Manual do Sistema

Michel Hecker Faria 12609690

Sprite

É perceptível que na parte 1 do projeto existiam muitos métodos repetidos, a saber, métodos de movimentação, métodos *getters* e *setters*, colisão, etc. Neste projeto, a solução para sanar essas repetições foi criar a classe `Sprite`, que fica responsável por fornecer todos estes métodos. Dessa forma, todos os objetos que fazem uso dessas funções, que em geral são os objetos interativos como `Spaceship`, `Rock`, etc., são filhos da classe `Sprite`. Com isso, o projeto ficou extremamente mais modularizado e simplificado. Duas funções precisaram do `Overide`, porém, isso foi necessário somente por beleza do código, pois era possível utilizar o método pai. Percebi que ficaria melhor sobrescrever os dois métodos do que manipular desnecessariamente os métodos da classe pai.

Colisão

A colisão dos `Sprite` neste trabalho também foram melhoradas. Desta vez todos os *sprites* possuem um retângulo de colisão. Na classe `Bullet`, há um método que verifica colisões. Sendo assim, basta utilizar o método da `Bullet` e colocar como parâmetro o retângulo de colisão do objeto desejado. Em minha opinião, essa era a melhor forma de verificar colisões, outras formas exigiam muitos `if`.

Funcionamento dos Alienígenas

Na parte 1 do projeto foi adotado uma estratégia diferente da usada nesta parte. Na parte 1 foi utilizada uma matriz de `Integer`, o que, particularmente, não foi do meu agrado, pois existiam muitos `for` durante o código, além de que, da maneira que abordei, foi preciso fazer outras duas matrizes, totalmente desnecessárias. Desta vez, com base nestes erros da parte 1, utilizei um `ArrayList` do tipo `Alien`. Utilizando uma lista, a situação ficou muito mais prática e simples, pois foi possível utilizar o *Java for-each Loop*, o que tornou tudo ainda mais prático.

A classe `Cenario` possui dois métodos de verificação que, dado um parâmetro que indica a posição x, eles retornam se essa posição está em uma das paredes. Dessa forma, não foi preciso repetir métodos de verificação de ”colisão”com a parede. Os alienígenas, por exemplo, fazem uso deste método. Ao se moverem, é chamado este método para verificar se algum *alien* está enconstando na parede, caso esteja, todos os alienígenas se movem para baixo e para direção oposta, trocando a direção de movimento.

Os alienígenas não são logicamente diferentes, somente graficamente. Ou seja, a única diferença entre os alienígenas é a imagem representativa dele. A única exceção para isso é o *alien* especial, que surge a primeira vez quando os monstros mudam de direção. Ele não atira contra o *player*, e se move uma vez, da esquerda para direita. Seu único objetivo é fornecer pontuação extra para o jogador.

Game Manager

Essa classe é a mais importante para o funcionamento do jogo pois ela administra tudo o que acontece no *game*, desde as aparições inicias de todos os personagens, movimentação dos mesmos, tela final, contabilização de pontos e até o movimento da nave. a Classe se divide em três métodos principais:

- `Start()`: Começa o jogo e faz todas as configurações inciais, tal como posições inicias dos *alien*, da nave, desenho do menu, etc.
- `Update()`: Acontece a cada atualização gráfica. É responsável por atualizar o jogo, ou seja, as posições de objetos gráficos que andam, como a nave, os alienígenas, as balas, serão modificadas nesta função. Outras muitas funcionalidades são efetuadas por meio dessa função, como o controle de colisões, controle de *delay*, etc.
- `Finish()`: Finaliza o jogo, mudando para a tela final e mostrando as informações finais.

Som

Foi criada uma classe `Sound` responsável por todo manejamento dos efeitos sonoros e músicas do *game*. A principal classe para som foram as classes `Media` e `MediaPlayer`. De primeira, foi utilizada a classe `Clip`, porém ocorreram alguns problemas e algums usuários do `StackOverflow` recomendaram em um *post* estas classes.

Cada vez que é necessário emitir um som, a classe `Sound` é instanciada, o som desejado é escolhido e é invocada a função `play`. Dessa forma, existem muitas novas instâncias de som durante o andamento do jogo.

Gráfico

A parte gráfica foi muito decisiva para a beleza e qualidade do projeto. De início, comecei utilizando o software *Scene Builder*, porém desisti, uma vez que, utilizando ele, todas as informações relacionadas ao gráfico do *game* ficariam em um único arquivo, o que me desagradou profundamente. Sendo assim, pesquisando, encontrei informações sobre o pacote *Animation* e as classe *Canvas*. A utilização das mesmas pareceram mais práticas e até melhores do que o *Scene Builder*. Sendo assim, migrei para ambas. Realmente, foi mais prático utilizá-las, pois, de bônus, ainda vinha um parâmetro que indicava o tempo atual da animação, o que ajudou muito na criação do jogo.

Acerca do desenvolvimento das imagens, todas foram desenvolvidas por mim, Michel Hecker Faria, e pela aluna do Instituto de Arquitetura e Urbanismo, Cecília Pietra Avelar Diniz, que ajudou na criação de imagens. Portanto, o *background* da tela de menu, os personagens, as rochas, o *background* da tela de jogo, tudo foi criado por nós, o que, em minha opinião, tornou o jogo mais pessoal e divertido, pois nele haviam ideias criativas de arte e design.