1 - Telas de interação

Foram geradas imagens e, posteriormente, adaptadas conforme erros e anomalias, que servissem de interface de interação entre o usuário e o programa, permitindo o usuário ter acesso ao jogo, às informações e à opção de sair do jogo.

Para o funcionamento das imagens, criamos retângulos com as dimensões dos botões (utilizando a função sf::IntRect ‘nome da variável‘ (dimensões dos pixels do botão)) que fossem sensíveis ao clique do mouse, de modo que, ao serem ativados, dariam window.clear() da tela em questão e daria window.draw() na tela de opção desejada. Para verificar o clique do mouse, utilizamos a verificação do event.type, durante o ‘.poolEvent’, se foi identificado clique no mouse (‘event.type == Event::MouseButtonPressed’) e se o botão clicado foi o botão esquerdo do mouse (event.mouseButton.button == Mouse::Left). Assim, ao verificar se houve clique no mouse e se o clique foi feito no botão esquerdo, é utilizado um ‘.getposition’ que pega a posição do cursor do mouse dentro da janela e, então, é feito if-else que verifica se o botão criado (utilizando IntRect) contém (utilizamos a função ‘.contanis()) a posição do mouse dentro da sua área clicável.

1.1 - Menu Inicial

A imagem do menu inicial apresenta 3 opções ao usuário:

- Jogar
    - Abre a tela jogável do jogo

- Créditos
    - Abre a tela de créditos dos envolvidos no projeto
    - Contém informações do tipo:
        - Nome
        - RA
        - Retrato pixelado

- Sair
    - O jogo é fechado

1.2 - Tela *Game-Over*

Quando o personagem é capturado por um fantasma, a tela automaticamente é alterada para a tela de derrota. O usuário possui duas opções:

- Voltar ao menu
    - O usuário é redirecionado ao menu inicial do jogo

- Jogar novamente

- O jogo é novamente iniciado - após o clique o personagem reaparece e aguarda mais um clique para o jogo ser iniciado.

1.3 - Tela *Win*

Quando o personagem coletar os cinco coletáveis disponíveis pelo mapa, o jogo é encerrado e o usuário vence. A tela é alterada para a arte de WIN que apresenta o botão de “Retornar ao menu” que permite o usuário retornar ao menu inicial e decidir se quer jogar novamente, ver os créditos do jogo ou sair.

2 - Sprites

Para maior imersão do usuário com o jogo, foram implementados os sprites apropriados para os personagens: um menino e quatro meninas.

Como o nome do jogo sugere, o personagem controlável é um menino nerd de calça, jaqueta e óculos. Sua missão é fugir de 4 meninas, que são identificadas com 4 cores diferentes de cabelo: loira, ruiva, morena e castanha.

Foi criado também um sprite para o ícone do jogo, que é atribuído ao ícone da janela e na barra de tarefas enquanto o programa está aberto. A implementação foi feita utilizando a função ‘.setIcon’.

3- Portais

Para permitir uma maior dinamicidade durante a jogatina, foi implementada em cantos específicos da biblioteca (mapa do jogo) portais acessíveis pelo jogador. Cada portal é identificado com cor e, ao acessá-lo, a posição do personagem é transportada para o outro portal de cor correspondente. As meninas (fantasmas) não são transportadas ao passar pelo portal.

A implementação do portal foi feita utilizando o mesmo raciocínio dos botões clicáveis das telas interativas. Foi criado um retângulo nas dimensões dos pixels dos portais que, assim que as bordas externas da sprite do personagem intersectar as dimensões do retângulo, a interação é confirmada e a posição é atualizada. Foi utilizada a função ‘.intersects’ (intersecta) em função de ‘.getGlobalBounds’ (permite acessar as dimensões e a posição do objeto desenhado (sprite) dentro da janela).

O controle dos portais foi feito utilizando a função ‘.getElapsedTime’ que permite a contagem do tempo passado. Após passar pelo portal, é estabelecido um tempo de cooldown de 0.4 segundos, que impede que o personagem entre em loop de teletransporte entre os portais.

Após acessar o portal, a direção do personagem é atualizada para a direção que diverge da colisão com as paredes, de modo que se o personagem entrar, por exemplo, olhando para baixo olhando para um portal e seja transportado para outro portal onde possua uma parede se continuar olhando para baixo, a direção de

movimentação é atualizada para onde tem caminho livre. Assim, a movimentação entre os portais se mantém fluída e dinâmica.

4 - Coletáveis

Assim como no ‘pacman’ no nosso jogo, o personagem vence a partida se conseguir coletar coletáveis espalhados pelo mapa. Os coletáveis são: lápis, borracha, livro, régua e óculos. Suas posições estão definidas em posições específicas e inalteráveis no mapa. São geradas uma de cada vez e de modo aleatório, ou seja, ao iniciar o jogo, por exemplo, se o primeiro coletável for o livro e depois for a régua, ao reiniciar a partida, o primeiro coletável pode ser uma borracha e seguido do óculos. As posições já estão definidas, mas sua ordem segue um controle de randomização.

A coleta dos coletáveis é feita seguindo o mesmo raciocínio implementado no portal: assim que as dimensões da sprite do personagem intersecta os retângulos interativos com as dimensões do item coletável, este é adquirido e outro é gerado em ordem aleatória pelo mapa.

O controle dos sprites, da área interativa, da variável de controle para verificar se o item foi coletado (booleano que é iniciado como false e, assim que o personagem coletar o item, é atualizado para true. Assim que for feita a randomização do novo coletável a ser mostrado na tela, se for sorteado um coletável já coletado, o controle booleano impede que ele seja gerado na tela e garante que seja feito um novo sorteio) e a ordem que será mostrada os coletáveis, com índices de 0 à 4 (que serão embaralhados usando a função ‘random_shuffle’) foram todos feitos utilizando um vetor da biblioteca Vector, que permite maiores opções de controle das variáveis e dos valores atribuídos a cada uma.

Assim que o colectável for obtido, é mostrado uma miniatura deste na parte superior da tela, funcionando como um score, que permite que o usuário tenha um *feedback* visual de quantos e de quais itens foram coletados até o momento. A impressão dos coletáveis é feita utilizando a mesma variável que é iniciada com *false* e que verifica se o item já foi coletado. Caso a variável seja *true*, o coletável é impresso na sessão de *score*. As posições dos coletáveis são pré-determinadas e não estão em ordem de coleta.

A coleta dos coletáveis está diretamente ligada à movimentação das meninas. No ínicio do jogo, todas possuem movimentação aleatória, logo, se movem sem nenhum destino específico pelo mapa. Após a coleta do primeiro item, uma menina passa a perseguir o personagem. Após o segundo coletável ser pego, a segunda menina é direcionada a perseguí-lo e assim é feito até o 4° coletável. Assim que estiverem as quatro meninas o seguindo, também aumentado, assim que pegar o penúltimo item, a velocidade delas, tornando a conclusão do jogo ainda mais desafiadora para o usuário, dando um gosto a mais e satisfação extra ao jogador após a vitória.

5- Música e efeitos sonoros

Para dar um toque a mais ao jogo, foram implementados arquivos de som ao jogo que respondem conforme ações *in-game.* A implementação foi feita utilizando a própria SFML, por meio da biblioteca `<SFML/Audio.hpp>`.

Assim que o usuário clicar para 'jogar' no menu inicial, inicia-se uma música de fundo que dura enquanto o usuário estiver jogando. Assim que o personagem coletar um item, é atribuído um efeito sonoro que dá um *feedback* auditivo ao usuário. No momento que for pego por uma das meninas, a música de fundo é interrompida, é iniciado um efeito sonoro de som de beijo, a tela fica estática enquanto estiver acontecendo o efeito sonoro (aproximadamente 1.5 segundos) e posteriormente é mostrada a tela de *game-over*. Quando o jogador coletar os cinco itens, a música de fundo é interrompida e é iniciado um efeito sonoro de vitória e a tela de *win* é mostrada na tela do jogador.

6- Movimentação dos fantasmas

Para a movimentação dos fantasmas, foi utilizado diferentes algoritmos para realizar a tarefa, os vilões que se movimentam de forma aleatória são resultado da função 'mover_vilao_aleatorio()', a qual verifica as possíveis direções de movimento sem ir em direção a uma parede, escolhe uma dessas direções, tentando o máximo possível não se movimentar para a direção inversa de onde veio, com uma trava de segurança que, caso ele não consiga se movimentar para outra direção, ele vai para a direção inversa.

Ademais, para a movimentação de perseguição dos vilões, a função 'mover_vilao_bfs()' utiliza um algoritmo de busca reconhecido por jogos nesse estilo, o algoritmo de busca em largura (BFS), onde ele testa diversos caminhos possíveis, desviando de obstáculos, até encontrar o melhor caminho para interceptar o personagem.

Além disso, foi adicionado uma função auxiliar, 'pos_ocupada()', que ajuda as movimentações dos vilões, retornando se já existe algum outro vilão naquela posição, caso sim, esse vilão não pode ir para a posição desejada, caso contrário, o vilão se movimenta.