Treme concentra-se nos desastres naturais de terremoto e tsunami. No início do jogo, o jogador pode escolher uma de duas personagens para ser seu avatar, um rapaz chamado Sunami ou uma rapariga chamada Terramota. Em seguida, o jogador começa a jogar cada um dos cinco níveis do jogo. Cada nível apresenta um desafio diferente e ensina uma lição valiosa ao jogador, como o que fazer em caso de terremoto ou como preparar um kit de emergência. Além disso, após cada desafio, são feitas algumas perguntas ao utilizador. Embora essas perguntas não sejam necessárias para passar ao próximo nível, servem para avaliar o conhecimento e compreensão do utilizador sobre o que foi aprendido no nível jogado. A componente séria deste jogo reside nas competências ensinadas em cada nível e nas perguntas realizadas ao final de cada um. A Figura 2.1 ilustra uma fase do jogo Treme-Treme.

Figura 2.1: Desafio no Treme-Treme.

Outro jogo relevante foi o Disasters Master. Este jogo foi desenvolvido pelo governo dos Estados Unidos em conjunto com a FEMA (Federal Emergency Management Agency), como parte da iniciativa pública da Ready.gov [24], e foi direcionado para crianças entre 6-11 anos. O jogo consiste numa história em quadrinhos que narra as aventuras de um grupo de jovens num acampamento de verão no Colorado, que posteriormente se dispersam por diferentes partes dos Estados Unidos, cada um enfrentando um desastre natural específico. Os desastres naturais abordados no jogo incluem incêndios florestais, tornados, furacões, apagões, incêndios em casa, tempestades, tempestades de inverno, ondas de calor, terremotos e tsunamis. Durante o jogo, os jogadores seguem a narrativa de um dos personagens principais e são apresentados a várias perguntas que devem ser respondidas corretamente para ganhar pontos. Ao acumular um número específico de pontos, o jogador recebe uma senha para desbloquear o próximo nível [9]. A componente séria deste jogo está presente nas perguntas feitas aos jogadores. Cada pergunta transmite informação sobre como agir diante de cada tipo de desastre natural. A Figura 2.2 ilustra a parte sobre Tsunamis do jogo Disasters Master.

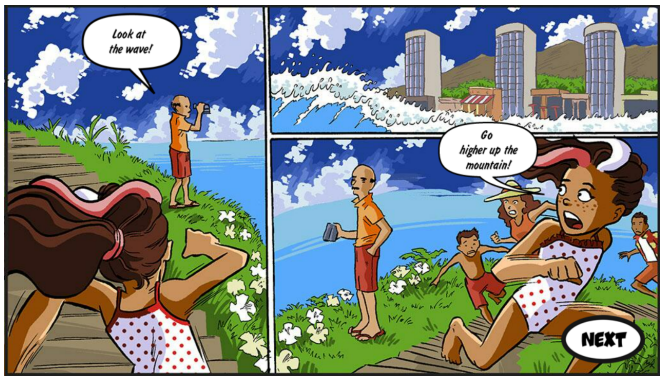

Figura 2.2: Página do capítulo sobre tsunamis do Disaster Master.

O Stop Disasters é outro jogo com objetivos semelhantes. Foi desenvolvido pela United Nations Office for Disaster Risk Reduction (UNDRR) em colaboração com a Playerthree, uma empresa de jogos digitais sediada em Londres. Foi projetado para crianças entre 9-16 anos [24]. O objetivo do jogo é construir infraestruturas e defesas ao redor de uma comunidade para reduzir o impacto dos seguinte tipos de desastres naturais: furacões, incêndios florestais, terremotos, enchentes e tsunamis [27]. Durante o jogo, o jogador tem recursos limitados que devem ser usados de forma estratégica para construir as infraestruturas e defesas, planeando a melhor maneira de proteger a comunidade. A abordagem séria do jogo está em ensinar às crianças como a localização e os materiais de construção das infraestruturas podem influenciar na redução do

impacto dos desastres naturais. As Figuras 2.3 e 2.4 mostram dois screenshots do jogo Stop Disasters.

Figura 2.3: Comunidade a proteger.

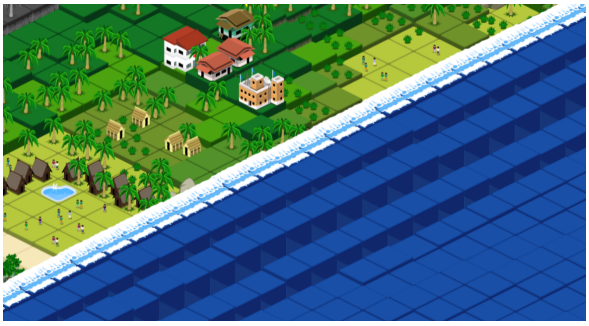

Figura 2.4: Tsunami.

Em relação a jogos mais dedicados aos tsunamis, foram propostos os jogos Earth Girl e Earth Girl 2: Preparing for the Tsunami. Ambos os jogos foram desenvolvidos por cientistas e artistas de jogos do Earth Observatory of Singapore e são destinados a crianças entre 8-15 anos [24]. No jogo Earth Girl, o jogador assume o papel de uma jovem do Sudeste Asiático que precisa salvar sua família e amigos de desastres naturais, como enchentes, vulcões e tsunamis [11]. O jogo é composto por seis níveis e inclui quizzes sobre os desastres naturais. Cada nível apresenta um desafio diferente, onde o objetivo é salvar o máximo de pessoas possível. O jogador também possui poderes especiais para combater os desastres naturais. A abordagem séria do jogo está presente nos quizzes e nos desafios de cada nível. Por exemplo, no nível do tsunami, o jogador deve empurrar as pessoas para longe do mar. A Figura 2.5 demonstra um dos níveis do jogo Earth Girl.

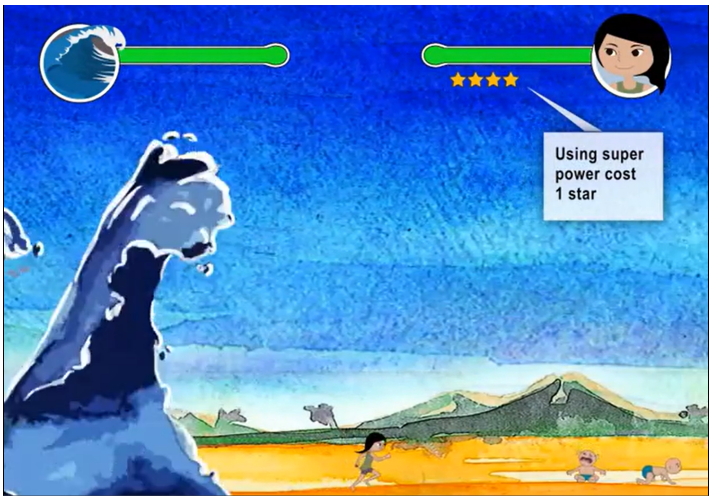

Figura 2.5: Earth Girl a salvar pessoas de um tsunami.

Por outro lado, em Earth Girl 2: Preparing for the Tsunami, o jogador é o protagonista principal do jogo, enquanto a Earth Girl aparece como anfitriã e guia. Este jogo enfatiza a preparação e habilidades de sobrevivência do jogador [12]. Este jogo baseia-se em interagir com as pessoas locais para aprender sobre o perigo dos tsunamis, escolher as ferramentas corretas com base num orçamento, explorar a cidade e tomar decisões estratégicas. O objetivo final de cada nível é salvar o máximo de pessoas possível. A parte séria deste jogo está na gestão de recursos, exploração do local e interação com as pessoas para desenvolver estratégias de evacuação em situações de tsunami. As Figuras 2.6, 2.7 e 2.8 expõem os vários desafios do jogo Earth Girl 2: Preparing for the Tsunami.

Figura 2.6: Interação com a comunidade.

Figura 2.7: Gerenciamento de recursos.

Figura 2.8: Plano de evacuação.

Por fim, apresentamos o jogo Tanah - The Tsunami and Earthquake Fighter. Este jogo é um projeto colaborativo desenvolvido pelo Global Disaster Preparedness Centre, UNESCO-Bangkok e USAID, com o apoio do Asean Coordinating Center for Humanitarian Assistance on Disaster Management (AHA Center) e da Indonesian Red Cross Society [24]. O jogo é destinado a crianças entre 7-12 anos. Em Tanah, o jogador assume o papel de uma personagem chamada Tanah, que está aprendendo a se preparar e se proteger contra terremotos e tsunamis. O jogo é do gênero plataforma e puzzle baseado em níveis. Os primeiros sete níveis focam-se em planos de preparação, como reconhecimento de alarmes de emergência, sinalização, planos de evacuação e preparação de um kit de emergência. Os níveis posteriores mostram como reagir durante um evento de terremoto e tsunami, enquanto os últimos níveis demonstram como se recuperar dos eventos ocorridos [30]. A abordagem séria deste jogo está nos objetivos de cada nível, que ensinam cada etapa de como se preparar, reagir e se recuperar de um terremoto ou tsunami. As Figuras 2.9 e 2.10 ilustram duas fases do jogo Tanah - The Tsunami and Earthquake Fighter.

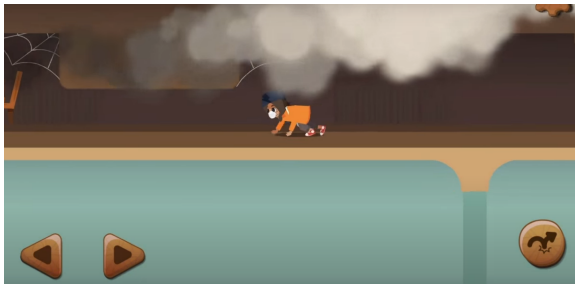

Figura 2.9: Navegação e puzzles.

Figura 2.10: Fuga da cidade.

Ao usar estes trabalhos como base, efetuou-se o planeamento da proposta para um jogo sério que aborda os perigos de um tsunami e ensina como se preparar para esse tipo de evento.

3

Método proposto

Neste capítulo é descrita a proposta deste trabalho e são discutidos os motivos que levaram a criar a mesma. Em seguida, são abordados com mais detalhes os diferentes componentes da proposta.

A proposta deste trabalho consiste em desenvolver um jogo casual para dispositivos móveis que aproveita um dos pontos mais forte dos jogos sérios para uma aprendizagem efetiva: a diversão [14]. Os jogos sérios devem explorar essa característica para manter os jogadores interessados e introduzir gradualmente os componentes a serem ensinados. Isso implica que os jogos sérios devem ser ricos em conteúdo, permitindo múltiplas sessões de jogo em vez de serem jogados apenas uma vez [28]. A ideia é criar um jogo sério que ofereça uma experiência de jogo acessível, interativa e de conclusão facil, traços comuns em jogos casuais [5]. Ao se envolver com o jogo, o jogador terá a oportunidade de realizar várias partidas consecutivas, em cada uma das quais é possível enfrentar falhas graciosamente. O conceito de enfrentar falhas graciosamente é uma das vantagens dos métodos de ensino baseados em jogos, onde o erro, em vez de ser indesejável, é encarado como um passo necessário no processo de aprendizagem. A falta de consequências adversas por falhas nos jogos, ao contrário do que ocorre na vida real, estimula o jogador a explorar e aplicar diferentes estratégias para atingir os objetivos estabelecidos [15]. Portanto, esta proposta foi desenvolvida com o intuito de permitir que, ao longo de várias sessões de jogo, o jogador desenvolva a capacidade de reagir a situações de tsunami e, por consequência, aprimore o seu desempenho no

jogo. Esta abordagem foi escolhida devido às vantagens oferecidas pelos dispositivos móveis para jogos casuais e pelo fato de representar uma estratégia distinta das discutidas na secção 2 sobre trabalhos relacionados.

Esta proposta foi inspirada no jogo "60 Seconds!" (Figura 3.1) e no género de jogos Roguelite. O jogo "60 Seconds!" consiste numa família que precisa coletar o máximo de recursos possível em 60 segundos e se esconder num abrigo à prova de bombas, pois há uma iminente explosão nuclear. Após se esconderem, a família deve sobreviver o máximo de tempo possível até que seja seguro retornar à superfície [1].

Figura 3.1: Jogo "60 Seconds!".

Esta jogabilidade encapsula perfeitamente o que deve ser feito numa situação de tsunami, ou seja, colecionar recursos, procurar abrigo em lugares altos e afastados do mar e sobreviver até que seja seguro retornar à cidade. Devido a essas similaridades, o jogo sério proposto apresenta uma jogabilidade inspirada em "60 Seconds!", com algumas adaptações para se adequar ao perigo de um tsunami, em vez de uma bomba atômica.

Por outro lado, o género Roguelite incorpora elementos de jogabilidade de outro género de jogos como plataforma ou jogos de tiro, adicionando a mecânica de morte permanente, onde o jogador precisa iniciar uma nova partida após morrer [8]. Os jogos Roguelite são conhecidos por dependerem não apenas da habilidade do jogador, mas também de elementos persistentes que o auxiliam [25]. Essas características são benéficas para o jogo proposto, já que um dos objetivos é que o jogador jogue várias partidas sem se aborrecer. Ao introduzir a morte permanente e elementos externos que auxiliam o jogador, como uma loja onde ele pode comprar recursos para ajudar na próxima partida, é possível manter o jogador entretido ao longo das sessões de jogo.

Agora que foi apresentada a proposta para este projeto, o proximo foco é explicar como o componente sério deste jogo é inserido. Como mencionado anteriormente, o utilizador participará de várias iterações do jogo, que podem ser definidas como um ciclo composto por partida, pontos e loja (Figura 3.2).

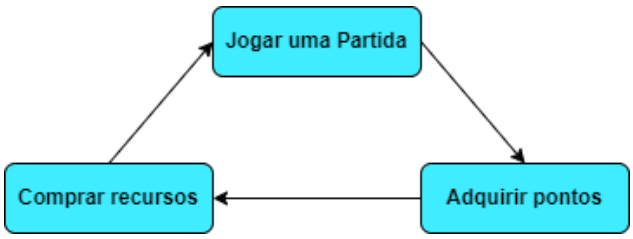

Figura 3.2: Ciclo de evolução do jogador.

A partida engloba o próprio jogo e seus diversos níveis. O jogador deve tentar escapar do tsunami ao progredir pelos vários níveis, cada um oferecendo um desafio único. Estes desafios exploram diferentes habilidades do jogador, cativando sua atenção e, por conseguinte, aprimorando o processo de aprendizagem [6]. Independentemente de conseguir escapar ou morrer, o jogador ganhará pontos. O sistema de pontos mede o desempenho do jogador ao longo dos níveis e atribui uma quantidade específica de pontos com base nas ações do utilizador. Esses pontos são a moeda do jogo e podem ser utilizados na loja. A loja contém recursos que o jogador pode adquirir para melhorar seu desempenho na próxima partida. Esses recursos funcionam como elementos externos presentes em jogos do género Roguelite. Este ciclo é essencial para a aprendizagem do utilizador, pois oferece um ambiente onde ele pode experimentar, por tentativa e erro, o que é útil e o que é desnecessário para sobreviver a um tsunami. Nas primeiras partidas, o jogador pode não saber o que fazer e provavelmente morrerá por causa do tsunami. No entanto, a cada nova partida, o jogador adquire uma compreensão maior do que é necessário para sobreviver. Através do sistema de pontos, o jogador pode avaliar seu desempenho e obter informação sobre o que precisa melhorar e prestar atenção. Por fim, a loja disponibiliza recursos essenciais para sobreviver a um tsunami. Ao comprá-los e testá-los, o jogador percebe a importância de tais recursos. Todos estes componentes juntos permitem que o jogador aprenda a tomar as melhores decisões em situações de tsunami e entenda por que essas decisões são as mais adequadas.

Com a ideia central do ciclo de evolução do jogador explicada, em seguida é explicado de forma mais detalhada cada um dos seus componentes. Começando pela partida, devido ao jogo ser projetado para sessões rápidas, cada partida consiste em três níveis: “Escapar de Casa”, “Escapar da Cidade” e “Sobrevivência”.

No primeiro nível, “Escapar de Casa”, o jogador é apresentado a uma Cutscene que alerta sobre a iminência de um tsunami. Esse aviso pode ser um terremoto ou uma sirene. Após a Cutscene, o jogador tem um tempo limitado para fugir para um local seguro antes que o tsunami o atinja. Ele pode optar por sair da casa a qualquer momento ou investir um pouco mais de tempo para recolher recursos ao redor. Cada recurso é destacado com um marcador de cor, indicando sua acessibilidade: amarelo para fora de alcance, verde para capturável e vermelho para inacessível. As Figuras 3.3, 3.4 e 3.5 ilustram o funcionamento dos marcadores dos objetos que podem ser capturados pelo jogador.

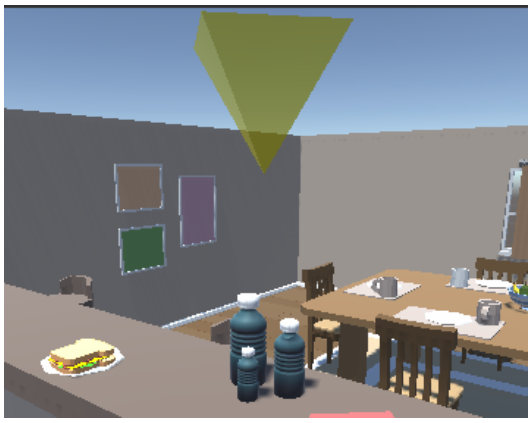

Figura 3.3: Marcador amarelo.

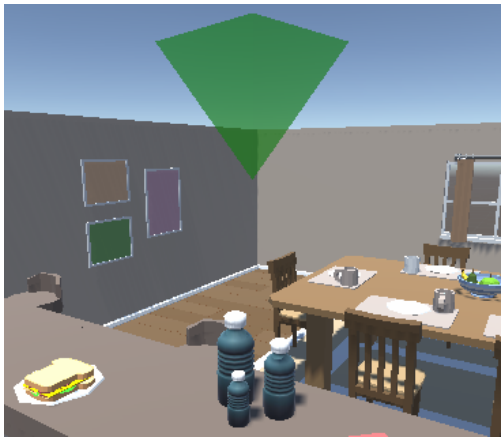

Figura 3.4: Marcador verde.

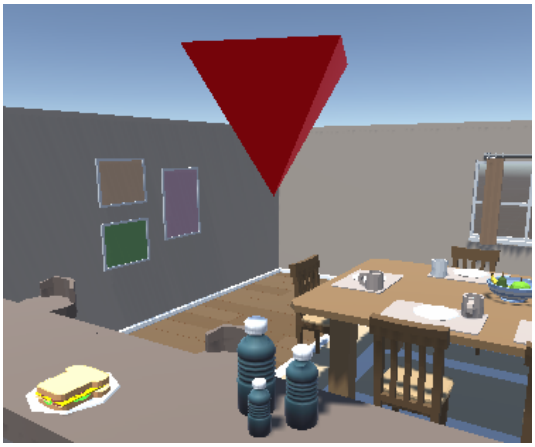

Figura 3.5: Marcador vermelho.

O jogador possui um inventário limitado (Figura 3.6) para armazenar os recursos, sendo que alguns deles ocupam mais espaço do que outros. Portanto, é essencial decidir quais recursos são mais importantes antes de sair de casa.

Figura 3.6: Inventário.

Após escapar de casa, o jogador deve procurar um ponto alto e distante da costa para evitar o tsunami. A cidade pela qual o jogador foge é um labirinto de prédios e casas, com obstáculos que bloqueiam ou atrasam o progresso do mesmo (Figuras 3.7 e 3.8).

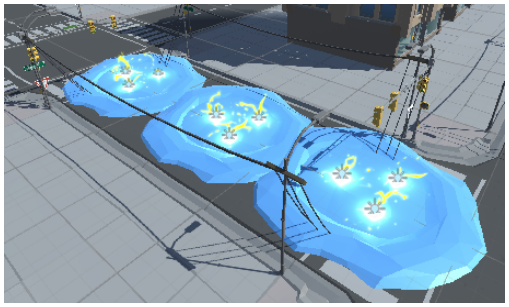

Figura 3.7: Água eletrizada.

Figura 3.8: Engarrafamento.

A cidade é gerada proceduralmente através do algoritmo Wave Function Collapse [35], garantindo que cada partida seja única e evitando a falta de inovação em cada tentativa, o que mantém o jogador interessado mesmo após várias partidas. A Figura 3.9 ilustra 3 cidades geradas proceduralmente.

Figura 3.9: Três cidades geradas proceduralmente.

Além dos obstáculos, a cidade também abriga cidadãos em pânico (Figura 3.10) devido ao perigo do tsunami. Esses cidadãos são Non-player characters (NPCs) que podem ser ajudados pelo jogador. Ao ajudar um NPC, ele passa a navegar pela cidade em busca de uma saída, assim como o jogador. Quando um NPC encontra a saída (Figura 3.11), ele escapa da cidade. No entanto, ajudar um NPC custa tempo, e se o jogador gastar muito tempo ajudando vários NPCs, corre o risco de não conseguir escapar do tsunami a tempo. Portanto, o jogador deve avaliar se tem condições de ajudar outras pessoas sem comprometer sua própria sobrevivência.

Figura 3.10: NPC em panico.

Figura 3.11: NPC fugindo.

Quando o jogador escapa da cidade, o tsunami atinge a área e ele precisa sobreviver com os recursos recolhidos em sua casa. O estágio de sobrevivência funciona como uma sequência de eventos (Figura 3.12), em que o jogador deve sobreviver por vários dias, e cada dia é definido por três eventos fixos e um evento aleatório.

Figura 3.12: Diagrama de sequência de eventos.

Os eventos fixos representam as necessidades básicas do ser humano, como fome, sede e higiene. O evento aleatório é escolhido um do conjunto de eventos, como ferimentos, encontro de sobreviventes, busca por recursos e falta de energia. Cada evento oferece duas opções de interação para o jogador: usar um recurso ou ignorá-lo. O uso de um recurso gasta um dos seus usos, e quando o número de usos de um recurso acaba, ele desaparece. Utilizar um recurso geralmente traz consequências positivas ou neutras. Por outro lado, ignorar um evento tende a trazer consequências negativas ou neutras para o jogador.

O evento de ferimentos indica que o jogador foi ferido e precisa tratar de sua ferida. O evento de encontro com outros sobreviventes significa que há sobreviventes pedindo ajuda ao jogador. O evento de busca por recursos indica que o jogador está em busca de recursos úteis, enquanto o evento da falta de energia, que só ocorre quando o jogador possui um recurso eletrónico, informa que um dos seus dispositivos está ficando sem energia. Cada evento oferece opções de utilizar recursos ou ignorá-los, resultando em consequências diversas. Cabe ao jogador descobrir qual é a melhor ação a tomar em cada situação.

O jogador possui quatro medidores que indicam seu estado geral: fome, sede, higiene e saúde (Figura 3.13). Quando um desses medidores se esgota, o jogador morre. Ignorar os eventos fixos diminui os medidores de fome, sede e higiene, e ignorar o evento de ferimento diminui o medidor de saúde.

Figura 3.13: Medidores de fome, sede, higiene e saúde.

Este ciclo de eventos continua até que o jogador morra ou seja possível retornar à cidade. Independentemente do resultado, a partida termina, e o jogador recebe uma quantidade de pontos de acordo com suas ações no jogo.

Finalizada a componente das partidas, o sistema de pontuação, que serve como a moeda do jogo e é de extrema importância para a evolução do jogador. Cada vez que o jogador completa um nível, ele recebe mil pontos. Ou seja, ao sair de casa, escapar da cidade ou sobreviver até ser seguro voltar para a cidade, o jogador ganha mil pontos. Esta recompensa garante que o jogador não fique preso e seja incapaz de progredir no jogo, mesmo se enfrentar desafios e tomar decisões menos acertadas. Além disso, cada nível avalia certos parâmetros que concedem mais pontos ao jogador quando ele toma boas decisões. No primeiro nível, o jogo avalia o tempo que o jogador demora para sair de casa e os recursos que ele recolheu. No segundo nível, o jogo apenas avalia o tempo que o jogador levou para fugir da cidade. No terceiro nível, o jogo verifica o número de dias que o jogador sobreviveu. Assim, para cada nível a pontuação é calculada da seguinte forma:

1. Sair de casa
   - Tempo decorrido: $(TempoRestante/TempoInicial) \times 1000$
   - Recursos recolhidos: $NrecursosCorretos \times 100$
2. Escapar da cidade
   - Tempo decorrido: $(TempoRestante/TempoInicial) \times 1000$
3. Sobrevivência
   - Dias sobrevividos: $NdiasSobrevividos \times 500$

Esses parâmetros avaliados concentram-se nas decisões que as pessoas devem tomar de forma acertada durante um tsunami. O tempo decorrido evidencia a importância de reagir rapidamente. Os recursos obtidos indicam quais itens são mais importantes numa situação de tsunami, com base em referências [31] e [4]. Por fim, os dias sobrevividos avaliam a capacidade do jogador de gerir seus recursos para sobreviver o máximo de tempo possível. A Figura 3.14 mostra o ecrã apresentado ao utilizador com a pontuação obtida quando termina uma partida.

Figura 3.14: Interface da pontuação.

O último componente que compõe o ciclo de evolução do jogador é a loja. Na loja, o jogador pode gastar os pontos que ganhou para adquirir novos recursos essenciais, que incluem:

- Mochila;
- Pilhas;
- Comida enlatada;
- Kit médico;
- Lanterna;
- Radio;
- Garrafas de água.

Cada um desses recursos é fundamental para sobreviver a um tsunami. A loja oferece suporte ao jogador para enfrentar os desafios e ensina quais recursos devem sempre estar disponíveis em situações de tsunami. A Figura 3.15 ilustra a aparência da loja.

Figura 3.15: Interface da loja.

Em resumo, neste capítulo foi apresentado o método proposto neste trabalho com o objetivo de informar o utilizador como sobreviver numa situação de tsunami, simulada através de um jogo que coloca o jogador num ambiente onde as ações tomadas podem significar a vida ou a morte. Através do ciclo de jogar, ganhar pontos e obter melhorias, o jogador poderá aprender por experiência própria o que fazer numa situação de tsunami.

4

Implementação

Neste capítulo, apresentamos as principais mecânicas, algoritmos e ferramentas implementadas no jogo desenvolvido. O jogo consiste em três níveis, cada um com desafios distintos dos demais. Portanto, cada nível possui suas próprias mecânicas exclusivas. Além disso, foram implementadas ferramentas que são utilizadas em mais de um nível ao longo do jogo. As mecânicas e algoritmos serão discutidos nível por nível, começando com o primeiro, seguido pelo segundo e, por fim, o terceiro nível. Em seguida, apresentaremos as ferramentas que são usadas em mais de um nível. O código para algumas das secções a seguir apresentadas está presente no Apêndice C.

4.1 Primeiro nível

No primeiro nível, o jogador se encontra dentro de uma casa e presencia uma cutscene na qual é informado de que um tsunami está prestes a ocorrer. O objetivo principal deste nível é recolher os recursos espalhados pela casa e escapar antes que o tsunami alcance o jogador. Cada recurso é identificado por um marcador para ajudar o jogador a identificar quais objetos podem ser capturados. O primeiro nível não é muito complexo, sendo as únicas mecânicas exclusivas a cutscene e o marcador de recursos.

4.1.1 Cutscene

No início do jogo, uma cutscene é exibida para indicar que um tsunami está prestes a ocorrer, seja por meio de um terremoto ou do som de uma sirene. A cutscene exibida é selecionada aleatoriamente.

As cutscenes são criadas usando a tecnologia "PlayableDirector" do Unity, que controla certos elementos com base numa Linha Temporal (Timeline Asset). As Linhas Temporais permitem controlar objetos, iniciar eventos, reproduzir áudio e executar outras funções ao longo de um determinado período de tempo, o que possibilita a criação de cutscenes. No entanto, durante uma cutscene, todas as entidades controladas por ela não podem ser movidas por outros componentes até que a cutscene termine. Isso significa que o jogador não consegue controlar seu personagem até que a cutscene seja concluída. Esse aspecto é um inconveniente, pois, no caso da cutscene da sirene, pretende-se que ela seja reproduzida durante a maior parte do primeiro nível.

Para resolver esse problema, foi criada uma Linha Temporal que controla apenas a animação do personagem, juntamente com outras duas Linhas Temporais para o som e outros efeitos das cutscenes do terremoto e da sirene.

As duas Linhas Temporais para os sinais de tsunami controlam áudio e eventos que ativam efeitos especiais e informam quando o nível começa. A Linha Temporal da sirene reproduz o som da sirene e, em seguida, inicia um evento para tocar a música padrão do primeiro nível. Por outro lado, a Linha Temporal do terremoto controla o som do terremoto e inicia um evento para mover a câmera e criar um efeito de tremor. O movimento da câmera é realizado pelo *script* "CameraShaker", que a movimenta com base no ruído de Perlin. O código para este *script* pode ser verificado no Apêndice C.

Devido ao fato de a câmera possuir o componente "CinemachineVirtualCamera", ela sempre seguirá o jogador e não permitirá qualquer tentativa de alterar sua posição. No entanto, o componente "CinemachineVirtualCamera" aceita a introdução de ruído de Perlin para movimentar a câmera ao redor da mesma posição. Dessa forma, é possível movimentar a câmera e adicionar um efeito de movimento aleatório para conferir mais realismo.

A escolha da cutscene a ser apresentada no primeiro nível é determinada pelo *script* "TimeLineRandomizer", que seleciona aleatoriamente a Linha Temporal a ser usada. Este *script* pode ser verificado no Apêndice C.

Em resumo, no primeiro nível, uma de duas cutscenes que sinalizam o início de um tsunami pode ser exibida. Cada cutscene possui sua própria Linha Temporal com áudio e eventos que adicionam efeitos à cutscene.

4.1.2 Marcador de recursos

No primeiro nível, o jogador possui o objetivo de recolher os utensílios mais importantes para sobreviver a um tsunami. Para facilitar esta tarefa, foi construído o *script* "ItemPointer". O código para este *script* pode ser verificado no Apêndice C.

O *script* "ItemPointer" é colocado nos objetos que podem ser capturados pelo utilizador. Este *script* guarda o CollectibleObject associado ao objeto e confere atributos que o utilizador pode manipular para criar um marcador ou indicativo (Figura 4.1) de que o objeto em questão pode ser capturado pelo utilizador.

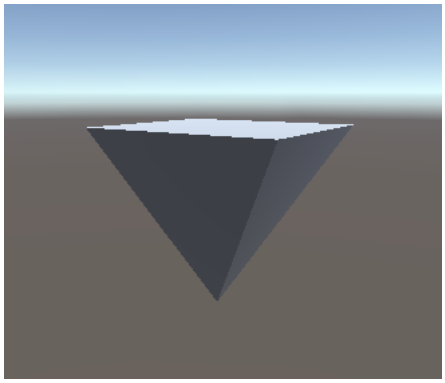

Figura 4.1: Marcador.

Por este meio, o *script* "ItemPointer" facilita a tarefa de capturar os CollectibleObjects e fornece informação visual sobre o estado do objeto.

4.2 Segundo nível

O segundo nível envolve a fuga da cidade em direção a um local elevado e distante do mar. O jogador é inserido numa cidade gerada proceduralmente, o que significa que a composição da cidade é diferente a cada vez que uma partida é iniciada. Dentro da cidade, há obstáculos distribuídos de forma a criar um labirinto que o jogador deve atravessar para escapar. Além disso, há NPCs espalhados pela cidade, em estado de choque, e é responsabilidade do jogador decidir se tem tempo suficiente para ajudá-los. Este segundo nível é o mais complexo de todos, principalmente devido ao desafio que toma uma forma diferente a cada partida, tornando-o único. É um componente crucial do jogo, pois apresenta o maior número de algoritmos e mecânicas para evitar a monotonia. Além disso, é neste nível que o tsunami pode ser avistado. Os principais elementos deste nível incluem o tsunami, a cidade, os obstáculos e os NPCs.

4.2.1 Tsunami

O tsunami é uma peça fundamental neste trabalho, por consequente foi desenvolvido um *script* capaz de representar o mar e a transformação do mesmo num tsunami.

O tsunami é construído a partir de vários vértices distribuídos num quadrado de uma dimensão específica, sendo 1 vértice o menor número possível. A partir destes vértices são criados Tiles constituídos por 4 vértices adjacentes, dentro destes Tiles são formados dois triângulos (Figura 4.2).

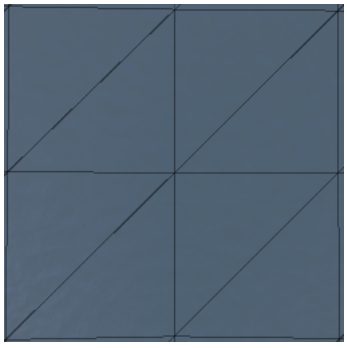

Figura 4.2: Triângulos formados dentro dos Tiles.

A forma como estes triângulos são construídos indica o sentido para onde a textura será demonstrada. Se os vértices estão conectados no sentido dos ponteiros do relógio, a textura será apresentada na parte superior dos Tiles. Por outro lado, se estão conectados no sentido contrário aos ponteiros do relógio, a textura aparecerá na parte inferior dos Tiles. Neste caso são construídos triângulos para que o tsunami seja visível na parte de superior e na parte de inferior. De seguida, é definido o valor dos mapeamentos UVs. Os UVs mapeiam as coordenadas da textura que serão aplicadas no tsunami, aumentando ou diminuindo a resolução da mesma (Figuras 4.3 e 4.4).

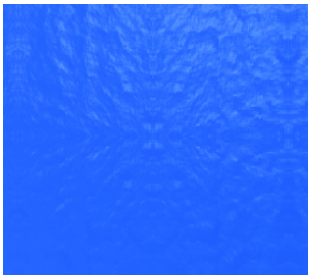

Figura 4.3: Alta resolução.

Figura 4.4: Baixa resolução.

No final da construção, são executados os métodos "RecalculateBounds" e "RecalculateNormals" para redefinir os limites do tsunami e os vetores normais do mesmo. Estes métodos são necessários sempre que os vértices de um objeto são modificados, de forma a refletir as mudanças efetuadas.

A textura foi criada utilizando o Universal Render Pipeline (URP), uma ferramenta desenvolvida pelo Unity para proporcionar uma forma fácil e eficiente de criar gráficos otimizados para uma ampla gama de plataformas [34]. Com o URP, foram aplicadas uma série de transformações e algoritmos para tornar a textura mais semelhante ao mar (Figura 4.5).

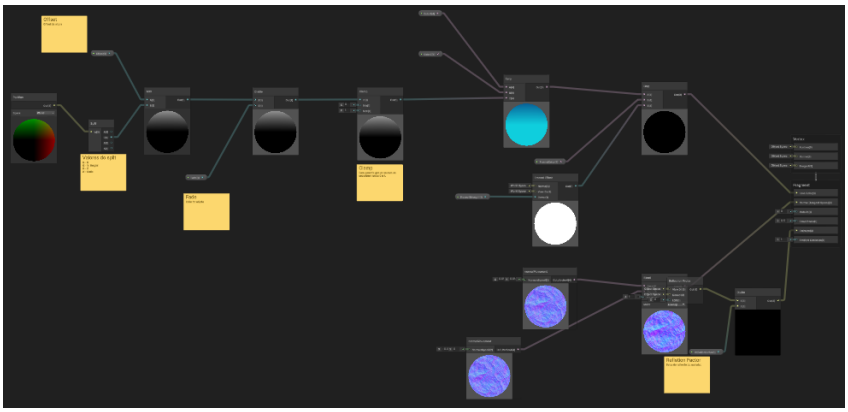

Figura 4.5: URP.

O primeiro passo para tornar a textura mais semelhante ao mar foi a atribuição de cores. Na Figura 4.6, busca-se a posição no eixo y da textura para torná-la dependente da altura. Em seguida, é aplicada uma operação de adição para dividir o material numa parte mais escura e outra mais clara e uma operação de divisão para criar uma transição suave entre as duas partes. Estas operações proporcionam o efeito de a água estar mais clara quando está mais próxima da superfície e mais escura quando está mais profundamente submersa. Ao aplicar as cores, é possível ver o efeito desejado.

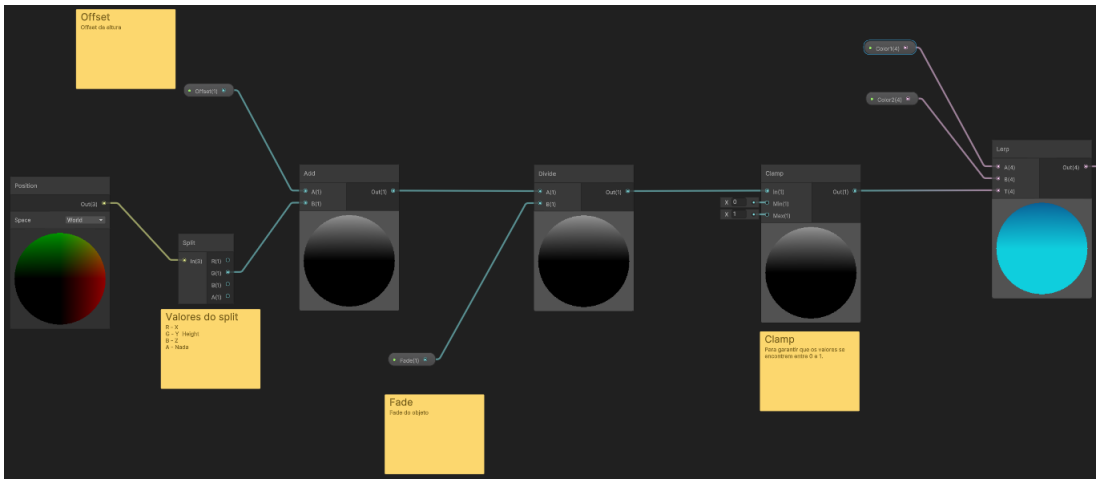

Figura 4.6: Divisão entre a parte mais escura e clara do mar.

O próximo passo foi a aplicação do efeito Fresnel. O efeito Fresnel é um fenômeno em que, com base no ângulo de incidência, ou seja, no ângulo entre a linha de visão e a superfície do objeto que está sendo observado, o fator de reflexão muda [16] (Figura 4.7). O efeito Fresnel foi implementado para conferir um aspecto realista à textura do mar (Figura 4.8).

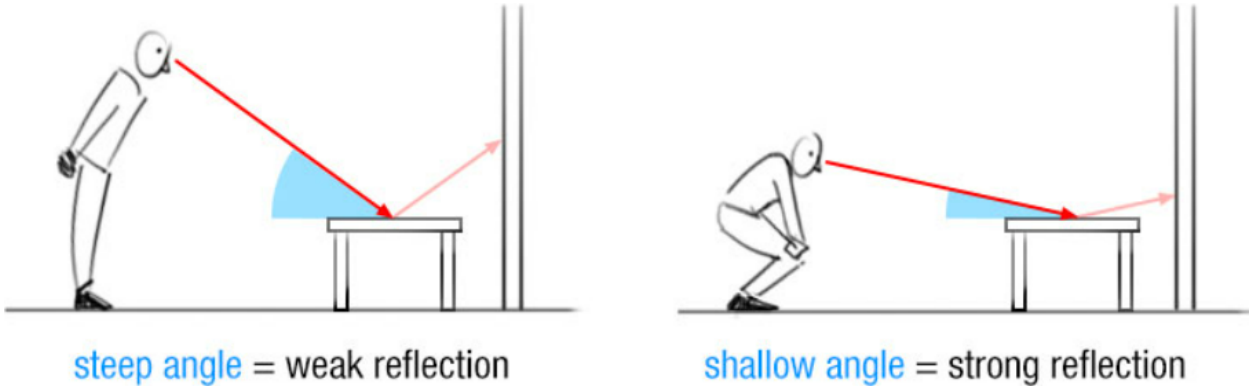

Figura 4.7: Fator de reflexão consoante o ângulo entre a linha de visão e uma superficie.

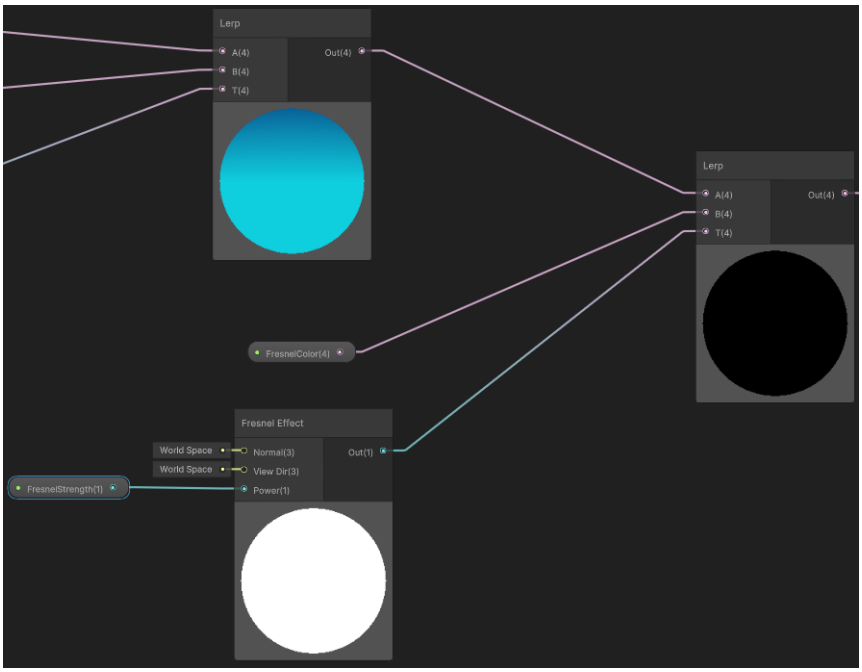

Figura 4.8: Aplicação do efeito Fresnel.

Por outro lado, foi utilizado um mapa de normais para fazer com que a luz refletisse na textura como se fosse água. No entanto, além da aplicação do mapa de normais, também foi aplicado movimento a ele. Ao aplicar movimento ao mapa de normais, consegue-se dar à textura do mar a sensação de movimento. Neste caso, foram aplicados dois movimentos distintos para alcançar um efeito mais realista e menos uniforme na movimentação da água (Figura 4.9).

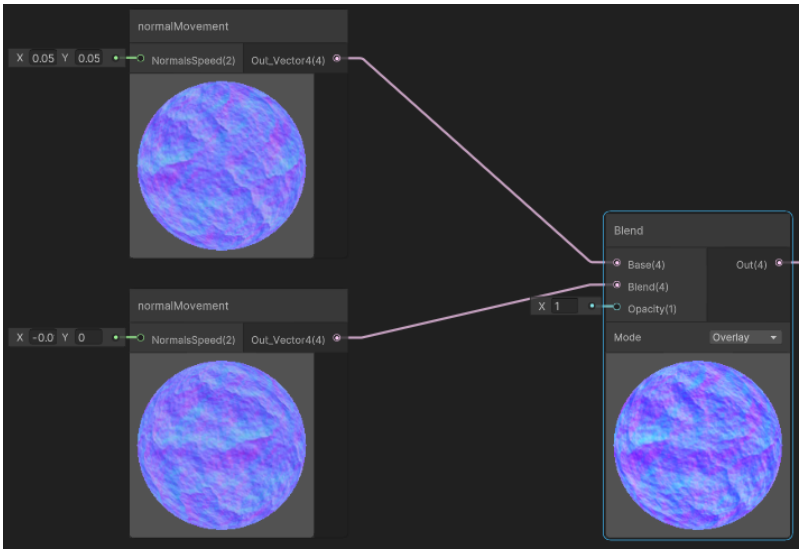

Figura 4.9: Aplicação de movimento ao mapa de normais.

Por fim, foi implementado um efeito de reflexão na textura para que ela possa refletir objetos ao seu redor como a água (Figura 4.10).

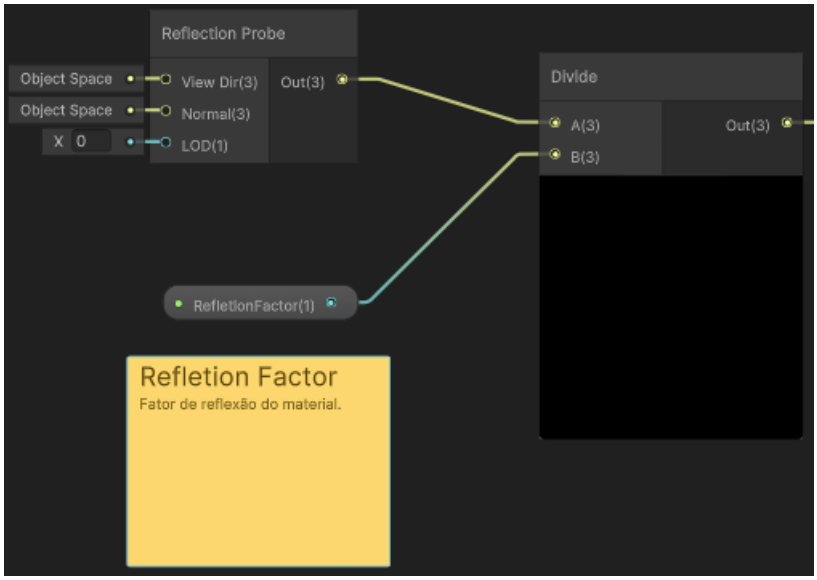

Figura 4.10: Aplicação do efeito de reflexão.

Com o tsunami construído, foi implementado o efeito de onda sobre o mesmo. O efeito de onda foi formado utilizando o Perlin Noise. O Perlin Noise introduz um efeito aleatório que as ondas do mar possuem, para além de permitir tornar as ondas mais suaves ou ruidosas ao alterar a variação dos valores introduzidos. Em conjunto com o Perlin Noise, também foi criado um *struct object* chamado Octave que guarda vários atributos para a manipulação do Perlin Noise, simplificando a manipulação das ondas do tsunami.

O Octave guarda os valores de velocidade, escala, altura e de se a onda é alternada. Cada um destes valores manipula os seguintes aspetos:

- **Velocidade** - velocidade com que a onda se mexe;
- **Escala** - comprimento de onda do tsunami, com valores mais pequenos apresentando um maior comprimento de onda e valores mais altos um menor comprimentos de onda;
- **Altura** - altura do tsunami;
- **Onda alternada** - define como a onda é calculada com o Perlin Noise.

Se a onda for alternada a altura de cada vértice é modificado consoante o Perlin Noise e uma função de cosseno para transmitir uma impressão de que água não está a se mover num sentido em concreto, por outro lado se a onda não for alternada ela segue uma direção determinada pela sua velocidade (Figuras 4.11 e 4.12).

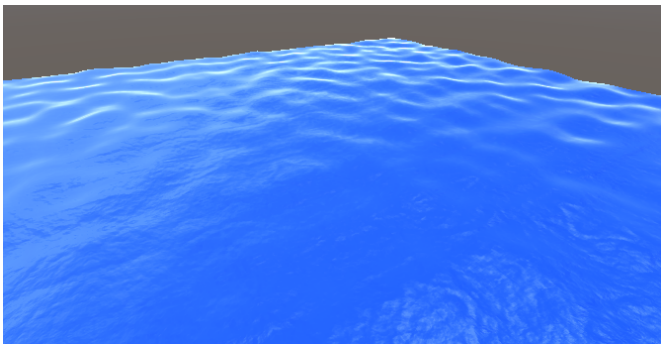

Figura 4.11: Onda não alternada.

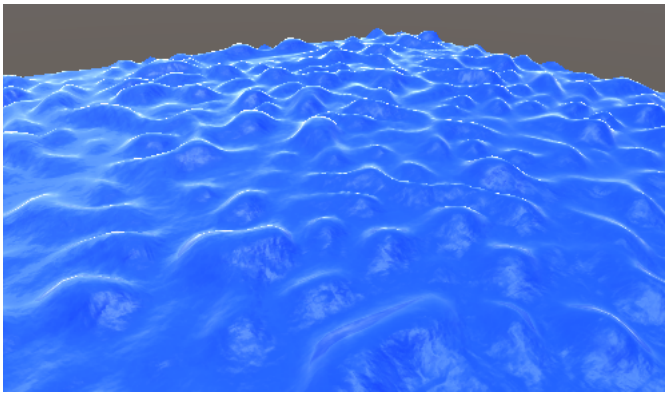

Figura 4.12: Onda alternada.

A partir da utilização do Octave é possível criar duas instâncias diferentes de valores para dois estados diferentes, uma para quando o mar está calmo e outra para o tsunami, ou criar múltiplas instâncias para serem usadas no mesmo estado, concedendo um efeito mais aleatório e realista à onda.

Por fim, foram implementados vários controlos na transição do mar calmo para o tsunami, como por exemplo uma altura mínima e uma suavização na transição. Devido à natureza aleatória do Perlin Noise, a transição do mar calmo para o tsunami pode não ser natural devido à diferença de valores. Porém com base na característica do Perlin Noise de devolver o mesmo valor para o mesmo conjunto de coordenadas, é possível ajustar os controlos implementados para suavizar a transição (Figuras 4.13 e 4.14).

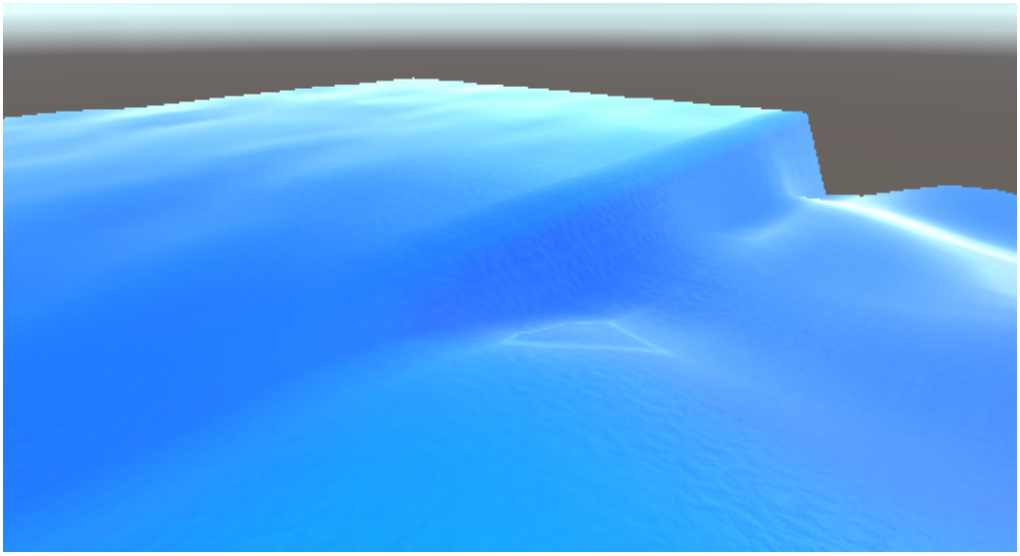

Figura 4.13: Onda não suavizada.

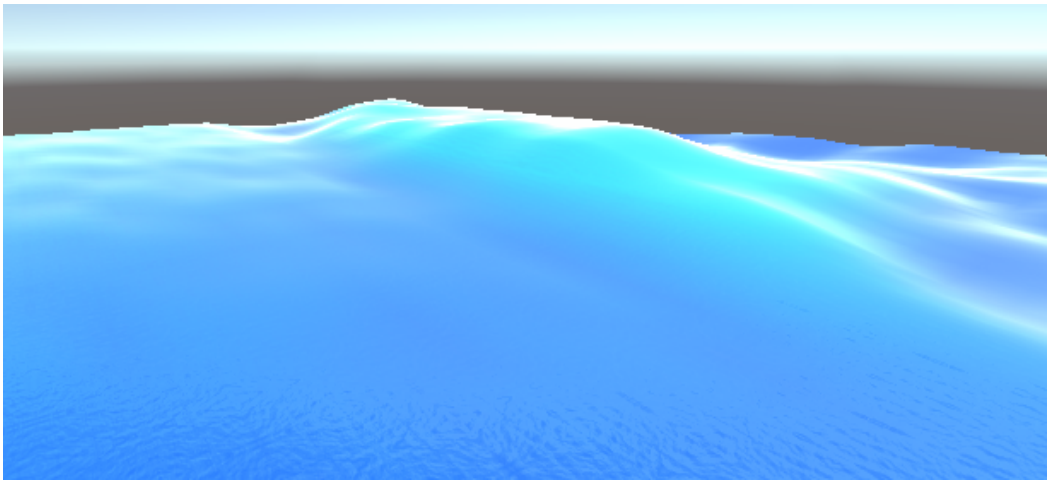

Figura 4.14: Onda suavizada.

4.2.2 ML-Agent

O ML-Agent é uma ferramenta de código aberto que possibilita a utilização de ambientes Unity para o treino de agentes inteligentes por meio de métodos de reforço e imitação. No contexto deste projeto, esta ferramenta foi utilizada para desenvolver NPCs que buscam escapar de um tsunami, com o propósito de infundir realismo ao jogo, um aspecto fundamental para a eficácia do aprendizado num jogo sério [14].

Antes de treinar o agente, é necessário determinar o que ele observa no seu ambiente. Para tornar o agente realista, ele deve agir de maneira semelhante ao jogador, com base nas mesmas observações. O jogador consegue ver os obstáculos ao seu redor e para onde deve se dirigir para escapar do tsunami. Portanto, o agente também precisa ter uma noção do espaço ao seu redor e de para onde deve fugir.

O agente obtém informações sobre o ambiente usando o componente "RayPerceptionSensor3D". Esse componente cria raios ao redor do agente que são capazes de detectar o ambiente (Figura 4.15). Cada raio fornece informações, como se atingiu algum objeto, a distância até o objeto atingido e se esse objeto possui uma determinada *tag*. Dessa forma, é possível fornecer ao agente uma ideia do que está ao seu redor. Além disso, o agente recebe uma observação de um vetor de direção. Esse vetor indica a direção que o agente deve seguir para alcançar seu objetivo. O vetor é normalizado para evitar problemas durante o processo de aprendizagem, impedindo que números grandes nas observações tenham mais importância do que deveriam.

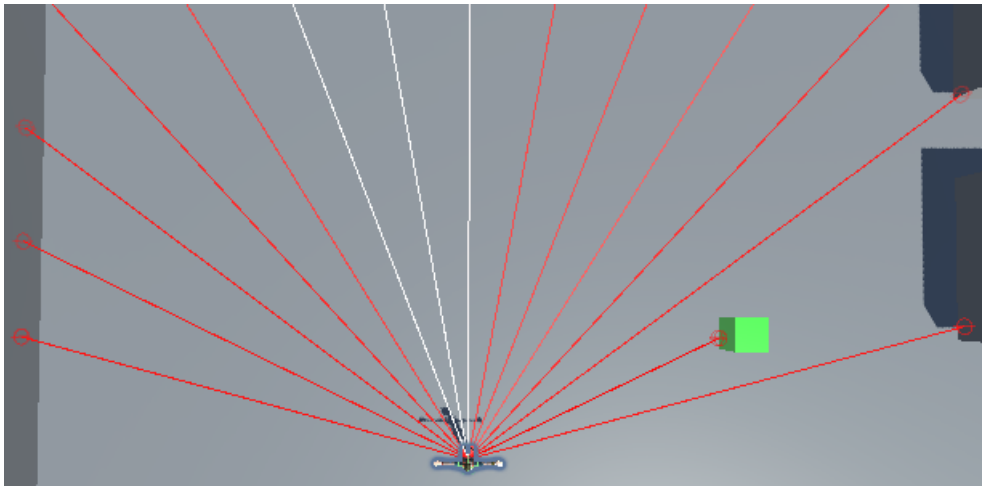

Figura 4.15: Raios do "RayPerceptionSensor3D".

Após a conclusão das observações, é necessário configurar as ações que o agente pode realizar. O agente deve se mover da mesma forma que o jogador. Nesta primeira iteração, foi utilizado o mesmo código que o *script* "ThirdPersonController". Este *script* serve para que o utilizador consiga controlar o seu personagem no jogo e servirá de base para desenvolver um controlador para o agente. Foram efetuadas duas modificações no código. Primeiro, foi alterado o mesmo para receber os valores do agente. O utilizador pode movimentar e girar o personagem usando o botão de direção (Figura 4.16), que por sua vez regista a posição em x e y para onde foi movido para aplicar o movimento ao personagem.

Figura 4.16: Botão de direção.

O agente reproduzirá esses valores de x e y para aplicá-los ao código de movimento do personagem. A segunda modificação foi tornar a rotação dependente apenas do movimento do personagem, pois o jogador pode ajustar a câmera, afetando a forma como a rotação é realizada durante o movimento. Como o agente não precisa ajustar a câmera, essa função foi removida do código de movimento do agente.

O último passo antes de começar a treinar o agente é definir o ambiente e o sistema de recompensas. O ambiente é uma cidade com edifícios e obstáculos que o agente deve evitar. Tanto os edifícios quanto os obstáculos possuem a *tag* "Obstacle", para que o agente saiba o que deve evitar. O agente e o objetivo são colocados aleatoriamente no ambiente, sendo que o objetivo possui a *tag* "Objective", indicando para onde o agente se deve direcionar. Quando o agente colide com um obstáculo, o ambiente volta ao estado inicial e o agente recebe uma recompensa negativa. No entanto, se o agente colide com o objetivo, o ambiente volta ao estado inicial e o agente recebe uma recompensa positiva. Adicionalmente, foi introduzida uma pequena recompensa negativa a cada movimento do agente para motivar o mesmo a não ficar parado no ambiente.

Agora que o ambiente está preparado, começou-se a efetuar os testes. Nas primeiras iterações os resultados não foram muito positivos, com a média das recompensas dos episódios situando-se abaixo de zero (Figuras 4.17 e 4.18).

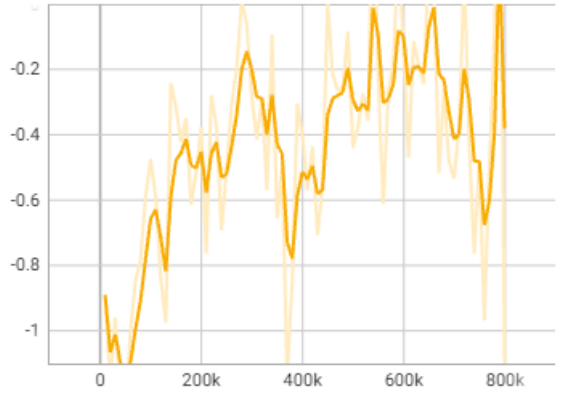

Figura 4.17: Gráfico das recompensas acumuladas em função do número de episódios.

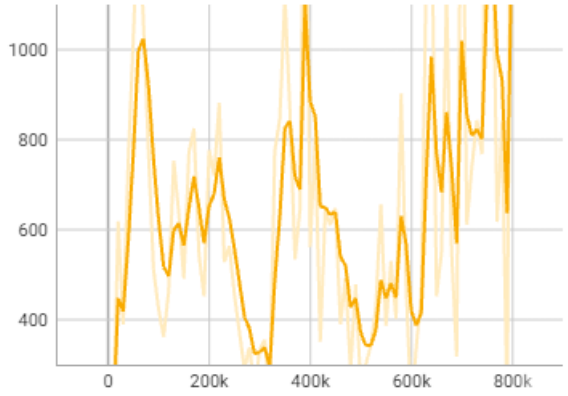

Figura 4.18: Gráfico do tempo total por episódio em função do número de episódios.

No final desses testes, o agente apresentava movimentos tremidos e demonstrava medo ao se aproximar de obstáculos e espaços estreitos, sendo capaz de alcançar o objetivo apenas quando este estava próximo. Nos testes subsequentes, foram feitas modificações nas configurações de aprendizagem do agente para verificar se um agente mais simples ou mais complexo se sairia melhor no ambiente criado. No entanto, as alterações realizadas não mudaram o resultado da aprendizagem, indicando que o problema estava no agente ou no ambiente em si.

Nas Figuras 4.19 e 4.20 são apresentados os resultados obtidos com o agente mais simples e nas Figuras 4.21 e 4.22 com o agente mais complexo.

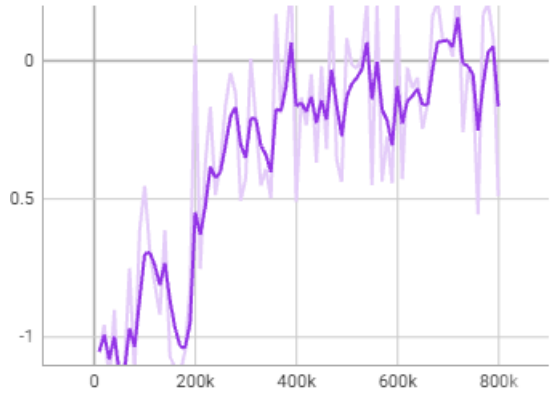

Figura 4.19: Gráfico das recompensas acumuladas em função do número de episódios.

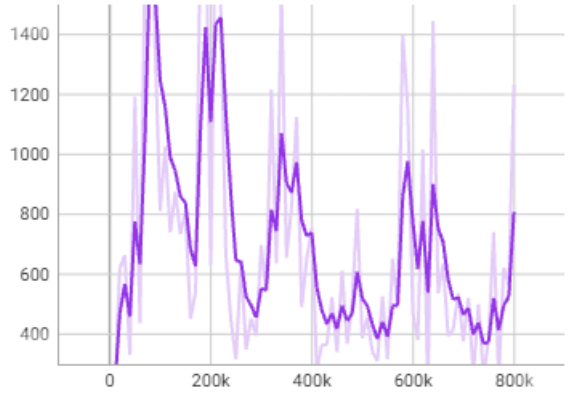

Figura 4.20: Gráfico do tempo total por episódio em função do número de episódios.

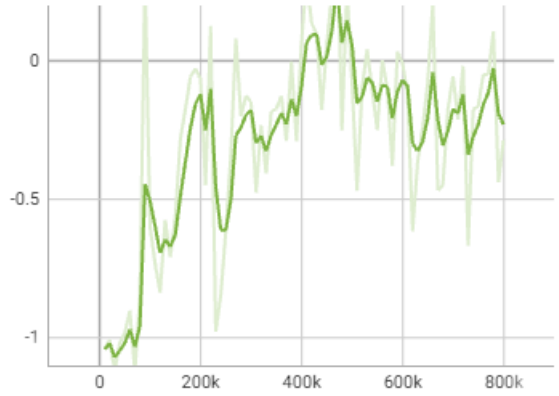

Figura 4.21: Gráfico das recompensas acumuladas em função do número de episódios.

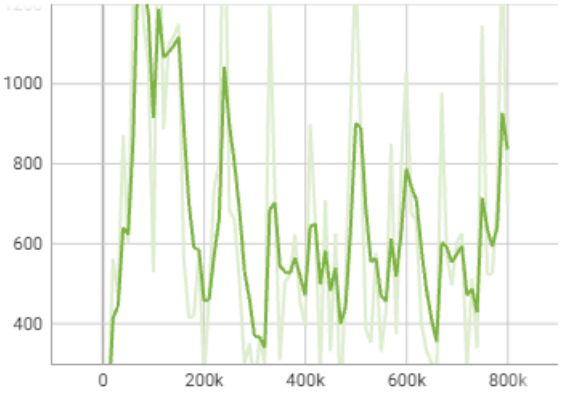

Figura 4.22: Gráfico do tempo total por episódio em função do número de episódios.

Para resolver esse problema, foi criado um ambiente mais simples, composto por polígonos e contendo menos obstáculos e caminhos menos estreitos. Ao treinar o agente nesse ambiente, foram obtidos resultados bastante positivos, com a média dos episódios se aproximando do valor máximo e o tempo necessário para completá-los diminuindo consistentemente (Figuras 4.23 e 4.24).

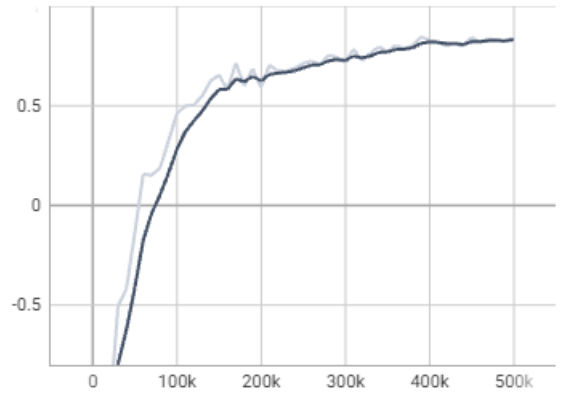

Figura 4.23: Gráfico das recompensas acumuladas em função do número de episódios.

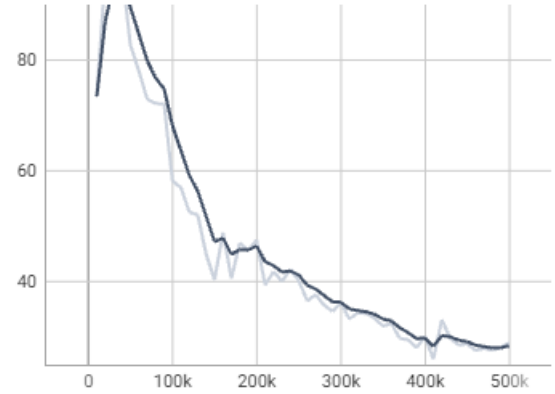

Figura 4.24: Gráfico do tempo total por episódio em função do número de episódios.

O agente treinado foi colocado na cidade e conseguiu navegar nela sem muitos problemas, evitando obstáculos e percorrendo espaços estreitos com facilidade. Esse teste destacou a importância de treinar os agentes em ambientes mais simples antes de colocá-los em ambientes mais complexos.

No entanto, embora o agente tenha conseguido alcançar seu objetivo, seus movimentos não eram realistas. Durante a busca pelo objetivo, o agente girava em torno de si mesmo e, ao se aproximar de um obstáculo, ele avançava e recuava múltiplas vezes antes de contorná-lo. Esta falta de realismo mostrou que o código de movimento do agente não era adequado. Portanto, foi implementada uma nova forma para o agente controlar seu movimento. O código de movimento anterior foi removido, exceto pela parte que controlava as animações do personagem. No novo modelo construído, o agente controla a rotação do personagem e a intensidade do movimento para frente. No modelo anterior, a rotação era uma consequência dos valores x e y introduzidos pelo agente, mas neste novo modelo o agente controla diretamente a rotação do personagem para conferir mais estabilidade. Além disso, neste novo código, o agente só pode se mover para frente e controlar a velocidade do movimento, evitando movimentos de avançar e recuar ao se aproximar de obstáculos.

Após estas alterações, o agente foi treinado novamente no ambiente simples. As recompensas médias permaneceram as mesmas, mas o tempo necessário para completar os episódios aumentou (Figuras 4.25 e 4.26).

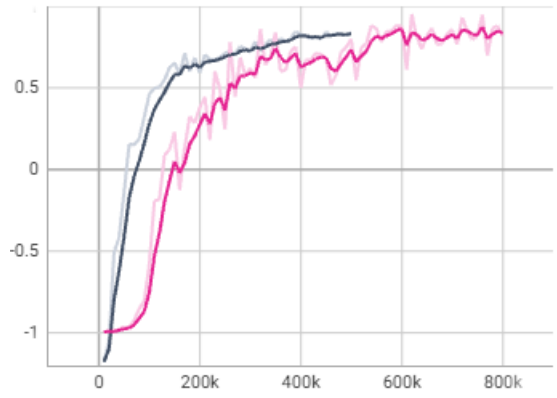

Figura 4.25: Comparação das recompensas acumuladas entre o novo código (púrpura) e o código original (preto).

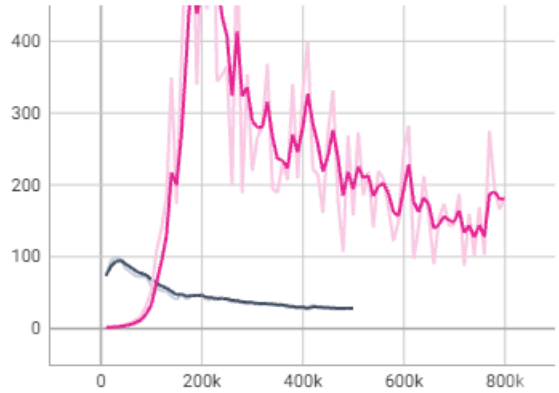

Figura 4.26: Comparação do tamanho total dos episódios entre o novo código (púrpura) e o código original (preto).

O custo do aumento dos episódios foi considerado aceitável, pois o agente ainda era capaz de navegar na cidade de forma mais realista do que nos exemplos anteriores.

Concluindo, a partir da tecnologia ML-Agents e dos testes efetuados, foi possível construir um agente capaz de navegar na cidade e escapar do tsunami de forma realista.

4.2.3 NpcObstacleController

No segundo nível, os NPCs estão em estado de choque e confusos, sem saber o que fazer. O jogador tem a opção de dedicar um pouco do seu tempo para fugir do tsunami para ajudar um NPC, e ao fazer isso, o NPC começa a agir por conta própria para tentar escapar do tsunami. A transferência de estado do NPC é realizada pelo *script* "NpcObstacleController", que reduz o tempo para fugir do tsunami e ativa a autonomia do NPC. O código para este *script* pode ser verificado no Apêndice C.

4.2.4 Construção da cidade

A cidade do segundo nível é gerada proceduralmente, o que significa que cada iteração do jogo apresenta uma cidade única. O algoritmo utilizado para realizar essa geração é o "Wave Function Collapse" (Colapso da Função de Onda).

O nome desse algoritmo deriva de um efeito na mecânica quântica, no qual um sistema está em superposição, ou seja, em diferentes estados ao mesmo tempo, até que um efeito externo afete o sistema e o coloque num único estado. O algoritmo "Wave Function Collapse" utiliza o mesmo princípio para resolver problemas. Um exemplo da aplicação desse algoritmo é o jogo Sudoku (Figura 4.27), no qual o jogador deve preencher cada célula com um número de 1 a 9, sem repetições em linhas, colunas ou blocos 3x3.

			6					3
8					5			
			4			5	2	
				7	2			
	7	6			4			
5	4	2	3			8		
	3	8	1	4			9	5
7				3				
	2			6	8	3		7

Figura 4.27: Estado inicial do Sudoku.

O algoritmo "Wave Function Collapse" começa a resolver esse problema ao verificar inicialmente todos os estados possíveis para cada célula. Dessa forma, todas as células entram num superestado, onde possuem mais de um estado possível. A partir desse ponto, o algoritmo seleciona uma célula com base num critério específico. Esse critério consiste em escolher a célula com o menor número de estados possíveis, ou seja, aquela com menor entropia. Essa seleção da célula de menor entropia ajuda a reduzir a possibilidade de o algoritmo gerar uma solução impossível. Ao selecionar a célula, o "Wave Function Collapse" escolhe um dos estados possíveis para ser o estado final da célula e, em seguida, todas as células restantes são atualizadas de acordo com as regras do Sudoku (Figura 4.28).

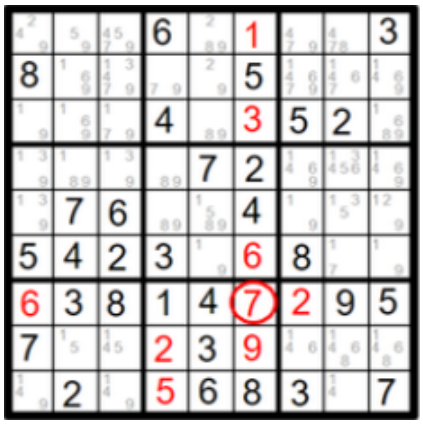

Figura 4.28: Seleção da célula e atualização das células restantes.

Esse processo é repetido várias vezes até que todas as células estejam preenchidas e o problema seja resolvido (Figura 4.29).

Figura 4.29: Segunda seleção e atualização.

Após a descrição do funcionamento do "Wave Function Collapse", vai ser explicado as regras de construção da cidade nas quais o algoritmo é aplicado. A cidade é composta por Tiles (Figura 4.30). Esses Tiles são quadrados que representam partes da cidade, como edifícios, estradas ou casas. Cada lado de um Tile possui um tipo pertencente a uma das seis categorias:

- Calçada;
- Estrada;
- Edifício;
- Edifício-estrada;
- Porto;
- Qualquer tipo.

A regra básica para a construção da cidade é que dois Tiles adjacentes devem ter lados do mesmo tipo, ou pelo o menos um dos lados deve ser de qualquer tipo. No entanto, cada Tile possui certas restrições únicas que são aplicadas a essas regras. Por exemplo, os Tiles que representam edifícios não podem ter outros tipos de Tiles que não sejam estradas em sua parte frontal (Figura 4.31). Isso evita a ocorrência de casos em que os edifícios tenham suas partes frontais voltadas umas para as outras. A partir dessas regras, o "Wave Function Collapse" é capaz de gerar cidades de forma procedural.

Figura 4.30: Um Tile.

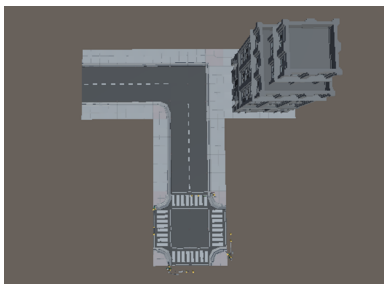

Figura 4.31: Ligações entre Tiles.

Por outro lado, embora a aplicação da escolha da menor entropia reduza a probabilidade de erro, é possível que sejam geradas combinações impossíveis, resultando num mapa com buracos onde não houve Tiles que se encaixaram com os Tiles adjacentes (Figura 4.32).

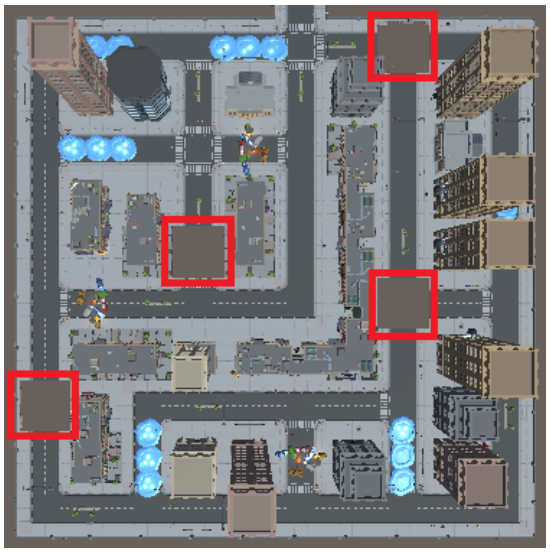

Figura 4.32: Cidade com buracos.

Portanto, sempre que o algoritmo atualiza as células, é verificado se existe uma combinação impossível com a atual configuração. Se houver, toda a cidade é refeita. Embora esta verificação resolva o problema das combinações impossíveis, pode tornar a criação da cidade bastante lenta, uma vez que a mesma é reconstruída sempre que ocorre uma combinação impossível. Para resolver esse problema, foi utilizada a técnica de não propagação. Esta técnica consiste em guardar o estado da cidade antes de selecionar um Tile para a célula corrente. Se o Tile selecionado provocar uma combinação impossível ao atualizar os outros Tiles, a cidade retorna ao estado antes da célula ter o seu estado selecionado e o Tile selecionado anteriormente é descartado. Este método acelera bastante o tempo de criação da cidade, com a única fraqueza sendo que é apenas guardado o estado antes da decisão. Se a combinação impossível envolver múltiplas células já selecionadas, então a cidade é refeita.

A cidade deve funcionar como um labirinto para o jogador, com obstáculos que bloqueiam certos caminhos. A criação do labirinto foi feita utilizando o algoritmo "Recursive Backtracker" [20]. Este algoritmo seleciona uma célula aleatoriamente e começa a visitar as células vizinhas, ignorando células já visitadas. Quando o algoritmo chega a um beco sem saída, retorna às células que já visitou para ver se existe alguma célula vizinha que ainda não tenha sido visitada. O algoritmo continua esse processo até que todas as células tenham sido visitadas. A Figura 4.33 ilustra o algoritmo "Recursive Backtracker" [20].

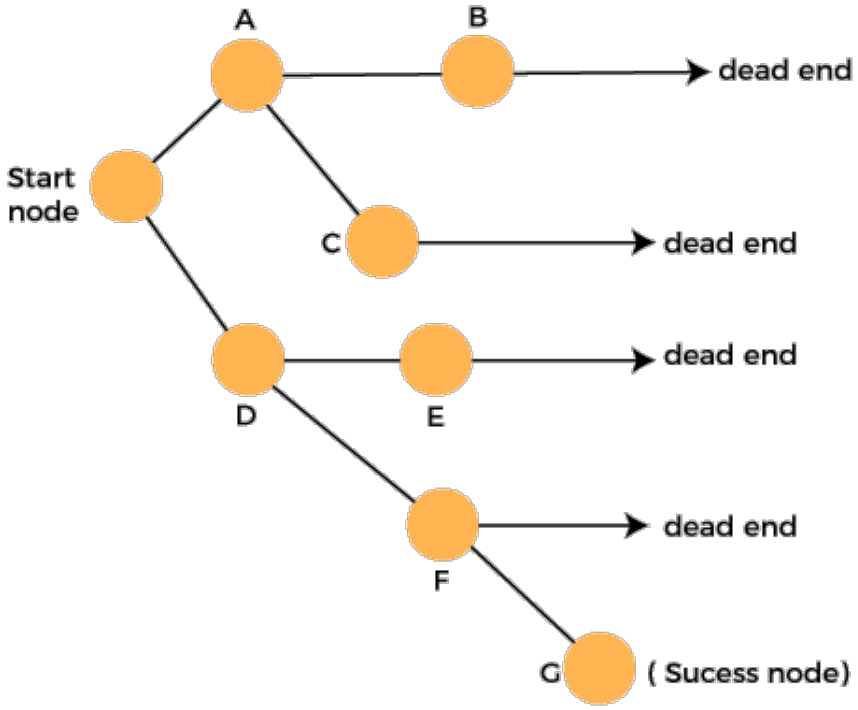

Figura 4.33: Funcionamento do algoritmo "Recursive Backtracker".

Na construção da cidade, o "Recursive Backtracker" [20] é aplicado nos Tiles que constituem a estrada da cidade, já que o jogador vai navegar por esses Tiles e os obstáculos devem se localizar nos mesmos para impedir o progresso do jogador. Durante o "Recursive Backtracker" [20], sempre que o algoritmo encontra uma célula que já visitou, um obstáculo é colocado na célula corrente, deixando o restante das células livres. Dessa forma, é possível colocar obstáculos na cidade sem correr o risco de bloquear o jogador e garantindo que sempre exista um caminho livre entre a entrada e a saída.

O último passo na construção da cidade é a colocação dos NPCs. Os NPCs navegam pela cidade na mesma forma que o jogador, procurando a saída e evitando obstáculos. Por consequente, os NPCs são colocados dentro da área da cidade e fora de qualquer edificio, obstáculo ou outro NPC.

4.3 Terceiro nível

O terceiro nível possui como objetivo sobreviver com os recursos recolhidos no primeiro nível até que seja seguro retornar à cidade. Este nível difere dos dois níveis anteriores, pois, em vez de controlar diretamente o personagem, o jogador é confrontado com uma série de eventos nos quais deve tomar decisões estratégicas para garantir sua sobrevivência. Embora tenha uma abordagem diferente em comparação aos níveis anteriores, este nível é possui apenas um componente que gera um sistema de eventos projetado para apresentar desafios e situações ao jogador.

4.3.1 Gestor de eventos

O terceiro nível consiste numa sequência de eventos em que o jogador tem a opção de usar um recurso capturado no primeiro nível ou ignorar o evento. O objetivo principal desse nível é gerir os recursos disponíveis da melhor forma possível até que seja seguro retornar à cidade.

Os eventos do terceiro nível implementam a interface "IEvent", que define as funções básicas que cada evento deve ter. Essa interface fornece funções para:

- Texto do evento;
- Recursos permitidos para uso no evento;
- Resultado ao usar um determinado recurso;
- Texto do resultado;
- Resultado ao ignorar o evento;
- Texto resultante de ignorar o evento.

Com essas funções, é possível construir eventos que possuam sua própria lógica para testar o jogador, sem que outros componentes precisem ter conhecimento dessa lógica.

O controle dos eventos é realizado pelo *script* "EventManager". Esse *script* mantém o registro de todos os eventos possíveis no terceiro nível e também registra o estado do jogador para saber se ele está vivo ou não. O terceiro nível segue um ciclo específico de eventos, começando com eventos fixos e depois um evento aleatório. Os eventos fixos representam as necessidades humanas, como alimentação, hidratação e higiene. Por outro lado, os eventos aleatórios ocorrem apenas uma vez em cada ciclo e apresentam um desafio diferente a cada iteração. Este ciclo está representado na Figura 3.12

O "EventManager" cria esse ciclo por meio de uma máquina de estados, com um estado para cada evento fixo e um único estado para os eventos aleatórios. Quando o jogador usa um recurso ou ignora um evento, o "EventManager" verifica o estado atual e aplica a função apropriada fornecida pela interface "IEvent" para determinar se o jogador concluiu com sucesso o evento. O jogador tem sucesso num evento quando aplica o recurso correto para aquele evento específico. O sucesso num evento fixo implica que o jogador cuidou de si mesmo e não está mais próximo da morte, enquanto o sucesso num evento aleatório pode resultar em recompensas positivas, como obter um novo recurso, negativas, como perder um recurso, ou não ter nenhuma consequência. No entanto, o jogador pode falhar num evento ao usar um recurso inadequado ou ignorar o evento. A falha num evento fixo significa que o jogador não cuidou de si mesmo e está mais próximo da morte, enquanto a falha num evento aleatório pode ter consequências, como a perda de um recurso. Ignorar um evento nem sempre significa que o jogador falhou nesse evento, pois em algumas situações ignorar pode ser a melhor solução. Dessa forma, o "EventManager" consegue reproduzir o ciclo de eventos do terceiro nível.

Em resumo, no terceiro nível, o jogador passará por vários ciclos de eventos que avaliam sua habilidade em gerir recursos. Os ciclos são criados pelo *script* "EventManager" e cada evento implementa as funções básicas da interface "IEvent".

4.4 UI

4.4.1 UI Toolkit

O UI Toolkit do Unity é uma ferramenta para criar interfaces utilizador em jogos e outras aplicações. Este Toolkit oferece vantagens como a facilidade de uso, flexibilidade, personalização, otimização e integração com outras ferramentas do Unity.

O UI Toolkit é projetado para ser intuitivo e fácil de aprender, mesmo para utilizadores iniciantes. Isso significa que os desenvolvedores podem criar interfaces utilizador complexas sem precisar de um conhecimento profundo em programação. Da mesma forma, oferece uma grande variedade de componentes prontos para uso, como botões, campos de texto, barras de progresso e muito mais. Além disso, o UI Toolkit também permite que os desenvolvedores personalizem a aparência dos componentes, incluindo cores, fontes e tamanhos. De forma semelhante, também oferece uma ampla variedade de opções de animação, o que pode tornar a interface utilizador mais atraente e dinâmica.

Outra característica do UI Toolkit é a sua otimização para desempenho e a integração com as outras ferramentas do Unity. Esta ferramenta foi projetada para ser rápida e eficiente, o que significa que as interfaces utilizador criadas com esta ferramenta carregam rapidamente e são responsivas. Para além disto, a sua integração permite criar interfaces utilizador que interajam com outros elementos do jogo, como sistemas de pontuação, menus de opções e muito mais. Isto torna o desenvolvimento mais fácil e eficiente.

Em resumo, o UI Toolkit do Unity é uma ferramenta para a criação de interfaces utilizador em jogos e outras aplicações. Com sua facilidade de uso, flexibilidade, personalização, otimização para desempenho e integração com outras ferramentas do Unity, oferece uma ampla variedade de vantagens para os desenvolvedores que desejam criar interfaces utilizador de qualidade.

4.4.2 Interface das UI

As interfaces utilizador do jogo são controladas por *scripts* que implementam a interface IUIControler. Esta interface contém os métodos “Activate” e “Deactivate” que controlam o que é visível e interagivel quando uma UI está ativa ou desativa respetivamente. Adicionalmente, ela também confere o evento “onPause” com a intenção ser utilizado numa situação de pausa, sair da pausa (se a User Interface (UI) for o menu de pause) ou aceder a opções do menu de pausa (como o menu principal), se este evento não for conveniente ele pode ser ignorado. O código para este *script* pode ser verificado no Apêndice C.

4.5 Ferramentas

4.5.1 CollectibleObjectScritableObject e ItensManagerScriptableObject

Os "CollectibleObjectScritableObject"e "ItensManagerScriptableObject"são Scriptable Objects, ou seja, são contentores de dados capazes de armazenar uma grande quantidade de informações independentemente das instâncias de classe. Estes Scriptable Objects são usados para armazenar os atributos e métodos de todos os objetos recolhíveis neste jogo e manipulá-los. Os *scripts* para "CollectibleObjectScritableObject"e "ItensManagerScriptableObject"podem ser encontrados no Apêndice C.

Em situações em que há risco de tsunami, é importante estar preparado antes da evacuação. Pessoas que vivem em áreas sem risco de tsunami ou onde esses eventos são raros podem não saber quais recursos são necessários para se preparar adequadamente. Para ajudar o jogador a entender quais itens são mais importantes e por quê, foram incluídos vários itens com diferentes níveis de utilidade neste jogo. Estes recursos são usados em várias partes do jogo, maioritariamente no primeiro nível e no terceiro nível, ou seja eles são referenciados em vários *scripts* ao longo do código. O "CollectibleObjectScritableObject"foi desenvolvido para facilitar a manipulação destes recursos e otimizar seu uso. O "CollectibleObjectScritableObject"armazena os seguintes atributos de um recurso:

- Ícone
- Durabilidade máxima do item
- Durabilidade atual do item
- Espaço ocupado no inventário
- Preço na loja
- Se o item já foi comprado
- Descrição
- Função do item
- Tipo do item

O "CollectibleObjectScritableObject"é um Scriptable Object, pois uma das grandes vantagens dos mesmos é que eles se comportam como *assets*, ou seja, as informações para cada recurso já estão carregadas antes do início do jogo. Por funcionarem como *assets*, as informações nos Scriptable Objects sobrevivem às transições de níveis sem a necessidade de instanciação constante. Essa característica melhora o desempenho do jogo e reduz as dependências entre componentes. Componentes que precisam acessar um recurso específico só precisam acessar o Scriptable Object correspondente, sem consumir memória ou depender de outro *script* para instanciar o objeto.

Um Scriptable Object é intencionado a ser somente de leitura, porem os seus valores podem ser alterados durante o tempo de jogo e voltam ao normal quando o mesmo termina. Devido a estas características, foi criado o *script* "ItensManagerScriptableObject"para criar um Scriptable Object capaz de armazenar e remover os recursos que o jogador possui consigo a dado momento do jogo. Desta forma, o "ItensManagerScriptableObject"funciona como o sistema de inventário do jogador. Este Scriptable Object é usado por todos os *scripts* responsáveis por gerenciar o inventário do jogador e o mesmo fornece métodos para realizar essas ações.

Em resumo, existem vários recursos que o utilizador deve testar para determinar quais são os mais úteis numa situação de tsunami. Cada recurso é um Scriptable Object, "CollectibleObjectScritableObject,"contendo informações sobre o item. O inventário do jogador também é um Scriptable Object, "ItensManagerScriptableObject,"que mantém o controle dos itens que o jogador possui a qualquer momento.

4.5.2 CollectItens

No primeiro e segundo nível o jogador consegue interagir com o ambiente à sua volta, mais especificamente ele consegue capturar recursos e ajudar NPCs respetivamente. O *script* responsável por esta interação é o “CollectItens”. O código para este *script* pode ser verificado no Apêndice C.

No primeiro nível, o objetivo é capturar recursos encontrados na casa. O *script* “CollectItens” permite detetar objetos colecionáveis dentro do alcance do jogador por meio de colisões com esses objetos. Quando uma colisão com um objeto colecionável é detetada, o “CollectItens” utiliza o *script* “ItemPointer” para fornecer informações visuais sobre os recursos disponíveis para capturar e os adiciona a uma lista de recursos dentro do alcance. Ao capturar um recurso, o primeiro item da lista dos recursos dentro

do alcance é selecionado. O "CollectItens" utiliza o "ItensManagerScriptableObject" para gerir os recursos capturados pelo jogador e manter a sincronia entre os elementos presentes no jogo. As informações visuais podem ser vistas nas Figuras 3.3, 3.4 e 3.5.

No segundo nível, o jogador pode ajudar NPCs, seguindo um esquema semelhante ao de capturar recursos. No entanto, apenas é guardada a informação sobre a pessoa mais próxima, em vez de uma lista. Quando o jogador ajuda uma pessoa, o "CollectItens" utiliza o "NpcObstacleController" para transmitir o comando de salvamento ao NPC.

Em resumo, o "CollectItens" permite ao jogador capturar recursos no primeiro nível e salvar NPCs no segundo nível.

4.5.3 Timer

O primeiro e o segundo nível apresentam um temporizador que indica o tempo que o jogador tem até o tsunami atingir a cidade. Os dois níveis partilham o mesmo tempo, ou seja quanto mais tempo o utilizador gasta no primeiro nível, menos tempo ele possui no segundo nível.

A consistência do tempo entre os níveis é garantida pelo *script* estático "Timer". Por ser estático, esse *script* não precisa ser instanciado, mantendo-se constante durante o jogo e podendo ser acessado por qualquer outro script. O "Timer" permite atualizar o tempo com base no tempo decorrido, adicionar ou subtrair tempo, obter o tempo atual em formato de texto e a percentagem do tempo restante, sendo que o tempo total é considerado cem por cento. O código para este *script* pode ser verificado no Apêndice C.

Em resumo, o uso do *script* estático "Timer" permite manter a mesma informação de tempo ao longo dos níveis e oferece funções para manipular essa informação.

4.5.4 Game Manager

O componente "Game Manager" controla e fornece funções para estabelecer a transição entre Scenes. Este componente não é destruído durante a mudança de Scenes e é acessível a qualquer componente. O código para este *script* pode ser verificado no Apêndice C.

O jogo sério desenvolvido possui quatro Scenes principais: o menu principal, primeiro nível, segundo nível e terceiro nível. A transição de uma Scene para outra pode ser um processo simples, como a transição do menu principal para o primeiro nível, onde é apenas esperado que o primeiro nível seja carregado. Por outro lado, a transição do primeiro nível para o segundo nível precisa aguardar que o segundo nível seja carregado e que a construção da cidade esteja completa. O componente "Game Manager" fornece funções para executar tanto transições simples como mais complexas, onde a Scene corrente pode definir quando está carregada.

Por fim, o componente "Game Manager" também gere as interfaces de carregamento e pontuação, fazendo a interface de carregamento aparecer sempre que uma transição de Scene ocorre e apagando os valores da interface de pontuação quando uma partida acaba.

5

Resultados

Esta secção concentra-se no processo de obtenção e discussão dos resultados. A primeira parte é dedicada à metodologia utilizada para a recolha de dados, enquanto a segunda parte se concentra na avaliação do jogo através dos resultados obtidos.

5.1 Metodologia

A avaliação da jogabilidade (usabilidade e experiência do utilizador) do jogo foi realizada através de testes com utilizadores. Para recolher o feedback dos utilizadores foi utilizado um questionário/guião que conduzia os testes. Estes testes foram realizados por 10 utilizadores. Os utilizadores foram introduzidos ao questionário após receberem a aplicação desenvolvida neste projeto. Caso não dispusessem de um smartphone adequado para a execução do jogo, um dispositivo apropriado foi fornecido para garantir a participação completa na avaliação. Os participantes responderam individualmente ao questionário, sendo informados de que o propósito era avaliar a eficácia do jogo sério desenvolvido para informar sobre os perigos dos tsunamis.

O questionário é dividido em seis secções: "Caracterização do Utilizador", "Usabilidade", "Elementos de Jogo", "Experiência do Utilizador", "Experiência do Utilizador após o Jogo" e "Componente Didática". A secção "Caracterização do Utilizador" coleta informações físicas e nível de conhecimento dos utilizadores. A secção "Usabilidade" avalia a usabilidade do jogo. A secção "Elementos de Jogo" captura as opiniões

dos utilizadores sobre mecânicas específicas presentes no jogo. As secções "Experiência do Utilizador" e "Experiência do Utilizador após o Jogo" investigam a experiência que o JS porpocionou aos jogadores como jogo. Por fim, a secção "Componente Didática" avalia o conhecimento adquirido pelos participantes sobre o tema. O questionário utilizado pode ser consultado no anexo A.

Por meio deste questionário, é possível recolher dados de vários utilizadores diferentes enquanto se avalia a aplicação desenvolvida e sua efetividade. Dessa forma, informações valiosas serão obtidas para aprimorar o jogo e a experiência do utilizador relacionada à conscientização do perigo dos tsunamis.

5.2 Caracterização do Utilizador

A primeira parte é a "Caracterização do Utilizador". Esta secção recolhe informações sobre o utilizador, assim como seu conhecimento atual sobre como reagir a tsunamis. Isto possibilita uma comparação entre o que o utilizador já sabia e o que ele aprendeu.

As primeiras perguntas são focadas nas características do utilizador, como idade, género e se ele vive perto de uma área costeira. O jogo sério desenvolvido pretende ser adequado para qualquer grupo de pessoas, e obter estas informações ajuda a entender como o jogo é percebido pelos diferentes grupos.Assim, foram realizados testes a 10 utilizadores, 50% de cada género e distribuído por várias escalas etárias (Figuras 5.1 e 5.2).

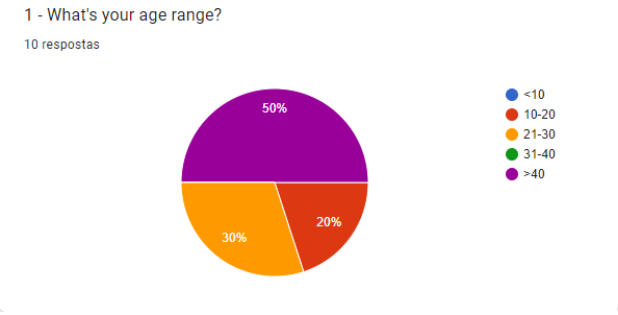

Figura 5.1: Idade dos respondentes.

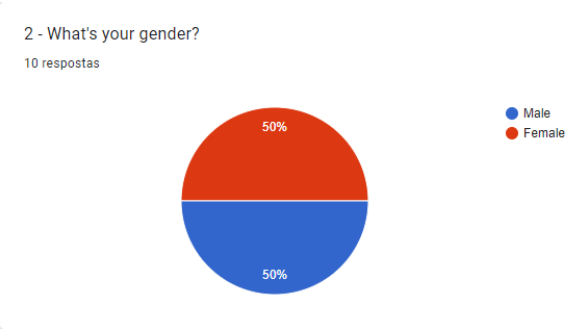

Figura 5.2: Género dos respondentes.

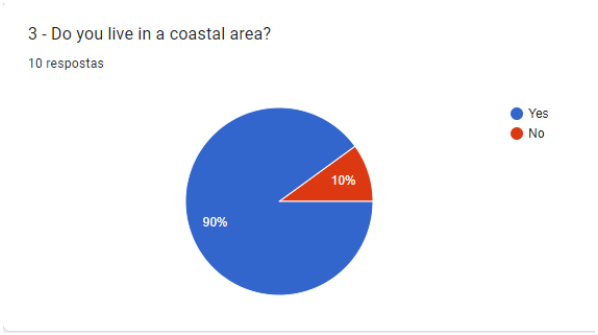

Figura 5.3: Respondentes que vivem numa área costeira.

Perguntas subsequentes avaliam o conhecimento do utilizador sobre como reagir numa situação de tsunami. Especificamente, as perguntas destacam o conhecimento sobre o que deve ser preparado numa situação de tsunami, os sinais naturais de um tsunami e o que é mais importante em tal situação.

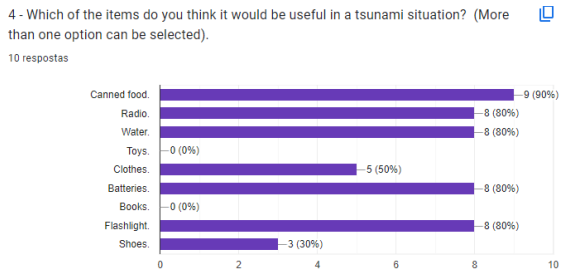

Figura 5.4: Recursos úteis numa situação de tsunami.

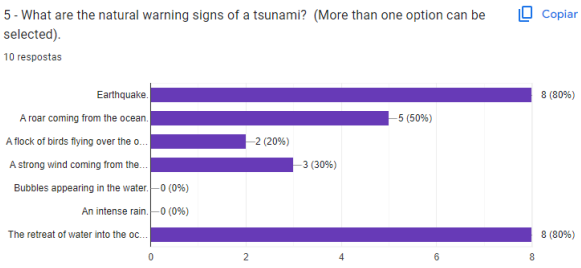

Figura 5.5: Os Sinais naturais de um tsunami.

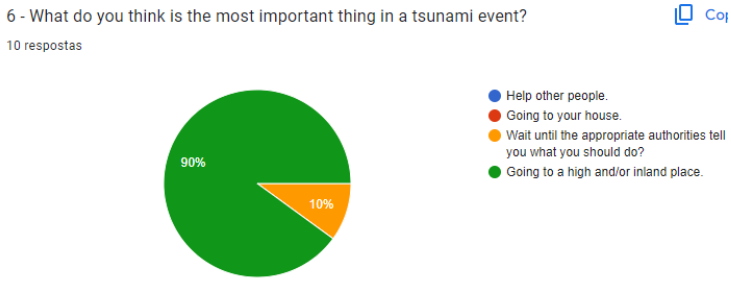

Figura 5.6: A ação mais importante numa situação de tsunami.

Como pode ser observado nas Figuras 5.4, 5.5 e 5.6, os respondentes do questionário mostram ter conhecimento sobre como reagir a um tsunami. No que diz respeito ao preparo para uma situação de tsunami, a maioria escolheu os recursos mais importantes, como comida enlatada, rádio, água, pilhas e lanterna. Da mesma forma, a pergunta sobre sinais naturais mostra que os respondentes sabem associar terramotos

e o recuo do mar a tsunamis, embora haja um pouco de desconhecimento em relação a sinais menos conhecidos, como um barulho alto vindo do mar e bolhas aparecendo na água. Por fim, a pergunta sobre o que é mais importante numa situação de tsunami demonstra que a maioria dos respondentes compreende as prioridades. Esse nível de conhecimento pode ser resultado do facto de que, conforme mostrado na Figura 5.3, quase todos os respondentes vivem em áreas costeiras.

Apesar do nível de conhecimento sobre o perigo de tsunamis, existem algumas lacunas no entendimento dos respondentes. Portanto, será possível avaliar a efetividade do JS desenvolvido para abordar essas lacunas.

5.3 Usabilidade

A secção "Usabilidade" emprega o System Usability Scale (SUS) [29] para avaliar a usabilidade da aplicação. O SUS é uma ferramenta para medir a usabilidade de um sistema, consistindo em dez perguntas com respostas numa escala do tipo Likert [19] com 5 opções entre "Discordo muito" e "Concordo muito", permitindo que o utilizador ofereça uma visão subjetiva da usabilidade do sistema. O SUS é geralmente utilizado após o utilizador ter tido a oportunidade de usar o sistema, mas antes de qualquer reflexão sobre o sistema em si. Em outras palavras, os utilizadores devem registrar sua resposta imediata, em vez de pensar na pergunta por muito tempo [29].

O sistema de pontuação do SUS funciona da seguinte forma: para cada pergunta, é associada uma pontuação de 1 a 5, dependendo se o utilizador respondeu "Discordo muito" ou "Concordo muito", respectivamente. Em seguida, os seguintes passos são executados:

- Calcular X = Soma dos pontos das perguntas impares - 5;
- Calcular Y = 25 - Soma dos pontos das perguntas pares;
- Calcular a Pontuação = $(X + Y) \times 2.5$.

O raciocínio por trás dessa fórmula tem como objetivo colocar a pontuação máxima a 100. Assim, a pontuação máxima do SUS é 100, com cada pergunta correspondendo a 10 pontos. As perguntas ímpares possuem uma conotação positiva e pretendem conferir uma pontuação máxima de 10 se o utilizador responder "Concordo muito" ou uma pontuação mínima de 0 se o utilizador responder "Discordo muito". Para garantir isso, um ponto é subtraído do valor de cada pergunta ímpar, de modo que a pontuação fique entre 0 e 4, o que resulta na pontuação mínima de 0. Posteriormente, quando multiplicado por 2.5, a pontuação máxima por pergunta se torna 10 [21].

Por outro lado, as perguntas pares possuem uma conotação negativa. Se a resposta fôr "Concordo muito", a pontuação é mínima, e se a resposta fôr "Discordo muito", a pontuação é máxima. Para atingir esse efeito, 5 é subtraído do valor de cada pergunta par. Dessa forma, se a resposta for "Concordo muito", a pontuação é 0, e se a resposta for "Discordo muito", a pontuação é 4. No final, multiplicar o valor de cada pergunta por 2.5 assegura que todas tenham uma pontuação máxima de 10 [21].

Compreendendo como a pontuação do SUS funciona, segue-se a análise das respostas (Tabela 5.1) dadas pelos utilizadores ao SUS durante os testes de usabilidade.

		Perguntas										
		1	2	3	4	5	6	7	8	9	10	Total
Participantes	1	3	3	3	2	2	3	3	1	3	3	55
	2	3	3	3	3	5	1	3	2	3	3	62.5
	3	5	1	3	3	4	1	5	2	5	4	70
	4	4	1	5	2	5	1	5	1	4	2	90
	5	3	4	4	5	3	3	3	4	2	3	40
	6	3	3	2	2	2	2	2	2	3	3	62.5
	7	4	4	4	4	4	4	4	3	3	5	47.5
	8	5	1	5	1	5	1	5	1	5	1	100
	9	3	1	2	1	5	1	5	1	4	1	85
	10	5	1	4	1	5	1	5	1	4	2	92.5
Média		3.8	2.2	3.5	2.4	4	1.8	4	1.8	3.6	2.7	70.5

Tabela 5.1: Tabela do SUS.

Embora a pontuação do SUS varie entre 0 e 100, ela não possui o mesmo significado que uma percentagem [21]. A tabela a seguir indica como interpretar a pontuação do SUS.

Pontuação	Nota	Avaliação
> 80.3	A	Excelente
68 - 80.3	B	Bom
68	C	Okay
51 – 68	D	Mau
< 51	F	Horrível

Tabela 5.2: Tabela da pontuação do SUS.

A pontuação média para o JS desenvolvido é de 70.5 pontos. De acordo com a tabela 5.2, o jogo não apresenta problemas de usabilidade relevantes. No entanto, o desvio padrão dos resultados é bastante alto ($\sigma = 18.5$)), indicando uma variação significativa nas opiniões dos utilizadores. Ao analisar a Tabela 5.1, é evidente que várias pontuações atribuídas pelos respondentes classificam a aplicação como "Mau" ou até mesmo "Terrível". Por outro lado, também existem utilizadores que classificam o JS como "Excelente". Essa ampla gama de resultados contribui para a classificação "Boa" da aplicação e para o alto desvio padrão.

A variação nos valores pode ser resultado do facto de que o JS foi projetado para que os utilizadores aprendam por si mesmo por meio de sua experiência de jogo, tornando o processo de aprendizagem e todo o jogo dependentes da habilidade do utilizador em jogos. Portanto, utilizadores que não estão familiarizados com jogos ou não sabem como lidar com eles podem encontrar dificuldades em usar a aplicação. Isto coloca um dilema, pois uma usabilidade deficiente pode comprometer outros elementos do jogo, como o processo de aprendizagem [2]. A compreensão dos respondentes em relação aos elementos do jogo é explorada na próxima secção.

5.4 Elementos do jogo

Os "Elementos do Jogo" analisam a dificuldade do utilizador em compreender certos aspectos do jogo. O jogo foi desenvolvido com a ideia de que o utilizador esteja sempre experimentando o que pode fazer nele, mas esperar que o jogador aprenda tudo por conta própria pode não ser muito vantajoso para um JS. Esta secção visa descobrir se permitir que o utilizador aprenda as mecânicas do jogo por conta própria é eficaz ou não.

As perguntas são formuladas no formato "Eu não tive dificuldade..." com o objetivo de formar as respostas no mesmo estilo do SUS, variando numa escala de cinco opções entre "Discordo muito" e "Concordo muito". Em todas as perguntas, a opção "Concordo muito" é a mais positiva, enquanto "Discordo muito" é a mais negativa.

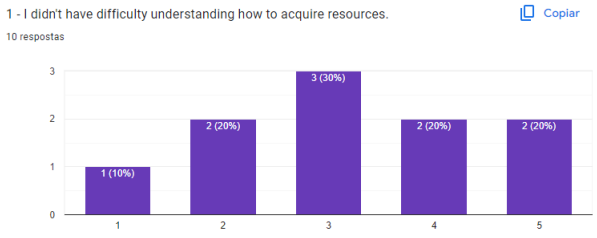

Figura 5.7: Recolher recursos.

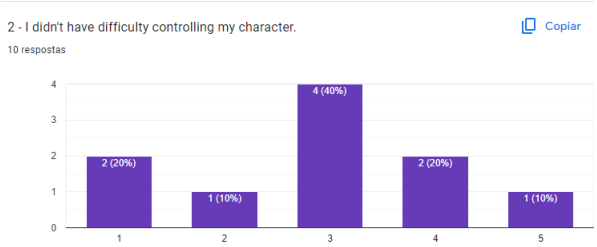

Figura 5.8: Controlar personagem.

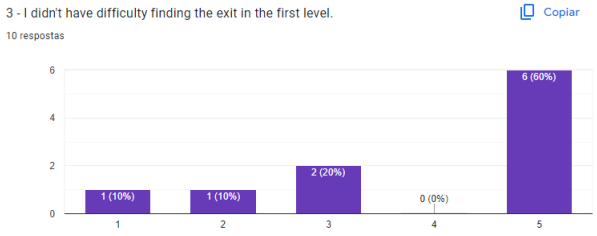

Figura 5.9: Encontrar a saída da casa.

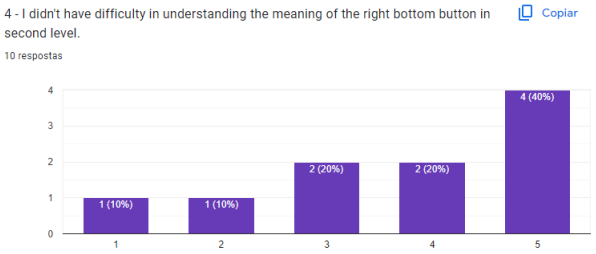

Figura 5.10: Botão de ajuda.

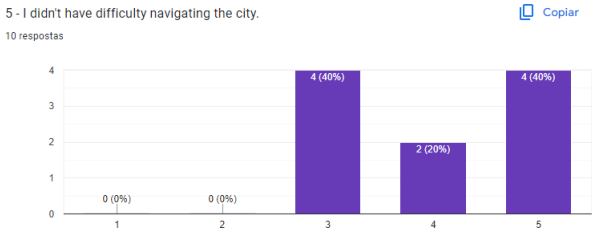

Figura 5.11: Navegar na cidade.

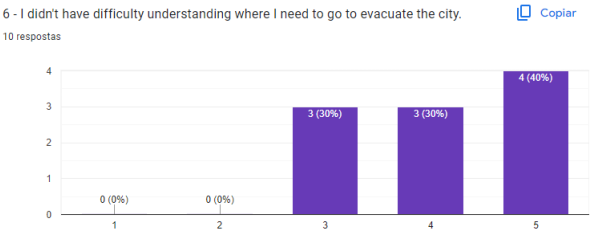

Figura 5.12: Por onde evacuar.

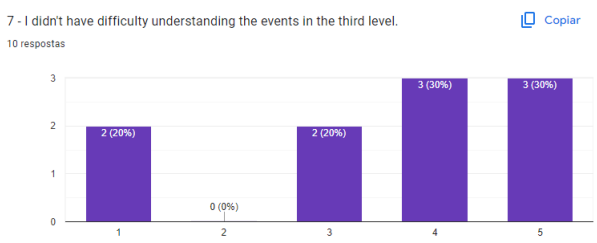

Figura 5.13: Eventos do 3º nível.

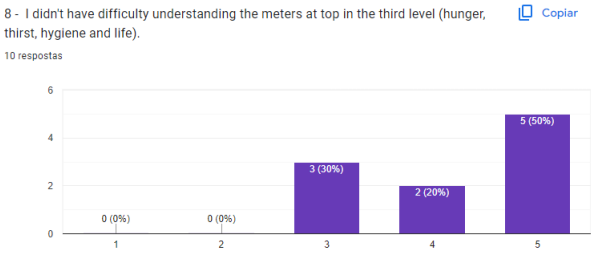

Figura 5.14: Medidores no 3º nível.

Conforme observado nas Figuras 5.7 a 5.14, há uma grande variação nas respostas, principalmente nas perguntas sobre a recolha dos recursos, controle do personagem, botão para ajudar pessoas e eventos do terceiro nível. Exceto pelo botão de ajuda, os outros elementos são bastante importantes para progredir no jogo; a falta de compreensão sobre esses elementos prejudica a experiência do jogador, o que pode explicar os resultados obtidos na secção 5.3.

A partir destes resultados, foi possível concluir que permitir que o utilizador aprenda as mecânicas do jogo por conta própria não é eficaz, pois isso depende do conhecimento prévio do utilizador sobre jogos. Essa dependência faz com que jogadores experientes aprendam facilmente as mecânicas do jogo, enquanto jogadores inexperientes têm extrema dificuldade em fazê-lo.

5.5 Game Experience

A "Experiência de Jogo" é composta por duas secções: "Experiência do Utilizador" e "Experiência do Utilizador após o Jogo". Cada uma dessas partes se concentra em examinar os sentimentos e pensamentos dos jogadores enquanto jogavam e após jogar o jogo, respectivamente.

Essa parte do questionário foi baseada no The Game Experience Questionnaire [17], que exemplifica um modelo de questionário que avalia três módulos de um jogo: a parte central, a parte social e a parte após o jogo. A parte central foca a experiência que o jogo proporciona ao jogador. A parte social investiga o envolvimento psicológico e comportamental do jogador ao interagir com outras entidades sociais. Por fim, a parte após o jogo avalia como os jogadores se sentiram após jogarem várias partidas do jogo. A "Experiência do Utilizador" usa a parte central e a "Experiência do Utilizador após o Jogo" usa a parte após o jogo. A parte social não foi implementada devido à ausência de outras entidades sociais com as quais o jogador pudesse interagir no jogo. Com base neste modelo, é possível avaliar o jogo com base nas experiências dos jogadores.

O sistema de pontuação desta secção agrupa determinadas perguntas para avaliar as qualidades do jogo. Na parte "Experiência do Utilizador", as seguintes qualidades são representadas pelas respectivas perguntas:

- **Tensão/Irritação**: 6ª pergunta;
- **Desafio**: 5ª pergunta;
- **Efeito Negativo**: 2ª e 4ª perguntas;
- **Efeito Positivo**: 1ª e 3ª perguntas;

Na parte de "Experiência do Utilizador após o jogo", as seguintes qualidades são representadas pelas respetivas perguntas:

- **Experiência Positiva**: 1ª, 5ª, 7ª, 8ª e 12ª perguntas
- **Experiência Negativa**: 2ª, 4ª, 6ª, 11ª e 14ª perguntas;
- **Cansaço**: 10º e 13º perguntas;
- **Retornar à Realidade**: 3ª e 9ª perguntas;

As perguntas e seus respectivos números podem ser verificados no Apêndice A. Cada uma das perguntas em cada secção possui cinco respostas, atribuindo valores de 0 a 4. Para avaliar a qualidade do jogo, é calculada a média dos valores das perguntas que representam essa qualidade. Por exemplo, a qualidade "Cansaço" da parte "Experiência do Utilizador após o Jogo" é calculada pela média das 10ª e 13ª perguntas dessa secção. As tabelas 5.3 e 5.4 demonstram a avaliação da "Experiência do Utilizador" e da "Experiência do Utilizador após o Jogo" para cada participante. Os valores também serão apresentados em formato de percentagem para facilitar a leitura.

Resultados da "Experiência do Utilizador":

	Participantes										
	1	2	3	4	5	6	7	8	9	10	Média
Tensão/Irritação	2	1	0	1	2	1	2	0	2	2	1.3
Desafio	4	1	1	4	2	1	2	3	2	2	2.2
Efeito Negativo	2	1	0	0	1	1	2	0	2	0	0.9
Efeito Positivo	2	3	4	3	2.5	3	2	4	2	4	2.95

Tabela 5.3: Resultados da "Experiência do utilizador".

Experiência	Média	Desvio Padrão	Percentagem
Tensão/Irritação	1.3	0.78	32.5
Desafio	2.2	1.1	55
Efeito Negativo	0.9	0.83	22.5
Efeito Positivo	2.95	0.79	73.7

Tabela 5.4: Média, desvio padrão e percentagem da "Experiência do utilizador".

Como pode ser observado mas tabelas 5.3 e 5.4, os jogadores, em geral, partilham a mesma opinião em relação à "Tensão/Irritação", "Efeito Negativo" e "Efeito Positivo" do jogo. Com "Tensão/Irritação" e "Efeito Negativo" apresentando valores baixos e "Efeito Positivo" apresentando um valor alto, é possível concluir que os utilizadores tiveram uma boa experiência ao jogar o JS, apesar das dificuldades mencionadas na secção 5.4. No entanto, o desvio padrão para "Desafio" mostra uma variação nas opiniões. A pergunta principal para avaliar o desafio é se os jogadores sentiram a pressão do tempo; portanto, a variação nesse aspecto indica que alguns jogadores sentiram-se aflitos com o limite de tempo imposto até o tsunami chegar, enquanto outros não.

Resultados da "Experiência do Utilizador após o Jogo":

	Partecipantes										
	1	2	3	4	5	6	7	8	9	10	Média
Experiência Positiva	1.2	2.6	3.6	2.2	2.2	2.4	1.8	3.8	2.6	3	2.5
Experiência Negativa	0.2	1.4	0.6	1	2	1.4	2	0	0.5	0	0.9
Cansaço	2.5	1	0	1	1	1.5	2	0	1.5	0	1
Retornar à Realidade	2.5	2	0	1.5	1	1	2	0	1	0	1.1

Tabela 5.5: Resultados da "Experiência do utilizador após o jogo".

Experiência	Média	Desvio Padrão	Percentagem
Experiência Positiva	2.5	0.74	62.5
Experiência Negativa	0.9	0.73	22.5
Cansaço	1	0.61	25
Retornar à Realidade	1.1	0.86	27.5

Tabela 5.6: Média, desvio padrão e percentagem da "Experiência do utilizador após o jogo".

Ao verificar as tabelas 5.5 e 5.6, relativas aos resultados da experiência após o jogo, é possível notar que os respondentes têm opiniões semelhantes em relação à "Experiência Positiva", "Experiência Negativa", "Cansaço" e "Retornar à Realidade". "Experiência Negativa" e "Cansaço" possuem valores baixos, indicando que após jogarem o jogo, os utilizadores não se importaram muito com os aspectos negativos do jogo nem se sentiram exaustos por jogá-lo. O "Retornar à Realidade" com uma percentagem baixa indica que os utilizadores não estavam tão envolvidos no JS a ponto de não conseguirem parar de jogar. Por fim, a "Experiência Positiva" possui um valor acima da média, indicando que o JS despertou emoções positivas nos utilizadores.

Com base nas informações obtidas, é possível concluir que a parte do jogo do JS desenvolvido proporcionou uma boa experiência aos jogadores. Apesar das dificuldades na usabilidade mencionadas nas secções 5.3 e 5.4, o sistema criado foi capaz de oferecer uma boa experiência aos utilizadores.

5.6 Componente Didática

A "Componente Didática" avalia os conhecimentos adquiridos pelos jogadores após participarem do jogo sério. A primeira parte desta secção será usada para comparar as respostas na "Caracterização do Utilizador" para determinar se a percepção do utilizador sobre o risco de tsunamis mudou, enquanto a segunda parte se dedica a determinar se os utilizadores aprenderam outros conhecimentos proporcionados pelo JS desenvolvido.

5.6.1 Comparação de resultados

Primeiramente, as respostas na "Caracterização do Utilizador" serão comparadas com as obtidas na "Componente Didática". Começando pelos sinais naturais de um tsunami (Figura 5.15).

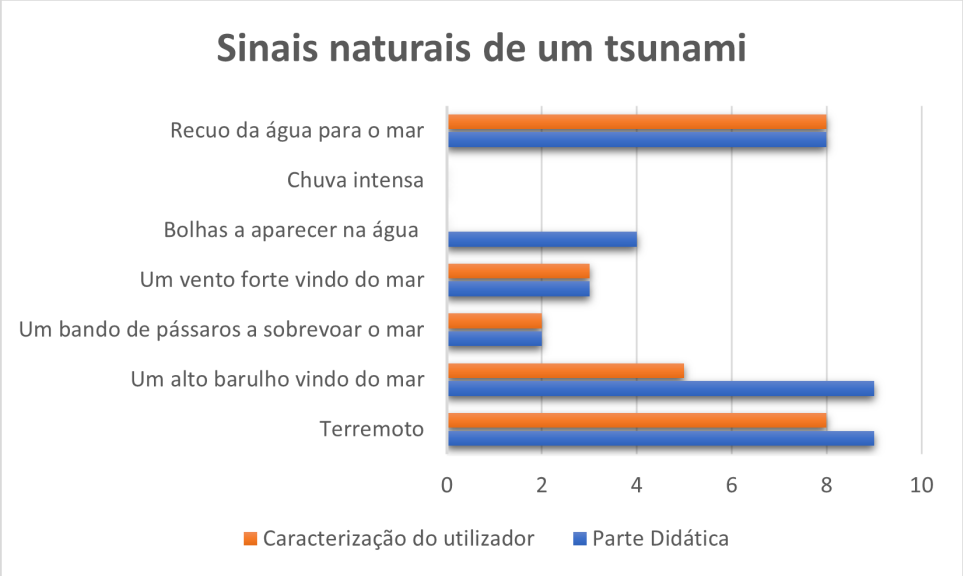

Figura 5.15: Comparação de respostas: Sinais naturais de um tsunami.

Nesta pergunta, houve uma melhoria nas respostas. Apesar de o JS não ter alterado a opinião dos utilizadores que escolheram as opções "Um bando de pássaros voando sobre o mar" e "Um forte vento vindo do mar", o jogo foi capaz de ensinar os principais sinais naturais de um tsunami, mais precisamente "Terramoto", "Um barulho alto vindo do mar", "Bolhas aparecendo na água" e "O recuo da água para o mar". A próxima pergunta é referente ao que colocar num Kit de Emergência (Figura 5.16).

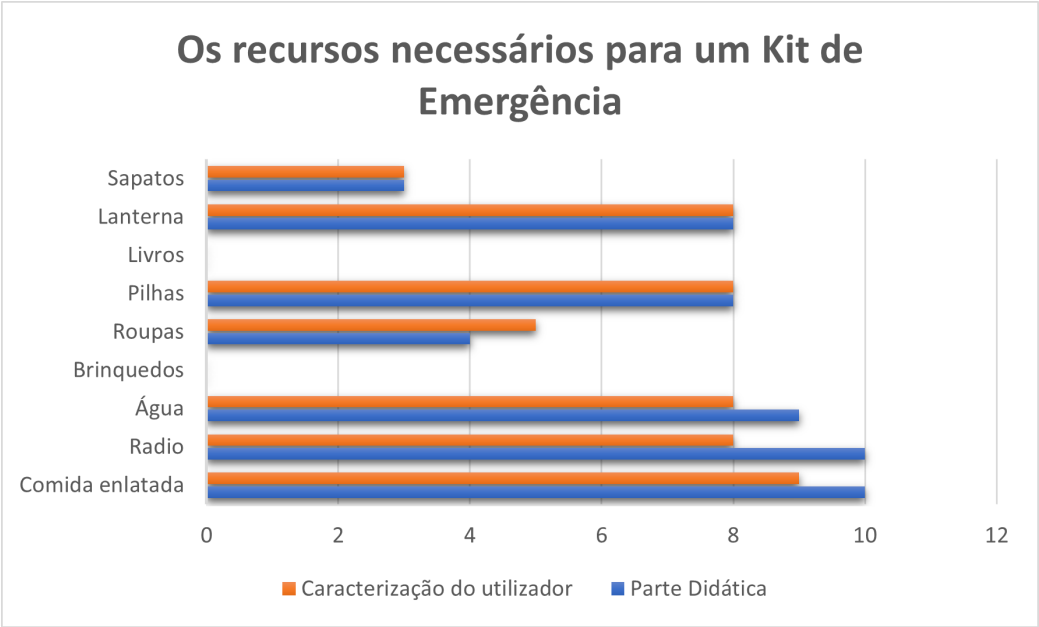

Figura 5.16: Comparação de respostas: Recursos para o Kit de Emergência.

A pergunta sobre os recursos de um Kit de Emergência apresenta os mesmos resultados que a pergunta anterior. Nesse caso, o JS teve dificuldade em desincentivar o uso de roupas e sapatos no Kit de Emergência, mas foi capaz de reforçar a necessidade de ter comida, água e um rádio para comunicação. A última pergunta foca-se no que é mais importante numa situação de tsunami (Figura 5.17).

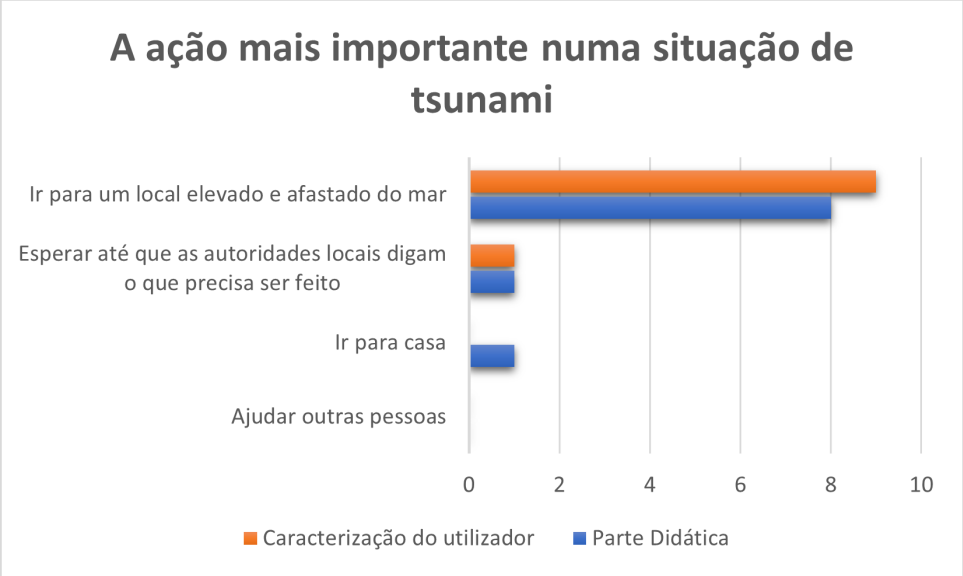

Figura 5.17: Comparação de respostas: A ação mais importante numa situação de tsunami.

Ao contrário das questões anteriores, esta pergunta apresentou resultados piores. A grande maioria dos utilizadores continuou a escolher a opção correta, mas um respondente mudou sua resposta correta para "Ir para casa e pegar tudo o que é importante". Esta resposta pode ter surgido do facto de os jogadores sempre começarem dentro de casa em cada rodada de jogo e precisarem recolher recursos para sobreviver ao tsunami. Esta mecânica pode ter induzido o jogador a erro.

Ao analisar cada uma das perguntas, é possível observar que o JS desenvolvido é capaz de transmitir conhecimento aos jogadores. Embora não consiga alterar alguns pressupostos que os utilizadores possam ter, na grande maioria dos casos, o JS é capaz de educar e reforçar as ideias básicas para sobreviver a um tsunami.

5.6.2 Obtenção de novos conhecimentos

As próximas perguntas foram formuladas para avaliar duas mecânicas implementadas no JS: autoaprendizagem e texto no ecrã de carregamento.

Como mencionado anteriormente, uma das principais características do JS desenvolvido é permitir que o jogador aprenda por si mesmo enquanto joga. Ao permitir que o jogador cometa erros e os corrija, o aprendizagem torna-se mais compreensível e duradouro. No entanto, como a proposta deste jogo é ser rápido para que os utilizadores possam iniciar uma nova partida quando terminam a anterior, algumas lições sobre como lidar com tsunamis podem ser comprometidas. Para resolver esse problema, a informação comprometida foi colocada no ecrã de carregamento entre níveis, permitindo que o jogador que aceda a essa informação enquanto aguarda o carregamento do nível.

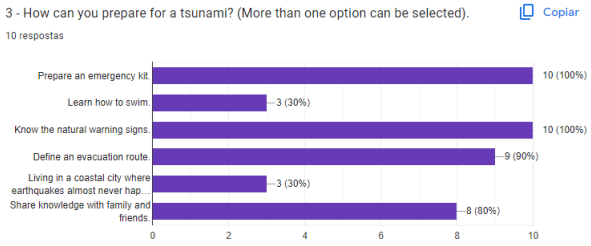

Figura 5.18: Como se preparar para um tsunami.

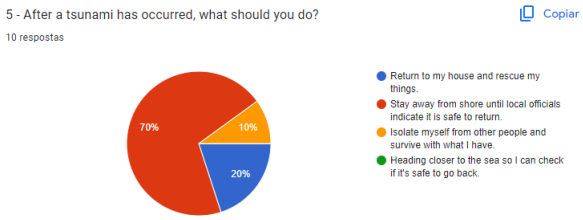

Figura 5.19: O que fazer depois de um tsunami.

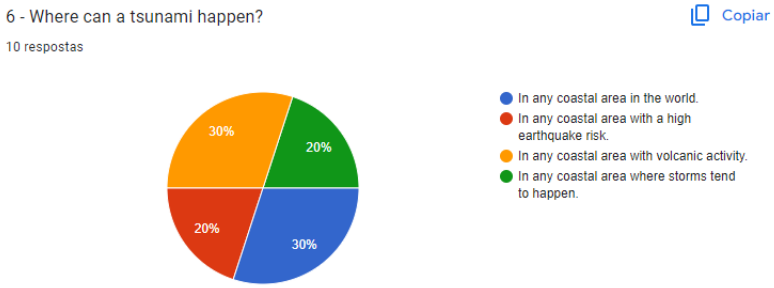

Figura 5.20: Onde tsunamis podem acontecer.

Como é possível verificar nas tabelas 5.18 a 5.20, as perguntas obtiveram bons resultados, com exceção da pergunta na Figura 5.20. Como se preparar para um tsunami e o que fazer após um tsunami são lições bem integradas no JS. A autoaprendizagem foi efetivo para a maioria dos jogadores, o que torna o mesmo uma boa mecânica para se implementar em jogos sérios. Por outro lado, a perguna sobre onde um tsunami pode ocorrer obteve respostas bastantes variadas. Esta informação é apresentada no texto da tela de carregamento, tornando este método a única forma de obter a mesma. Pelo o que se pode verificar, a mecânica de implementar informações que não estão presentes na jogabilidade em ecrãs de carregamento não é muito efetiva para o processo de aprendizagem.

Com base nestes testes, é possível concluir o JS é capaz de transmitir informações sobre como reagir em caso de tsunami. A mecânica de autoaprendizagem provou-se efetiva no papel de transmitir conhecimentos, ao contrário da mecânica de colocar informação textual nos ecrãs de carregamento.

6

Conclusões e Trabalho Futuro

Os jogos sérios representam um campo onde alguns tópicos têm sido pouco explorado no âmbito educacional, com poucos estudos avaliando sua eficácia na aquisição de conhecimento. Com o avanço da tecnologia, os métodos de ensino também precisam evoluir, e os jogos sérios emergem como uma rota promissora a seguir. A interatividade oferecida por esses jogos supera os métodos de ensino tradicionais, fundamentados na relação professor-aluno. No entanto, como o próprio nome sugere, os jogos sérios têm um elemento de entretenimento, o que os torna envolventes para os jogadores. Embora esse fator seja uma vantagem ao prender a atenção dos utilizadores, ele também apresenta desafios, pois muitas vezes elementos educacionais são sacrificados para melhorar a jogabilidade. Portanto, os jogos sérios não devem substituir inteiramente os métodos de ensino convencionais, mas sim atuar como ferramentas complementares que coexistem e auxiliam na retenção de informações ensinadas.

A principal proposta deste trabalho foi desenvolver um jogo sério para dispositivos Android, com o intuito de ensinar os jogadores sobre os perigos dos tsunamis. A escolha de um ambiente para dispositivos Android motivou a criação de um jogo casual, onde os jogadores pudessem repetir o jogo várias vezes, adquirindo conhecimento ao longo do processo. Com esse objetivo em mente, foram explorados jogos sérios com o tema de tsunamis para entender os elementos essenciais desse tipo de jogo, assim como jogos casuais com temas semelhantes para entender como integrar a abordagem casual.

Durante o desenvolvimento, foram investigadas abordagens envolvendo inteligência artificial e geração procedural, a fim de proporcionar entretenimento e reduzir a repetição dentro do jogo. Isso resultou na implementação da tecnologia ML-Agents e do algoritmo "Wave Function Collapse". A tecnologia ML-Agents permitiu a criação de agentes treinados para realizar uma tarefa específica, nesse caso, escapar da cidade antes que o tsunami a atingisse. A inclusão desses agentes trouxe um nível de realismo ao jogo, já que eles também precisavam encontrar uma rota de fuga sem possuir conhecimento prévio da mesma, da mesma forma que os jogadores. Além disso, o algoritmo "Wave Function Collapse" foi utilizado para gerar a cidade de forma procedural, alterando sua aparência a cada iteração do jogo. Essa mudança na organização da cidade serviu para manter o interesse dos jogadores e diminuir a monotonia da jogabilidade. Por meio dessas técnicas, foi possível adicionar mais variação e conteúdo às sessões de jogo dos utilizadores.

A avaliação do jogo foi conduzida com participantes de diversas faixas etárias, uma vez que o objetivo era ensinar pessoas independentemente de sua idade. A usabilidade foi avaliada como boa, com uma pontuação média de 70.5 no SUS. No entanto, ao analisar individualmente os resultados de usabilidade para cada participante, foi observada uma variação significativa nos resultados, causada pelo facto de o jogo ser projetado para permitir que os utilizadores aprendam com seus próprios erros. Essa característica torna o jogo mais dependente da habilidade do jogador, o que pode resultar numa experiência de usabilidade negativa para aqueles sem experiência em jogos ou sem a capacidade de reconhecer os padrões existentes nos mesmos. Esse desafio foi especialmente evidente na dificuldade que alguns utilizadores enfrentaram para compreender certas mecânicas do jogo. Os resultados, assim como os testes de usabilidade, revelaram uma variação considerável, indicando que o método de autoaprendizagem nem sempre é adequado quando o objetivo é atingir um público amplo.

No entanto, ao aplicar o "Game Experience Questionnaire" para avaliar a experiência oferecida pelo jogo como um todo, ficou claro que os jogadores tiveram uma experiência positiva, mesmo aqueles que não avaliaram bem a usabilidade. Por fim, ao examinar o conhecimento adquirido pelos jogadores, foi possível constatar que o jogo foi eficaz na transmissão de informações, validando a abordagem de autoaprendizagem como uma forma eficaz de educação dos utilizadores. No entanto, mecânicas focadas na transmissão de informações que não fazem parte da experiência direta do jogo, como texto nos ecrãs de carregamento, não se mostraram tão eficazes para ensinar os jogadores.

Apesar dos desafios de usabilidade derivados da mecânica de autoaprendizagem, o jogo desenvolvido conseguiu proporcionar uma experiência agradável aos jogadores e transmitir conhecimento sobre como reagir a situações de tsunami. Como resultado, o principal objetivo deste trabalho, a criação de um jogo sério para dispositivos móveis capaz de educar sobre desastres naturais, foi alcançado.

Embora os objetivos principais tenham sido cumpridos, não muda o facto de o jogo ainda apresentar problemas. Para futuros desenvolvimentos, as seguintes melhorias podem ser implementadas para aprimorar a experiência do jogo e a eficácia da aprendizagem:

- Incluir uma fase de tutorial para orientar os jogadores sobre comandos e mecânicas do jogo;
- Adicionar mais cenários além da casa inicial, introduzindo outras situações onde as pessoas possam se encontrar durante um tsunami;
- Incorporar mais sinais de alerta de tsunami, além do terramoto e da sirene;
- Introduzir outras formas de fuga, como evacuação vertical;
- Aprimorar o sistema de eventos no terceiro nível, tornando-o menos monótono e mais compreensível.

Referências

[1] Robot Gentleman. "60 Seconds! Reatomized". (), URL: https://robotgentleman.com/60seconds/.

[2] Jonathan Moizer, Jonathan Lean, Elena Dell'Aquila, Paul Walsh, Alphonsus (Alfie) Keary, Deirdre O'Byrne, Andrea Di Ferdinando, Orazio Miglino, Ralf Friedrich, Roberta Asperges & Luigia Simona Sica, "An approach to evaluating the user experience of serious games", *Computers & Education*, vol. 136, páginas 141–151, 2019.

[3] Steven Hawthorn, Rui Jesus & Maria Ana Baptista, "TSUNAMI RISK COMMUNICATION: ASSESSING KNOWLEDGE GAPS AND SUITABLE SERIOUS GAMES", *Journal of Tsunami Society International*, vol. 41, n.º 1, 2022.

[4] "Build A Kit". (out. de 2022), URL: https://www.ready.gov/kit.

[5] Jussi Kuittinen, Kultima Annakaisa, Johannes Niemelä & Janne Paavilainen, "Casual games discussion", páginas 105–112, nov. de 2007.

[6] Juho Hamari, David Shernoff, Elizabeth Rowe, Brianno Coller, Jodi Asbell-Clarke & Teon Edwards, "Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning", *Computers in Human Behavior*, ago. de 2016.

[7] National Geophysical Data Center / World Data Service: NCEI/WDS Global Historical Tsunami Database. NOAA National Centers for Environmental Information. "NCEI/WDS Global Historical Tsunami Database, 2100 BC to Present". (jul. de 2021), URL: https://data.noaa.gov/metaview/page?xml=NOAA/NESDIS/NGDC/MGG/Hazards/iso/xml/G02151.xml&view=getDataView#.

[8] Zeno Rogue. "What is the difference between Roguelike and Roguelite?" (Abr. de 2022), URL: https://zenorogue.medium.com/what-is-the-difference-between-roguelike-and-roguelite-4c0fdc403db.

[9] "Disaster Master". (), URL: https://www.ready.gov/kids/games/data/dm-english/index.html.

[10] Julian Alvarez, Damien Djaouti, Sandy Louchart, Yoann Lebrun, Nabil Zary, Sophie Lepreux & Christophe Kolski, "A formal approach to distinguish Games, Toys, Serious Games & Toys, Serious Re-purposing & Modding and Simulators", *IEEE Transactions on Games*, páginas 1–13, jun. de 2022.

[11] Earth Observatory of Singapore. "Earth Girl". (2013), URL: http://earthgirlgame.com/.

[12] Earth Observatory of Singapore. "Earth Girl Tsunami". (), URL: https://earthgirl2.com/earth-girl-tsunami/.

[13] B. Yurdaarmagan, Ceren Melek, Burak Merdenyan, O. Cikrikcili, Yucel Salman & Hong-In Cheng, "The effects of digital game-based learning on performance and motivation for high school students", *ICIC Express Letters*, vol. 9, páginas 1465–1469, jan. de 2015.

[14] Emmanuel Fokides, Penelope Atsikpasi, Polyxeni Kaimara & Ioannis Deliyannis, "Factors Influencing the Subjective Learning Effectiveness of Serious Games", *Journal of Information Technology Education:Research*, vol. 18, páginas 437–466. Out. de 2019.

[15] Jan Plass, Bruce Homer & Charles Kinzer, "Foundations of Game-Based Learning", *Educational Psychologist*, vol. 50, páginas 258–283, out. de 2015.

[16] Dorian Iten. "Understanding the Fresnel Effect". (), URL: https://www.dorian-iten.com/fresnel/.

[17] IJsselsteijn, W. A., de Kort, Y. A. W., & Poels & K., "The Game Experience Questionnaire", *Technische Universiteit Eindhoven*, 2013.

[18] Nicolas Wenk & Stéphane Gobron, "Reinforcing the difference between Simulation, Gamification, and Serious Game", em *proceedings of the Gamification & Serious Games Symposium 2017 (GSGS'17)*, HES-SO HE-Arc, ed., 2017, páginas 1–3.

[19] Ankur Joshi, Saket Kale, Satish Chandel & Dinesh Pal, "Likert Scale: Explored and Explained", *British Journal of Applied Science & Technology*, vol. 7, páginas 396–403, jan. de 2015.

[20] Peter Gabrov ˇsek, "Analysis of Maze Generating Algorithms", *The IPSI BgD Transactions on Internet Research*, vol. 15, n.º 1, jan. de 2019.

[21] Will T, "Measuring and Interpreting System Usability Scale (SUS)", *UX Research*, dez. de 2021.

[22] Antonia Ypsilanti, Ana B. Vivas, Teppo Räisänen, Matti Viitala, Tuula Ijäs & Donald Ropes, "Are serious video games something more than a game? A review on the effectiveness of serious games to facilitate intergenerational learning", *Springer Science+Business Media*, 2014.

[23] Aleksandra Solinska-Nowak, Piotr Magnuszewski, Margot Curl, Adam French, Adriana Keating, Junko Mochizuki, Wei Liu, Reinhard Mechler, Michalina Kulakowska & Lukasz Jarzabek, "An overview of serious games for disaster risk management – Prospects and limitations for informing actions to arrest increasing risk", *International Journal of Disaster Risk Reduction*, 2018.

[24] Steven Hawthorn, Rui Jesus & Maria Ana Baptista, "A Review of Digital Serious Games for Tsunami Risk Communication", *International Journal of Serious Games*, vol. 8, n.º 2, 2021.

[25] Josh Bycer. "The Roguelike Debate – Roguelikes vs Roguelites". (nov. de 2019), URL: https://www.gamedeveloper.com/design/the-roguelike-debate---roguelikes-vs-roguelites.

[26] Clark C. Abt, *Serious Games*. Viking Press, 1970.

[27] UNDRR. "Stop Disasters". (), URL: https://www.stopdisastersgame.org/#1540988157744-7c3ef168-edc2.

[28] Werner Ravyse, Seugnet Blignaut, Verona Leendertz & Alex Woolner, "Success factors for serious games to enhance learning: a systematic review", *Virtual Reality*, vol. 21, mar. de 2017.

[29] John Brooke, "SUS - A quick and dirty usability scale", *Usability Eval. Ind.*, vol. 189, nov. de 1995.

[30] "A gaming application that teaches players how to protect themselves against natural disasters: Tanah – The Tsunami & Earthquake Fighter". (abr. de 2017), URL: https://www.cepal.org/en/notas/videojuego-que-ayuda-protegerse-frente-desastres-naturales-tanah-terremotos-tsunamis.

[31] "Treme-Treme". (), URL: https://www.treme-treme.pt/.

[32] Jorge León, Patricio A. Catalán & Alejandra Gubler, "Assessment of Top-Down Design of Tsunami Evacuation Strategies Based on Drill and Modelled Data", *Front. Earth Sci.*, dez. de 2021.

[33] National Tsunami Warning Center. "Tsunami Frequently Asked Questions". (ago. de 2023), URL: https://www.tsunami.gov/?page=tsunamiFAQ.

[34] Unity. "Universal Render Pipeline overview". (), URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/index.html.

[35] Hwanhee Kim, Seongtaek Lee, Hyundong Lee, Teasung Hahn & Shinjin Kang, "Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm", páginas 1–4, ago. de 2019.

Questionário

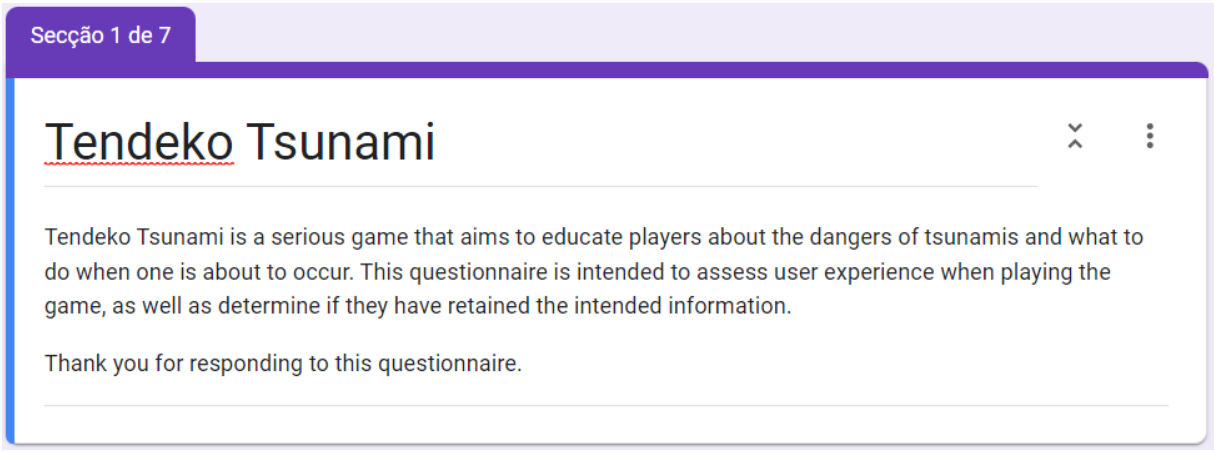

Figura A.1: Introdução do questionário.

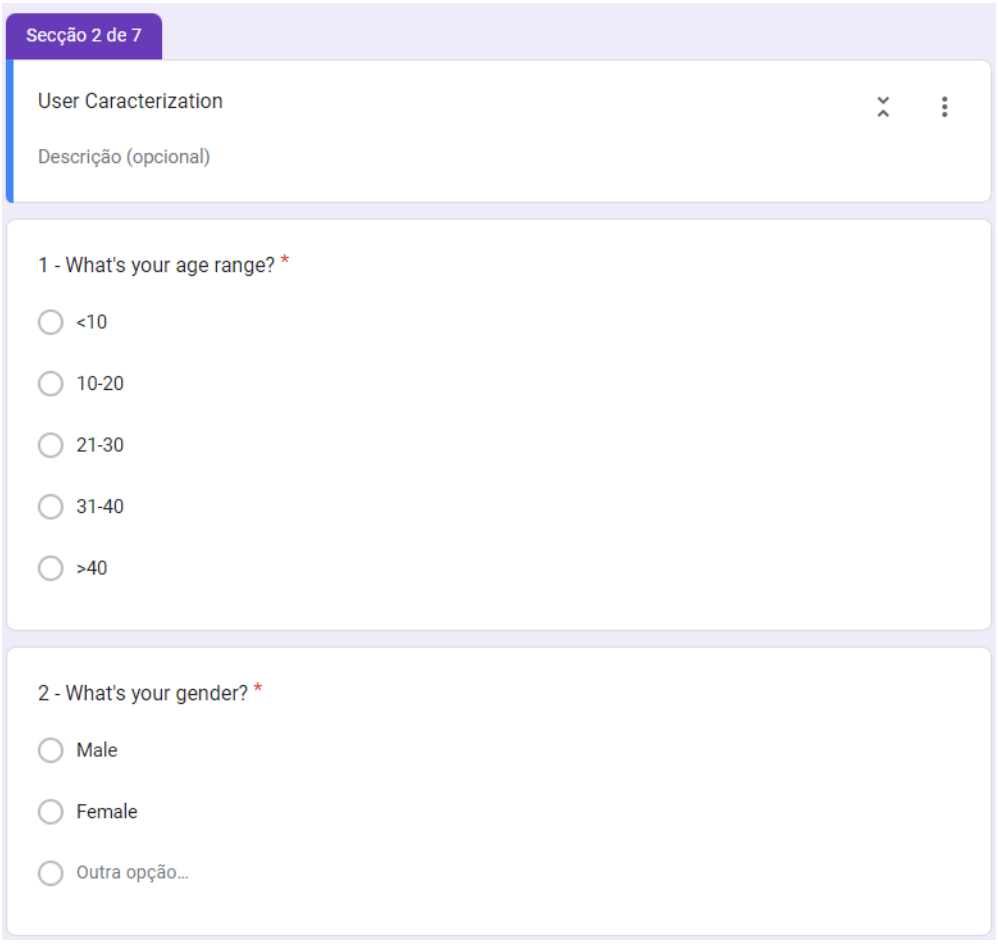

Figura A.2: Caracterização do usuário: 1-2.

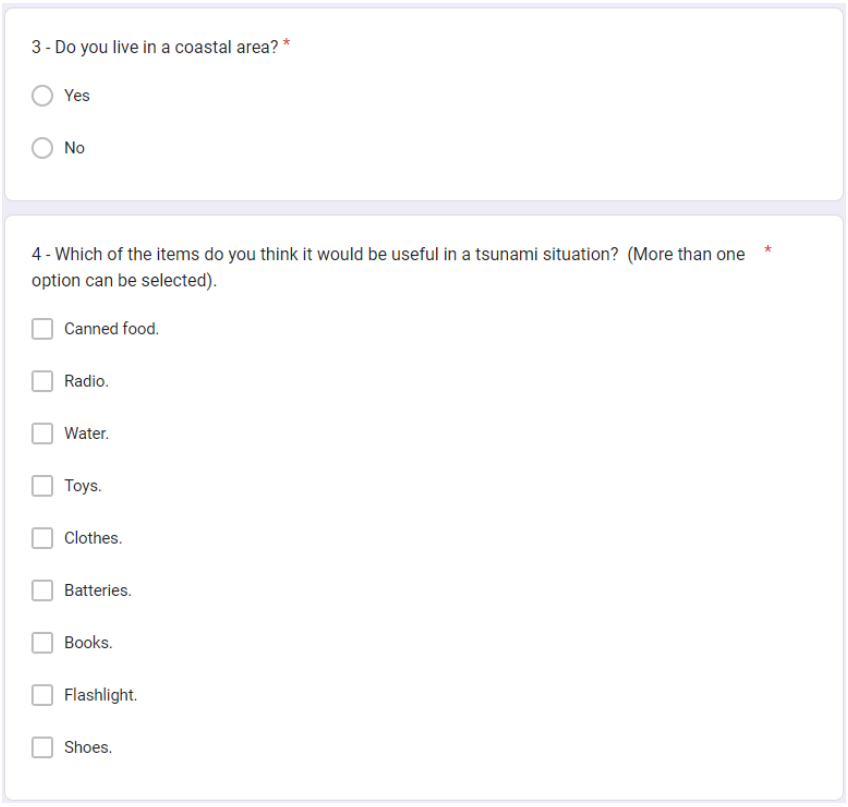

Figura A.3: Caracterização do usuário: 3-4.

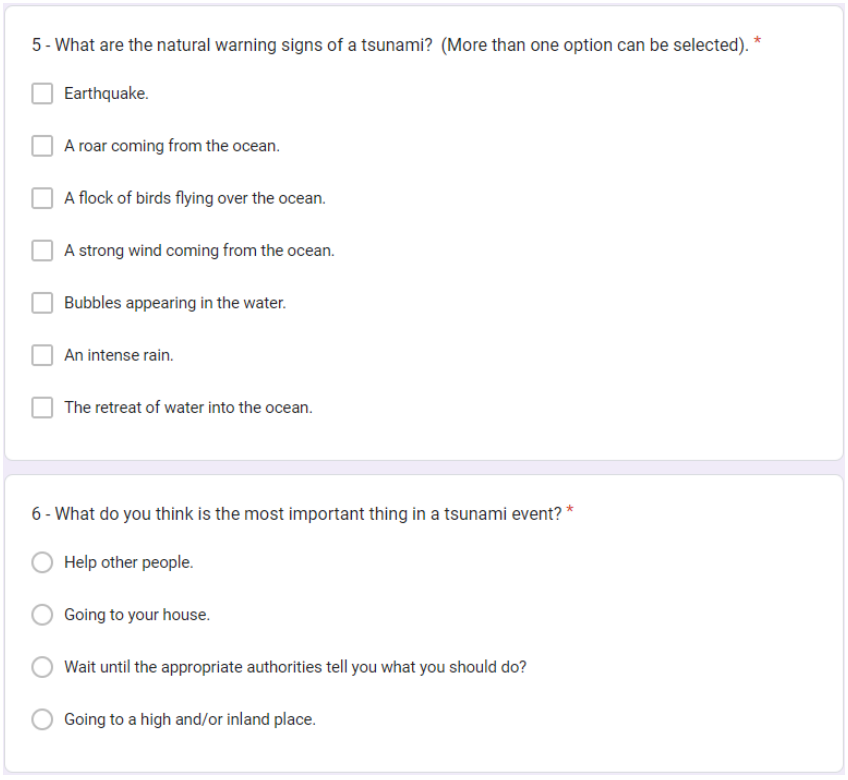

Figura A.4: Caracterização do usuário: 5-6.

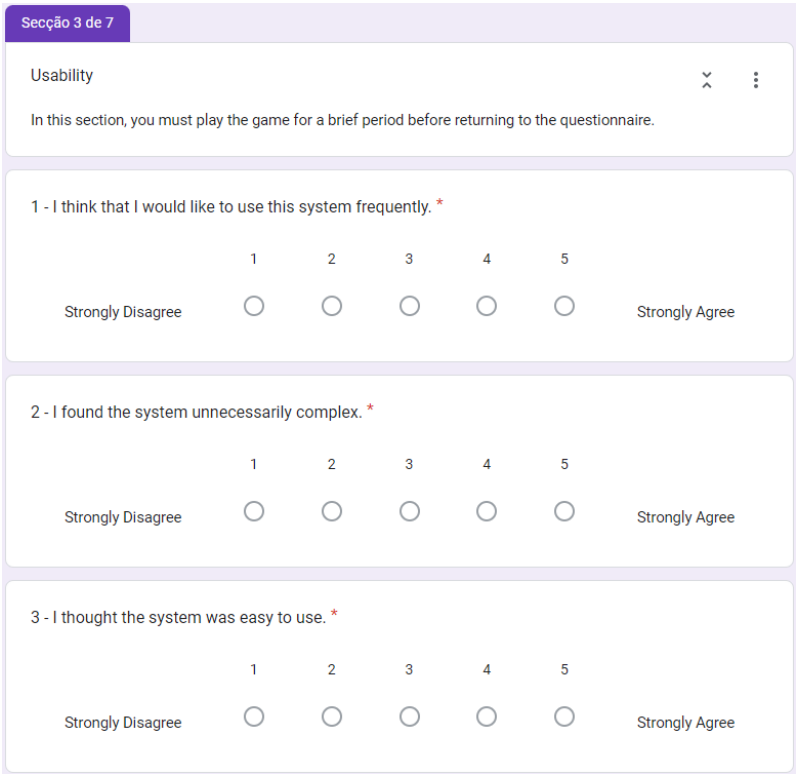

Figura A.5: Usabilidade: 1-3.

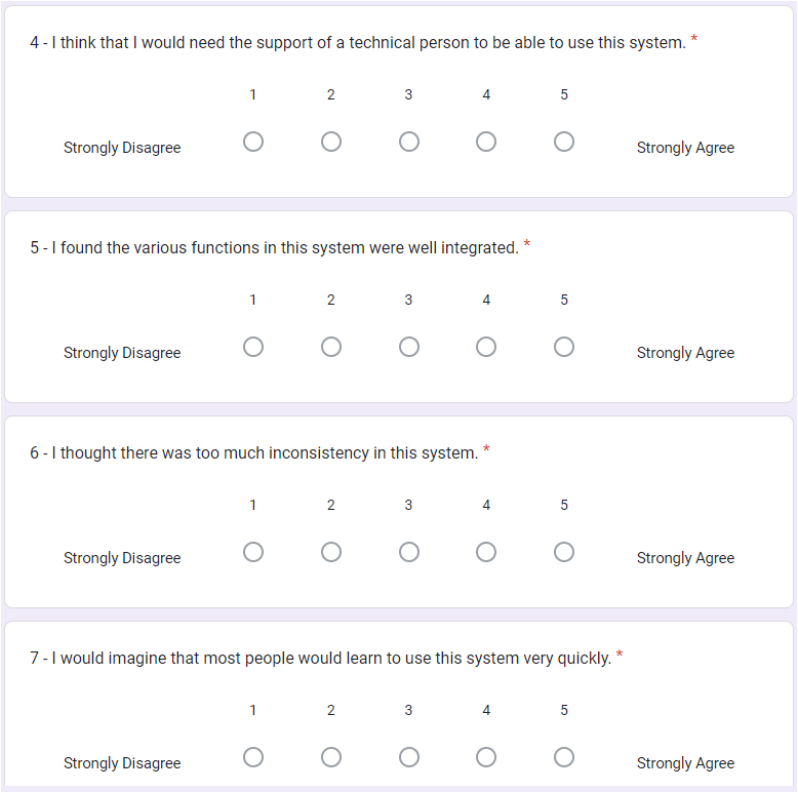

Figura A.6: Usabilidade: 4-7.

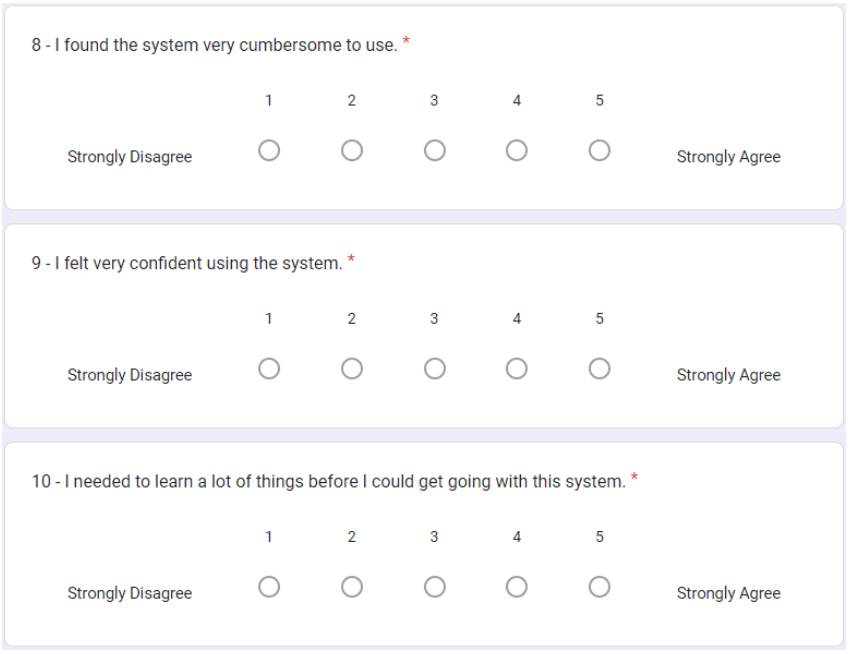

Figura A.7: Usabilidade: 8-10.

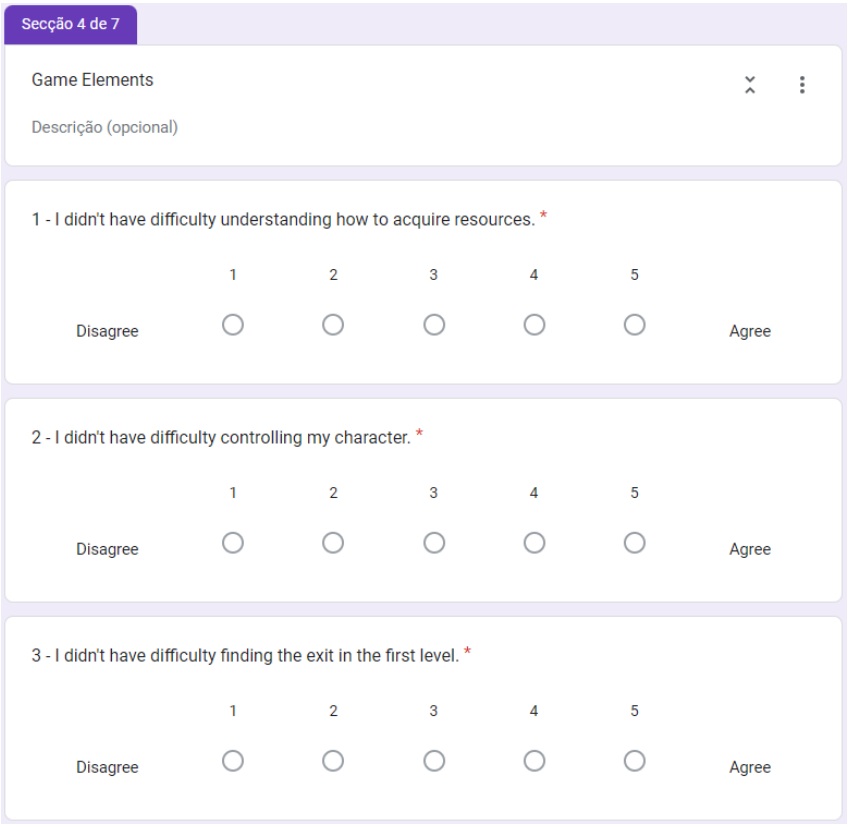

Figura A.8: Elementos do jogo: 1-3.

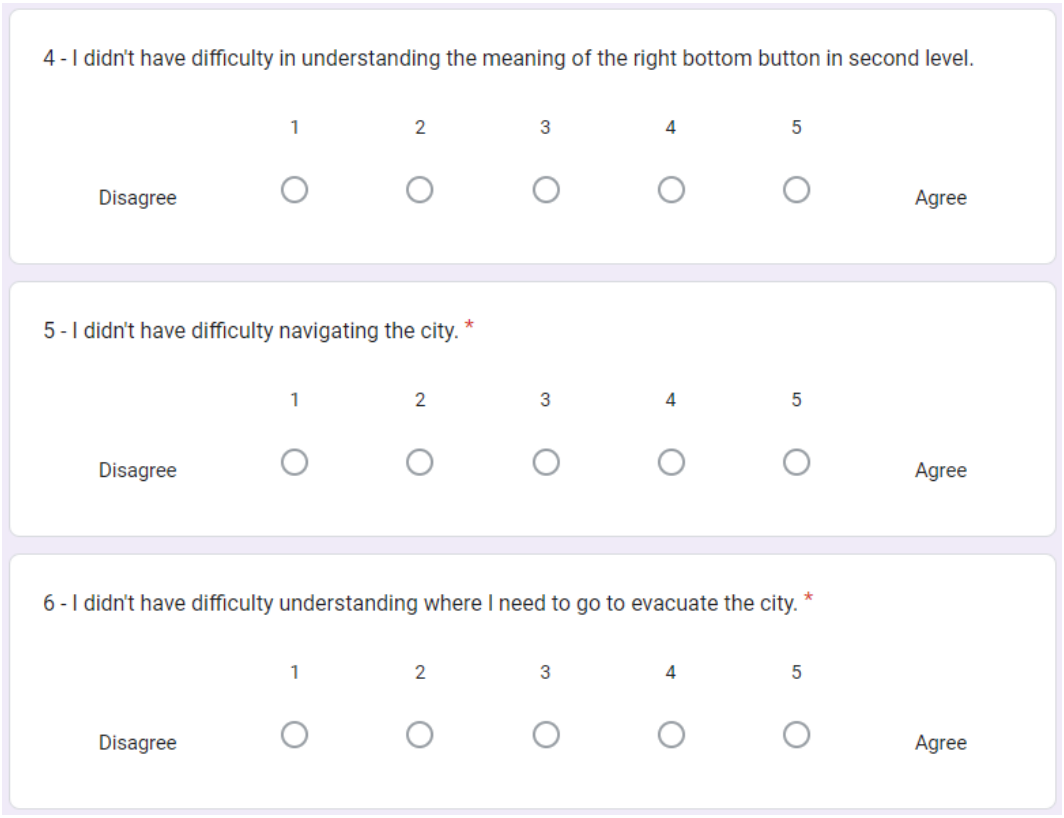

Figura A.9: Elementos do jogo: 4-6.

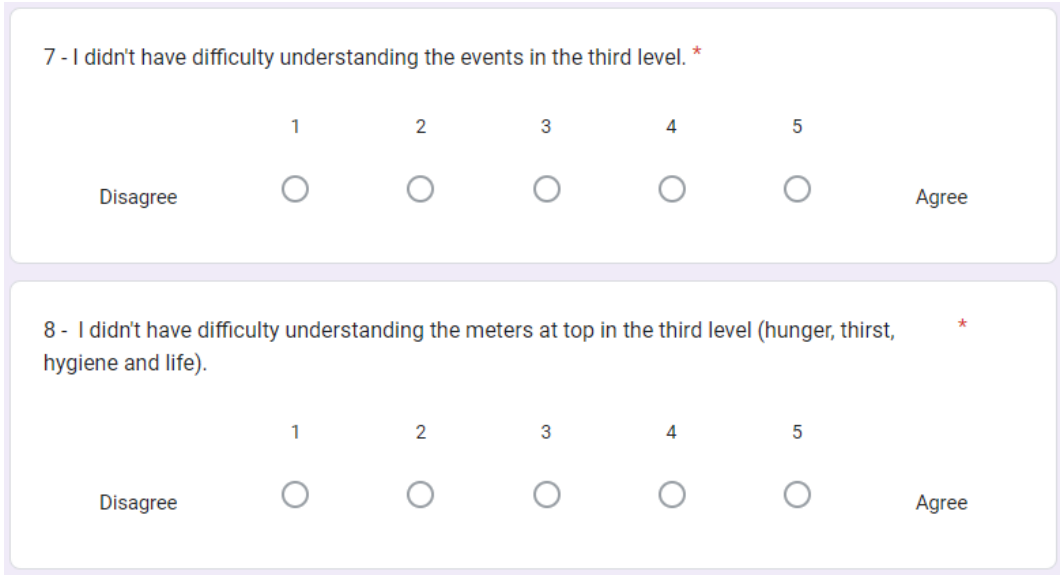

Figura A.10: Elementos do jogo: 7-8.

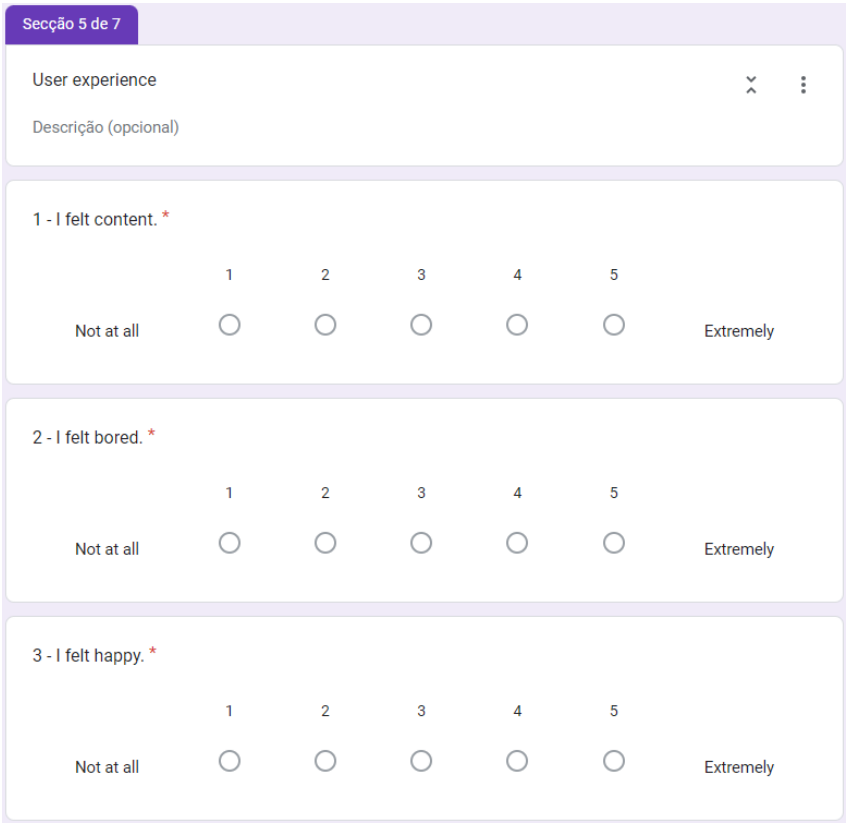

Figura A.11: Experiência do utilizador: 1-3.

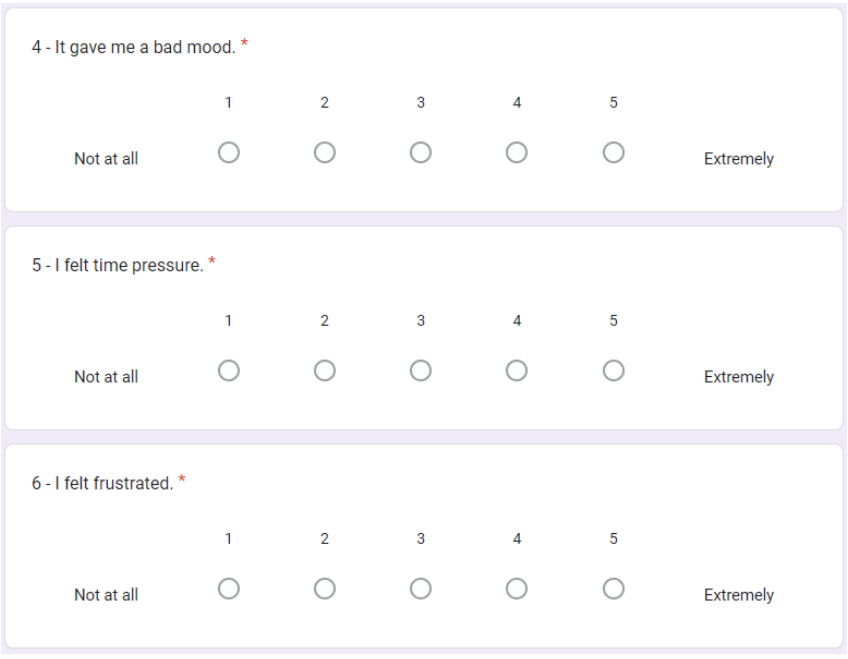

Figura A.12: Experiência do utilizador: 4-6.

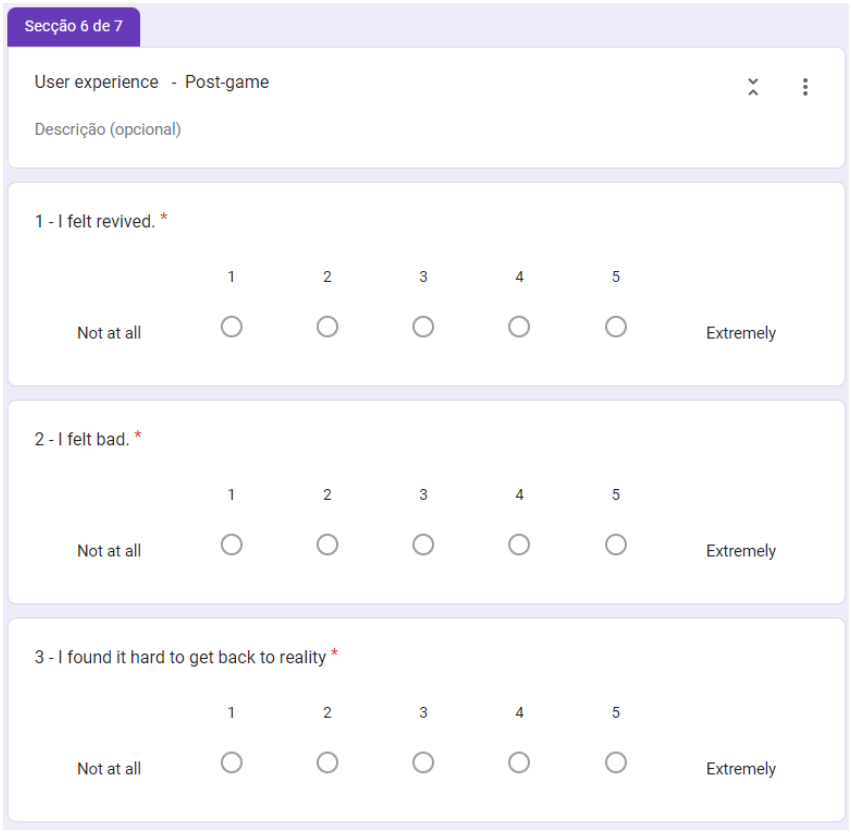

Figura A.13: Experiência do utilizador após o jogo: 1-3.

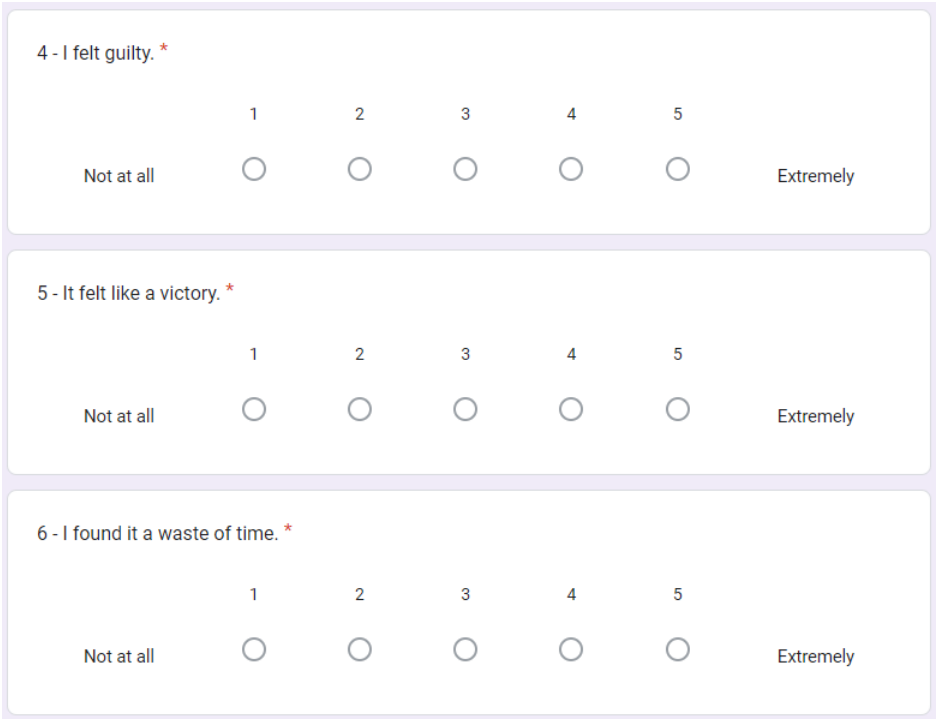

Figura A.14: Experiência do utilizador após o jogo: 4-6.

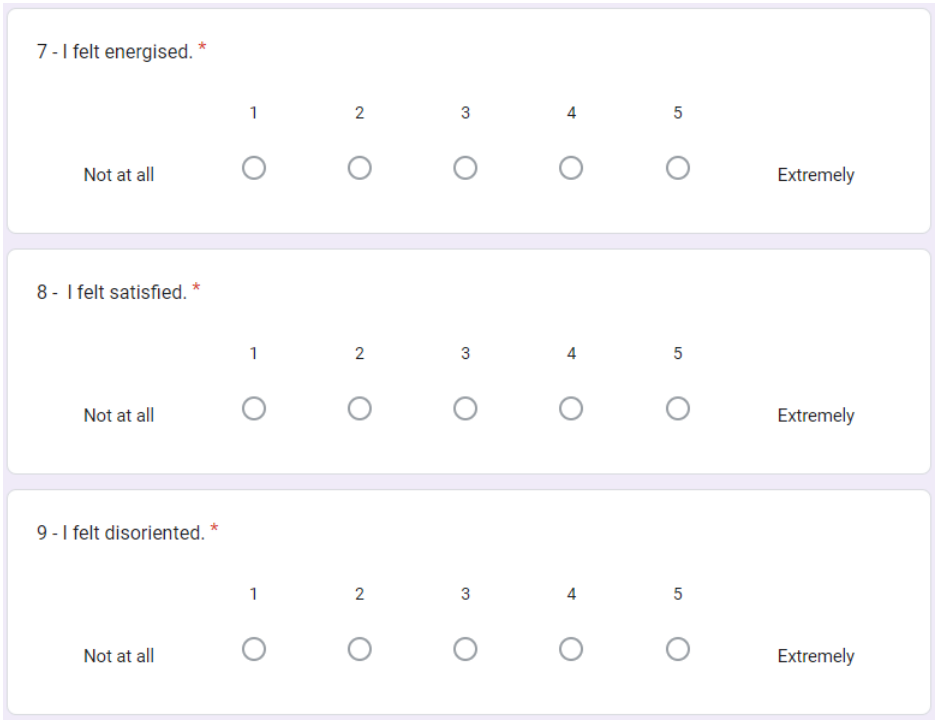

Figura A.15: Experiência do utilizador após o jogo: 7-9.

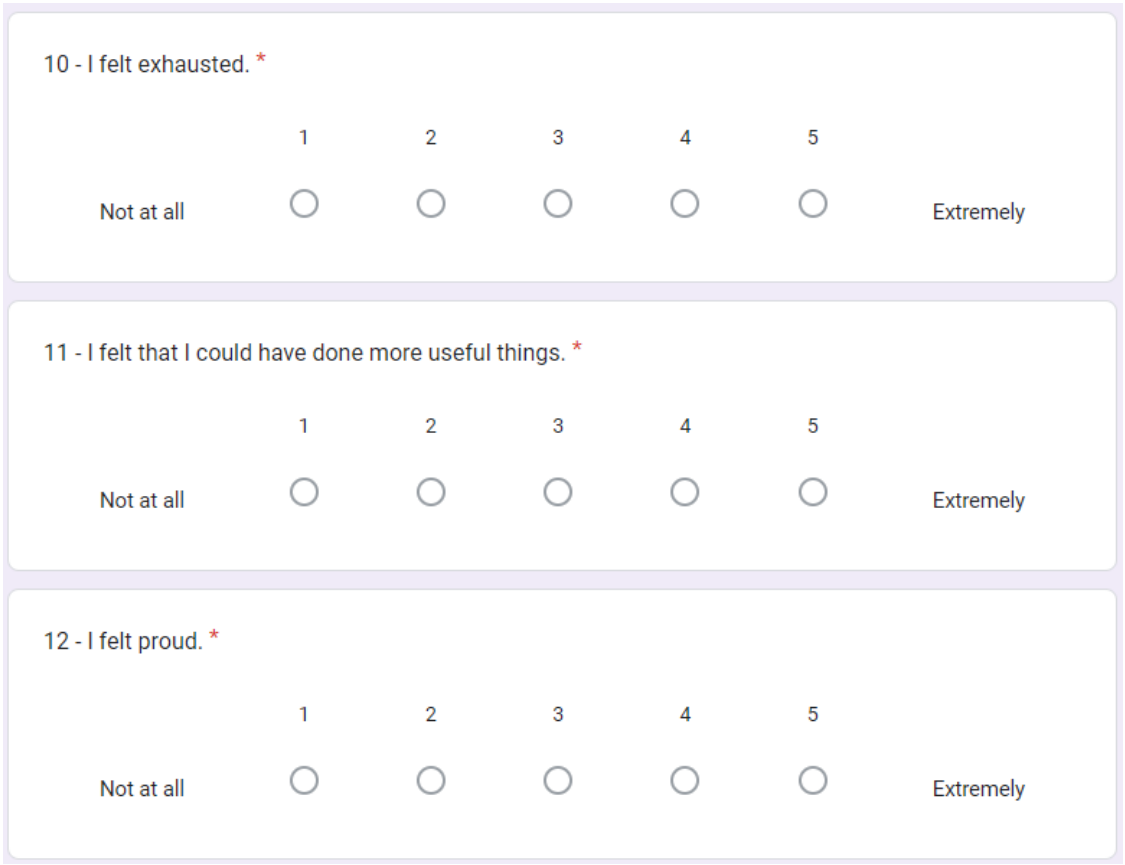

Figura A.16: Experiência do utilizador após o jogo: 10-12.

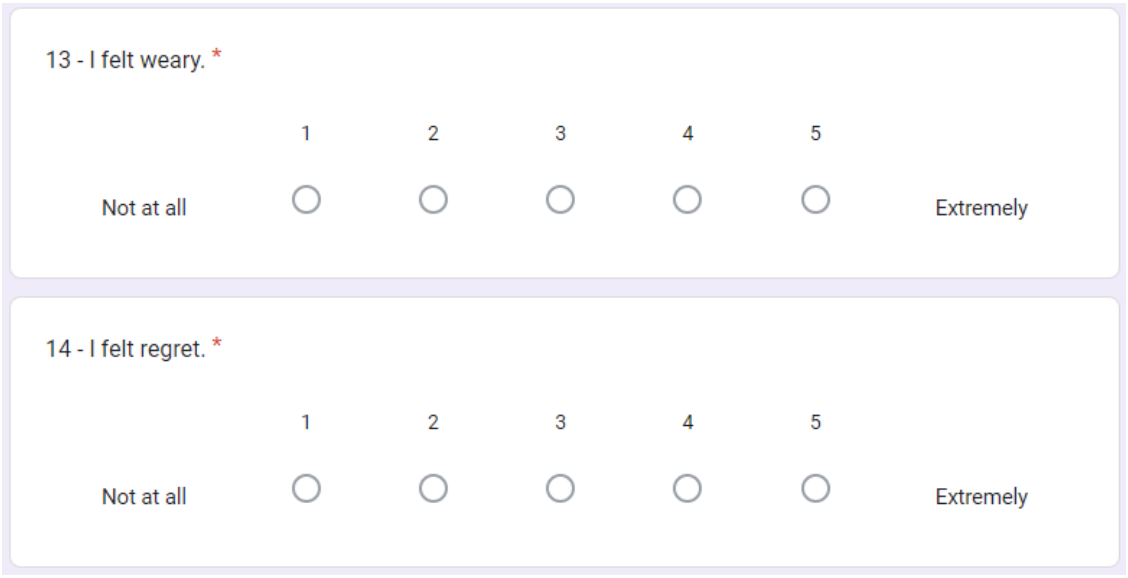

Figura A.17: Experiência do utilizador após o jogo: 13-14.

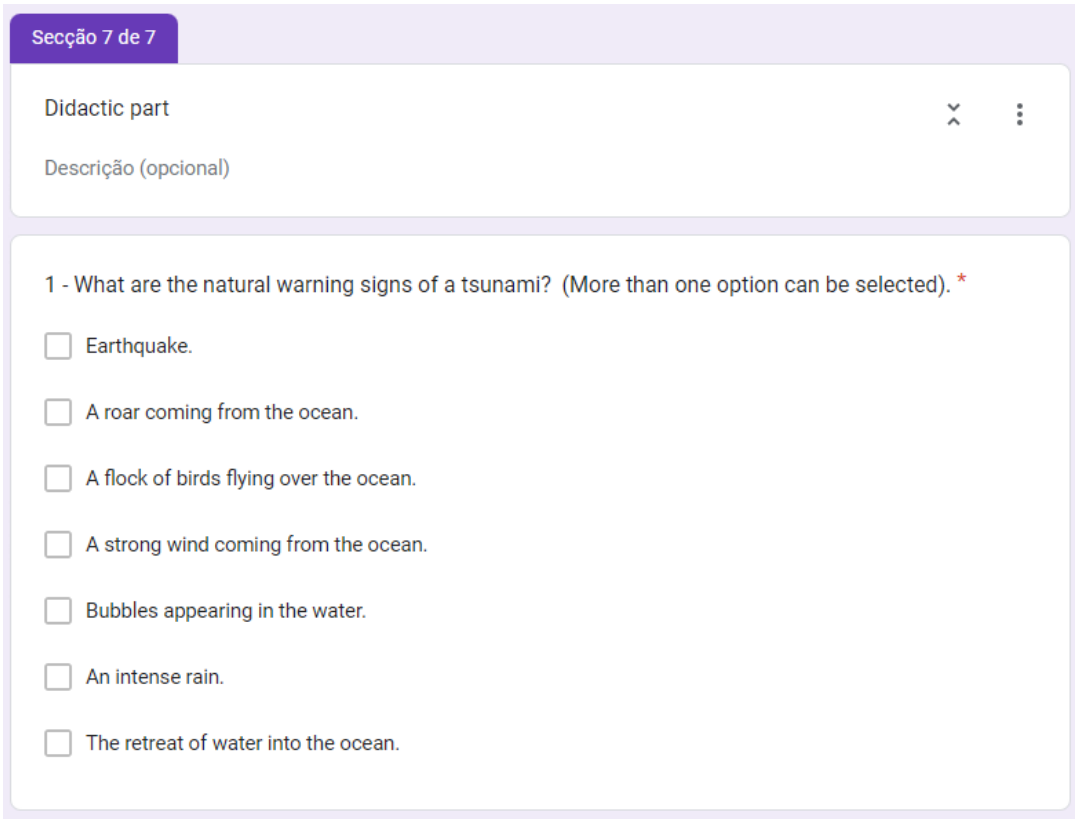

Figura A.18: Parte didática: 1

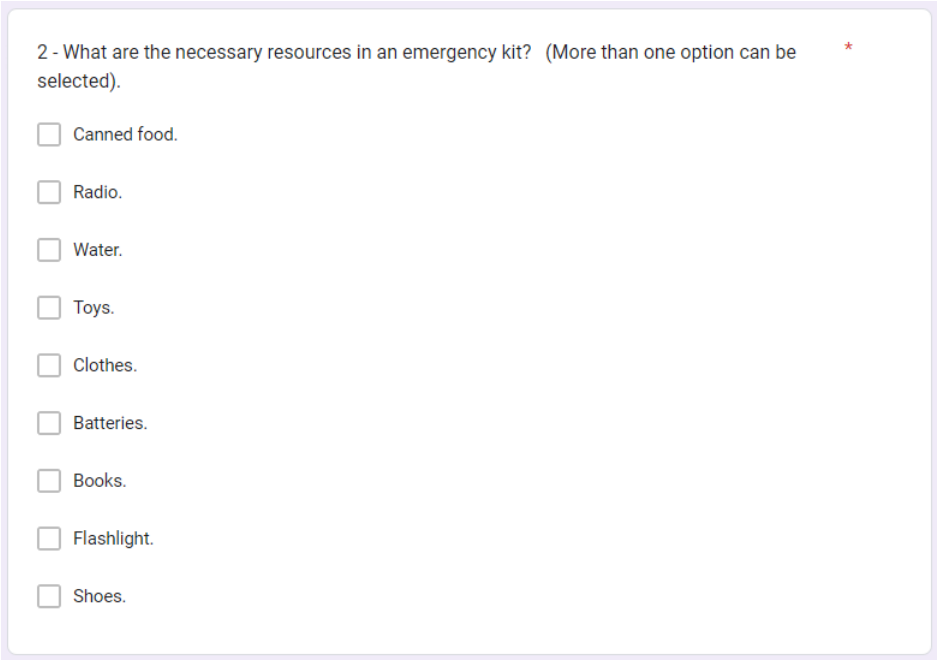

Figura A.19: Parte didática: 2

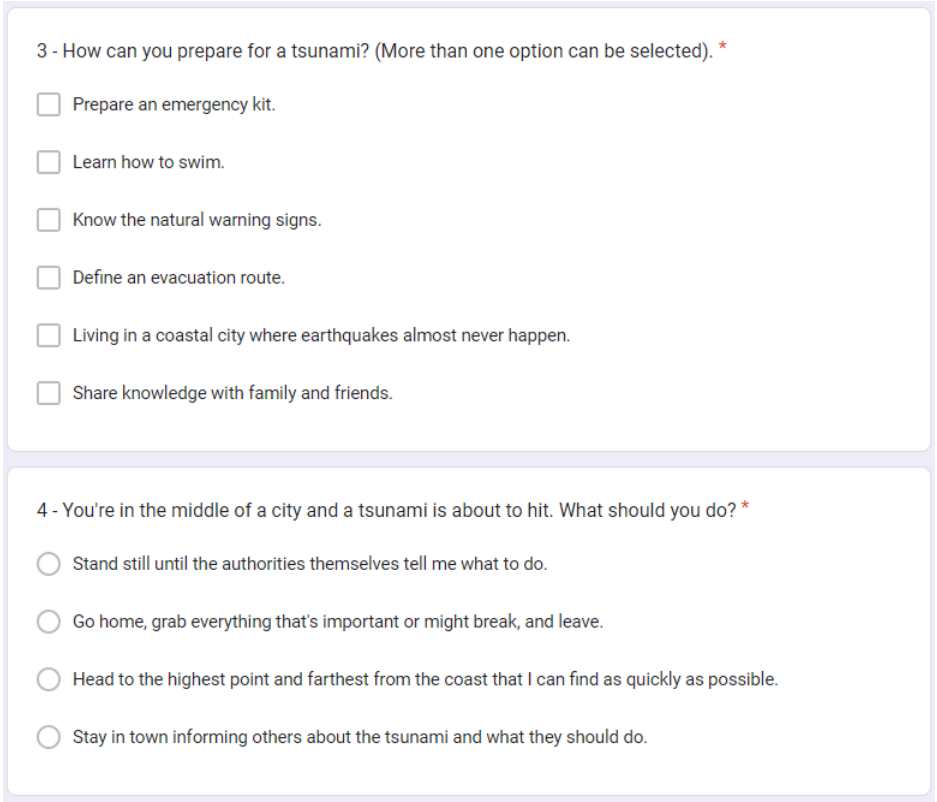

Figura A.20: Parte didática: 3-4.

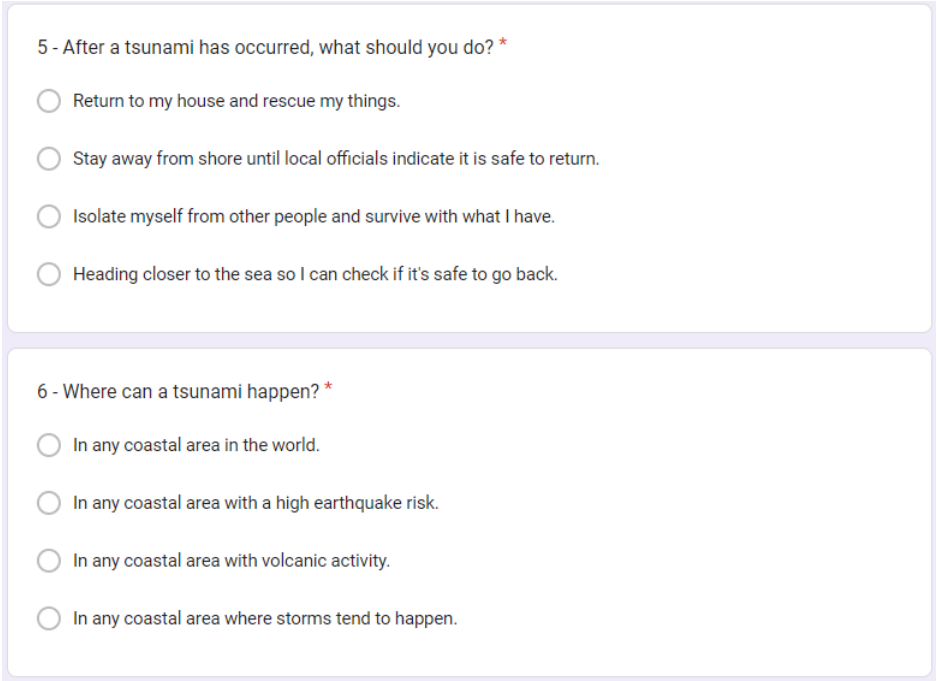

Figura A.21: Parte didática: 5-6.

B

Respostas do questionário

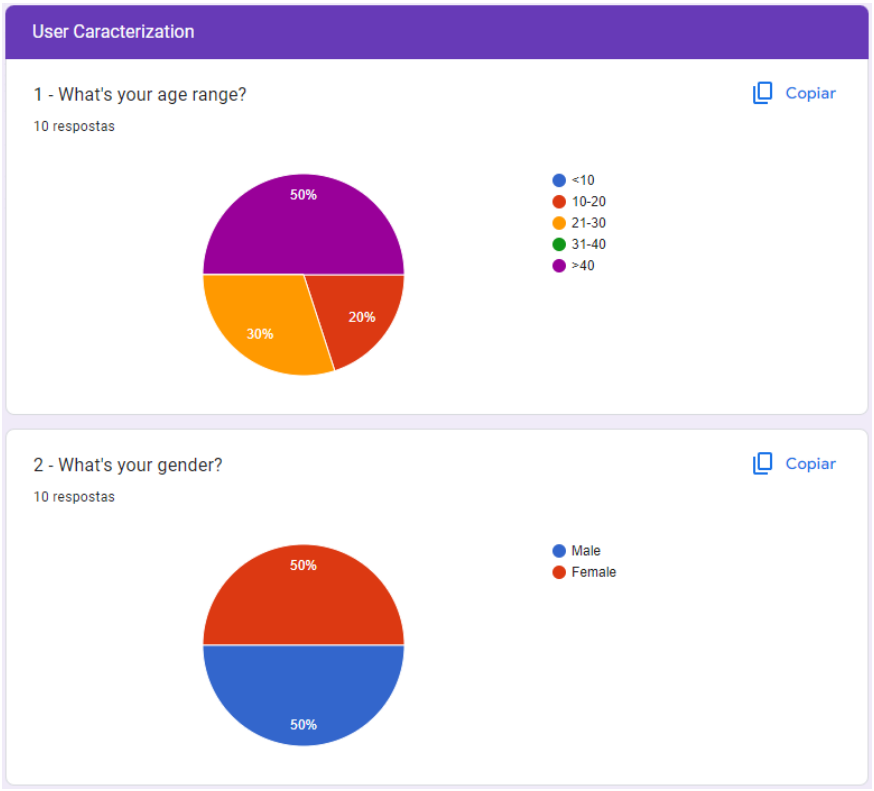

Figura B.1: Respostas da caracterização do usuário: 1-2.

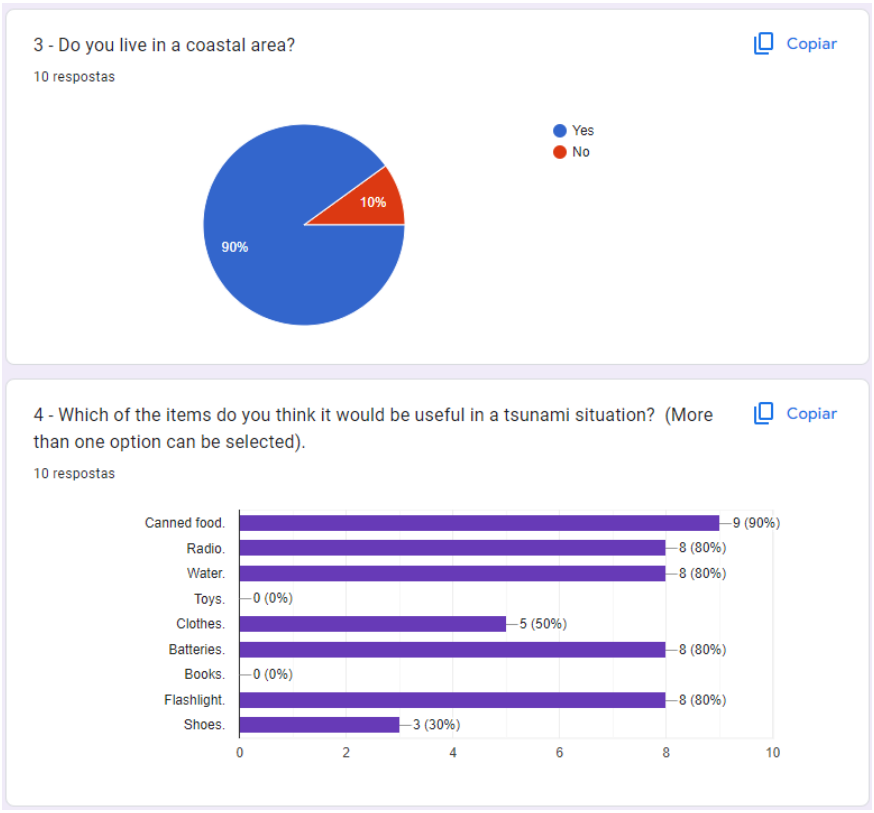

Figura B.2: Respostas da caracterização do usuário: 3-4.

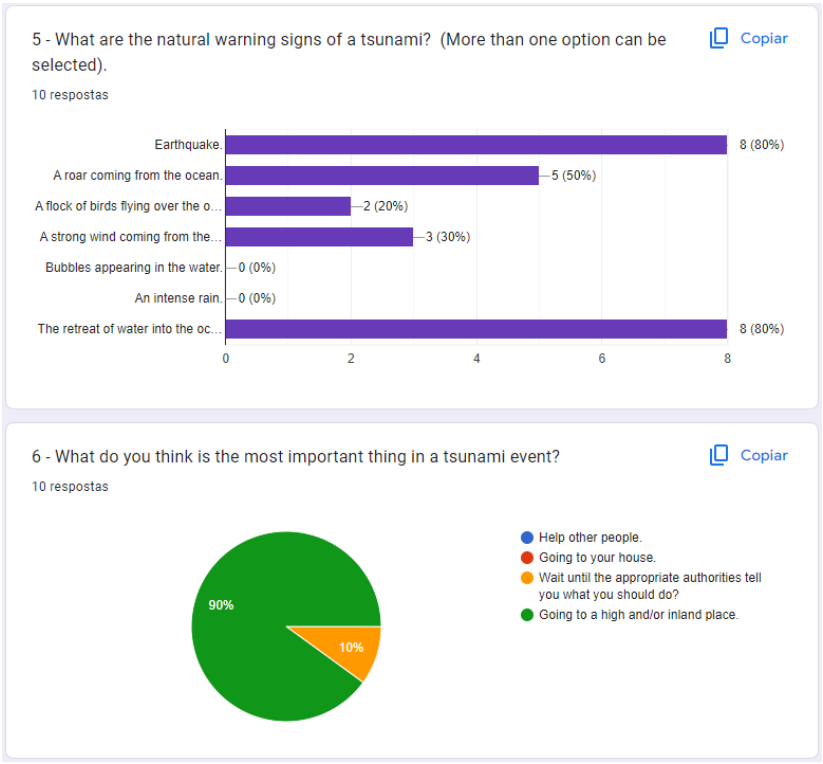

Figura B.3: Respostas da caracterização do usuário: 5-6.

Figura B.4: Respostas da usabilidade: 1-2.

Figura B.5: Respostas da usabilidade: 3-4.

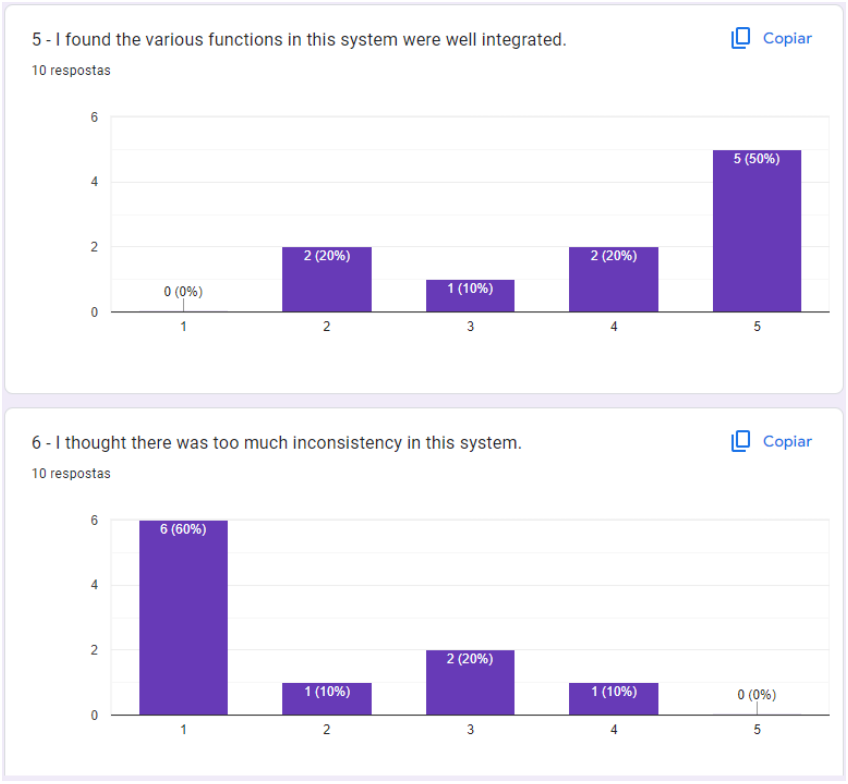

Figura B.6: Respostas da usabilidade: 5-6.

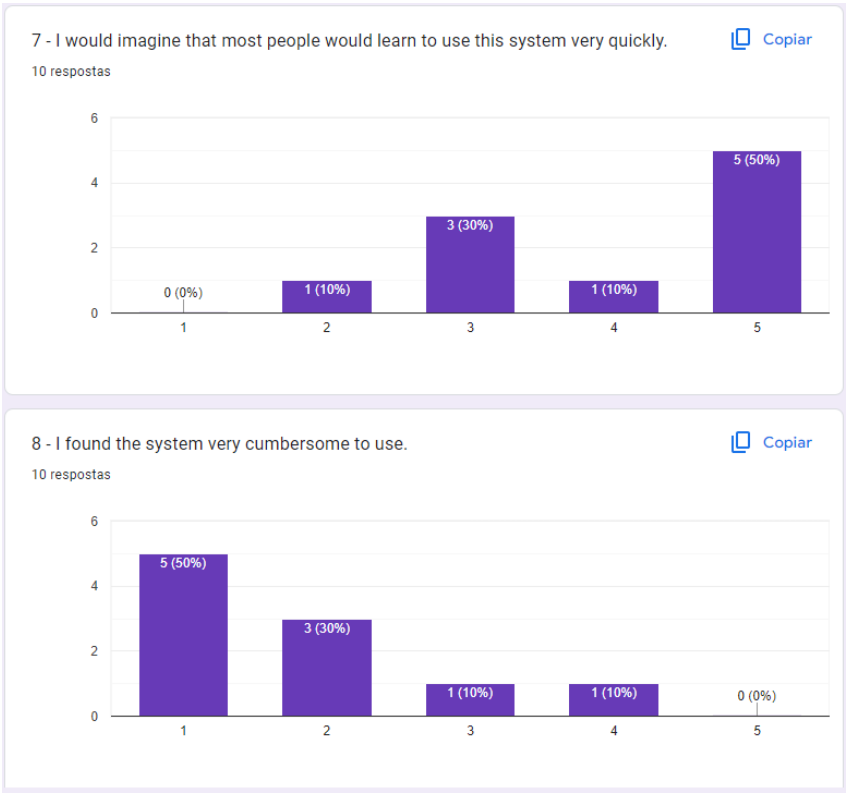

Figura B.7: Respostas da usabilidade: 7-8.

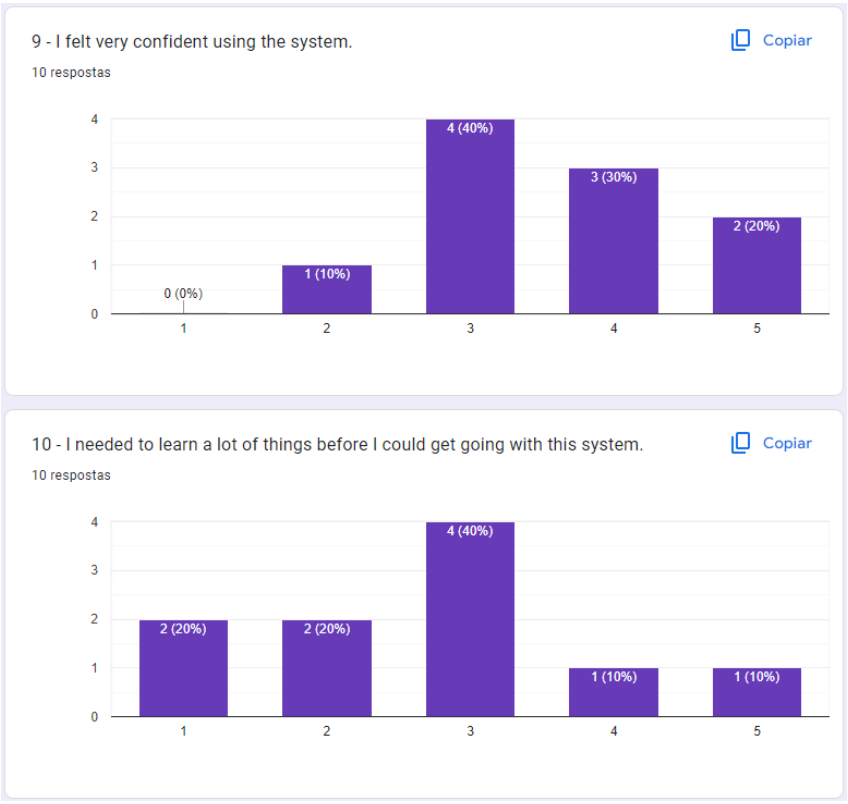

Figura B.8: Respostas da usabilidade: 9-10.

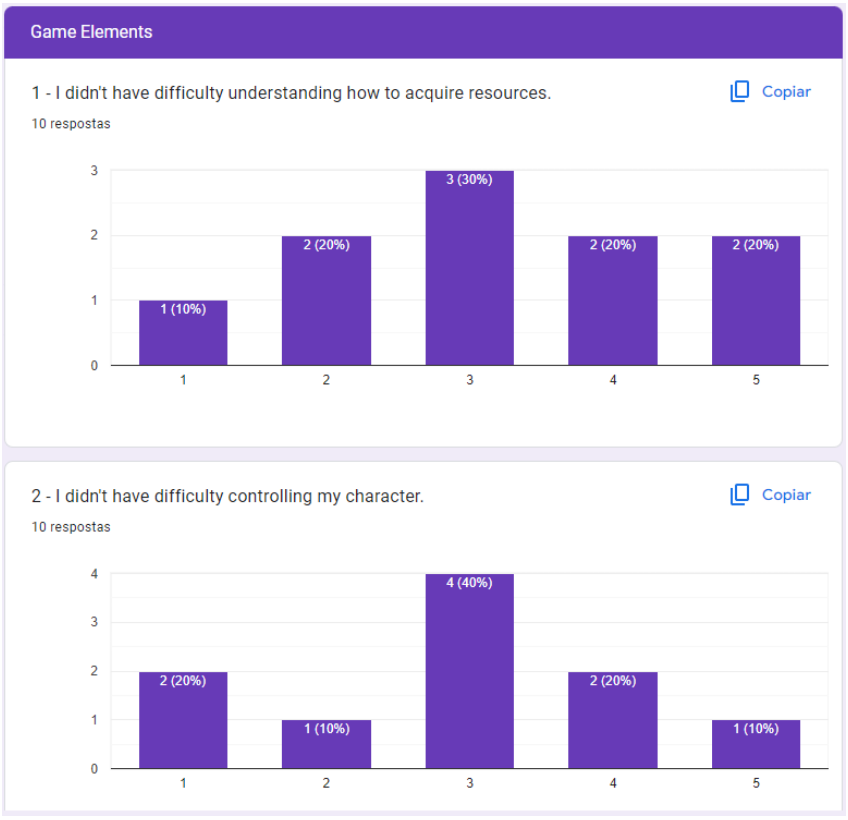

Figura B.9: Respostas dos elementos do jogo: 1-2.

Figura B.10: Respostas dos elementos do jogo: 3-4.

Figura B.11: Respostas dos elementos do jogo: 5-6.

Figura B.12: Respostas dos elementos do jogo: 7-8.

Figura B.13: Respostas da experiência do utilizador: 1-2.

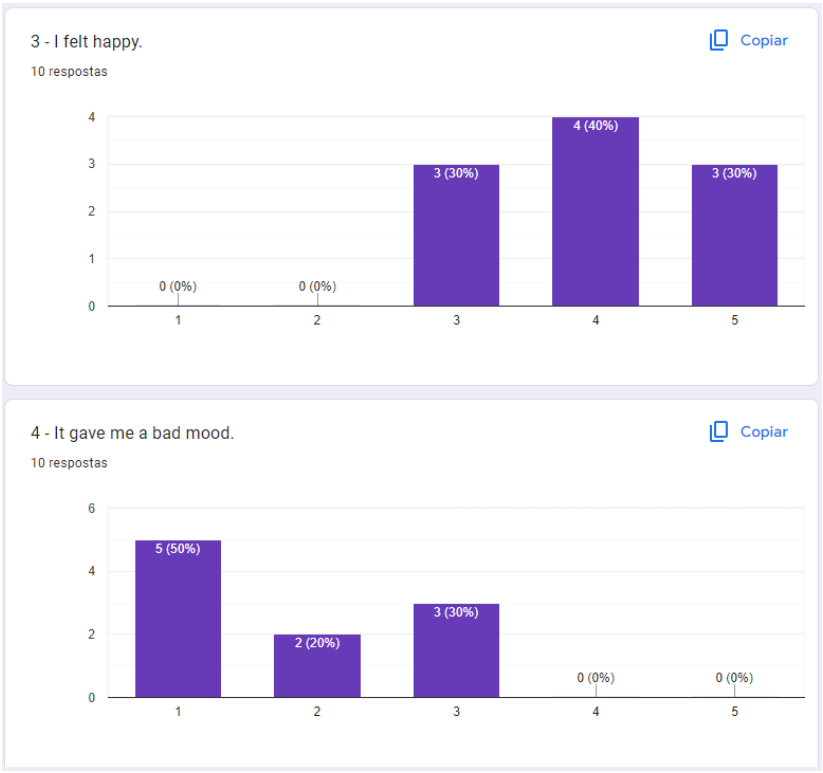

Figura B.14: Respostas da experiência do utilizador: 3-4.

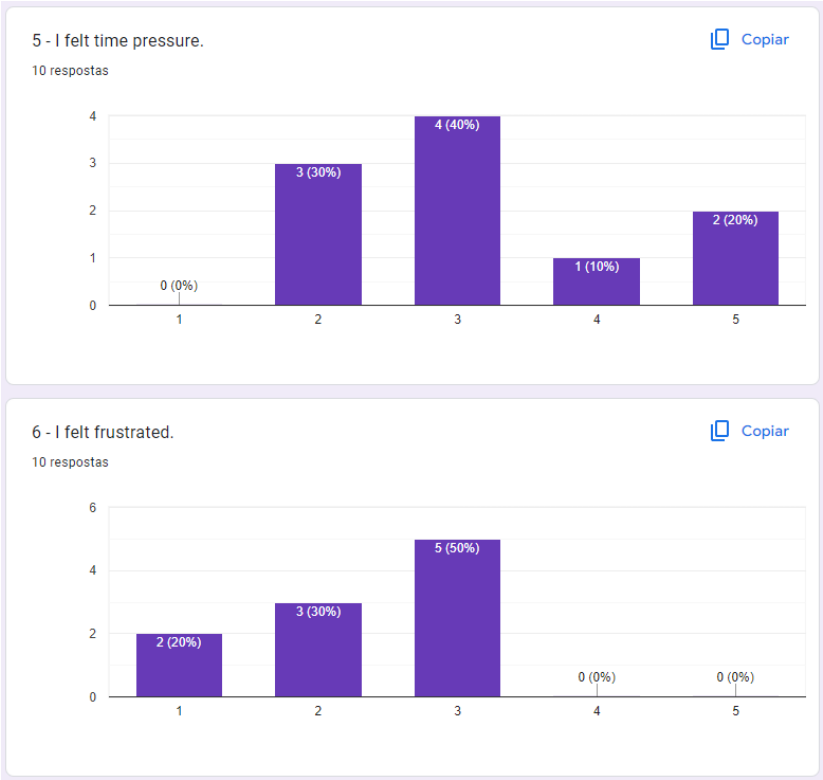

Figura B.15: Respostas da experiência do utilizador: 5-6.

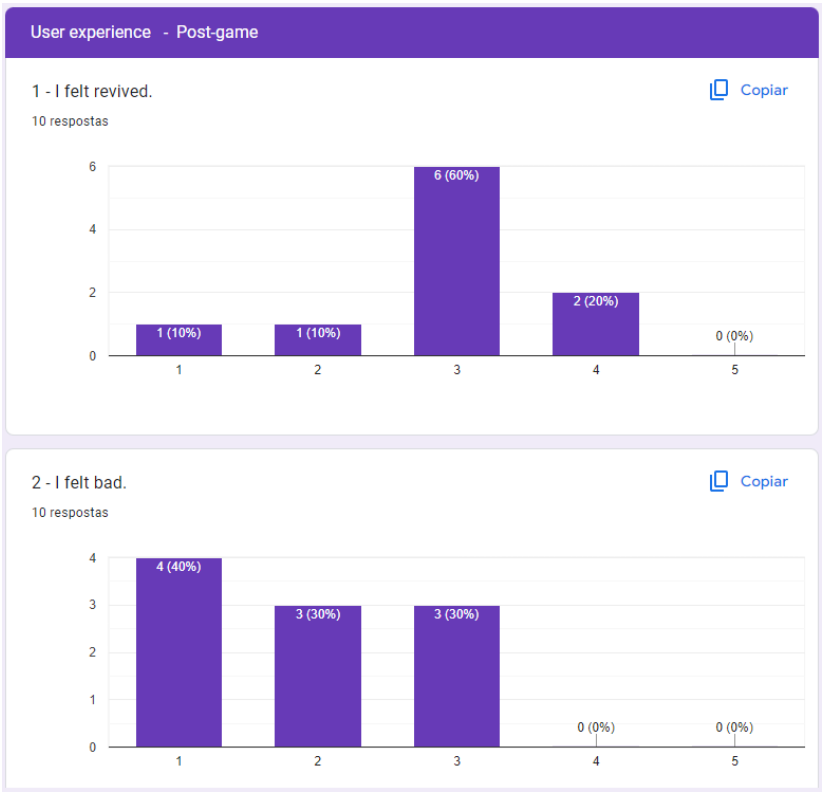

Figura B.16: Respostas da experiência do utilizador após o jogo: 1-2.

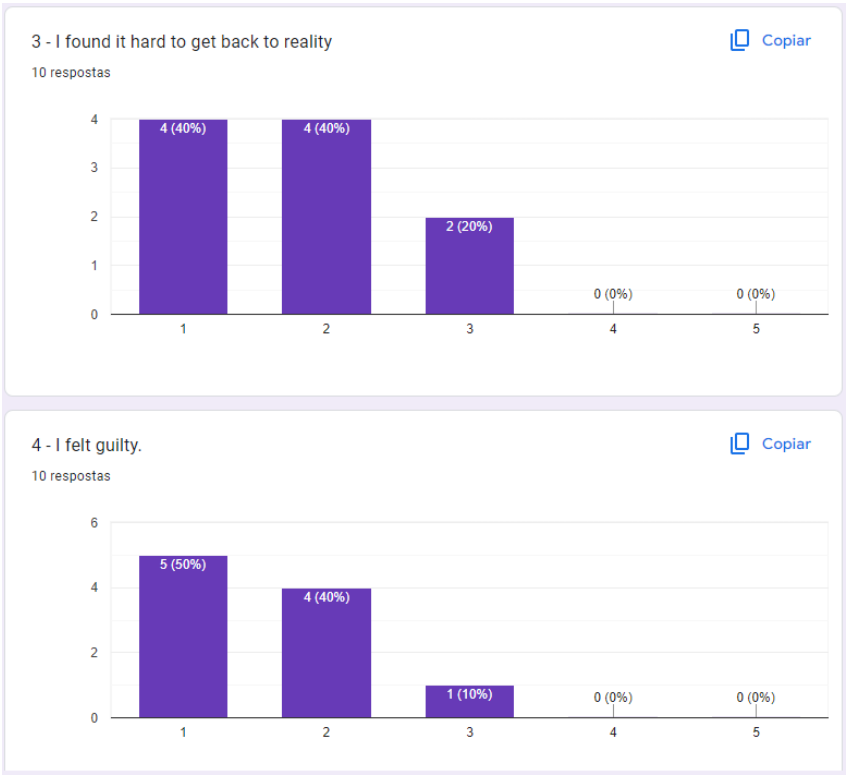

Figura B.17: Respostas da experiência do utilizador após o jogo: 3-4.

Figura B.18: Respostas da experiência do utilizador após o jogo: 5-6.

Figura B.19: Respostas da experiência do utilizador após o jogo: 7-8.

Figura B.20: Respostas da experiência do utilizador após o jogo: 9-10.

Figura B.21: Respostas da experiência do utilizador após o jogo: 11-12.

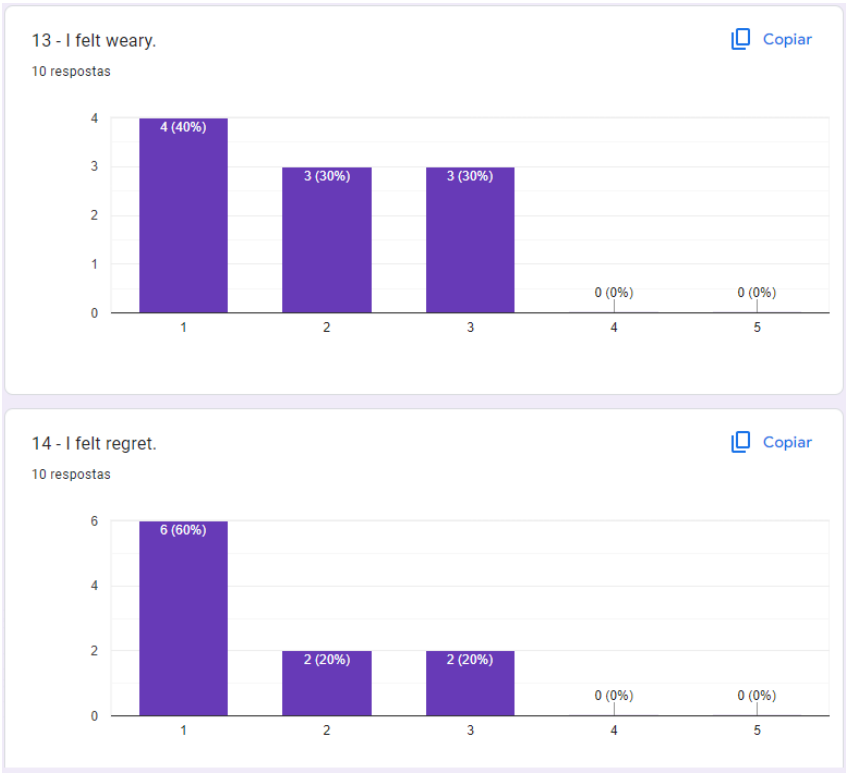

Figura B.22: Respostas da experiência do utilizador após o jogo: 13-14.

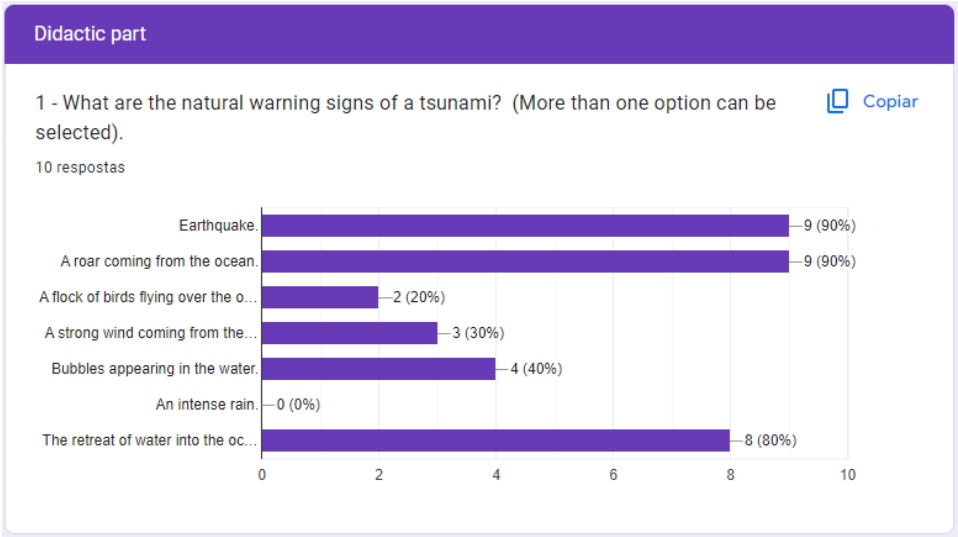

Figura B.23: Respostas da parte didática: 1.

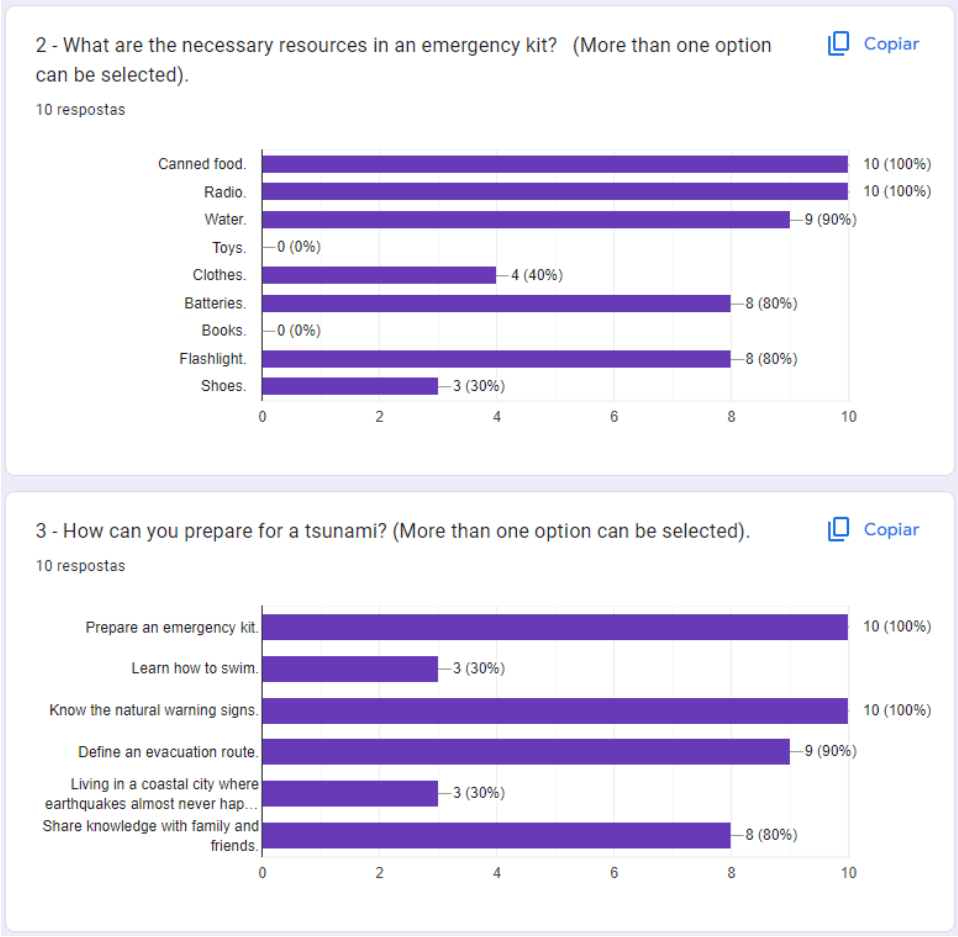

Figura B.24: Respostas da parte didática: 2-3.

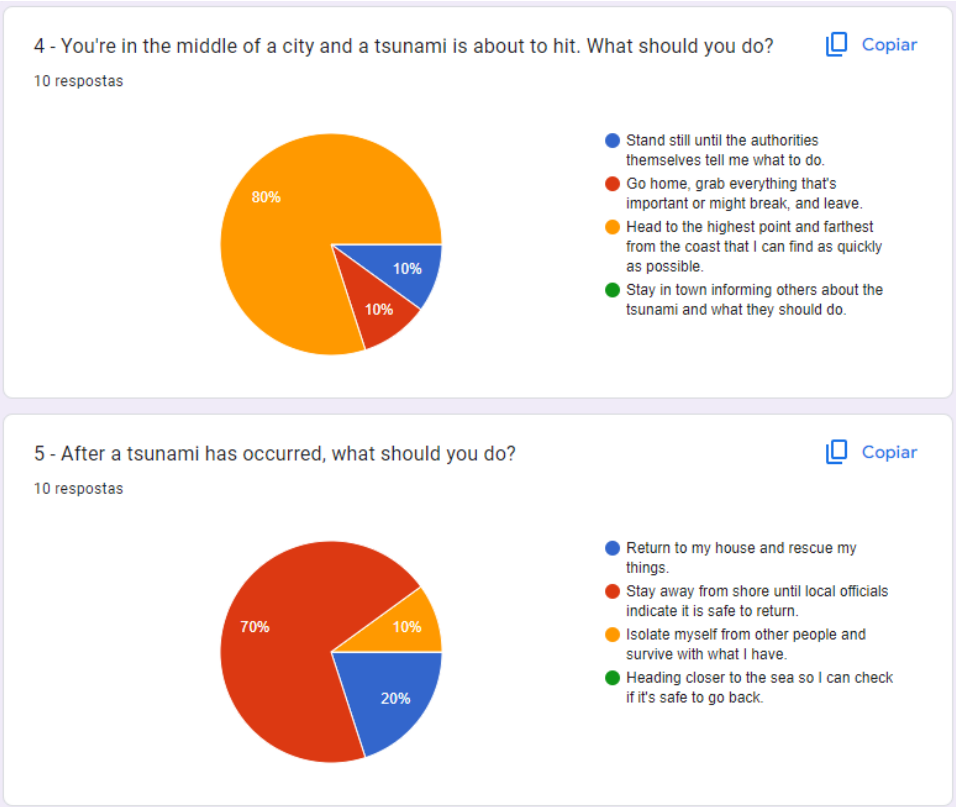

Figura B.25: Respostas da parte didática: 4-5.

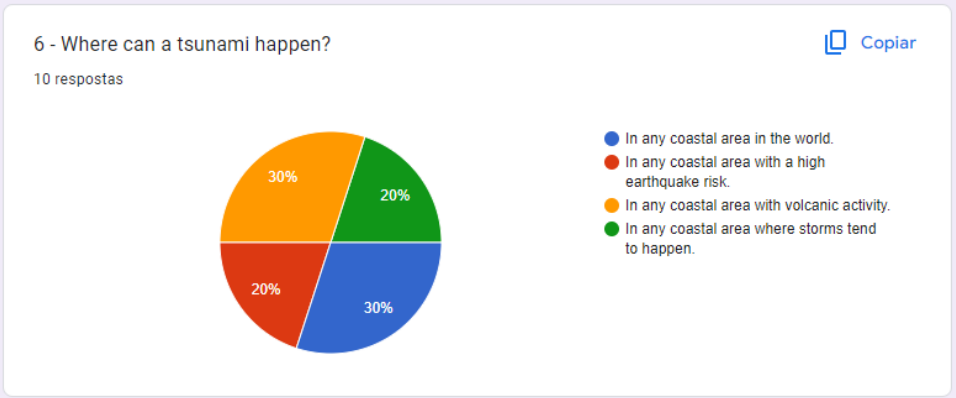

Figura B.26: Respostas da parte didática: 6.

Código

C.1 CameraShaker

```
public class CameraShaker : MonoBehaviour
{
    //Amplitude do ruído de Perlin.
    [SerializeField]
    private float amplitudeGain = 1;
    //Frequência do ruído de Perlin.
    [SerializeField]
    private float frequencyGain = 1;

    //Câmera virtual.
    CinemachineVirtualCamera vcam;
    //Canal de ruído da câmera virtual.
    CinemachineBasicMultiChannelPerlin channelPerlin;

    private void Awake()
    {
        vcam = GetComponent<CinemachineVirtualCamera>();
        channelPerlin = vcam.GetCinemachineComponent<
    CinemachineBasicMultiChannelPerlin>();
    }

    //Função para movimentar a câmera com o ruído de Perlin.
    public IEnumerator Shake(float duration)
    {
        channelPerlin.m_AmplitudeGain = amplitudeGain;
        channelPerlin.m_FrequencyGain = frequencyGain;
        float elapsedTime = 0f;
        while (elapsedTime < duration)
        {
            elapsedTime += Time.deltaTime;
            yield return null;
        }
        channelPerlin.m_AmplitudeGain = 0;
        channelPerlin.m_FrequencyGain = 0;
    }
}
```

C.2 TimeLineRandomizer

```
public class TimeLineRandomizer : MonoBehaviour
{
    //Componenete PlayableDirector.
    private PlayableDirector director;

    //As Timelines a serem escolhidas.
    public PlayableAsset[] timelines;

    //Usar a Timeline durante o Awake
    public bool playOnAwake;

    void Awake()
    {
        director = GetComponent<PlayableDirector>();
        if (timelines.Length > 0)
        {
            director.playableAsset = timelines[Random.Range(0, timelines.
    Length)];
            if (playOnAwake)
                director.Play();
        }
    }
}
```

C.3 ItemPointer

```
public class ItemPointer : MonoBehaviour
{
    //Se o recurso está disponível por padrão.
    [SerializeField]
    private bool isDefaultItem;

    //O recurso que está a ser apontado.
    [SerializeField]
    private CollectibleItemScriptableObject collectibleItem;

    public CollectibleItemScriptableObject Item
    {
        get { return collectibleItem; }
```

```
14      }
15
16      //GameObject do apontador.
17      [SerializeField]
18      private GameObject pointer;
19
20      [SerializeField]
21      //Material para quando o recurso pode ser recolhido.
22      private Material AlternateMaterial;
23
24      //Material padrão do apontador.
25      [SerializeField]
26      private Material DefaultMaterial;
27
28      //Material para quando o recurso não pode ser recolhido.
29      [SerializeField]
30      private Material ImpossibleMaterial;
31
32      //Componente para alterar a cor do apontador.
33      private Renderer pointerRenderer;
34
35      //Velocidade de rotação do apontador.
36      [SerializeField]
37      private float pointerRotationSpeed = 1f;
38
39      //Velocidade de movimento do apontador.
40      [SerializeField]
41      private float pointerMovementSpeed = 1f;
42
43      //Escala do apontador.
44      [SerializeField]
45      private float pointerScale = 1f;
46
47      //Desvio da posição inicial do apontador.
48      [SerializeField]
49      private Vector3 pointerOffset;
50
51      //Posição do apontador no eixo do y.
52      private float posY;
53
54      private void Awake()
55      {
56          if (!isDefaultItem)
57              this.gameObject.SetActive(collectibleItem.Bought);
58      }
```

```
59
60      // Start is called before the first frame update
61      void Start()
62      {
63          pointer = Instantiate(pointer, this.transform.position +
    pointerOffset, Quaternion.Euler(180, 0, 0), this.transform);
64          posY = pointer.transform.position.y;
65          pointer.transform.localScale = new Vector3(pointerScale,
    pointerScale, pointerScale);
66          pointerRenderer = pointer.GetComponent<Renderer>();
67          if (pointerRenderer == null)
68              pointerRenderer = pointer.AddComponent<Renderer>();
69          pointerRenderer.material = DefaultMaterial;
70      }
71
72      // Update is called once per frame
73      void Update()
74      {
75          pointer.transform.Rotate(Vector3.up * pointerRotationSpeed * Time.
    deltaTime);
76          pointer.transform.position = new Vector3(pointer.transform.position
    .x, posY + (Mathf.Sin(Time.time * pointerMovementSpeed) / 2), pointer.
    transform.position.z);
77
78      }
79
80      public void Reachable()
81      {
82          pointerRenderer.material = AlternateMaterial;
83      }
84
85      public void Unreachable()
86      {
87          pointerRenderer.material = DefaultMaterial;
88      }
89
90      public void Impossible()
91      {
92          pointerRenderer.material = ImpossibleMaterial;
93      }
94
95      public void PickItem()
96      {
97          this.gameObject.SetActive(false);
98      }
```

```
99
100     public void PutDownItem()
101     {
102         this.gameObject.SetActive(true);
103         Unreachable();
104     }
105 }
```

C.4 NpcObstacleController

```
1  public class NpcObstacleController : MonoBehaviour
2  {
3      //Animação do NPC.
4      private int animIDSaving;
5      private Animator animator;
6
7      //Indica se o NPC está a salvo.
8      [SerializeField]
9      private bool isSave;
10
11     //Tempo que se perde ao ajudar o NPC.
12     [SerializeField]
13     private int subtractMin;
14     [SerializeField]
15     private int subtractSec;
16
17     //Caixas de texto que aparecem em torno do NPC.
18     [Header("Particles")]
19     [SerializeField]
20     private ParticleSystem[] particleSystems;
21
22     //Variação em x das caixas de texto.
23     [SerializeField]
24     [Range(0.0f, 1f)]
25     private float xPosVariation;
26
27     //Variação em y das caixas de texto.
28     [SerializeField]
29     [Range(0.0f, 1f)]
30     private float yPosVariation;
31
```

```
32      //Variação em x das caixas de texto.
33      [SerializeField]
34      [Range(0.0f, 1f)]
35      private float zPosVariation;
36
37      //Texto que aparece nas caixas de texto.
38      [Header("LivelyChatBubbles")]
39      public ChatMouthpiece chatBubble;
40      [SerializeField]
41      private string[] chatTexts;
42
43      public bool IsSave
44      {
45          get { return isSave; }
46      }
47
48      // Start is called before the first frame update
49      void Start()
50      {
51          animator = GetComponent<Animator>();
52          animIDSaving = Animator.StringToHash("isSave");
53
54          if (particleSystems.Length > 0)
55              StartCoroutine(PlayingParticles());
56      }
57
58      public void Save()
59      {
60          if (!isSave)
61          {
62              if (chatBubble != null && chatTexts.Length > 0)
63                  chatBubble.Speak(chatTexts[Random.Range(0, chatTexts.Length
    )]);
64              animator.SetBool(animIDSaving, true);
65              isSave = true;
66          }
67      }
68
69      public void OnThankingEnd()
70      {
71          Timer.SubtractTimer(subtractMin, subtractSec);
72      }
73
74      IEnumerator PlayingParticles()
75      {
```

```
76          while (!isSave)
77          {
78              ParticleSystem particle = particleSystems[Random.Range(0,
        particleSystems.Length)];
79              if (particle.isStopped)
80              {
81                  particle.gameObject.transform.position = transform.position
        + new Vector3(Random.Range(-xPosVariation, xPosVariation),
82
                    1 + Random.Range(0f, yPosVariation),
83
                    Random.Range(-zPosVariation, zPosVariation));
84                  particle.Play();
85              }
86              yield return new WaitForSeconds(Random.Range(0.7f, 1f));
87          }
88      }
89  }
```

C.5 IUIControler

```
1   public interface IUIControler
2   {
3       public event EventHandler OnPause;
4       public void Activate();
5       public void Deactivate();
6   }
```

C.6 CollectItens

```
1   public class CollectItens : MonoBehaviour
2   {
3       //Tamanho padrão do inventário
4       private int maxItens = 3;
5
6       //Lista com os recursos dentro do alcance.
7       List<ItemPointer> inReachItens;
```

```
8       //Lista dos recursos coletados.
9       List<ItemPointer> itens;
10
11      //NPC dentro do alcance do Player.
12      NpcObstacleController npcObstacleController;
13
14      //O gerenciador de recursos.
15      [SerializeField]
16      private ItensManagerScriptableObject itensManager;
17
18      // Start is called before the first frame update
19      void Start()
20      {
21          inReachItens = new List<ItemPointer>();
22          itens = new List<ItemPointer>();
23          maxItens = PlayerPrefs.GetInt("BackPack", 0) == 1 ? 6 : maxItens;
24      }
25
26      private void OnEnable()
27      {
28          //Quando um item é removido, é atiada a função RemoveItem.
29          itensManager.removeItemEvent.AddListener(RemoveItem);
30      }
31
32      private void OnDisable()
33      {
34          //Remoção do Listener.
35          itensManager.removeItemEvent.RemoveListener(RemoveItem);
36      }
37
38      private void OnTriggerEnter(Collider other)
39      {
40
41          //Colisão com obsjetos colecionáveis e NPCs.
42          other.TryGetComponent<ItemPointer>(out ItemPointer collectibleItens
        );
43          other.TryGetComponent<NpcObstacleController>(out
        npcObstacleController);
44          if (collectibleItens != null)
45          {
46              if (itens.Count < maxItens && itens.Count + collectibleItens.
        Item.Space <= maxItens)
47                  collectibleItens.Reachable();
48              else
49                  collectibleItens.Impossible();
```

```
50              inReachItens.Add(collectibleItens);
51          }
52      }
53
54      private void OnTriggerExit(Collider other)
55      {
56          //Colisão com objetos colecionáveis e NPCs.
57          other.TryGetComponent<ItemPointer>(out ItemPointer collectibleItens
    );
58          if (collectibleItens != null)
59          {
60              collectibleItens.Unreachable();
61              inReachItens.Remove(collectibleItens);
62          }
63          other.TryGetComponent<NpcObstacleController>(out
    NpcObstacleController npcObstacleControllerExit);
64          if (npcObstacleController == npcObstacleControllerExit)
65              npcObstacleController = null;
66      }
67
68      //Coletar um recurso.
69      public void PickItem()
70      {
71          if (inReachItens.Count > 0 && itens.Count + inReachItens[0].Item.
    Space <= maxItens)
72          {
73              ItemPointer collectibleItem = inReachItens[0];
74              inReachItens.Remove(collectibleItem);
75              collectibleItem.PickItem();
76              for (int i = 0; i < collectibleItem.Item.Space; i++)
77                  itens.Add(collectibleItem);
78
79              itensManager.AddItem(collectibleItem.Item);
80          }
81
82          UpdateInReachItens();
83      }
84
85      //Salvar um NPC.
86      public void SavePerson()
87      {
88          if (IsTherePerson())
89              npcObstacleController.Save();
90      }
91
```

```
92      public bool IsTherePerson()
93      {
94          return (npcObstacleController != null && !npcObstacleController.
    IsSave);
95      }
96
97      //Remover um recurso.
98      public void RemoveItem(CollectibleItemScriptableObject item)
99      {
100         ItemPointer collectibleItem = itens.Find(collectibleItem =>
    collectibleItem.Item == item);
101         if (collectibleItem != null)
102         {
103             collectibleItem.PutDownItem();
104             itens.RemoveAll(item => item == collectibleItem);
105         }
106
107         UpdateInReachItens();
108     }
109
110     //Atualizar o estado dos recursos que estão perto do jogador.
111     private void UpdateInReachItens()
112     {
113         foreach (ItemPointer collectibleItens in inReachItens)
114         {
115             if (itens.Count < maxItens && itens.Count + collectibleItens.
    Item.Space <= maxItens)
116                 collectibleItens.Reachable();
117             else
118                 collectibleItens.Impossible();
119         }
120     }
121
122     //Retornar os recursos coletados.
123     public List<CollectibleItemScriptableObject> GetCollectibleItens()
124     {
125         return itensManager.GetCollectibleItens();
126     }
127
128     //Colocar novos valores para os itens.
129     public void SetCollectibleItens(List<ItemPointer> itens)
130     {
131         this.itens = itens;
132     }
133 }
```

C.7 CollectibleItemScriptableObject

```
1  public class CollectibleItemScriptableObject : ScriptableObject
2  {
3      public enum ItemFunction { EAT, DRINK, SEARCH, HEALTH, SUPPLY,
       ENTERTAINMENT, HYGIENE};
4
5      public enum ItemType { CHARGER, ELETRONIC, NONE};
6
7      [SerializeField]
8      private Texture2D icon;
9      [SerializeField]
10     private ItemType itemType;
11     [SerializeField]
12     private ItemFunction itemFunction;
13
14     [SerializeField]
15     [Min(1)]
16     private int maxDurability;
17     [SerializeField]
18     [Min(1)]
19     private int durability;
20     [SerializeField]
21     [Min(1)]
22     private int space;
23     [SerializeField]
24     [Min(1)]
25     private int itemPrice;
26
27     [SerializeField]
28     private bool bought;
29
30     [SerializeField]
31     private string itemDescription;
32
33     [SerializeField]
34     private string itemName;
35
36     public string ItemDescription
37     {
```

```
38          get { return itemDescription; }
39      }
40
41      public int ItemPrice
42      {
43          get { return itemPrice; }
44      }
45
46      public int Durability
47      {
48          get { return durability; }
49      }
50
51      public int Space
52      {
53          get { return space; }
54      }
55
56      public Texture2D Icon
57      {
58          get { return icon; }
59      }
60
61      public ItemFunction Function
62      {
63          get { return itemFunction; }
64      }
65
66      public ItemType Type
67      {
68          get { return itemType; }
69      }
70
71      public bool Bought
72      {
73          get { return bought; }
74      }
75
76      public string ItemName
77      {
78          get { return itemName; }
79      }
80
81      private void OnEnable()
82      {
```

```
83          ResetItem();
84      }
85
86      private void OnDisable()
87      {
88          ResetItem();
89      }
90
91      public void LoseDurability()
92      {
93          durability -= 1;
94      }
95
96      public void BuyItem()
97      {
98          bought = true;
99          PlayerPrefs.SetInt(itemName, 1);
100     }
101
102     public void ResetItem()
103     {
104         durability = maxDurability;
105         bought = PlayerPrefs.GetInt(itemName, 0) == 1;
106     }
107 }
```

C.8 ItensManagerScriptableObject

```
1  public class ItensManagerScriptableObject : ScriptableObject
2  {
3      [SerializeField]
4      private List<CollectibleItemScriptableObject> itens;
5
6      [System.NonSerialized]
7      public UnityEvent<CollectibleItemScriptableObject> addItemEvent;
8
9      [System.NonSerialized]
10     public UnityEvent<CollectibleItemScriptableObject> removeItemEvent;
11
12     public void AddItem(CollectibleItemScriptableObject item)
13     {
```

```
14          itens.Add(item);
15          addItemEvent?.Invoke(item);
16      }
17
18      public void RemoveItem(CollectibleItemScriptableObject item)
19      {
20          itens.Remove(item);
21          item.ResetItem();
22          removeItemEvent?.Invoke(item);
23      }
24
25
26      public List<CollectibleItemScriptableObject> GetCollectibleItens()
27      {
28          return itens;
29      }
30
31      public void RemoveItens()
32      {
33          foreach(CollectibleItemScriptableObject item in itens.ToArray())
34              RemoveItem(item);
35      }
36
37      public void SetCollectibleItens(List<CollectibleItemScriptableObject>
        itens)
38      {
39          this.itens = itens;
40      }
41
42      private void OnEnable()
43      {
44          addItemEvent ??= new UnityEvent<CollectibleItemScriptableObject>();
45
46          removeItemEvent ??= new UnityEvent<CollectibleItemScriptableObject
        >();
47      }
48  }
```

C.9 Timer

```
public static class Timer
{
    private static float initialTime = 1;
    private static float timeInSeconds = 0;
    private static int minutes = 0;
    private static int seconds = 0;

    public static float TimeInSeconds
    {
        get { return timeInSeconds; }
    }

    public static float InitialTime
    {
        get { return timeInSeconds; }
    }

    public static void SetTime(int min, int sec)
    {
        timeInSeconds = min * 60 + sec;
        initialTime = timeInSeconds;
        minutes = min;
        seconds = sec;
    }

    public static void UpdateTimer(float timePassed)
    {
        if (timeInSeconds > 0)
        {
            timeInSeconds = timeInSeconds - timePassed > 0 ? timeInSeconds
   - timePassed : 0;

            minutes = Mathf.FloorToInt(timeInSeconds / 60);
            seconds = Mathf.FloorToInt(timeInSeconds % 60);
        }
    }

    public static void SubtractTimer(int min, int sec)
    {
        timeInSeconds = Mathf.Max(timeInSeconds - (min * 60 + sec), 0);
    }

```

```
42      public static void AddTimer(int min, int sec)
43      {
44          timeInSeconds += (min * 60 + sec);
45      }
46
47      public static string GetCurrentTime()
48      {
49          return string.Format("{0:00}:{1:00}", minutes, seconds);
50      }
51
52      public static float GetTimePercentage()
53      {
54          return (timeInSeconds / initialTime) * 100;
55      }
56  }
```

C.10 GameManager

```
1   public class GameManager : MonoBehaviour
2   {
3
4       public UILoadScreenControler loadScreen;
5       public UIGameEndControler endScreen;
6       public ItensManagerScriptableObject itensManager;
7       public bool resetValue;
8       //Flag ready controlada pelo level.
9       public bool isReady = true;
10      //Flag ready controlado pelo loading
11      private bool isLoadingReady = true;
12
13      public bool IsLoadingReady
14      {
15          get { return isLoadingReady; }
16      }
17
18      private void Awake()
19      {
20          if(resetValue)
21              PlayerPrefs.DeleteAll();
22          DontDestroyOnLoad(this.gameObject);
23          DontDestroyOnLoad(loadScreen.gameObject);
```

```
24          DontDestroyOnLoad(endScreen.gameObject);
25      }
26
27      // Start is called before the first frame update
28      void Start()
29      {
30          RemoveItens();
31          endScreen.OnPause += HandleReplay;
32          endScreen.OnExit += HandleExit;
33          loadScreen.OnAdvance += HandleOnAdvance;
34          LoadLevelBlackScreen("MainMenu");
35      }
36
37      public void LoadLevel(string levelName)
38      {
39          StartCoroutine(LoadSceneAsync(levelName));
40      }
41
42      public void LoadLevelWithWaiting(string levelName)
43      {
44          isLoadingReady = false;
45          StartCoroutine(LoadSceneAsync(levelName));
46      }
47
48      public void LoadLevelBlackScreen(string levelName)
49      {
50          loadScreen.DisableElements();
51          StartCoroutine(LoadSceneAsync(levelName));
52      }
53
54      public void ResetLoadLevel(string levelName)
55      {
56          endScreen.ResetScore();
57          RemoveItens();
58          LoadLevel(levelName);
59      }
60
61      IEnumerator LoadSceneAsync(string levelName)
62      {
63          AsyncOperation operation = SceneManager.LoadSceneAsync(levelName);
64
65          loadScreen.Activate();
66
67          while (!operation.isDone || !isReady || !isLoadingReady)
68          {
```

```
69              if (operation.isDone && isReady)
70                  loadScreen.LevelReady();
71              yield return null;
72          }
73          Time.timeScale = 1;
74          AudioListener.pause = false;
75          loadScreen.Deactivate();
76      }
77
78      void HandleExit(object sender, EventArgs e)
79      {
80          ResetLoadLevel("MainMenu");
81      }
82
83      void HandleReplay(object sender, EventArgs e)
84      {
85          ResetLoadLevel("FirstLevel");
86      }
87
88      void HandleOnAdvance(object sender, EventArgs e)
89      {
90          LoadingIsReady();
91      }
92
93      private void RemoveItens()
94      {
95          itensManager.RemoveItens();
96      }
97
98      public void LoadingIsReady()
99      {
100         isLoadingReady = true;
101     }
102 }
```