UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RICARDO DOS CASAES BELO

SegurLab 2D: Desenvolvimento de um App Educativo Multiplataforma sobre Segurança no Laboratório de Química

Monografia apresentada como requisito parcial, para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Leandro Krug Wives
Coorientador: Prof. Dr. Marcelo Eichler

Porto Alegre

2021

AGRADECIMENTOS

Agradeço aos meus pais, Sandra e Antônio Ricardo, que sempre me acompanharam e apoiaram durante toda essa longa jornada acadêmica.

À minha amada Ingrid Vieira e sua coelhinha Yue, por terem me trazido paz de espírito e tranquilidade durante períodos difíceis.

Ao Prof. Marcelo Eichler, por ter me dado apoio e esta oportunidade incrível, que veio a se tornar a realidade deste trabalho.

Agradeço, também, ao Prof. Leandro Wives, por ter me orientado nesta grande tarefa.

Finalmente, muito obrigado a todos meus amigos, professores e colegas de curso, que de uma forma ou de outra, me ajudaram a chegar onde estou.

RESUMO

Este trabalho detalha o desenvolvimento, tanto no âmbito de *design*, como também de implementação de um software/jogo educativo, voltado ao aprendizado de conceitos pertinentes à segurança de um laboratório químico. Devido à necessidade vital desse aprendizado, para estudantes de áreas relacionadas à química, foi constatado que uma forma mais acessível e lúdica ajudaria neste requisito, trazendo leveza para a assimilação do conteúdo. Para isso, foi idealizado o "SegurLab 2D", um jogo multiplataforma que providencia atividades que exercitam o conhecimento em sinalização e procedimentos de segurança em laboratório químico. A pré-produção do projeto foi feita junto com o Instituto de Química da UFRGS, na forma de um TCC em Licenciatura em Química, onde foram levantados e analisados vários elementos de conteúdo didático do SegurLab 2D. A implementação do software, documentada aqui, foi realizada tendo esta pré-produção como base, porém com alguns ajustes que se mostraram necessários durante o desenvolvimento. É mostrada uma análise das ferramentas que foram consideradas para o projeto, tendo a *engine* Renpy como escolha mais adequada, devido ao seu fácil uso e adequação aos requisitos do projeto, juntamente com experiência prévia com sua linguagem de programação, Python. Também são discutidos elementos criados na parte de *design* gráfico, além de uma breve discussão do ambiente de aprendizado para o SegurLab 2D, ou seja, na segurança do laboratório químico.

Palavras-chave: Software. Educativo. Química. Aprendizado. Segurança. Laboratório. Desenvolvimento.

SegurLab 2D: Development of a Multiplatform Educational App for Chemistry Laboratory Safety

ABSTRACT

The following work details the development, both in design and implementation, of an educational software/game that entails the learning of concepts and principles of chemistry laboratory safety. Due to this subject being vital for students in related fields of chemistry, it was found that making this content more accessible and ludic would help in its assimilation, keeping the content easy to understand in a game approach. To this end, SegurLab 2D was conceptualized, being a multiplatform game that presents ways to exercise the user's knowledge of subjects such as GHS Pictograms or NFPA 704's Safety Squares. The project, which uses Renpy as a development platform and supervised by UFRGS' Chemistry Institute, also shows how the games, originally in the physical medium, made their transition to a virtual one. The usage and modification of the tools involved is also detailed, giving credit to their simplicity in the development's work. Finally, this work will also discuss the original games in their learning environments, in other words, how they relate to laboratory safety.

Keywords: Software. Educational. Chemistry. Learning. Safety. Laboratory. Development

LISTA DE FIGURAS

Fig. 2.1 - Versão idealizada do Quiz..... 15
Fig. 2.2 - Exemplo de Versão Implementada do Quiz..... 16
Fig. 2.3 - Tela do Identificando Risco..... 17
Fig. 2.4 - Gabarito de uma Tela no Jogo..... 18
Fig. 3.1 - Jogo XeNUBi..... 21
Fig. 3.2 - Personagem principal do nível Tumbaga ……..……. 23
Fig. 3.3 - Tela de apresentação, menu inicial e níveis 24
Fig. 4.1 - Exemplo de Arte Digital do Quiz..... 31
Fig. 4.2 - Alguns Exemplos de Arte para o Identificando Riscos..... 32
Fig. 4.3 - Evolução e Variações das Imagens..... 32
Fig. 5.1 - Grafo de Estados do Quiz..... 36
Fig. 5.2 - Grafo de Estados do Identificando Riscos..... 43
Fig. 5.3 - Cartão de Instruções do Identificando Riscos…….....…….....……. 45
Fig. 5.4 - Uma das *Screens* do Identificando Riscos, com alguns elementos selecionados.. 49
Fig. 5.5 - Botão de Confirmação de Respostas, no canto inferior direito..... 51

LISTA DE TABELAS

Tabela 2.1 – Situações de Risco e Frequências …….. 18

Tabela 5.1 – Nomenclatura das Imagens do Identificando Riscos …….…….. 47

LISTA DE ABREVIATURAS E SIGLAS

GHS - *Globally Harmonized System of Classification and Labelling of Chemicals*

HOG - *Hidden Object Game*

NFPA – *National Fire Protection Association*

ABNT – Associação Brasileira de Normas Técnicas

NBR – Norma Brasileira

EPI – Equipamento de Proteção Individual

EPC – Equipamento de Proteção Coletiva

RAM – *Randomic Access Memory*

PC – *Pocket Computer*

API – *Application Programming Interface*

GML – *Game Maker Language*

GMS – *Game Maker Studio*

GIMP - GNU Image Manipulation Program

GNU - *GNU's Not Unix!*

SUMÁRIO

1 INTRODUÇÃO 11

2 O QUE É O SEGURLAB 2D 13

2.1 Requisitos do Projeto 13

2.2 Quiz 14

2.3 Identificando Riscos..... 16

3 REVISÃO BIBLIOGRÁFICA 20

3.1 XeNUBi: Jogo da Tabela Periódica 21

3.2 Ferreiros e Alquimistas 21

3.3 Quizmica 23

3.4 Resultados 24

4 ESCOLHAS DE DESIGN 26

4.1 Escolha da *Engine* para o Projeto 26

4.1.1 *Unreal Engine 4 e Unity* 27

4.1.2 GameMaker Studio 28

4.1.3 Renpy 28

4.1.4 Godot 29

4.2 Retrospectiva da Escolha da *Engine* 30

4.3 Produção e *Design* da Arte 31

5 IMPLEMENTAÇÃO 34

5.1 Organização de Arquivos no Desenvolvimento 34

5.1.1 Implementação no Renpy 35

5.1.1.1 Implementação do Quiz: Inicialização 35

5.1.1.2 Implementação do Quiz: *Loop* Principal ....…….....…..……..…… …… 38

5.1.1.3 Implementação do Quiz: Exemplo da Parte Lógica de uma Questão ….....……..… 39

5.1.1.4 Implementação do Quiz: Exemplo da *Screen* de uma Questão …….....…..……. 40

5.1.2 Implementação do Identificando Riscos: Inicialização……. …...……..……... 42

5.1.2.1 Implementação do Identificando Riscos: *Loop* Principal ……..…… 45

5.1.2.2 Implementação do Identificando Riscos: Parte Lógica da *Screen* Principal ….……. 46

5.1.2.3 Implementação do Identificando Riscos: *Screen* do Jogo ……....……..… 49

5.1.2.4 Implementação do Identificando Riscos: Parte Lógica da *Screen* de Revisão ...…… 51

5.1.2.5 Implementação do Identificando Riscos: *Screen* de Revisão …….……...… 54

6 PESQUISA DE USABILIDADE ..…….……..…..…… 57

7 CONCLUSÃO …….……...……..……. 59

REFERÊNCIAS ……... ... …. ……...…...…. 61

1 INTRODUÇÃO

Ressaltar o quão indispensável a Computação é para a educação, chegando a ser um clichê, mas um clichê que deve ser destacado inúmeras vezes, tamanha é sua contribuição para a área. Seja pelo indispensável ensino do Pensamento Computacional no âmbito escolar, para que desde cedo, alunos tenham contato e domínio sobre formas informatizadas de aprendizado (KONG, ABELSON, 2019) ou pela informatização dos métodos de ensino em todos os âmbitos, inclusive no ambiente universitário.

Nas palavras de Ghensev: “Os *games* voltados à educação não são exclusivos para jovens e crianças, se bem aplicados podem ser utilizados para pessoas de todas as idades atingindo resultados satisfatórios” (GHENSEV, 2010). Nesse cenário, a inserção de elementos de jogos educativos em diversas áreas de ensino se torna muito relevante e cada vez mais acessível.

Os laboratórios de ensino e pesquisa de química apresentam riscos à saúde das pessoas que frequentam esses ambientes. Reagentes que são manuseados de forma incorreta, sem o cuidado dos riscos envolvidos com esses produtos, tornam o trabalho no laboratório mais perigoso. O não uso de equipamentos de proteção individuais, assim como, condutas inapropriadas dentro do laboratório aumentam os riscos. Ao longo da formação, a segurança no laboratório pode ficar distante do cotidiano dos estudantes, o que possibilita algum descuido em momentos cruciais, provocando a ocorrência de acidentes com danos. Um jogo educativo, que aborda o tema segurança no laboratório em um dispositivo móvel, auxiliando no permanente contato desses conteúdos com a vida dos estudantes, visto que grande parte deles usam esses dispositivos, como os *smartphones* e PCs. Os jogos permitem a criação de situações, que oferecem algum tipo de risco, mas de maneira controlada e inofensiva. Jogos com temas envolvendo segurança no laboratório, auxiliam os jogadores em como agir durante um acidente ou qual o comportamento mais correto para a prevenção de acidentes dentro do laboratório (CUNHA, 2019).

Neste trabalho, descreve-se o desenvolvimento de um *software* desse tipo, denominado SegurLab 2D, inserido na área de segurança de um laboratório químico, conhecimento que é de competência de diversas áreas acadêmicas e de uma grande importância para a formação de alunos de tais áreas. O trabalho tem destaque, embora trate de um tema educacional simples, por ser específico nesta área do ensino, utilizando uma combinação de ferramentas e métodos não encontrada em projetos similares, sendo um protótipo dessa configuração.

O objetivo deste trabalho de conclusão consiste, portanto, na documentação minuciosa de desenvolvimento do SegurLab 2D, um jogo sério que avalia o conhecimento do usuário nas áreas de segurança no laboratório de química.

Para tanto, este documento está estruturado da seguinte forma: no capítulo seguinte há uma descrição da área educacional na qual o projeto aborda, ressaltando sua importância dentro do contexto do projeto, além de uma breve descrição das atividades, que compõem o SegurLab 2D. No terceiro capítulo, são apresentados trabalhos relacionados, i.e., projetos similares para comparação e avaliação. No quarto capítulo, são abordadas as escolhas tomadas no âmbito de *design*, tanto para desenvolvimento técnico, quanto artístico. Em seguida, é feito um detalhamento completo do código desenvolvido, com o intuito de servir como material de consulta para futuro desenvolvimento do projeto. No sexto capítulo é apresentado uma pequena pesquisa de usabilidade, realizada junto ao público alvo do *software*, com seus resultados.

Para finalizar, apresenta-se o capítulo de conclusões, detalhando as experiências que ocorreram durante o projeto, além de previsões para o futuro..

2 O QUE É O SEGURLAB 2D

A área de segurança no ambiente laboratorial compreende uma grande e variada gama de diferentes competências, que permitem a um estudante ou profissional de área relacionada utilizar a instalação e seus equipamentos, minimizando riscos e acidentes, sejam estes graves ou não. A desconsideração ou abandono destas práticas em qualquer ambiente de laboratório, configura uma atitude irresponsável (ALI et al. 2017) e assim, é de prima importância a sua assimilação no aprendizado e, posteriormente, sua utilização permanente no laboratório de química.

Essas competências vão desde a utilização correta de equipamentos sofisticados como incubadoras e freezers, a sinalização de elementos e perigos associados e, até, o comportamento e atitudes humanas que devem ser praticadas ou evitadas dentro de um ambiente seguro. Dentro desse contexto, o *game* educativo SegurLab 2D contempla estes dois últimos itens, apresentando dois módulos de atividades de aprendizado.

O SegurLab 2D apresenta um questionário sobre os pictogramas GHS e NFPA 704, também conhecidos como "Diamantes de Hommel[1]". Com ambos, o aluno testa sua memória e compreensão destes dois itens frequentemente presentes num laboratório.

O software também testa o conhecimento do usuário sobre comportamentos no ambiente laboratorial, através do módulo chamado "Identificando Riscos", primeiramente concebido como uma digitalização de uma atividade didática, mas posteriormente transformado para melhor se adaptar no formato de *videogame,* através do modelo de jogo *Hidden Object*[2].

2.1 Requisitos do Projeto

A concepção do projeto teve dois requisitos: o documento inicial de pré-produção e as reuniões de equipe. Os dois principais requisitos funcionais, foram a realização dos dois módulos, Quiz e Identificando Riscos, que precisavam atender a vários mecanismos de jogo, além da implementação de seus recursos visuais e uma estrutura de interface para suportá-los, detalhados nos subcapítulos seguintes.

No escopo de requisitos não funcionais, a prioridade seria o desenvolvimento multiplataforma, mobile e PC, com o objetivo adicional de baixos requisitos de *hardware* e

[1] Diamante de Hommel ou diagrama de Hommel é uma simbologia aplicada em diversos países que busca mostrar o nível de periculosidade dos elementos químicos presentes em um produto.

[2] É um gênero de videogame de quebra-cabeça em que o jogador deve encontrar itens de uma lista que estão escondidos dentro de uma cena.

software, visando o funcionamento no maior número possível de equipamentos e sistemas operacionais, buscando-se, então, o desenvolvimento nas plataformas Windows, Linux e Androide. Ao mesmo tempo, tornou-se necessário ferramentas de desenvolvimento que atendessem a essas necessidades e à limitação financeira do projeto. Detalhes no capítulo quatro.

2.2 Quiz

Esta atividade assemelha-se com perguntas padrões de múltipla escolha, aplicadas em provas e testes, com o foco na memorização e associação dos pictogramas apresentados. Cada pergunta é apresentada na forma de um cartão, numerado e colorido de acordo com a sua dificuldade e tema de pergunta. Cada cartão dá ao usuário quatro opções de resposta (ver Fig. 2.1). Para deixar a atividade mais lúdica, foram colocados *easter eggs*[3], na forma de referências à cultura popular, como séries de TV, *Anime*[4], filmes e etc.

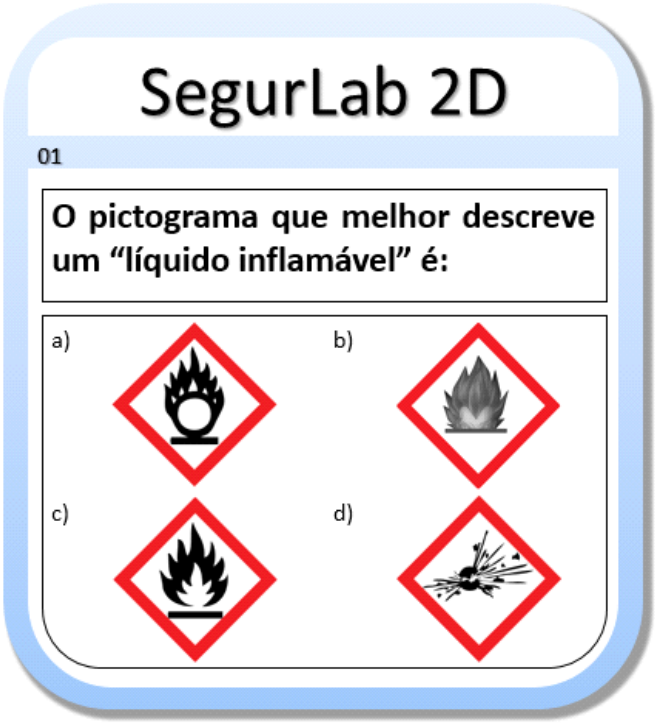
O objetivo do jogador é responder a todas as perguntas corretamente. É dado um tempo limite para cada cartão, baseado na dificuldade previamente escolhida e ele é pontuado pelo número e rapidez nos seus acertos. Ao final um escore é apresentado, mostrando sua pontuação final, além de uma lista de erros e acertos.

Foram produzidos trinta cartões de acordo com documentos e pesquisas prévias do projeto (um Trabalho de Conclusão em Licenciatura em Química), nas quais tiveram apoio e participação de professores da Área de Segurança de Laboratório, que apontaram e ajudaram a desenvolver o questionário utilizado. Na figura 2.1 é mostrado o *mock up* de um cartão, feito durante esta fase de pré-produção, com destaque para o *easter egg* de cultura popular na

[3] Em informática, um ovo de páscoa - ou *easter egg*, tradução para o inglês, como é mais conhecido - é qualquer coisa oculta, podendo ser encontrada em qualquer tipo de sistema virtual, incluindo músicas, filmes, videogames etc. Nunca deve ser confundido com mensagem subliminar, pois esta é uma coisa que o ser humano não percebe logo à primeira vista. Este termo se dá ao fato de que ovos de páscoa contém surpresas no seu interior, sejam chocolates ou brinquedos, daí a origem da expressão para sistemas de informática.

[4] Anime ou animê (como é dito no Brasil) é o nome dado para o tipo de desenho animado produzido no Japão.

Fig.2.1 - Versão idealizada do Quiz



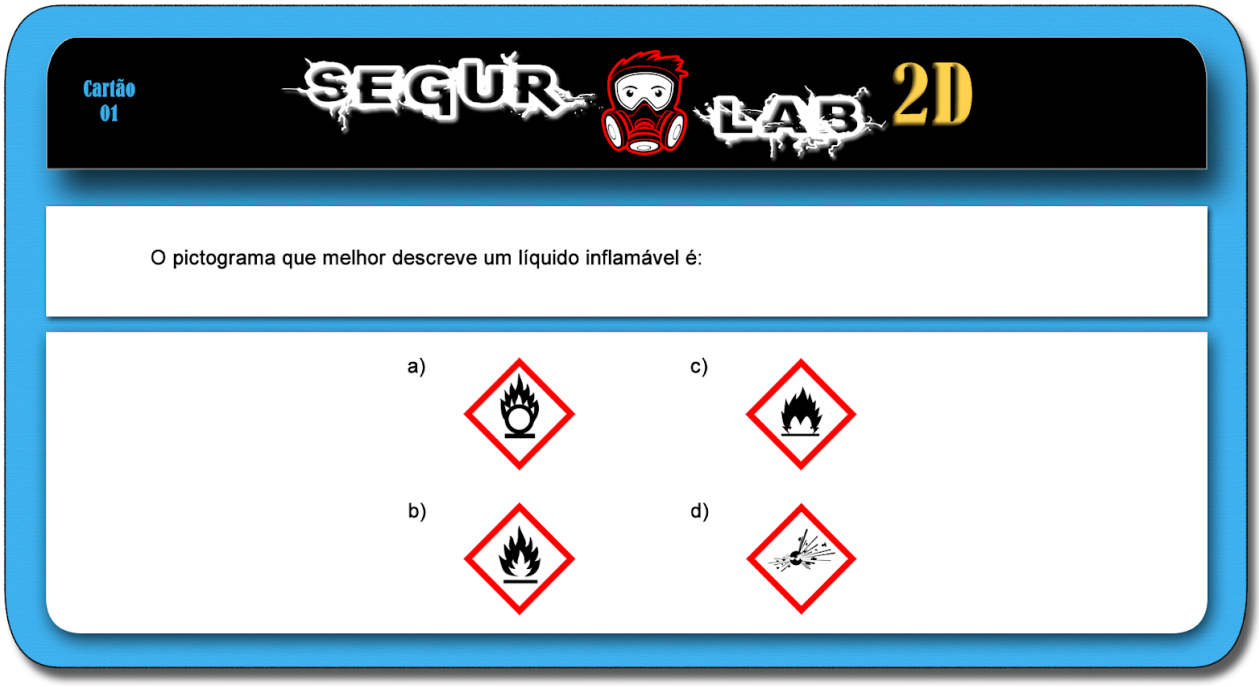
Fonte: (CUNHA, 2019)

opção B, onde a cabeça do personagem Vegeta, do anime *Dragon Ball Z,* se assemelha com o pictograma de líquido inflamável (CUNHA, 2019). O outro elemento importante são as cores de cada cartão, cada cor representa os seguintes conteúdos:

- Azul: o foco destes cartões é a identificação dos nove pictogramas de acordo com o Sistema Global Harmonizado de Classificação e Rotulagem de Produtos Químicos (GHS) e a norma da ABNT NBR 14725;
- Vermelho: além dos pictogramas, os cartões abordam as frases de perigo conforme o GHS e a norma da ABNT NBR 14725;
- Verde: nestes cartões, o jogador deverá interpretar o Diagrama de Hommel;
- Roxo: estes cartões relacionam o Diagrama de Hommel com os pictogramas.

Estes dois elementos, os *easter eggs* em alguns cartões e sua coloração foram mantidos na implementação, porém os cartões tomaram o formato padrão *widescreen*, padrão para PCs e Mobile. Abaixo, temos o exemplo do mesmo cartão, na sua versão final, onde o usuário seleciona uma das opções, clicando tanto na letra da opção ou no símbolo escolhido. Já para as questões sem imagens, o jogador pode selecionar o próprio texto da resposta.

Fig.2.2 - Exemplo de Versão Implementada do Quiz



Fonte: Própria

2.3 Identificando Riscos

Nesta atividade, o jogador irá participar num jogo estilo *Hidden Object Game,* ou seja, um *game* que apresenta uma cena, onde o usuário terá que achar elementos escondidos (Fig. 2.3). No caso do SegurLab 2D, o foco é a análise da cena apresentada e localização dos itens que apresentam perigo no ambiente laboratorial, o que o diferencia de um HOG tradicional.

Fig.2.3 - Tela do Identificando Riscos

Fonte: Própria

O objetivo do jogador é selecionar todos os itens que ele julga apresentarem riscos e após confirmar sua seleção, através de um botão. Feito isso, o jogo apresenta um gabarito da cena atual, com pontuação, mostrando todos os riscos presentes para que o usuário possa conferir seus erros e acertos. Posteriormente, ele também pode clicar nestes elementos de perigo e receber uma breve descrição de seu potencial problema para o laboratório, fixando o aprendizado.

Fig.2.4 - Gabarito de uma Tela no Jogo

Fonte: Própria

Para o Identificando Riscos foram produzidas cinco telas compostas por diversas imagens individuais, totalizando em torno de quarenta imagens diferentes. As imagens de risco foram desenhadas baseadas numa listagem, constatada no material de pré-produção (CUNHA, 2019). Essa lista, com quase 70 itens, foi montada a partir de uma pesquisa investigativa feita junto a alunos e professores de áreas relacionadas à utilização do laboratório de química, identificando as situações mais comuns de risco que são presenciadas.

Consta a seguir, a Tabela 2.1 dos itens que foram julgados de fácil visualização numa representação 2D, em função da capacidade de design da equipe em realizá-las. Essas escolhas também foram avaliadas e aprovadas pelo coordenador do projeto, juntamente com sua ordem de frequência avaliada.

Tabela 2.1 - Situações de Risco e Frequências

#	Frequência	Situação
1	6	Não usar os óculos (ou retirar para alguma observação)

2	4	Ajoelhar-se para verificar o menisco
3	4	Manipular material tóxico fora da capela
4	4	Não rotular o material
5	3	Óculos de segurança inapropriado (tamanho, desconfortável)
6	3	Óculos de proteção com a lente riscada
7	3	Não fechar frascos
8	3	Não usar EPI
9	2	Descartar luvas de forma errada
10	2	Derramar reagente e/ou não limpar a balança
11	2	Usar vidraria rachada
12	2	Derramamento sem o devido tratamento
13	2	Pipetar com a boca
14	2	Chama próxima de líquido inflamável
15	2	Solvente inflamável próximo a uma tomada, fonte de ignição
16	1	Usar vidraria lascada
17	1	Não ter EPC
18	1	Colocar um frasco perto do rosto
19	1	Pessoas com cabelos longos e soltos
20	1	Material no chão
21	1	Uso de banco ou cadeira para subir
22	1	Material dos estudantes dentro do laboratório
23	1	Vidraria (em geral) muito próxima do final da bancada
24	1	Derramamento na bancada
25	1	Rosto próximo do líquido sendo vertido
26	1	Usar celular

Fonte: (CUNHA, 2019).

3 REVISÃO BIBLIOGRÁFICA

Para a divulgação, tanto para leitores como para qualquer um interessado na produção de um trabalho de conclusão acadêmica, a revisão bibliográfica deste trabalho seguiu a máxima: “é preciso definir os tópicos chaves, autores, palavras, periódicos e fontes de dados preliminares.” (DANE, 1990).

Dentro dessas diretrizes, foram elaboradas perguntas e palavras-chaves para pesquisa de material relacionado nesta revisão. Posteriormente foram levantadas fontes de dados preliminares para análise e encontradas as referências citadas neste texto.

Perguntas-Chaves:

- Existe algum projeto semelhante ao SegurLab 2D?
- Quais *softwares* educativos existem na área de segurança de laboratório?
- Quais Técnicas de Design são usadas em um jogo educativo para nível acadêmico?
- Como se avalia um jogo/*software* educativo perante o público alvo?

Palavras-Chaves:

- Jogo
- Educativo
- Aprendizagem
- Desenvolvimento
- *Software*
- *Design*
- Química
- Laboratório

Foram encontrados vários projetos de escopo educacional na área química, como o projeto XeNUBi (EICHLER, 2013), Ferreiros e Alquimistas (EICHLER, 2018), ou o Quizmica (SANTOS et al. 2019), porém com enfoques em diferentes áreas de ensino. No caso dos exemplos, Tabela Periódica, História da Química e Radioatividade, respectivamente.

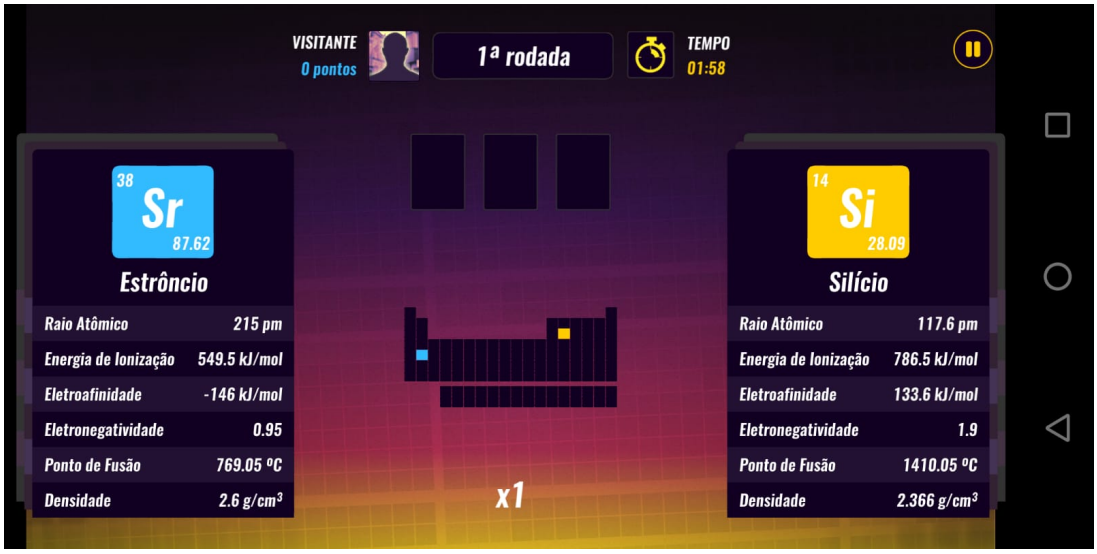
Esses projetos variam no escopo de implementação, tanto na parte de programação e ferramentas usadas, como também, em áreas externas e produção gráfica. Isso deu um caráter mais único ao SegurLab 2D.

A revisão bibliográfica, porém, veio com a grande utilidade na captação de ideias e sugestões, principalmente no período de desenvolvimento do projeto.

3.1 XeNUBi: Jogo da Tabela Periódica

O XeNUBi é um jogo educacional gratuito para telefones celulares e *tablets*, que permite ao usuário exercitar seu conhecimento, quanto às propriedades de um elemento químico. É um jogo de cartas voltado para o estudo da tabela periódica, com propostas de uso em redes sociais, na modalidade *multiplayer*.

Fig. 3.1 - Jogo XeNUBi



Fonte: Própria

O jogador deve analisar a posição dos elementos químicos e escolher qual propriedade da tabela periódica é superior à do elemento de seu oponente, o computador, tudo baseado nos conhecimentos científicos de química. Tem uma identidade visual, cujo conceito está apoiado no ambiente do estudante.

Cada jogador ganha ao iniciar o jogo cinco cartas com elementos químicos. Quando o jogador acerta a resposta, ganha uma das cartas do oponente. Quando erra, perde uma carta. O primeiro que atingir dez cartas, vence (EICHLER et al, 2013).

3.2 Ferreiros e Alquimistas

O *gameplay* de Ferreiros e Alquimistas é semelhante ao do jogo Minecraft©[5], ou seja, o jogador deve manipular itens para criar outros itens. Uma diferença é que nesse jogo o

[5]Minecraft© é um jogo eletrônico sandbox de sobrevivência criado pelo desenvolvedor sueco Markus "Notch" Persson e posteriormente desenvolvido e publicado pela Mojang Studios, cuja propriedade intelectual foi obtida pela Microsoft em 2014.

jogador tem um objetivo definido. Os níveis são compostos de vários cenários, nos quais o jogador pode navegar.

O nível Tumbaga, por exemplo, é composto pelos seguintes cenários: Casa, Mina de cobre, Rio de ouro, Plantação de milho e Criação de abelhas. O personagem do Tumbaga[6] não é exatamente um ferreiro, ele é um artesão e ourives, ou seja, trabalha com um processo de metalurgia. Os processos que ele vai aprender e os produtos que ele vai realizar, em cada etapa do jogo, são as transformações químicas. Nas próximas fases ele se tornará um ferreiro, aprendendo a produzir diversas ligas metálicas, para depois se tornar um alquimista. Haverá uma evolução do personagem no decorrer das etapas (ourives, aprendiz de ferreiro, ferreiro, aprendiz de alquimista, alquimista, mestre dos alquimistas, aprendiz de química, químico e finalmente mestre de química).

O objetivo do nível tumbaga será produzir cinco artefatos de tumbaga, que seriam progressivamente maiores e mais difíceis (com inscrições ou desenhos, o que torna a elaboração mais delicada). Alguns destes itens possuem propriedades como "durabilidade", o que significa que se desgastam com o uso. Outros dão "força" para o personagem, para que ele consiga realizar as tarefas.

Com iniciativa e idealização de Marcelo Eichler, professor do Instituto de Química da UFRGS e desenvolvido pelo NAPEAD - Núcleo de Apoio Pedagógico à Educação à Distância da UFRGS. Foi escolhido o uso da *engine* SOLPEO, pois é focada em jogos de ambiente isométrico em HTML5. A *engine* possui um conjunto de ferramentas, que roda em uma ampla gama de dispositivos, incluindo computadores, tablets e celulares, sem utilizar *plugins* (EICHLER et al, 2018).

[6] Tumbaga é o nome para a liga metálica não específica de ouro e cobre dado pelos conquistadores espanhóis aos apetrechos metálicos encontrados em uso difundido na Mesoamérica pré-colombiana e na América do Sul.

Fig. 3.2 - Personagem principal do nível Tumbaga

Fonte: (PERRY, 2014)

3.3 Quizmica

É um jogo digital do tipo ache os erros, que apresenta a aprendizagem sobre radioatividade. Elaborado com o *engine* MIT App Inventor[7], uma plataforma gratuita de desenvolvimento de jogos e aplicativos, que tem como intuito contribuir para a compreensão dos conteúdos sobre radioatividade, proporcionando uma aprendizagem mais sobre o assunto proposto, possibilitando que os estudantes façam uso do aplicativo de modo fácil e contundente. Conta com uma interface intuitiva, facilitando a utilização. Seu nome vem do acrônimo das palavras Quiz e Química.

[7] MIT App Inventor, também conhecido como App Inventor for Android, é uma aplicação código aberto originalmente criada pela Google, e atualmente mantida pelo Massachusetts Institute of Technology.

Fig. 3.3 - Tela de apresentação, menu inicial e tela de níveis

Fonte: SANTOS et al. (2019)

3.4 Resultados

Os projetos analisados foram concebidos com uma visão de um estudante/professor da área Química, apresentando algumas ferramentas diferenciadas para o desenvolvimento (MIT-Inventor e SOLPEO). Essas ferramentas, porém, não atendiam aos requisitos propostos para o SegurLab 2D, já que tinham enfoque em *Mobile* e *Web*, respectivamente. Assim, precisou-se de uma ferramenta de uso mais geral, abordado nos capítulos seguintes.

Sobre a questão de *gameplay*, os projetos apresentam comparações de jogabilidade com outros jogos similares, como Minecraft© ou o Super Trunfo. Nesse sentido, embora indiretamente, a análise destes projetos permitiu a tomada de decisão com a implementação de elementos não previstos e externos ao planejamento original do jogo. No caso do SegurLab 2D, foi a adoção do estilo *Hidden Object,* no módulo Identificando Riscos.

Quanto ao âmbito de aprendizado, nenhum dos projetos encontrados, ou aqui apresentados, tinham enfoque específico na segurança de laboratórios. Dentre os três destacados, somente o XeNUBi teve versão implementada disponível para *download*, tendo certa influência no *design* da interface, principalmente na versão *mobile* do SegurLab 2D.

Finalmente, no quesito de financiamento, todos os projetos analisados tiveram apoio de entidades de pesquisa na íntegra do projeto (CNPq e FAPERGS no caso do XeNUBi e Ferreiros e Alquimistas, por exemplo), enquanto que o SegurLab 2D teve apenas financiamento parcial de suas entidades.

4 ESCOLHAS DE *DESIGN*

No decorrer do desenvolvimento do SegurLab 2D, várias decisões de *design* foram tomadas, sendo elas tanto em áreas técnicas, visuais como educativas. Essas decisões foram tomadas dentro dos limites disponíveis, visando a viabilidade do projeto perante as dificuldades encontradas. Segue uma descrição e discussão em cada tópico.

É importante ressaltar, que o projeto foi realizado essencialmente em duas etapas, primeiramente foi feito o módulo Quiz, para em seguida ser realizado o módulo/atividade Identificando Riscos. Assim, cada um tem uma certa individualidade em vários aspectos, inclusive de como foram implementados em código.

4.1 Escolha do *Engine* para o Projeto

O SegurLab 2D tem o propósito de ensinar, através de atividades lúdicas, conceitos de segurança em laboratório. Assim, junto com uma documentação de pré-produção previamente realizada (CUNHA, 2019), foram identificadas as seguintes atividades:

- Uma seção de perguntas e respostas sobre pictogramas, Diagrama de Hommel e identificação e correlação de ambos com os riscos que apresentam.
- Um jogo de análise e identificação de imagens representando situações inadequadas no ambiente de laboratório.

Para o desenvolvimento do SegurLab 2D foi necessário a utilização de um *engine*[8] como ferramenta, visto que tal *software* permitiria facilidade para a construção dos diversos módulos que compuseram o projeto.

Como cada *engine* tem suas peculiaridades, baseado no projeto descrito do SegurLab 2D e o diálogo com o Prof. Marcelo Eichler, foram identificadas as seguintes prioridades e itens de especificação:

1. Apresentação gráfica totalmente 2D (como consta no próprio nome do *game*);
2. Necessidade lógica básica de randomização e escolhas por parte do jogador, interagindo com imagens e contando *scores;*
3. O *software* deve ter a maior longevidade possível (sem manutenção constante, devido a incompatibilidades com *software* e *hardware*);

[8] Motor de jogo, também conhecido pelo termo em inglês, *game engine*, ou simplesmente *engine*, é um programa de computador e/ou conjunto de bibliotecas, para simplificar e abstrair o desenvolvimento de jogos eletrônicos.

4. O *game* deve ter compatibilidade com PCs do Campus Universitário, não sendo necessário *hardware* mais especializado (placas de vídeo, memória RAM elevada, óculos de realidade virtual, etc.);
5. *Engine* com licença gratuita ou de código aberto;
6. Suporte a uma versão mobile como possibilidade.

Também foram propostos como requisitos importantes pela equipe de desenvolvimento:

1. A facilidade/experiência com a linguagem usada pela *engine*;
2. Suporte que ela tem perante a comunidade (para esclarecimentos de dúvidas e dificuldades que iriam surgir).

Com estes critérios, foram avaliadas as seguintes *engines*: Unreal Engine 4 e Unity, Game Maker, Renpy, Godot. Elas são detalhadas a seguir, com uma análise de prós e contras.

4.1.1 Unreal Engine 4 e Unity

Ambas as *engines* desfrutam de ampla popularidade e apoio de suas comunidades de desenvolvimento, algo muito importante já que o projeto foi realizado por apenas um desenvolvedor. Além de serem ferramentas de uso geral, sendo assim, bem flexíveis para eventualidades e mudanças de projeto, ambas possuem licenciamento gratuito para a escala de projeto do SegurLab 2D (Lee, J. 2016) (Lukosek, G. 2016). Como destaque consta a enorme lista de outros projetos bem sucedidos realizados nas duas *engines*, sendo estes muitas vezes multiplataforma (PC, Mac, Xbox, Playstation, sistemas mobile e outros).

Em resumo, os prós são:

- *Engines* de uso geral, 3D e 2D;
- Suporte para desenvolvimento para quase todas as plataformas e sistemas operacionais populares, inclusive *Xbox*, *Playstation* e *mobile*;
- Opções de licença *free*;
- Grande liberdade para mudanças e adições de *features,* à medida que forem necessárias;
- Ambas têm grande apoio de suas respectivas comunidades.

Os argumentos contrários:

- São *Engines* mais pesadas (principalmente a Unreal Engine) onde seria necessário trabalho adicional para deixá-las mais leves, certos requisitos, como *drivers* e *APIs* de vídeo (*DirectX*, *OpenGL*, etc.) atualizados seriam obrigatórios;
- Mais complexas na questão de programação (Unreal usa linguagem baseada em C++ e Unity em C#);
- Desenvolvimento multiplataforma não é automatizado. É necessário esforço adicional (e dependendo, até compra de *plugins/packages* adicionais) se for desejado que uma versão *mobile* seja feita.

4.1.2 GameMaker Studio

Prós:

- Foco para desenvolvimento de jogos 2D (em termos de mecânica e perspectiva, pois suporta renderização de gráficos e arte 3D);
- Ferramenta com suporte, robustez e farta documentação;
- Mais leve no *hardware* do que a Unity/Unreal.

Contras:

- Apenas módulos básicos são *free*, impossibilitando fácil desenvolvimento multiplataforma;
- Linguagem própria para *scripting*, a GML (embora tal linguagem não fuja dos padrões de programação);
- Vários elementos do GMS são focados para *games* de plataforma e/ou de câmera *top down,* que estão fora do escopo do projeto.

4.1.3 Renpy

Prós:

- Foco para narrativas/histórias interativas em 2D;

- Extremamente leve em requisitos de *hardware/software*;
- Código fonte aberto e totalmente *free;*
- Ferramenta para conversão automática do código para um modelo mobile Android;
- Utiliza Python como linguagem base, embora tenha sua própria API, que facilita mais ainda o desenvolvimento.

Contras:

- Impossibilidade de utilizar qualquer gráfico/arte 3D sem fazer modificações pesadas no *engine;*
- Dificuldade em trabalhar com animações. O Renpy tem foco maior em imagens estáticas, que mudam para outras imagens estáticas;
- Dificuldade em trabalhar com eventuais *features* que possam sair do escopo do atual projeto.

4.1.4 Godot

Prós:

- Foco para jogos 2D;
- Código fonte aberto e com suporte da comunidade;
- Suporte para todos os sistemas operacionais (*Mac, Linux* e *Windows*) e *mobile;*
- Gratuito;
- Supostamente fácil de aprender (utilizados em escolas como forma de aprendizado sem conhecimento prévio de programação).

Contras:

- Sem suporte para arte/renderização 3D;
- Foco da ferramenta é para jogos estilo plataforma e *point & click*, que não são o foco dos projetos;

- Será necessário montar do zero a lógica do SegurLab 2D;
- Utiliza C++/C# e sua própria linguagem de *script.*

4.2 Retrospectiva da Escolha do *Engine*

De acordo com os estudos e requisitos levantados, a *engine* "Renpy" se adequou com o projeto, visto facilidade e experiência prévia, tanto com Python como com a própria *engine*. O *software* ficou leve e acessível para PCs nos laboratórios do Campus Universitário.

Com essa escolha, o Renpy não deixou muita liberdade para adição de muita complexidade gráfica ao SegurLab 2D (animações em tempo real, objetos 3D, etc.), porém facilitou imensamente toda a lógica de menus, perguntas, respostas e interação 2D, que existe na documentação do projeto.

O Desenvolvimento do SegurLab 2D encontrou certos empecilhos no decorrer do desenvolvimento, particularmente na extensão de funcionalidades não previstas na pré-produção.

Uma das ideias propostas era a elaboração de uma ferramenta para a edição e adição dos cartões do Quiz, essa ideia se tornou difícil de ser alcançada à medida que quebraria tanto práticas convencionadas do Renpy, como também a facilidade de desenvolvimento multiplataforma do projeto. Como exemplo, funções de acesso a arquivos externos (imagens, textos e etc.) e sua manipulação iriam se tornar específicas de uma plataforma, já que acesso ao nível de Sistema Operacional seria necessário.

Outro grande empecilho veio na forma de limitações de arte que o *engine* suporta, como já mencionado na avaliação do Renpy, Arte 3D - renderização de modelos 3D, por exemplo não são suportados - isso tornou-se um problema, quando não houve a contratação de um bolsista para produção artística e foi necessário o desenho da arte 2D, ao invés de renderizá-la a partir de modelos 3D.

Ainda assim, a escolha do Renpy trouxe vários benefícios. O seu caráter e suporte a histórias/narrativas visuais pôde ser rapidamente adaptado para os requisitos do SegurLab 2D. As funções de manipulação de telas e *layers* foram mais que suficientes para tal.

Como previsto, o *engine* é bem leve em requisitos de sistema e hardware, tanto para PCs e *Mobile Android*, como também mostrando-se de fácil manuseio, para portabilidade multiplataforma.

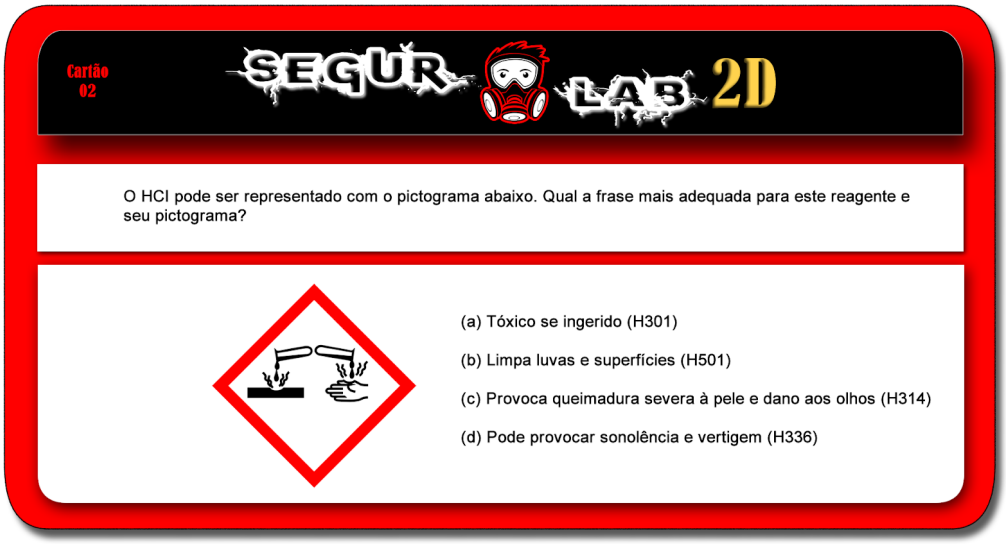
4.3 Produção e *Design* da Arte

Com o projeto sendo feito em duas etapas, a arte também seguiu esse princípio, mas aqui é importante ressaltar que era esperado um profissional (bolsista/estagiário) para realização desta tarefa, algo que não aconteceu. Portanto, a arte foi feita de acordo com os recursos disponíveis.

Foi utilizado o *software* aberto GIMP (*GNU Image Manipulation Program*) em todas as edições e manipulações de arte e imagem, não só por experiência prévia com o *software,* mas também pelo seu caráter gratuito, enquadrando-se na realidade do projeto. Igualmente ao Renpy, o GIMP também conta com uma comunidade ativa e prestativa, fator sempre importante durante empecilhos durante o desenvolvimento.

No jogo Quiz, a arte feita foi completamente digital para manter uniformidade, já que o texto das perguntas deve estar em uma fonte clara e de fácil visualização (especialmente na tela de um dispositivo mobile), o resto da arte acompanhou esta decisão.

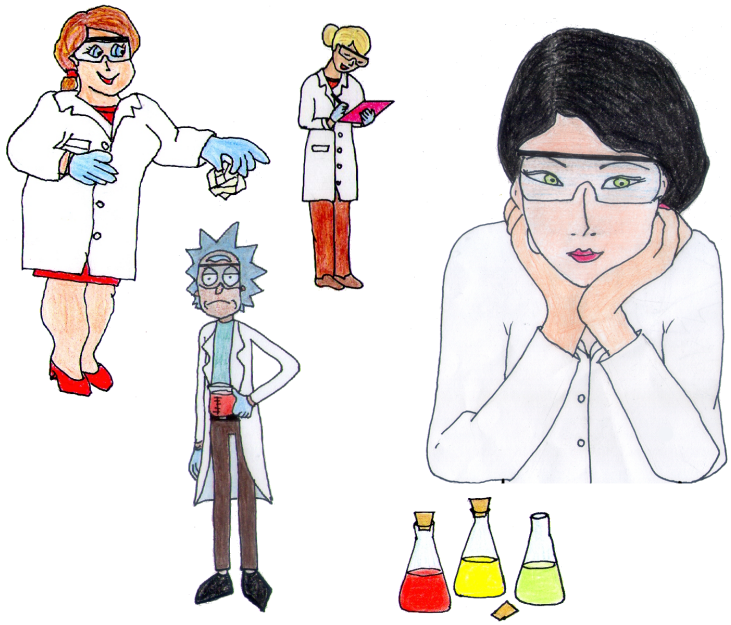
Fig.4.1 - Exemplo de Arte Digital do Quiz



Fonte: Própria.

Na atividade Identificando Riscos, devido a necessidade de apresentar cenas, objetos, pessoas em situações mais específicas e complexas, se optou por um estilo gráfico feito à mão, em parte por falta de experiência na realização destes requisitos, numa arte diretamente digital e também, por falta de material e equipamento disponível para o mesmo (ex: mesa de desenho digitalizadora).

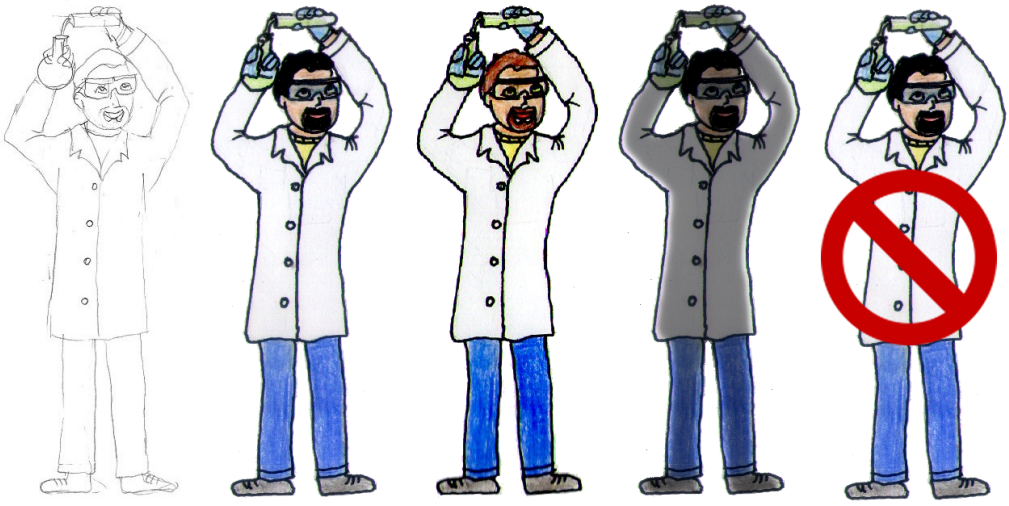
Fig.4.2 - Alguns Exemplos de Arte para o Identificando Riscos



Fonte: Própria.

A arte foi, então, realizada manualmente em papel, produzida a partir da observação de modelos artísticos online e aplicados criativamente para as cenas do jogo. Ela foi criada, primeiramente, com rascunho a lápis, passando por uma delineação com nankin, finalmente a coloração foi feita a lápis. Esta arte foi, então, digitalizada, onde retoques foram feitos (limpeza da imagem, adição de pequenos traços, etc.). O resultado final foi um estilo mais uniforme e natural, sem traços e coloração 100% limpa e lisa, dando uma certa identidade visual a esta parte do trabalho.

Fig.4.3 - Evolução e Variações das Imagens



Fonte: Própria.

Foram produzidas cerca de 40 imagens neste estilo, das quais foram criadas variações com efeitos para seleção, indicador de risco e afins. Estas variações foram produzidas digitalmente.

5 IMPLEMENTAÇÃO

O propósito deste capítulo foi fazer um detalhamento técnico sobre a implementação do SegurLab 2D, como, também, é esperado que esta informação seja de utilidade para a continuidade do projeto, a informação também fará um peculiaridades da *engine* Renpy, indo além da simples programação do jogo e entrando no detalhamento das bases/fundações necessárias, para o funcionamento do *software*.

5.1 Organização de Arquivos no Desenvolvimento

Em relação ao código fonte, foi utilizado o padrão de documentação e recomendado pelo Renpy. Temos quatro arquivos que incluem as informações e dados respectivos ao jogo. São eles: scripts.rpy, screens.rpy, gui.rpy e options.rpy. O fato desses arquivos estarem fortemente atrelados, é uma das razões pelas quais o *engine* mantém portabilidade multiplataforma automatizada.

Gui.rpy

Contém as informações textuais (fonte, tamanho, posicionamento, cores, etc.) de todo o texto gerado no jogo. Também contém caminhos para arquivos a serem usados na interface.

Options.rpy

Contém uma miscelânea de informações internas do jogo, como nome (para display pelo SO do usuário), versão, local de arquivos a serem carregados, transições de tela, etc.

Screens.rpy

Contém informações sobre os *layouts* das telas de jogo, por exemplo, a disposição de botões no Menu Principal é diferente da disposição do Menu de Opções ou da tela de jogo, seja o Quiz ou o Identificando Riscos.

Script.rpy

É onde a lógica do jogo propriamente acontece, tanto do Quiz, quanto do Identificando Riscos. Por exemplo: sequência lógica de perguntas, display aleatório das cartas (no caso do Quiz) e contagem de score e tempo, etc. O restante dos arquivos são de *assets* do jogo, em sua grande maioria imagens.

5.1.1 Implementação no Renpy

Antes do detalhamento do código fonte, é relevante destacar, que um esquema fundamental da implementação do SegurLab 2D segue conceitos pré-estabelecidos pelo Renpy, principalmente quando se trata de sua forma híbrida na utilização da linguagem Python. O conceito que mais marca o projeto é o de *screen* (ROTHAMEL, 2020).

Estas *screens* ou telas são um ambiente onde o Renpy permite a amostragem de informações e imagens para o usuário, isso vai desde a *interface* (Texto, menus, etc) até elementos propriamente dos jogos (imagens e botões). A outra grande função deste elemento é permitir uma fácil interação visual do usuário. São telas onde o *engine* facilita estas operações, dando amplo apoio para funções de mouse/teclado, coordenados com o que é apresentado na tela.

Assim, a implementação gira em torno do código fonte para o ambiente *screen,* onde é montado uma cena e colocado seus elementos tanto de interface como visuais. No código fora de tal ambiente, a maioria da lógica do jogo ocorre (cálculo de escores, progresso do usuário pelo jogo, etc.).

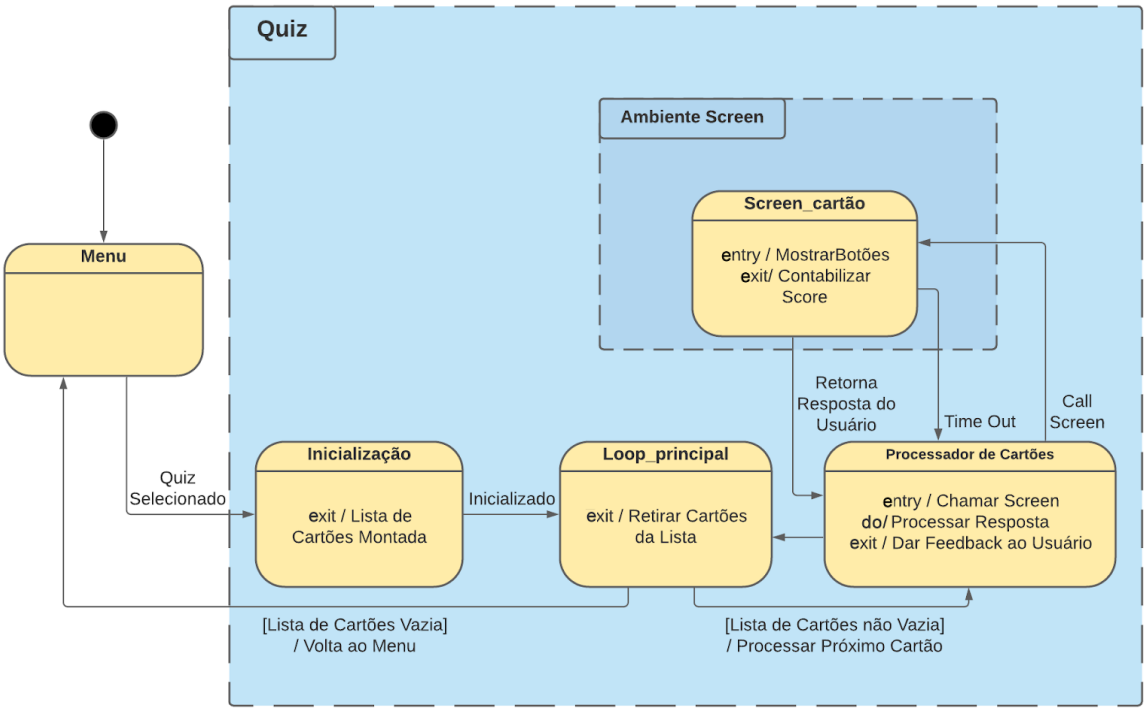
Os trechos de código fonte apresentados a seguir, serão todos pertencentes ao arquivo script.rpy. Também é destaque, que quaisquer arquivos de imagem (primariamente arquivos .png) se encontram na pasta "*images*" dentro do diretório "*game*", tanto no projeto não compilado como na versão final do SegurLab 2D.

5.1.1.1 Implementação do Quiz: *Inicialização*

O código abaixo começa no momento em que o usuário seleciona o Quiz no menu principal (menu é detalhado no arquivo screens.rpy). A partir disto, temos um redimensionamento de alguns botões da interface (*check2, checkbox e vcheckbox*) nas linhas 3 até 10.

Essas declarações associam um nome lógico, como "*vcheckbox*" ao arquivo correspondente. Na grande maioria dos casos, foi utilizado o mesmo nome do próprio arquivo por motivo de organização. Junto a isso temos outros atributos que podem ser associados a essa imagem.

Fig.5.1 - Grafo de Estados do Quiz



Fonte: Própria.

No caso presente de *"checkbox"* e *"vcheckbox"*, o atributo *zoom* faz um redimensionamento da imagem, para ser encaixado na tela (a maioria das imagens tem resolução e tamanho muito maiores do que os utilizados no SegurLab 2D. Sendo assim, elas apresentam boa qualidade em quaisquer telas 1080p, padrão, mais utilizadas em monitores e celulares atualmente).

```
1    label start:
2    #resize da UI:
3    image check2:
4        "check2.png"
5    image checkbox:
6        "box.png"
7        zoom 0.05
8    image vcheckbox:
9        "vbox.png"
10       zoom 0.05
11
12   $ renpy.block_rollback()
13   $ easy_list = ["Azul01","Azul02","...", "Red03"]
14   $ medium_list = ["Azul04","Azul05","...","Red06"]
```

```
15    $ hard_list = ["Roxo03","Roxo04","...","Azul09","Azul10"]
16    $ all_list =["Roxo01", "Roxo02", "...","Azul09","Azul10"]
17    $ score = 0
18    $ nrquestoes = 0
19    $ totalquestions = len(easy_list) + len(medium_list) +
len(hard_list)
20     scene chemp
21     show screen tutorial
22     jump pause
```

Na sequência (linhas 12 a 21), temos as declarações de variáveis, algumas são para limpar e *resetar* estas variáveis de controle (linhas 17,18), que definem o escore do jogador e o número total de questões respondidas, respectivamente.

Na linha 12, temos uma chamada importante, o "block_rollback()" aqui está bloqueando o jogador de voltar para trás em qualquer tela que ele se encontre. Essa função está desativada por padrão no Renpy (para fins de narrativa e *debugging)*. No caso do Quiz, mantê-la ativa permitiria ao jogador a possibilidade de "trapacear" nas suas respostas, já que ele poderia ver a resposta e voltar para trocá-la.

Das linhas 14 até 18, temos as inicializações das listas de perguntas, "*easy_list, medium_list, hard_list".* Representam listas, onde cada item é uma *string* que irá referenciar uma *label* no *loop* principal do jogo (ver subseção seguinte), que por sua vez referenciará uma *screen* de mesmo nome.

Temos uma separação de perguntas por dificuldade, além de uma lista com todas as perguntas, estas diferenciam as dificuldades que o jogador pode escolher no início do Quiz.

Foi omitida a lista completa de todas as perguntas para fins de legibilidade (denotado por "..." no código).

```
1 screen countdown:
2     timer 0.01 repeat True action If(time > 0,
true=SetVariable('time', time - 0.01), false=[Hide('countdown'),
Jump(timer_jump)])
3     bar value time range timer_range xalign 0.5 yalign 0.9
xmaximum 300 at alpha_dissolve
```

Como um *addendum*, o código acima se refere à criação do *timer* que é utilizado no Quiz. Na linha 2 temos a criação do *timer*, uma estrutura convenientemente pronta no Renpy.

O relógio tem sua unidade mínima em 0,01 segundos, uma granularidade suficiente para os fins do jogo. A variável "*time*" foi fixada para 3 minutos, para cada questão.

Acompanhando o cronômetro lógico interno, na linha 3, existe uma barra, que começa cheia e irá diminuindo, conforme o tempo de 3 minutos vai se esgotando.

5.1.1.2 Implementação do Quiz: *Loop Principal*

```
1 label questions_loop:
2     $ answer = "NONE"
3     $ disable_button = False
4     hide screen tutorial
5     if not easy_list == []:
6         $ random_jump_target = random.choice(easy_list)
7         $ easy_list.remove(random_jump_target)
8         $ renpy.call(random_jump_target)
9     elif not medium_list == []:
10        $ random_jump_target = random.choice(medium_list)
11        $ medium_list.remove(random_jump_target)
12        $ renpy.call(random_jump_target)
13    elif not hard_list == []:
14        $ random_jump_target = random.choice(hard_list)
15        $ hard_list.remove(random_jump_target)
16        $ renpy.call(random_jump_target)
17     "Você acertou [nrquestoes] de [totalquestions] questões e
seu score final é [score]. Clique para continuar."
18     "Fim da Demonstração!"
19    $ MainMenu(confirm=False)()
```

No trecho de código acima, temos o loop principal do jogo, nas linhas 2 a 4, novamente, *resetamos* as variáveis de respostas, que são preenchidas em cada pergunta do Quiz. Nos trechos de 6 a 16 percorremos as listas de perguntas aleatoriamente, porém em ordem, das perguntas mais fáceis para as mais difíceis. O "*random_jump_target*" escolhe, aleatoriamente, um *label* na lista de dificuldade correspondente. Este item é posteriormente removido através do método ".remove". E, por fim, é feita a chamada pelo Renpy para o *label* correspondente através do "*renpy.call(random_jump_target)*". Na linha 19, após a apresentação do escore e desempenho final, temos o retorno para o *main* menu.

5.1.1.3 Implementação do Quiz: *Exemplo da Parte Lógica de uma Questão*

Como já mencionado, a esquematização do Renpy e, portanto, do SegurLab 2D se divide em partes de cunho interativo, ditas *screens* e partes de cunho comum, como as demonstradas nesta subseção. O código abaixo parte da seleção da *label* "Azul01", que, no caso, representa o Cartão/Questão de mesmo nome. Essa *label* é referenciada no *Loop* Principal, mostrado na última seção.

Começamos imediatamente chamando a *screen* de nome correspondente na linha 1, que será detalhada na próxima subseção, mas para efeitos de entendimento, é suficiente saber que ela irá retornar na variável "*answer*" uma *string* ou caractere.

Na linha 2 temos uma chamada para apenas mostrar a imagem correspondente à *screen* prévia, através da função "*show"* do Renpy. Isso pode parecer redundante, mas é necessário pois há uma diferença entre fazer somente o *display* visual da questão e chamar todo o código de interface para responder à questão. Ao término de *"call screen Azul01 with dissolve"* é removido todos os elementos visuais, assim *"show screen Azul01"* permite que a questão continue de fundo visual, enquanto o feedback a seguir é mostrado.

Uma nota em relação a essas funções é o sufixo "with dissolve", sendo ele um dos vários efeitos visuais de transição de imagens fornecido pelo *engine*, nada mais que um simples *fade out/fade in* neste caso.

```
label Azul01:
1     call screen Azul01 with dissolve
2     show screen Azul01
3     $ disable_button = True
4     if answer =="timeout":
5         "Tempo Limite Atingido!"
6     if answer == "B":
7         $ score = score + 10 * timeremaining
8         "Correta! Seu score atual é : [score]. Clique para
continuar."
9         $ nrquestoes = nrquestoes + 1
10    else:
11        "Errada! Clique para continuar."
12   hide screen Azul01 with dissolve
13   jump questions_loop
```

Na sequência, temos na linha 3, uma outra consequência da linha 1, "*disable_button = True",* trava os botões pressionados durante a chamada da *screen*. Isso permite que mesmo que o usuário não possa mais interagir com os botões da pergunta, ele ainda poderá vê-los e

assim, associar sua resposta escolhida ao *feedback* associado.

Nas linhas 4 a 8 temos os casos de resposta. Neste exemplo, a resposta B é a correta, em caso de acerto é agregado um valor (dependente do tempo levado para responder) ao escore total do usuário. Também há o caso de tempo esgotado (*timeout*) e de erro, em ambos não há adição de escore.

Ao fim, nas linhas 12 e 13 temos a remoção da imagem de fundo (inserida na linha 2) e o retorno ao *loop* principal, seguindo para a próxima questão, onde todo o processo aqui descrito se repetirá.

5.1.1.4 Implementação do Quiz: *Exemplo da Screen de uma Questão*

Nesta parte do código, é apresentado um exemplo de implementação de uma *screen* no SegurLab 2D, como já mencionando a *screen* representa a parte interativa do Quiz. Nela é posicionado e colocado efetivamente os elementos que dependem da interação do usuário, como botões ou cronômetros visuais.

Tomamos como exemplo a *screen* Roxo03, no trecho de código abaixo. Na linha 2 temos a utilização de um *imagemap*, como pode ser inferido pelo nome, este elemento serve para mapear, em um sistema de coordenadas, um arquivo de imagem que terá propriedades e elementos adicionados a ele. No caso, a imagem selecionada é “Roxo03.png”, que é primeiramente fixada exatamente no meio da tela, através do comando *ground,* juntamente com o alinhamento dos eixos x e y pela metade (linha 3).

Na sequência nas linhas 4 até 8, temos a declaração de *hotspots*, áreas na imagem previamente mapeada que são clicáveis. No caso do exemplo, as coordenadas em parênteses correspondem à área na imagem que engloba cada uma das respostas de múltipla escolha: A, B, C e D. A *action* associada a esses *hotspots* é a escolha da resposta, a letra da resposta escolhida é guardada na variável *answer*, sendo o principal objetivo desta *screen*.

```
1 screen Roxo03:
2     imagemap:
3         ground "Roxo03.png" xalign 0.5 yalign 0.5
4         hotspot (630, 569, 254, 169) action
SetVariable("answer","A"), Return()
6         hotspot (630, 781, 252, 172) action
SetVariable("answer","B"), Return()
7         hotspot (1049, 564, 261, 177) action
SetVariable("answer","C"), Return()
8         hotspot (1050, 782, 254, 168) action
SetVariable("answer","D"), Return()
```

```
10      imagebutton:
11         idle "check3_idle.png"
12          if disable_button == False:
13             hover "box4.png"
14          selected_idle "box4.png"
15          selected_hover "box4.png"
16          focus_mask None
17          xpos 600
18          ypos 550
19          action SetVariable("answer","A"),Return()
20         "..."
21      if persistent.timed_choices:
22          bar:
23              xalign 0.5
24              xpos 1550
25              ypos 930
26              xsize 400
27              value AnimatedValue(old_value=0.0, value=1.0,
range=1.0, delay=timeout)
28          timer timeout action SetVariable("answer","timeout"),
Return()
```

Antes de continuar para as linhas 10 até 19, o leitor talvez já tenha notado que nestas linhas há uma redundância de função, o *imagebutton* declarado na linha 10 irá fazer a mesma função que o *hotspot* mencionado anteriormente. Essa redundância está presente por dois motivos, o primeiro é o fator legado, a imagem de cada questão foi feita em uma única imagem (tanto a parte artística quanto a parte textual). Isto apresenta alguns problemas, principalmente quando se trata de adicionar elementos visuais a esta imagem já pronta. O segundo motivo é que, neste contexto, o *hotspot* funciona melhor para a versão *Mobile*, onde a interface *touch* se conforma dentro deste contexto. O Contraponto é que para se ter um bom *feedback* visual no computador com um mouse, é necessário adicionar elementos em cima dessa imagem, elementos dinâmicos que realcem a escolha de resposta por parte do usuário. Neste último caso o *imagebutton* se encaixa bem, já que o *engine* proporciona métodos para tal. Uma solução foi colocar ambos sobrepostos na imagem, na versão PC do SegurLab 2D o *hotspot* é transparente ao usuário, já na versão mobile alguns elementos do *imagebutton* não são mostrados.

Dito isso, na linha 11 temos a declaração da imagem de *idle*/inatividade, esta é a imagem que aparece, quando a área do *imagebutton* não está sendo tocada pelo cursor do

mouse, neste caso dos botões de múltipla escolha no Quiz, todas as imagens *idle* são transparentes, para manter consistência com o *hotspot* descrito acima.

Estas imagens transparentes servem para delinear o botão em questão de coordenadas, que serão correspondidas com o *hotspot* do mesmo. Nas linhas 12 a 15 temos a declaração das outras imagens que compõem o *imagebutton*, no caso do Quiz, serão utilizadas a mesma imagem em todos os casos, já que o *feedback* importante é somente mostrar a resposta selecionada e clicada, sendo os dois estados essenciais o *idle* e o *hover* (quando o ponteiro do mouse está sob a área do botão). Na linha 12, também temos a variável booleana "*disable_button*", que como explicada na subseção anterior, é usada para trancar o *display* da opção selecionada e permitir o usuário ver a opção selecionada após sua escolha.

Nas linhas 16 a 19 temos alguns itens adicionais, além do posicionamento do *imagebutton*, temos a booleana "*focus_mask*", uma variável para controlar se o Renpy deve desconsiderar transparência na imagem *idle* como área "clicável" do botão, como dito anteriormente a imagem *idle*, neste caso, é totalmente transparente, por isso o parâmetro foi "setado" para falso.

A linha 20 é para motivos de legibilidade, pois é feito um *imagebutton* para cada opção de múltipla escolha da questão.

Nas linhas 21 a 28 temos a adição do *timer* mencionado no subcapítulo 5.2.2.1, aqui ele é simplesmente chamado, posicionado e alinhado corretamente, sendo adicionado a *screen.* Na linha 28 temos a chamada para que a resposta da pergunta seja "*timeout",* que, como visto no capítulo anterior, irá gerar um *feedback* correspondente, em outras palavras, o usuário ficou sem tempo para responder.

5.1.2 Implementação do Identificando Riscos: Inicialização

O trecho de código abaixo começa quando o usuário seleciona "Identificando Riscos" no menu principal. Nas linhas 1 até 7 iniciamos várias variáveis, com a maioria delas sendo de nome autoexplicativo. "*game2_answer_list*" é um *array* que irá conter tanto variáveis quanto *strings* que representarão os elementos que o jogador escolheu em cada cena.

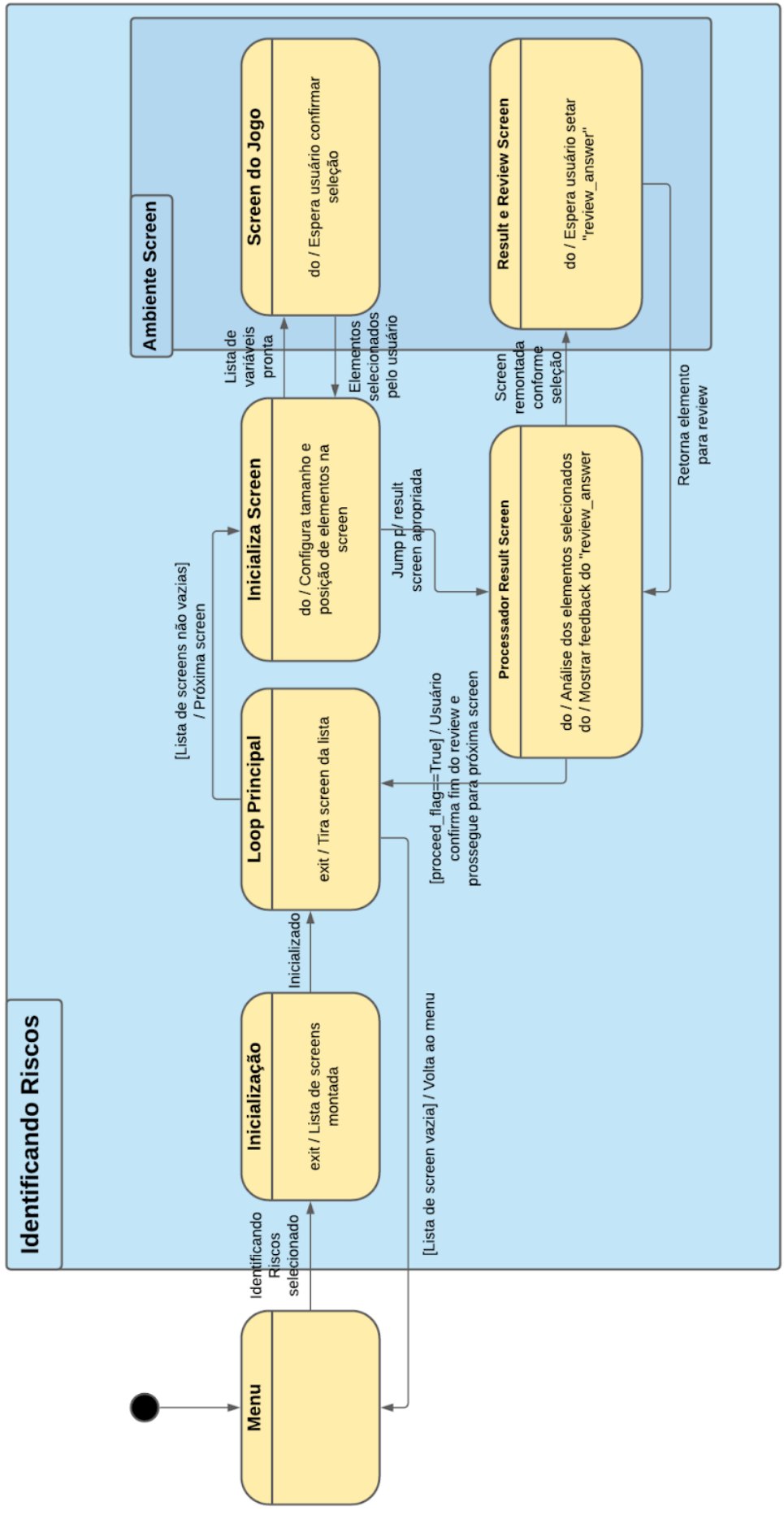


Fig. 5.2 - Grafo de Estados do Identificando Riscos

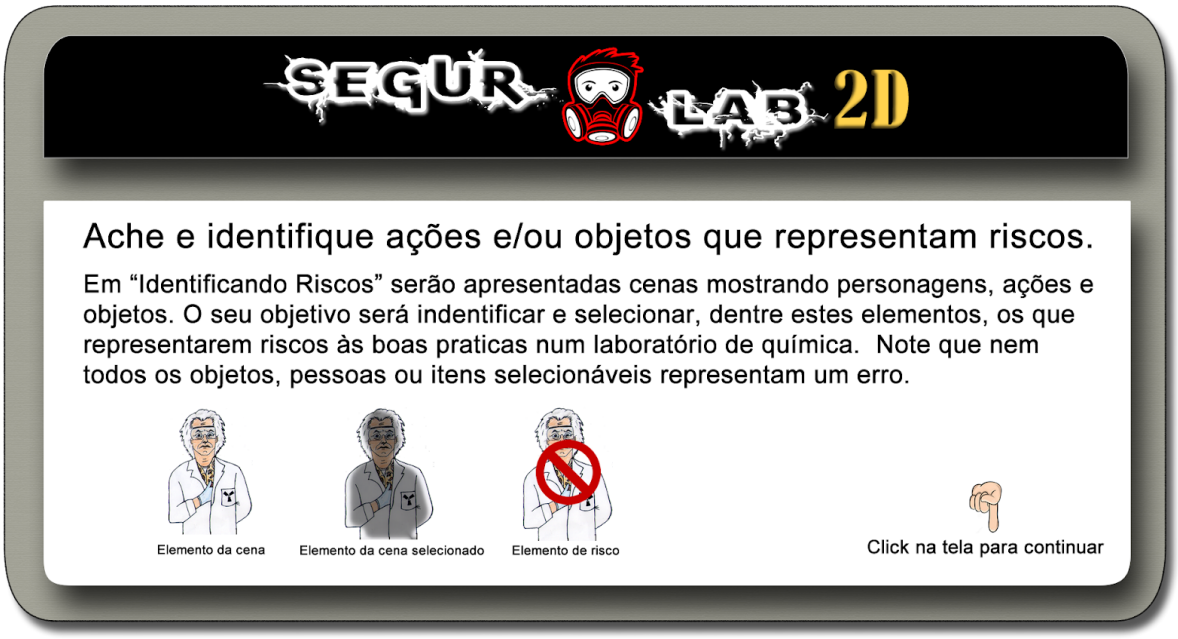
Fonte: Própria

O restante das variáveis são apenas resetadas para zero ou colocadas nos seus valores iniciais, "*screen_count*" indica o número de arranjos de cena contidas no jogo (cada arranjo contém diferentes imagens). Linha 8 chama o fundo de tela (utilizado durante a apresentação do tutorial do Identificando Riscos). Na linha 9 temos a inicialização da lista das cinco cenas atualmente contidas no jogo, cada elemento da lista é uma *string* que será utilizada para pular para uma *label* de mesmo nome no *loop* principal (explicado na próxima subseção).

```
# inicialização
1     $ game2_answer_list = []
2     $ score = 0
3     $ nrtelas = 0
4     $ screen_count = 5
5     $ total_acertos = 0
6     $ total_riscos = 0
7     $ total_score = 0
8     scene chemp
9     $ screen_list = ["screen01", "screen02",
"screen03","screen04","screen05"]
10    show instru with dissolve:
11        xalign 0.5 yalign 0.5
12    $ renpy.pause ()
13    hide instru
14    show img482:
15        xalign 0.5 yalign 0.5
```

Na sequência, das linhas 10 a 13, mostramos um cartão com instruções e objetivos do Identificando Riscos, sendo este alinhado na tela e com uma pausa para o jogador ter o tempo necessário para leitura, o cartão, então, é removido da tela e a imagem de fundo, que será utilizada por todas as cenas seguintes é colocada na tela (linhas 13 a 15).

Fig. 5.3: Cartão de Instruções do Identificando Riscos



Fonte: Própria

5.1.2.1 Implementação do Identificando Riscos: Loop Principal

Similar aos cartões do Quiz, o loop principal continua até à exaustão de uma lista que contém *labels* referentes a estruturas do jogo, nesse caso, as cenas contidas em "*screen_list*". Na linha 3, escolhemos uma cena aleatória contida na lista, que ficará guardada na variável "*random_jump_target*", esta cena selecionada é posteriormente retirada da lista (linha 4), diminui-se o contador de cenas e apresentamos a arte de fundo onde os atores e objetos irão aparecer (linhas 5 até 7). Finalmente fazemos a chamada para o *label* correspondente de "*random_jump_target*". Nas linhas 8 e 9, temos o final do jogo, com o retorno ao menu principal.

```
1 label telas_loop:
2     if not screen_list == []:
3         $ random_jump_target = random.choice(screen_list)
4         $ screen_list.remove(random_jump_target)
5         $ screen_count = screen_count - 1
6         show img482:
7             xalign 0.5 yalign 0.5
8         $ renpy.call(random_jump_target)
```

```
9     "Fim de Jogo! Seu Total de acertos foi de [total_acertos] de
[total_riscos], com o score final de [total_score], você terminou
[nrtelas] estagios!"
10    $ MainMenu(confirm=False)()
```

5.1.2.2 Implementação do Identificando Riscos: Parte Lógica da Screen Principal

No trecho a seguir será mostrado um exemplo de cena. Note que para cada cena, existe um trecho similar de código ao apresentado abaixo. Começamos por uma listagem de definições nas linhas 1 a 6, elas são elementos que serão utilizados para a contagem de *score* e providenciar *feedback* para o usuário, a exceção de "*results_counter*" e "*game2_answer_list*", que são elementos internos de cada cena e utilizados para o funcionamento da mesma.

Nas linhas 7 até 28, temos um exemplo de como uma imagem é introduzida no jogo (este padrão se repete para cada imagem), os valores com as posições X e Y são declarados, juntamente com o *zoom*/redimensionamento apropriado. Em seguida a imagem é propriamente declarada com seu nome lógico (por exemplo: "mochila_grande" e associado ao seu arquivo de imagem correspondente ("mochila.png"). Neste exemplo há várias variações da imagem "mochila_grande". Elas correspondem aos artefatos e efeitos que foram aplicados à imagem e tem propósitos variados. Na sequência deste subcapítulo, segue uma pequena tabela com os prefixos utilizados e seus significados para todas as imagens do Identificando Riscos.

```
1 label screen01:
2     $ score = 0
3     $ results_counter = 0
4     $ max_riscos = 4
5     $ game2_answer_list = []
6     $ erros = 0
7   #mochila_grande
8     $ mochila_grande_yalign = 0.7
9     $ mochila_grande_xalign = 0.05
10    $ mochila_grande_zoom = 0.5
11    image mochila_grande:
12        "mochila.png"
13        zoom mochila_grande_zoom
14    image mochila_grande_r:
15        "mochila_r.png"
16        zoom mochila_grande_zoom
17    image mochila_grande_pro:
```

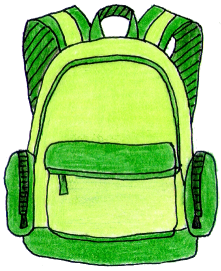
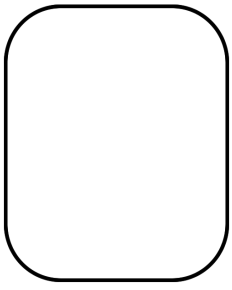
```
18          "mochila_pro.png"
19          zoom mochila_grande_zoom
20      image mochila_grande_rpro:
21          "mochila_rpro.png"
22          zoom mochila_grande_zoom
23      image mochila_grande_mold_s:
24          "mochila_mold_s.png"
25          zoom mochila_grande_zoom
26      image mochila_grande_mold_i:
27          "mochila_mold_i.png"
28          zoom mochila_grande_zoom
29      $ mochila_grande_var = True
30      $ game2_answer_list.append ("mochila_grande_var")
31      $ game2_answer_list.append (mochila_grande_var)

#Repete-se as linhas 7-28 para os demais elementos na cena

32      call screen game2_hiddenobject with dissolve
33      jump end_result_screen
```

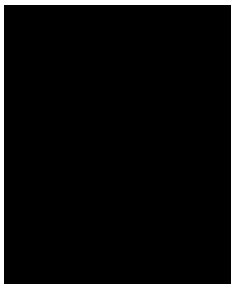
Como brevemente mencionado no subcapítulo de inicialização, "*game2_answer_list*" é uma lista carregando informações de forma implícita e binária, no caso do exemplo, o par de informações é uma *string* e um *boolean*, ambos de mesmo nome. A *string* serve para identificar o elemento na lista, enquanto que a variável *boolean,* diz ao programa se este elemento foi selecionado ou não (mostrado em detalhes na próxima subseção).

Por fim, na linha 32, chamamos a *screen* da cena em questão "*game2_hiddenobject*" que é onde efetivamente o usuário interage com o jogo, retornando desta, na linha 33, partimos para a *screen* dos resultados e *feedback*, ambas detalhadas nos subcapítulos seguintes.

Tabela 5.1 - Nomenclatura das Imagens do Identificando Riscos

Prefixos das Imagens	**Descrição**	**Exemplo**

Sem prefixo, ex.: mochila.png	Imagem padrão, tal como foi escaneada e sem efeitos adicionados	
Prefixo "_r", ex.: mochila_r.png	Imagem realçada com escurecimento das cores, indica que o usuário está com o ponteiro (mouse ou dedos na *touch screen* do celular) em cima de uma imagem selecionável.	
Prefixo "_pro", ex.: mochila_pro.png	Imagem utilizada no *feedback* do usuário, mostrada durante e após a *results_screen*, a imagem demonstra que ela é um risco que não foi identificado e selecionado, configurando um erro.	
Prefixo "_rpro", ex.: mochila_rpro.png	Imagem utilizada no *feedback* do usuário, mostrada durante e após a *results_screen*, a imagem demonstra que ela é um risco que foi corretamente identificado e selecionado, configurando um acerto.	
Prefixo "_mold_s", ex.: mochila_mold_s.png	Moldura utilizada somente durante a parte interativa da *results_screen*, ela realça todos os elementos de risco, estejam estes selecionados ou não, quando o usuário passa com o ponteiro do dispositivo sobre os mesmos.	

Prefixo “_mold_i” , ex.: mochila_mold_i.png	Imagem invisível (aqui colorida de preto para visualização), serve como uma máscara para a moldura acima, providenciando não somente coordenadas para a área de realce, mas também serve como o estado *default* da moldura descrita acima, ou seja, a moldura permanece invisível até ser realçada.	

Fonte: Própria

5.1.2.3 Implementação do Identificando Riscos: Screen do Jogo

Nesta *screen* é onde o jogador do Identificando Riscos terá a principal interação com o jogo, ela consiste em vários *imagebuttons,* dispostos na cena de forma condizente com a imagem de fundo. Abaixo, teremos o exemplo de um único *imagebutton*, mas o exemplo vale para qualquer um utilizado no jogo.

Fig. 5.4: Uma das *Screens* do Identificando Riscos, com alguns elementos selecionados

Fonte: Própria.

Similar ao jogo Quiz, nas linhas 1 até 4 e 7 até 10, há uma declaração do botão *idle* (linha 3) indicando a imagem mostrada quando o botão não está ressaltado pelo ponteiro do *mouse*. A imagem *hover* (linha 7) é mostrada quando esse realce ocorre pelo *mouse*. *Selected_idle* é para manter o botão com a imagem de realce, caso ele for selecionado. As demais linhas deste conjunto são de igual função, já explicadas no Quiz.

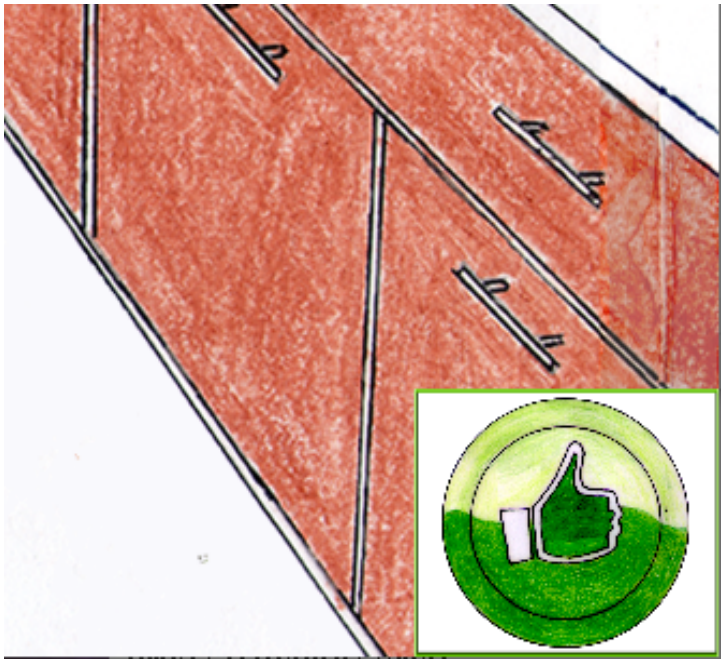
O destaque aqui fica, então, para as linhas 5 e 6, onde a lista "*game2_answer_list*" é utilizada. Através do método *index* é achado a *string* equivalente, a posição na lista é acrescida de "+1", que corresponde à sua variável *booleana* (como explicado na subseção anterior, esses dois elementos são sempre montados em ordem na lista). Assim, *action* na linha 5, corresponde à ação que o botão realiza cada vez que é clicado, um *toggle* na sua variável *booleana.* Na linha 6, o *selected* indica qual a variável é utilizada para mostrar se o botão está selecionado ou não (se a variável for *true* é utilizado a imagem atribuída no *idle*, se estiver *false*, a imagem mostrada é *selected_idle*).

```
1 screen game2_hiddenobject:
2     imagebutton:
3         idle "frasco_capela"
4         focus_mask True
5         action ToggleDict(game2_answer_list,
game2_answer_list.index("frasco_capela_var") + 1)
6         selected
(game2_answer_list[game2_answer_list.index("frasco_capela_var") +
1] == False)
7         hover "frasco_capela_r"
8         selected_idle "frasco_capela_r"
9         xalign frasco_capela_xalign
10        yalign frasco_capela_yalign

#Repete-se as linhas 1-10 de forma similar para os demais
elementos na cena

11    image "frame":
12        xalign frame_xalign
13        yalign frame_yalign
14    imagebutton:
15        idle "b1"
16        hover "b2"
17        xalign b1_xalign
```

```
18          yalign b1_yalign
19          action With(dissolve), Return()
```

Nas linhas 14 a 19, temos a implementação do botão que confirma as respostas, juntamente com seu *frame*, que serve para destacar o botão.

Fig. 5.5: Botão de Confirmação de Respostas, no canto inferior direito



Fonte: Própria.

5.1.2.4 Implementação do Identificando Riscos: Parte Lógica da Screen de Revisão

Como nos outros trechos apresentados, nas linhas 2 a 4, começamos com a inicialização de variáveis de erros e acertos dos usuários, além do iterador que percorrerá o *loop* seguinte.

Nas linhas 6 a 26, temos dois exemplos de como o programa remonta a seleção feita pelo usuário na *screen* descrita na subseção anterior. A partir da iteração da "*game2_answer_list*", conseguimos descobrir todas as seleções de itens feitas pelo usuário, já que como mencionado, esta estrutura carrega a informação booleana das mesmas.

```
1  label end_result_screen:
2      $ erros = 0
3      $ score = 0
```

```
4     $ results_counter = 0
5
6     while (results_counter < len(game2_answer_list)):
7         if (game2_answer_list[results_counter] ==
"frasco_capela_var"):
8             if (game2_answer_list[results_counter + 1] ==
False):
9                 $ erros = erros + 1
10                show frasco_capela_r:
11                    yalign frasco_capela_yalign
12                    xalign frasco_capela_xalign
13            else:
14                show frasco_capela:
15                    yalign frasco_capela_yalign
16                    xalign frasco_capela_xalign
17        if (game2_answer_list[results_counter] ==
"luva_caindo_var"):
18            if (game2_answer_list[results_counter + 1] ==
False):
19                $ score = score + 1
20                show luva_caindo_rpro:
21                    yalign luva_caindo_yalign
22                    xalign luva_caindo_xalign
23            else:
24                show luva_caindo_pro:
25                    yalign luva_caindo_yalign
26                    xalign luva_caindo_xalign
#demais itens na cena
27        $ results_counter = results_counter + 1
```

Como exemplo de imagem sem risco, na linha 7, procuramos pelo item de nome “frasco_capela_var”. Na linha 8, apresenta-se o caso onde o usuário selecionou este elemento durante o jogo e assim apresentamos a imagem com realce.

Na linha 13, caímos no caso onde o jogador não selecionou o elemento, ambos casos resultarão em imagens descritas na Tabela 5.1. O detalhe fica na linha 9, onde é contabilizado um erro do usuário, pois ele marcou uma imagem sem risco, neste exemplo.

Como exemplo de imagem com risco, na linha 17, procuramos pela variável do elemento ”luvas_caindo_var”. De forma análoga ao exemplo anterior, temos os dois mesmos

casos: na linha 18, o elemento foi selecionado, configurando um acerto, denotado na linha 19. E, na linha 23, temos a ocasião em que o elemento não foi selecionado. O destaque fica por conta de que em ambos os casos a imagem será substituída por sua equivalente - com e sem realce - denotada pelo seu sufixo descrito na Tabela 5.1.

Fica subentendido, que todos estes exemplos apresentados se repetem de forma homóloga para tantos quantos forem os elementos que estiveram presentes na sua *screen* associada.

```
1     $ proceed_flag = False
2     $ review_answer = "NONE"
3     $ score_points = score * 15
4     $ total_acertos = total_acertos + score_points
5     $ total_riscos = total_riscos + max_riscos
6     $ total_score = total_acertos + score_points
7     if (erros == 0):
8         "Você indentificou [score] de [max_riscos] riscos! Seu
score final é [score_points]. Clique para continuar."
9     if (erros > 0):
10        "Você indentificou [score] de [max_riscos] riscos, porém
[erros] items foram erroneamente marcados! Seu score final é
[score_points]. Clique para continuar."
11    while (proceed_flag == False):
12        "Clique nos items que representam riscos para maiores
informações. Senão clique no botão verde novamente para ir para o
proximo nivel"
13        call screen review_screens with dissolve
14        if (review_answer == "mochila_grande"):
15            "Mochila na mesa apresenta risco de contaminação,
tanto de agentes externos quanto internos! Mochilas, bolsas e
afins devem estar longe de areas de risco, como armários ou
cabides apropriados."
16        if (review_answer == "sem_ocolus"):
17            "Esta aluna não está utilizando proteção ocular,
pondo em risco sua visão!"
18        if (review_answer == "boneco_c"):
19            "Este aluno está se agachando no chão do
laboratório, colocando-se em risco devido a residuos presentes no
chão"
20        if (review_answer == "costas"):
21            "Este aluno está se desfazendo de suas luvas de
```

```
forma errada!"
22        if (review_answer == "luva_caindo"):
23            "As luvas devem ser descartadas no local apropriado
e não na lixeira comum!"
24        $ review_answer = "NONE"
25    jump telas_loop
```

No código acima, das linhas 1 a 12, preparamos para a chamada da *screen* que apresentará um *review*/gabarito da cena corrente, além de seu respectivo *feedback*, linhas 14 a 23. Além das variáveis de controle, nas linhas 1 e 2, temos a sistematização do sistema de *score*, nas linhas 3 a 6. Nas linhas 7 a 10, apresentamos uma tela de texto, informando ao usuário qual foi o seu desempenho, na cena/*screen* que acabou de concluir, informando valores como *score*, riscos identificados e elementos marcados erroneamente.

A partir da linha 11 entramos na seção de feedback, onde chamaremos a uma *screen*, linha 13, que permitirá interatividade com o usuário, onde será descrito na próxima subseção. Para entendimento deste exemplo de código, é suficiente informar que a *screen* "*review_screens*" modificará a variável "*review_answer*", fazendo com que mostrado uma das descrições exemplificadas nas linhas 14 até 23, que providenciarão uma explicação sobre cada elemento de risco mostrado na cena.

5.1.2.5 Implementação do Identificando Riscos: Screen de Revisão

A *screen* de revisão tem o propósito de dar interatividade ao usuário para identificar e clicar em elementos da cena dos quais ele tem dúvida,. como mencionado no último subcapítulo. O principal objetivo da chamada desta *screen* é modificar a variável "*review_answer*" com uma *string,* que indicará o objeto clicado e será processado pela parte lógica da revisão.

Neste trecho de código, são criados *imagebuttons* de maneira bem similar aos criados no módulo Quiz. Somente imagens que representam riscos são interativas, as demais imagens continuam na cena, porém sem nenhum botão associado.

Da linha 1 até a 9, temos um exemplo deste botão. Utilizamos uma imagem de máscara, como descrita na Tabela 5.1, para emoldurar a área "clicável", devido ao fato de que esta imagem ser invisível, precisamos desabilitar o método "*focus_mask*" da mesma forma ocorrida no jogo Quiz. Na linha 4 temos a principal função da *screen*, que neste exemplo, modificará "*review_answer*" para a *string* "mochila grande", imediatamente saindo do ambiente *screen* e retornando esta informação para a parte lógica descrita anteriormente.

Das linhas 5 até 9, apresentamos a imagem de moldura apropriada, neste caso "mochila_grande_mold_s", que pelo método *hover*, só será mostrada quando o usuário passar o ponteiro do *mouse* sobre a área delimitada por sua máscara. Vale ressaltar que, tanto a moldura quanto a máscara têm as mesmas resoluções, tamanhos e coordenadas para todas as imagens utilizadas nas cenas do Identificando Riscos, portanto não havendo necessidade de modificá-las em tempo de execução.

Da linha 10 até a 18, passamos por um outro exemplo similar ao descrito até então, este trecho de código irá se repetir de forma análogo, tantas vezes quantos forem os elementos de risco na *screen* associada.

```
screen review_screens:
1     imagebutton:
2         idle "mochila_grande_mold_i"
3         focus_mask None
4         action
SetVariable("review_answer","mochila_grande"),Return()
5         selected (review_answer == "mochila_grande")
6         hover "mochila_grande_mold_s"
7         selected_idle "mochila_grande_mold_s"
8         yalign mochila_grande_yalign
9         xalign mochila_grande_xalign

10    imagebutton:
11        idle "sem_ocolus_mold_i"
12        focus_mask None
13        action
SetVariable("review_answer","sem_ocolus"),Return()
14        selected (review_answer == "sem_ocolus")
15        hover "sem_ocolus_mold_s"
16        selected_idle "sem_ocolus_mold_s"
17        yalign sem_ocolus_yalign
18        xalign sem_ocolus_xalign
# Demais itens da cena
19    image "frame":
21        xalign frame_xalign
22        yalign frame_yalign

23    imagebutton:
24        idle "b1"
25        hover "b2"
26        xalign b1_xalign
```

```
27          yalign b1_yalign
28          action SetVariable("proceed_flag",True), Return()
```

Das linhas 19 até 28, temos novamente, a inserção do botão verde de confirmação. neste caso, trocará a variável booleana "*proceed_flag*", retornando para a parte lógica da *screen* de revisão, que por sua vez voltará para o *loop* principal do identificando Riscos e este, por sua vez, chamará a próxima cena ou terminará o jogo, dependendo de quantas cenas ainda estiverem na lista de cenas, "*screen_list*", como descrito no subcapítulo 5.1.2.2.

Em outras palavras, se um jogador começar o Identificando Riscos do começo ao fim, ele passará por 5 cenários diferentes, cada uma com sua respectiva inicialização, composição de elementos/figuras, *screens* principal e de revisão. Todas tendo implementação similar, porém única, a apresentada nos exemplos mostrados nos subcapítulos anteriores. Todo este processo é resumido na figura 5.2, mostrada no começo do capítulo 5.1.2.

6 PESQUISA DE USABILIDADE

Após a completude da primeira versão beta[9] do projeto, foi realizada uma pesquisa de usabilidade, com 10 questões, junto a 29 alunos e professores de áreas relacionadas à segurança de laboratórios químicos. A pesquisa contou com dez perguntas baseadas na Standard Usability Scale[10], adaptadas para a língua portuguesa, com algumas adições, objetivando conhecer o perfil do usuário e descobrir *bugs.*

Na primeira pergunta, os entrevistados foram questionados sobre qual plataforma foi utilizada para rodar o jogo. Os resultados demonstraram que 51,7% usaram a versão mobile (Android) e 48,3% a versão Windows. A versão Linux não foi utilizada.

Na segunda pergunta, com resposta numa escala de 1 a 5, houve um questionamento sobre a dificuldade de rodar o jogo no dispositivo escolhido. Em 79,3% dos casos, não houve dificuldade alguma (5), 13,8% responderam na escala 4, poucas dificuldades. E, 6,9% tiveram muita dificuldade (1).

Na sequência, a questão 3 perguntava quais seriam as dificuldades encontradas na questão 2. Dentre as respostas, ficaram mais caracterizadas: a dificuldade em descompactar o arquivo de distribuição do jogo e o retorno ao menu principal.

Na quarta pergunta, em escala de dificuldade de 1 a 5, foi questionado sobre a facilidade de navegação do Identificando Riscos. Aqui 48,3% acharam muito fácil, escala 5, 31% na escala 4, tiveram facilidade, 13,8%, tiveram pequena dificuldade e apenas 6,9 % tiveram dificuldade (2).

A quinta questão, em uma escala de 1 a 5, questiona o entendimento do objetivo proposto pelo módulo Identificando Riscos. Entre os entrevistados, 44,8% tiveram o entendimento pleno, escala 5; obtiveram um bom entendimento, 31%; tiveram alguma dificuldade no entendimento (3), 20,7% e somente 3,4% dos entrevistados acharam o objetivo muito confuso, escala 1.

A sexta questão é idêntica à quarta, porém avaliando a interface do módulo Quiz. Com 51,7%, muito fácil (5); 20,7%, tiveram facilidade (4); 24,1% tiveram alguma dificuldade (3) e somente 3,4% acharam muito difícil a navegação (1).

Na sétima questão, foi perguntado se o usuário havia encontrado inconsistências no jogo, como um todo. Com resposta não, 93,1% dos casos.

---

[9] Entende-se por versão beta, a primeira versão do *software*, onde todas as *features* planejadas foram implementadas. No caso do SegurLab 2D, isto significa que os dois módulos de jogo foram implementados.

[10] SUS - Método de averiguação do nível de usabilidade de um sistema.

A oitava pergunta dá continuidade à anterior, pedindo que fossem descritas as inconsistências encontradas. Duas ocorrências foram descritas: *bugs* na atualização do escore, no Identificando Riscos e possível erro ao clicar com o botão direito do mouse durante o jogo.

A nona pergunta se referia, em escala de 1 a 5, sobre se seus alunos/colegas de aula conseguiriam utilizar o SegurLab 2D. Do total, 72,4% afirmaram que certamente conseguiriam, (5); 20,7% conseguiriam (4) e 10,3% conseguiriam com alguma dificuldade.

A décima e última questão referia-se à utilização ou não, do jogo em sala de aula, em uma escala de 1 a 5. Certamente usariam: 65,5% (5); usariam, 20,7% (4) e 13,8% não souberam responder (3).

Ao final do questionário foi disponibilizado um campo para comentários, onde várias informações foram observadas. Dentre elas, podemos destacar: a realização de uma melhor explicação nas escolhas erradas, no Quiz; aperfeiçoamento das formas e atalhos na interface; erros ortográficos e remoção dos *bugs* encontrados, além de vários elogios ao jogo e a sua arte gráfica.

Como conclusão essa pesquisa teve resultados positivos em dois âmbitos, num primeiro momento houveram críticas e levantamento de problemas a serem resolvidos e num segundo o retorno na opinião dos entrevistados, foi, também, positivo, dando justificativa e estímulo para a continuidade do projeto.

7 CONCLUSÃO

O Projeto SegurLab 2D foi realizado durante um ano e meio, passando por um período de instabilidades, tanto no que tange à política adotada para a educação, quanto ao agravamento da pandemia do Covid-19.

No projeto inicial era previsto a contratação de dois bolsistas: um para o desenvolvimento da engenharia de *software*, juntamente com a sua implementação, e outro para o *design* e produção da arte gráfica. Devido à política de não incentivo à educação e pesquisa, houve uma redução dos investimentos governamentais, não sendo possível a contratação do segundo bolsista. Além disso, nos últimos seis meses do projeto, nem a bolsa existente foi renovada, ocasionando a necessidade de trabalho voluntário da parte do primeiro bolsista, em todas as áreas do projeto.

Na fase de pré-projeto, foi feito um estudo para a identificação da melhor ferramenta, que se adequasse aos requisitos do SegurLab 2D, sendo escolhido o *engine* Renpy. A análise desta ferramenta se mostrou, em termos gerais, correta, com facilidade de seu domínio e seus recursos.

Devido à restrições nos aspectos estruturais do *engine*, alguns princípios de organização do código, tais como, modularização em diferentes arquivos dos cartões e cenas do jogo, tiveram que ser suprimidas para facilitar a portabilidade para a plataforma Android. Outro aspecto com grande destaque, foi a interação direta com a comunidade e desenvolvedores do Renpy, cuja troca de ideias foi essencial para a implementação do código.

Uma consideração que vale ser comentada, foi as mudanças feitas no decorrer do projeto, com o objetivo de melhorar a jogabilidade. Originalmente o módulo Identificando Riscos teve uma aproximação muito maior com seu homólogo físico. Após experimentos e discussões com o orientador do projeto, ficou decidido que algumas mudanças seriam necessárias para adaptação, principalmente, ao meio mobile.

Com o passar do projeto, algumas novas ideias, como a realização de um editor de cartões para o módulo Quiz, foram contempladas e experimentadas no código, mas deixadas para futuras expansões do jogo. Também para o futuro foi pensado um mecanismo *in-game* para a captação do *feedback* do usuário, facilitando esse processo. Além disso, uma refatoração do código para sua melhor modulação é esperada, facilitando a manutenção e evolução futura.

O projeto SegurLab 2D proporcionou um largo ganho de experiências em vários aspectos: como na formulação e organização do projeto; experienciando o trabalho em equipe; no intercâmbio com elementos fora da área de informática; na apresentação em

eventos científicos; no desenvolvimento e aprendizagem na produção gráfica e na programação em Python.

Enfim, cabe ainda ressaltar, que todos esses elementos supracitados foram desenvolvidos em um período atípico de grandes turbulências e adversidades, consistindo num ganho de experiência no que diz respeito às formas de se contornar as adversidades.

REFERÊNCIAS

110 BEST 2D GAME ENGINES AS OF 2020. (2013, February 26). Retrieved February 14, 2020. Disponível em: https://www.slant.co/topics/341/~best-2d-game-engines#22

ABD ALI, Ali & Shaalan, Naser & Al-Dahhan, Wedad & Hairunisa, Nany & Yousif**,** Emad. (2017). *A Technical Evaluation of a Chemistry Laboratory: A Step Forward For Maintaining Safety Measures*. Oriental Journal of Physical Sciences. 2. 68-71. 10.13005/OJPS02.01.06.

CONFORTO, Edivandro & Amaral, Daniel & Luis, da Silva. (2011). Roteiro para Revisão Bibliográfica Sistemática: Aplicação no Desenvolvimento de Produtos e Gerenciamento de Projetos.

CUNHA, Silas Goulart da. Desenvolvimento do jogo SegurLab 2D: Uma alternativa para abordar segurança no laboratório no ensino de Química. *In*: Desenvolvimento do jogo SegurLab 2D: uma alternativa para abordar segurança no laboratório no ensino de química. Porto Alegre, 2019. Disponível em: https://lume.ufrgs.br/handle/10183/204284. Acesso em: 23 jan. 2021.

DANE, F. Research methods. Brooks/Cole Publishing Company: California, 1990.

EICHLER, Marcelo Leandro, et al. (2013). XeNUBi - Jogo sobre Propriedades Periódicas. Disponível em: https://lume.ufrgs.br/handle/10183/127151

EICHLER, Marcelo Leandro, et al. (2018). Jogo Ferreiros e Alquimistas. Disponível em: https://lume.ufrgs.br/handle/10183/178202

FAMULARO, J. (2017, September 28). *The best game engines for making your own 2D indie game.* Retrieved February 14, 2020. Disponível em: https://www.pcgamer.com/the-best-2d-game-engines

GAMEMAKER STUDIO. (2020, February 8). Retrieved February 14, 2020. Disponível em: https://en.wikipedia.org/wiki/GameMaker_Studio

GHENSEV, R. O. Uso dos Games na Educação. Dissertação de mestrado - Pós-graduação em Mídias Sociais, Centro Universitário Senac. 55f. São Paulo: SENAC, 2010.

KRÜGER, Verno. Segurança no laboratório, f. 65. 1996. 130 p.

LEE, J. (2016). *Learning Unreal Engine game development: a step-by-step guide that paves the way for developing fantastic games with Unreal Engine 4*. Birmingham, UK: Packt Publishing.

LETONSAARI, M., Karjalainen, L., & Selin, J. (2019, June 14). *Nonlinear Storytelling Method and Tools for Low-Threshold Game Development*. Retrieved February 15, 2020. Disponível em: https://journals.hioa.no/index.php/seminar/article/view/3074/3190

LUKOSEK, G. (2016). *Learning C# by developing games with Unity 5.x: develop your first interactive 2D platformer game by learning the fundamentals of C#*. Birmingham, UK: Packt Publishing

MANZUR, A., & Marques, G. (2018). *Godot engine game development in 24 hours: the official guide to Godot 3.0*. Indianapolis: Pearson.

PERRY, Gabriela, et al. (2014). Ferreiros e Alquimistas. Disponível em:https://www.researchgate.net/publication/324755267_Ferreiros_e_Alquimistas_um_jogo_educacional_sobre_Historia_da_Quimica

ROTHAMEL, Tom. RenPy Documentation. Disponível em: https://www.renpy.org/doc/html/screens.html. Acesso em: 22 dez. 2020.

SANTOS, Cícero Ernandes de Melo, et al. (2019). Construção de um jogo educativo em uma plataforma de desenvolvimento de jogos e aplicativos de baixo grau de complexidade: o caso do Quizmica - Radioatividade. Disponível em: https://seer.ufrgs.br/renote/article/download/95725/53815

SOARES, Márlon Herbert Flora Barbosa. *The ludic in chemistry: games and activities in chemistry teaching.*. 2004. 218 f. Tese (Doutorado em Ciências Exatas e da Terra) - Universidade Federal de São Carlos, São Carlos, 2004.