Fire Escape: Simulação baseada em jogos de escape room para treinamento de controle ou fuga de incêndio em realidade virtual.

1. Introdução

O simulador consiste em auxiliar pessoas que estão realizando treinamentos pertinentes a casos de incêndio domiciliar ou interessadas em experienciar e adquirir conhecimento sobre este cenário. Com propósito de entregar uma imersão mais fidedigna a um incêndio real, a aplicação será desenvolvida para plataformas móveis (smartphones) para trazer a experiência e pressão em uma simulação com auxílio de óculos para realidade virtual.

Este acessório óptico mencionado, tem como finalidade aumentar a imersão proporcionada pelo simulador para que a experiência do usuário possa ser mais fiel aos cenários do mundo real. Não é obrigatório o uso deste utensílio.

Esta aplicação baseia-se em jogos de modalidade *escape room*, cujo o objetivo traduz-se em resolver quebra cabeças contextualizados em algum cenário fictício ou real para concluir etapas do jogo até que o jogador consiga escapar do local. Jogos deste estilo possuem perspectiva do usuário em primeira pessoa, em outras palavras, os olhos do protagonista são os olhos do jogador.

De maneira similar, *Fire escape* incorpora elementos da modalidade acima para apresentar cenários que envolvem situações de incêndio em residências, na qual, o usuário deverá combater as chamas ou optar pela fuga para preservar sua vida. Com isto, objetos interativos dispostos no cenário terão importância para que o usuário entenda o que está no alcance para ser utilizado para controlar as chamas. No entanto, dependendo do utensílio usado, poderá agravar a situação ou ajudar no combate aos perigos.

Em adição, o simulador oferece a possibilidade do usuário experienciar situações de perigo diversas vezes. Com intuito de educar e enriquecer conhecimentos pertinentes a cenários de incêndio domiciliar mesmo falhando ou acertando nas decisões durante a simulação.

2. Especificação

Esta seção manifesta o modelo de processo para realização do projeto e elucida sobre as funcionalidades que o simulador deverá incorporar para que seja possível a definição dos requisitos de um modo geral. Além do mais, aborda restrições de projeto para contornar as tecnologias e ferramentas a serem usadas para o desenvolvimento.

2.1. Escopo do Projeto

O modelo de processo definido para a construção do simulador foi o Evolucionário, seguindo o padrão de Prototipação. Esta preferência foi adotada pelo fato do modelo proporcionar flexibilização entre as etapas de desenvolvimento do software e mudanças conforme o lançamento de versões aprimoradas, com isto, o simulador será refinado e ajustado a cada *release* estreado. Além disso, o modelo permite afinar requisitos de sistema, visto que ao realizar a validação do software, pode ser necessário mudanças pertinentes às funcionalidades.

2.2. Elicitação de Requisitos

De um modo geral, as funcionalidades que o simulador deve conter foram levantadas de maneira informal, visando a praticidade para realizar a especificação de requisitos. A seguir, será salientado brevemente sobre as funcionalidades e características do software:

1. O simulador deverá girar em torno de cenários domésticos;
2. Deve ser possível a locomoção do usuário pelos cenários;
3. Imprescindível a interação entre objetos domésticos;
4. Necessário situar o usuário em situações de risco que envolvam incêndio;
5. Elementos que auxiliem a imersão do usuário no simulador;
6. Elementos que auxiliem o *feedback* das ações do usuário durante qualquer interação.
7. O simulador deve operar em plataformas smartphones.

Dado os detalhes acima, adiante será realizada a Especificação de Requisitos.

2.3. Requisitos Funcionais

RF01	O usuário deve navegar pelo cenário olhando para indicadores.
RF02	O usuário deve segurar objetos do cenário apenas olhando para elas.
RF03	O usuário deve utilizar objetos para interagir com outros objetos que representam perigo (tomadas, frigideiras e etc).
RF04	O usuário deve movimentar sua visão.
RF05	O simulador deve verificar se o usuário extinguiu as chamas.
RF06	O simulador deve verificar se o usuário não conseguiu conter as chamas.
RF07	O simulador deve alertar sobre o tempo restante no cenário.
RF08	O simulador deve restringir certas ações do usuário dependendo do grau de incêndio.
RF09	O usuário deve executar ação de deitar-se em certas ocasiões.
RF10	O usuário deve executar ação de efetuar ligações telefônicas.
RF11	O simulador deve possuir um sistema de partículas pertinente às chamas.
RF12	Posicionamento aleatório de alguns objetos para cada cena.
RF13	As chamas devem indicar grau de incêndio no cenário.
RF14	O usuário deverá executar ação de largar objetos em mãos.

2.4. Requisitos Não Funcionais

RNF01	Ações do usuário devem possuir animações. (Usabilidade)
RNF02	Estímulo visual ao olhar para objetos. (Usabilidade)
RNF03	Simulador deve possuir efeitos sonoros. (Usabilidade)
RNF04	Interação entre objetos devem ativar animações. (Usabilidade)
RNF05	Indicadores de tempo e saúde do jogador. (Usabilidade)
RNF06	Tela de fim de jogo ao falhar. (Usabilidade)
RNF07	Tela de fim de jogo ao conseguir combater ou fugir das chamas. (Usabilidade)
RNF08	Móveis e objetos para cada cena. (Usabilidade)
RNF09	O simulador deve operar em dispositivos móveis Android. (Compatibilidade)
RNF10	Simulador não deve apresentar quedas de performance muito intensas. (Performance)

2.5. Elucidação dos Requisitos Funcionais

- RN01: Possibilidade de se locomover pelo cenário focando a visão em indicadores visuais.
- RN02: Focar a visão para ter a posse de objetos.
- RN03: Ao ter a posse de algum objeto, o usuário poderá realizar interações com objetos domésticos que apresentam risco de incêndio.
- RN04: A movimentação da câmera (‘olhos’ do jogador dentro do simulador).
- RN05: Caso o usuário consiga extinguir as chamas em um determinado cenário, algum evento ocorre.
- RN06: Caso o usuário não consiga extinguir as chamas em um determinado cenário, algum evento ocorre.
- RF07: Algum estímulo deve ser repassado ao jogador indicando tempo restante no cenário.
- RF08: Dependendo da situação e dos níveis do incêndio, o simulador deverá limitar ações que envolvam interação com objetos em incêndio.
- RF09: Em determinados cenários, deverá ser repassada através de uma ação, impressão de que o usuário está deitado.
- RF10: Em determinados cenários, deverá existir uma opção para simular uma chamada telefônica aos bombeiros.
- RF11: Simulador de partículas que controla a quantidade de visual de chamas no cenário.
- RF12: Alguns objetos palpáveis pelo usuário podem ser posicionados em diferentes lugares pelo cenário.
- RF13: Para cada estado de incêndio, deverá ser exibido visualmente uma etiqueta afirmando o nível das chamas.

- RF14: Ao ter um objeto em mãos, o usuário poderá soltar o objeto através de algum *input* (toque na tela ou clique de mouse).

2.6. Elucidação dos Requisitos Não Funcionais

2.6.1. Usabilidade

- RNF01: Criar animações indicando *feedback* visual a partir de uma ação executada pelo usuário.
- RNF02: Alguma indicação visual para o objeto que está sendo focado.
- RNF03: Efeitos sonoros para destacar alguma ação realizada dentro do simulador.
- RNF04: Objetos em que o usuário tem posse, ao interagir com objetos que representam perigo, devem reproduzir alguma animação.
- RNF05: Indicador visual que mostra o estado de saúde e tempo restante do usuário.
- RNF06: Caso o usuário não conseguir conter as chamas, uma cena indicando a falha do usuário ao conter/fugir das chamas será exibido.
- RNF07: Caso o usuário consiga conter as chamas, uma cena elogiando os esforços do usuário será exibida.
- RNF08: Para cada cenário do simulador, deve possuir móveis e utensílios domésticos.

2.6.2. Compatibilidade

- RNF09: Ambiente de operação do simulador destina-se a sistemas móveis Android.

2.6.3. Performance

- RNF10: O simulador desenvolvido deve possuir elementos visuais controlados para que a experiência do usuário não seja afetada caso o dispositivo apresente queda de performance.

2.7. Restrições de Projeto

A seguir serão citadas as tecnologias e ferramentas para o desenvolvimento do simulador.

2.7.1. Linguagem de Programação

Linguagem nativa da plataforma Unity 3D, C#.

2.7.2. IDE

Visual Studio Code como ferramenta *CASE* para a escrita dos códigos. Vem acompanhada juntamente com a plataforma Unity 3D.

2.7.3. Game Engine e Plataforma de Desenvolvimento

Unity 3D. Atua como a ferramenta principal para o desenvolvimento do simulador. Possui recursos e funcionalidades que auxiliam na construção de cenários, bem como *assets*, pacotes que contêm modelos gráficos e recursos diversos obtidos pela *asset store*, comunidade onde estes pacotes podem ser obtidos e importados para projetos Unity.
A plataforma conta com renderizador avançado que oferece utilidades para facilitar a manipulação de elementos gráficos e suporte para processamento gráfico.
Possui *game engine* própria para auxiliar na física de aplicações 3D e no manejo de *scripts* (Classes), artefatos que definem comportamentos ou regras para cenas e objetos contidos na aplicação.
Por fim, oferece importação de projetos para diversas plataformas.

Disponível em: https://unity.com/pt

2.7.4. Audacity

Software usado para edição de efeitos sonoros.

Disponível em: https://www.audacityteam.org/

2.7.5. Github

Plataforma utilizada para a hospedagem do código-fonte e documentação.

Disponível em: github.com

2.7.6. Git

Sistema utilizado para controle de versão.

Disponível em: https://git-scm.com/

2.7.7. Doxygen

Software de documentação automática, utilizada para criar um guia explicitando as classes do projeto.

Disponível em: https://www.doxygen.nl/index.html

3. Arquitetura

Esta seção elucida brevemente sobre alguns aspectos da arquitetura de Unity, tais como, elementos que compõem um projeto na *game engine*, acesso aos dados dos objetos em cena e ciclo de vida do *script*.

Um projeto Unity consiste resumidamente de um conjunto de cenas, modelos gráficos, *scripts* e *assets*.

Cada cena possui modelos de objetos gráficos que podem ser atribuídos *scripts* , artefato que define um conjunto de funcionalidades, referências pertinentes aos atributos e informações do objeto. Com isto, é possível acessar e modificar materiais, comportamentos (física do objeto) e atributos vinculados ao objeto via código. Ademais, o *script* possibilita a transição entre cenas, interação entre objetos e entre outras opções dentro do simulador.

3.1. *GameObject*

Classe da Unity responsável por conter informações dos objetos em cena, possibilitando o acesso ou modificação de comportamentos e atributos dos objetos existentes em cena.

3.2. Função *start()*

Método inicializador da classe. O *script* ao ser invocado, executará as instruções escritas dentro desta função em primeiro lugar. Bem semelhante a um método construtor. Por padrão, cada *script* possui *start()*.

3.3. Função *update()*

Método que executa suas instruções a cada *frame* processado dentro da cena. Por padrão, cada *script* possui *update()*.

3.4. MonoBehaviour

MonoBehaviour é a classe base da qual a maioria dos *scripts* Unity deriva. Ele oferece algumas funções do ciclo de vida de código que são mais fáceis para o desenvolvimento. De um modo geral, as classes de um projeto Unity herdam MonoBehaviour para que haja um controle mais específico na execução do código usufruindo de funções associadas ao ciclo de vida do *script*. Como exemplo, as funções *start()* e *update()* fazem parte da classe citada anteriormente.

A Figura 1 esclarece um pouco sobre o fluxo de execução do código baseado no ciclo de vida de MonoBehaviour:

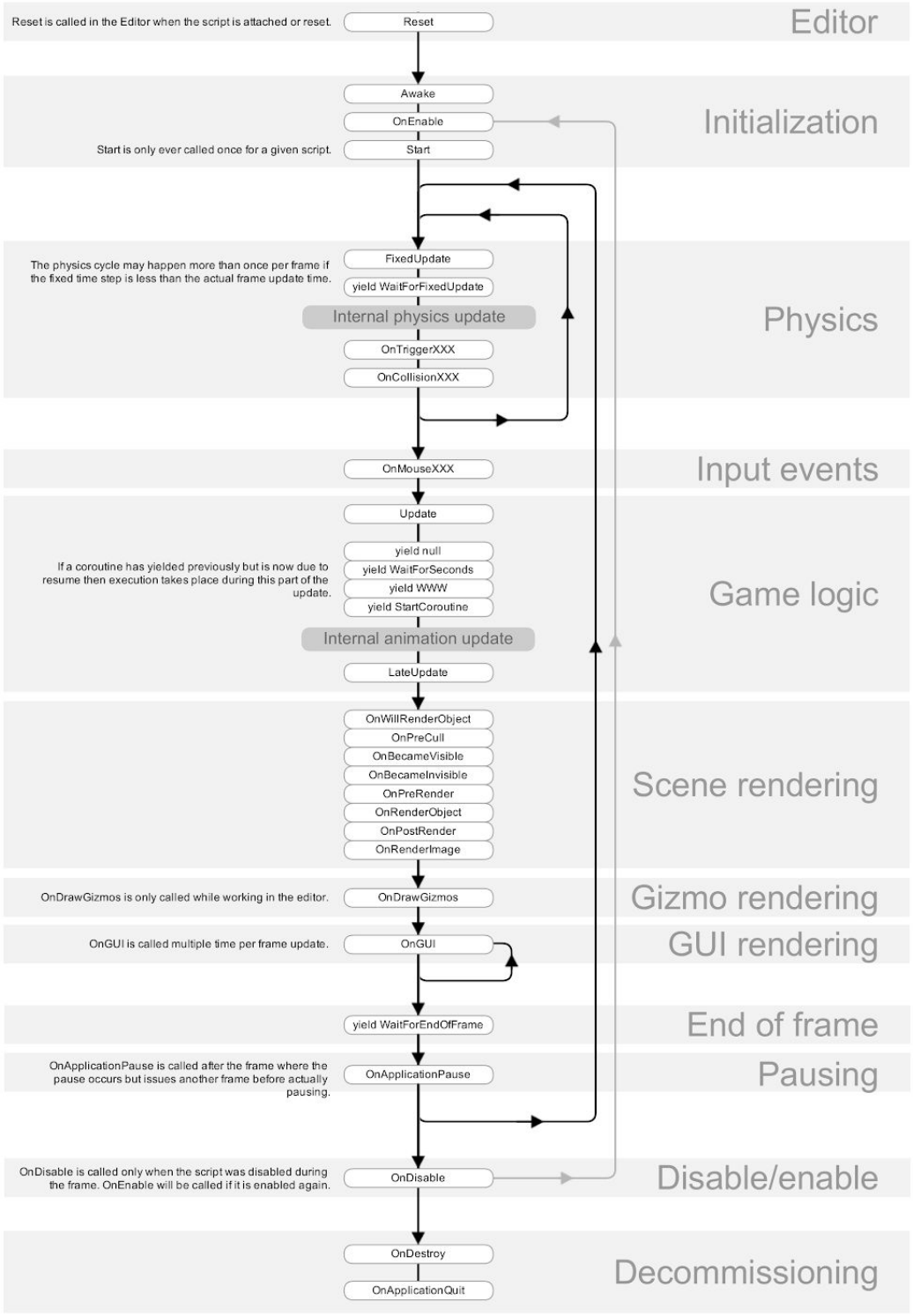

Figura 1: Ciclo de vida MonoBehaviour.

3.4. Diagrama de Classes

Abaixo, encontram-se duas figuras que representam o Diagrama de Classes do projeto. Para que fosse possível encaixar neste documento o Diagrama legivelmente, foi necessário recortar o artefato em duas metades. Em outras palavras, as Figuras 2 e 3 representam o mesmo Diagrama de Classes por completo.

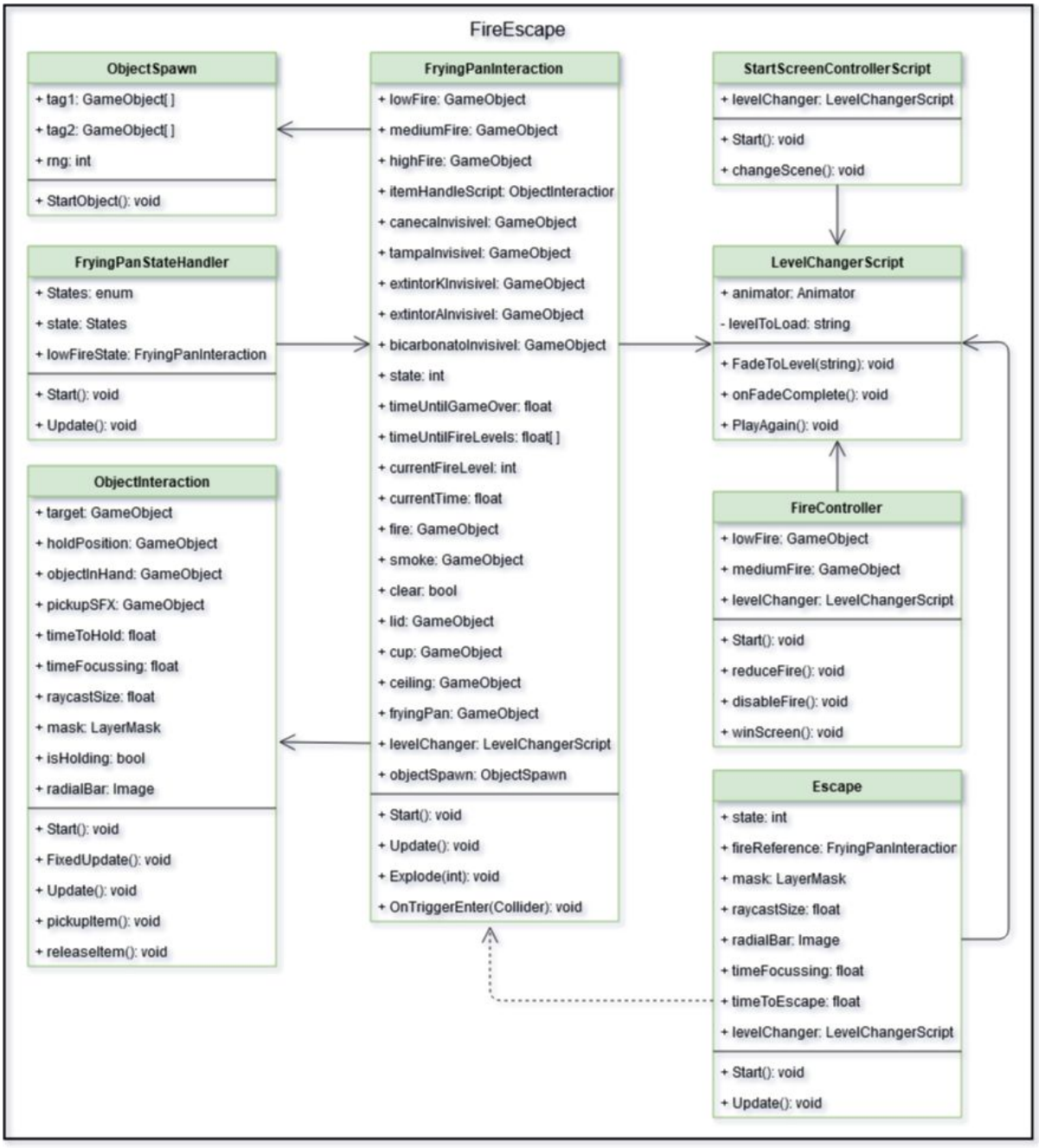

Figura 2: Primeira metade do diagrama.

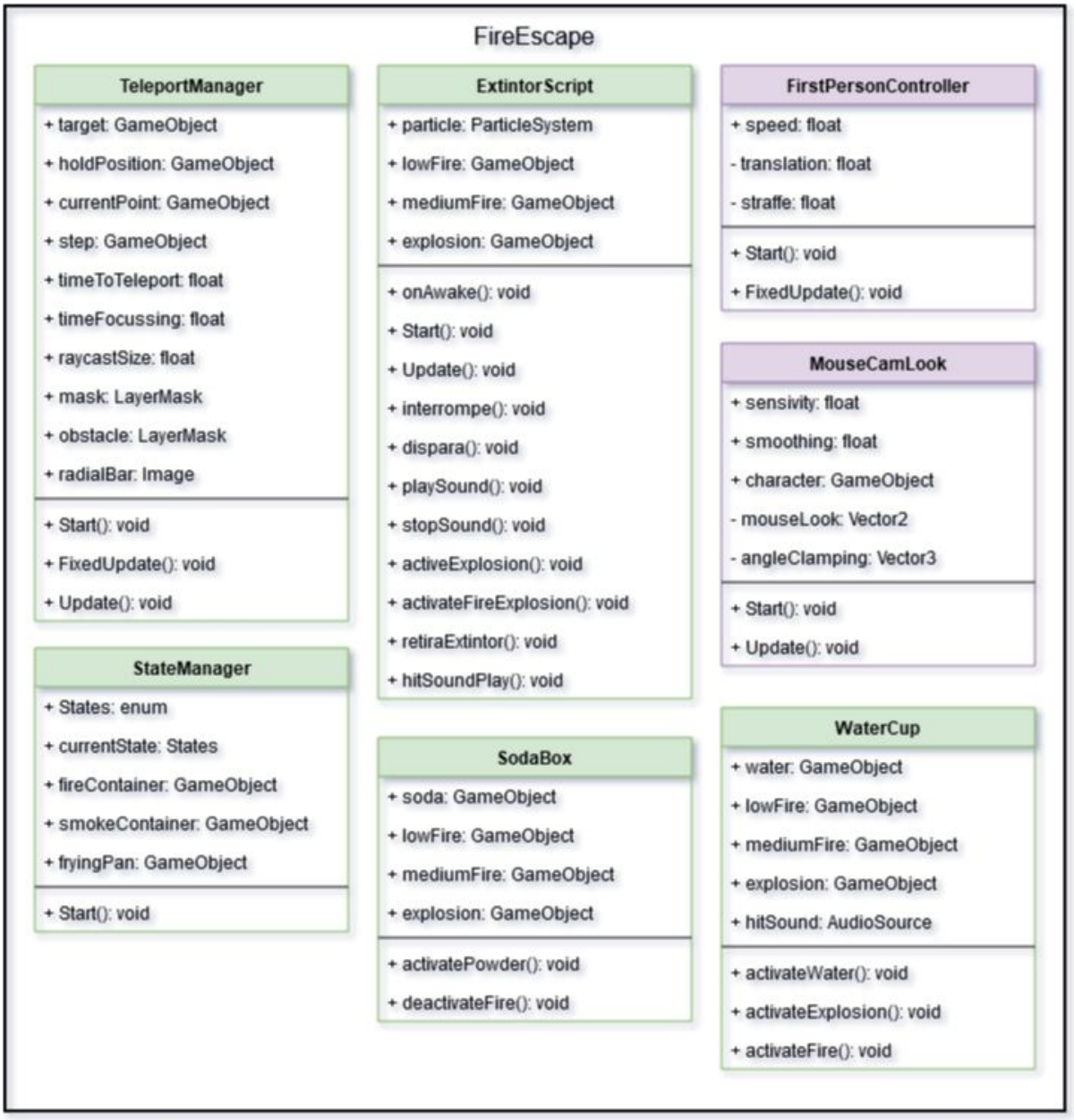

Figura 3: Segunda metade do diagrama.

As classes coloridas em lilás representarão módulos para auxiliar na testagem e adaptação para a versão desktop caso necessário.

As demais classes representarão o projeto em si, não sendo voltadas especificamente para alguma plataforma.

4. Teste de Software

Esta seção expõe critérios de teste usados e revisão sobre o andamento dos requisitos.

A aceitação do programa será realizada mediante a validação a partir de cada requisito incorporado na aplicação, verificando se as funcionalidades foram implementadas da forma esperada.

Especificamente, a metodologia consiste em implementar requisitos listados na seção 2 (especificação) e o artefato da seção 3 (arquitetura), verificando se a implementação planejada está de acordo com a descrição do requisito e validando se funcionalidade possui o comportamento esperado.

Entretanto, é bem provável o surgimento de defeitos, falhas e erros durante o desenvolvimento da aplicação. Neste caso, os problemas mencionados serão contornados via *Ad-Hoc*. Buscando extensivamente a presença de *bugs* durante a testagem das funcionalidades.

A seguir, será apresentado quais requisitos foram implementados na última revisão do projeto. Na terceira coluna, o valor ‘OK’ indica que o requisito foi satisfeito e implementado; O valor ‘NOK’ indica que o requisito não foi implementado; O valor ‘QOK’ indica que o requisito foi parcialmente implementado ou aceito.

4.1. Requisitos Funcionais

RF01	O usuário deve navegar pelo cenário olhando para indicadores.	OK
RF02	O usuário deve segurar objetos do cenário apenas olhando para elas.	OK
RF03	O usuário deve utilizar objetos para interagir com outros objetos que representam perigo (tomadas, frigideiras e etc).	OK
RF04	O usuário deve movimentar sua visão.	OK
RF05	O simulador deve verificar se o usuário extinguiu as chamas.	OK
RF06	O simulador deve verificar se o usuário não conseguiu conter as chamas.	OK
RF07	O simulador deve alertar sobre o tempo restante no cenário.	NOK
RF08	O simulador deve restringir certas ações do usuário dependendo do grau de incêndio.	OK
RF09	O usuário deve executar ação de deitar-se em certas ocasiões.	NOK
RF10	O usuário deve executar ação de efetuar ligações telefônicas.	NOK
RF11	O simulador deve possuir um sistema de partículas pertinente às chamas.	OK
RF12	Posicionamento aleatório de alguns objetos para cada cena.	OK

RF13	As chamas devem indicar grau de incêndio no cenário.	NOK
RF14	O usuário deverá executar ação de largar objetos em mãos.	OK

4.2. Requisitos Não Funcionais

RNF01	Ações do usuário devem possuir animações. (Usabilidade)	QOK
RNF02	Estímulo visual ao olhar para objetos. (Usabilidade)	QOK
RNF03	Simulador deve possuir efeitos sonoros. (Usabilidade)	OK
RNF04	Interação entre objetos devem ativar animações. (Usabilidade)	OK
RNF05	Indicadores de tempo e saúde do jogador. (Usabilidade)	NOK
RNF06	Tela de fim de jogo ao falhar. (Usabilidade)	OK
RNF07	Tela de fim de jogo ao conseguir combater ou fugir das chamas. (Usabilidade)	OK
RNF08	Móveis e objetos para cada cena. (Usabilidade)	OK
RNF09	O simulador deve operar em dispositivos móveis Android. (Compatibilidade)	OK
RNF10	Simulador não deve apresentar quedas de performance muito intensas. (Performance)	OK

5. Considerações Finais e Resultados

A versão atual do simulador é funcional e estável. Houve necessidade de criar duas *builds* da aplicação, uma para smartphones Android e outra para sistema operacional Windows, por conta da instalação do simulador em dispositivos móveis de diferentes marcas ser diversificado em certas nuanças.

Não obstante, no manual do usuário será esclarecido tais sutilezas e como proceder com a instalação do simulador em ambas plataformas.

Caso necessário, uma documentação citando classes, métodos e atributos foi gerada automaticamente pelo software *Doxygen*. O arquivo pode ser consultado dentro da pasta ‘Docs’ cujo nome é ‘index.html’

Por fim, segue abaixo imagens do simulador em execução:

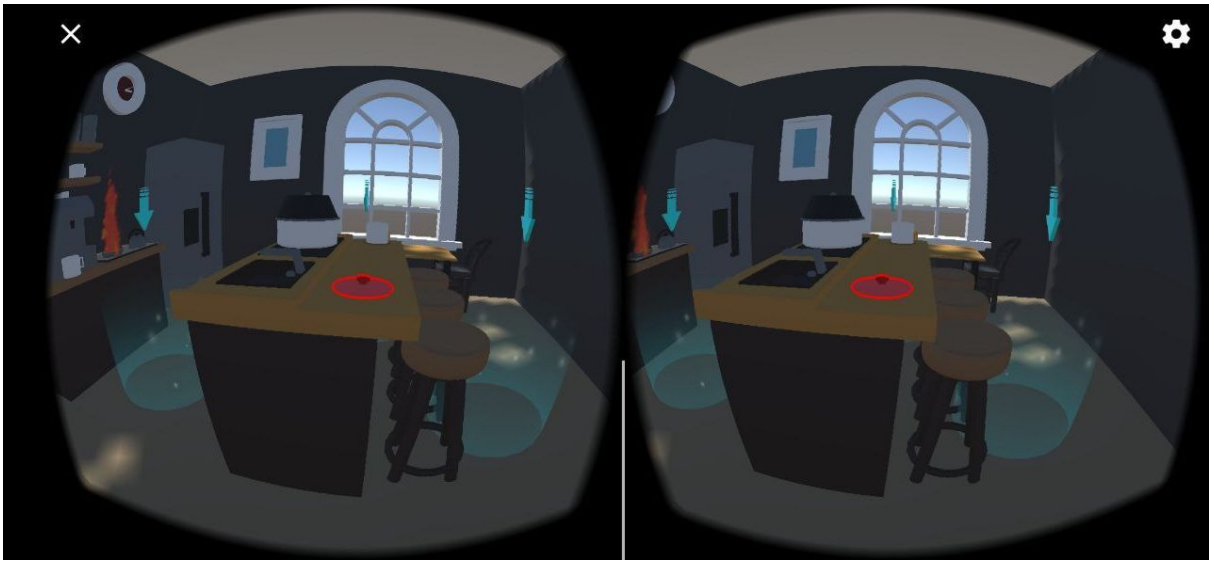

Figura 4: Usuário focando a visão no objeto. Em instantes, ele terá posse do item.

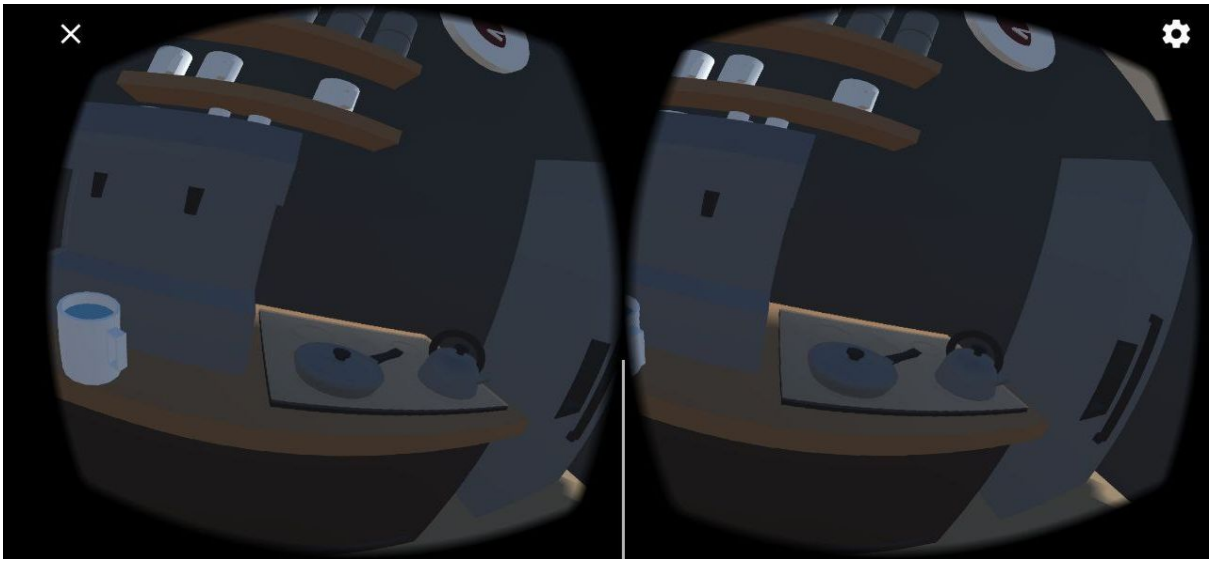

Figura 5: Com posse da tampa, o usuário interage com a frigideira apagando as chamas.