7. Inimigos

Para o controle dos inimigos dentro do jogo criamos uma classe chamada Inimigos, a qual herda de Atores, pois assim, teria um Ponto2D, bem como métodos e atributos para controlar colisões, explosões e disparos. Na classe Inimigos, foram declarados atributos próprios dos inimigos, como sua velocidade angular, chamada de vR, e seu ângulo. Também foi implementado um construtor próprio que definia esse angulo e velocidade angular, Inimigos foi declarado como uma classe abstrata, isso porque, não deve ser possível instanciar um Inimigo, apenas suas variações, com a classe abstrata declaramos métodos abstratos sendo eles: dispara() - que sobrescreve o dispara de Atores -, desenha() e atualizaEstado(). Isso foi adotado para que fosse possível controlar todos os Inimigos em uma única lista ligada de Inimigos dentro da classe Main.

Da classe Inimigos, herdam Inimigo1 e Inimigo2, classes as quais definem de fato as características de cada inimigo e podem ser instanciadas. Dentro destas foram implementados os métodos abstratos declarados em Inimigos e um construtor para cada, onde foi definido o raio fixo para cada tipo de inimigo, sendo 9.0 para o inimigo 1 e 12.0 para inimigo 2. Ambas as classes utilizam o método colission() da classe Atores para verificar sua colisão com os projéteis do player, sem alterar a função em nada.

Na classe Inimigo1 seu método dispara apenas verifica se o tempo para seu próximo tiro é maior do que o tempo atual do jogo e se sua posição y é menor que a do player, indicando que ainda não passou do player, caso ambas as condições sejam verdadeiras adiciona um novo projétil a lista ligada de projéteis inimigos, a qual é compartilhada por todos os inimigos, e então atualiza o tempo do seu próximo tiro. No método desenha, caso esteja explodindo desenha sua explosão baseada no tempo de início e fim dela, caso contrário apenas desenha o formato do inimigo 1, sendo este uma esfera ciano, a qual é definida na biblioteca GameLib. Já no método atualizaEstado são atualizadas as posições x e y baseado em sua velocidade e ângulo, caso esta instancia do inimigo não esteja explodindo chama o método de colisão e caso sua explosão tenha terminado, ou tenha saído da tela, retorna falso, indicando que o inimigo deve ser excluído da lista ligada de Inimigos.

Na classe Inimigo2, utilizamos um atributo booleano a mais chamado ShootNow, que indica o momento no qual o inimigo deve iniciar seus disparos, na função dispara verificamos se esse atributo é verdadeiro e caso seja insere um novo projétil na lista ligada de projéteis inimigos, baseando-se em um array de ângulos para calcular a direção que esse projétil deve seguir. No método desenha, a única diferença é o formato desenhado que ao invés de uma esfera ciano é um losango magenta. Já na classe atualizaEstado alguns cálculos são feitos para definir a movimentação do inimigo, já que ele se movimenta em uma elipse, por isso utilizamos algumas variáveis adicionais como a previousY e a threshold que servem para verificar as posições verticais dos inimigos e da tela, utilizando dessas variáveis a movimentação do inimigo é feita e o atributo ShootNow é atualizado,

depois a função dispara é chamada e são feitas as mesmas verificações de estado explodindo e posição que são feitas em inimigo 1.

9. Arquivos e Main

O jogo possuí um sistema de fases, no qual cada fase possuí uma estrutura previamente definida com quais inimigos e bosses serão instanciados em quais posições e momentos específicos, essas fases são definidas em um arquivo de configuração Config.txt encontrado dentro de uma pasta com o nome de arquivos, esta pasta se encontra dentro do diretório principal do nosso jogo, o diretório src, dentro da pasta arquivos se encontram os arquivos de configuração específicos para cada fase e o arquivo Config.txt. A leitura desses arquivos bem como a preparação das fases é feita dentro da classe Main e do método main, para tal, utilizamos de uma classe auxiliar chamada Instancia, a qual como o próprio nome diz armazena atributos específicos para cada instancia do jogo, esses atributos são os passados nos arquivos de configuração, são eles: nome da instancia (CHEFE ou INIMIGO), tipo (1 ou 2), vida, tempo, x e y. Utilizando dessa classe auxiliar, dentro da main criamos um array de listas ligadas de instancia, onde cada posição do array representa uma fase do nosso jogo, e cada lista ligada possuí dentro dela todas as instancias que precisarão ser inseridas no jogo.

No método *main*, portanto, definimos uma variável booleana que define o estado do nosso jogo, se for *true*, o jogo está em execução e se for *false* o jogo deve ser finalizado, temos também duas variáveis que armazenam o tempo inicial do jogo e a variação de tempo entre os *loops* principais do jogo. Definido essas variáveis, definimos o caminho para o arquivo de configuração do jogo, que caso o jogo esteja sendo executado pelo diretório src, será “arquivos/Config.txt” e caso não será “src/arquivos/Config.txt”, definido esse caminho, abrimos nosso arquivo de configuração utilizando a classe File da biblioteca java.io e utilizamos um *scanner* para iterar por este arquivo. Definindo a vida do player, a quantidade de fases e criando o *array* de fases baseado nesta quantidade. Então iteramos por cada um dos arquivos de fase e adicionamos na nossa lista ligada para aquela respectiva fase uma instancia com cada um dos atributos passados no arquivo.

Ainda na *main*, criamos uma lista ligada para os projeteis do *player*, instanciamos um *player*, criamos a lista ligada de inimigos, de projeteis inimigos, um iterador para iterar pela lista ligada de fases, um *array* para controlar as estrelas do nosso cenário, uma lista ligada de projeteis do *boss*, uma instancia de boss que será atualizada em cada fase, uma lista ligada de *power ups* e por fim diversas flags de controle que serão utilizadas para excluir projeteis e inimigos das respectivas listas ligadas.

No método *main*, portanto, definimos uma variável booleana que define o estado do nosso jogo, se for *true*, o jogo está em execução e se for *false* o jogo deve ser finalizado, temos também duas variáveis que armazenam o tempo inicial do jogo e a

variação de tempo entre os *loops* principais do jogo. Definido essas variáveis, definimos o caminho para o arquivo de configuração do jogo, que caso o jogo esteja sendo executado pelo diretório src, será “arquivos/Config.txt” e caso não será “src/arquivos/Config.txt”, definido esse caminho, abrimos nosso arquivo de configuração utilizando a classe File da biblioteca java.io e utilizamos um *scanner* para iterar por este arquivo. Definindo a vida do *player*, a quantidade de fases e criando o *array* de fases baseado nesta quantidade. Então iteramos por cada um dos arquivos de fase e adicionamos na nossa lista ligada para aquela respectiva fase uma instancia com cada um dos atributos passados no arquivo.

Ainda na *main*, criamos uma lista ligada para os projeteis do *player*, instanciamos um *player*, criamos a lista ligada de inimigos, de projeteis inimigos, um iterador para iterar pela lista ligada de fases, um *array* para controlar as estrelas do nosso cenário, uma lista ligada de projeteis do *boss*, uma instancia de boss que será atualizada em cada fase, uma lista ligada de *power ups* e por fim diversas flags de controle que serão utilizadas para excluir projeteis e inimigos das respectivas listas ligadas. No nosso *loop* principal do jogo, ou seja, um *while* que funciona enquanto a variável *running* (que define o estado do jogo) for *true,* nós atualizamos as variáveis do tempo atual do jogo e chamamos as funções atualizaEstado(...) e desenha(...) do *player* além disso, conferimos se a instancia atual no nosso iterador das fases é um inimigo ou chefe e instanciamos o respectivo inimigo ou chefe, caso seja um inimigo instanciamos ele na nossa lista ligada de Inimigos e caso seja um boss apenas instanciamos ele e atribuímos seu ponteiro a variável criada anteriormente na main, claro, sempre conferindo qual tipo de inimigo ou chefe estamos instanciando e atualizando o iterador para a próxima instancia da lista.

Depois de instanciados chamamos as funções de atualizaEstado e desenha para cada elemento do nosso jogo, ou seja, inimigos, chefes, projéteis de cada um deles, cenário e *powerUps.* Fazemos isso iterando pela lista ligada de cada um dos elementos e sempre conferindo o retorno do método atualizaEstado, para que caso seja falso a instancia atual seja deletada da lista. Na verificação do *boss* também implementamos para que caso ele seja destruído, o iterador das fases seja alterado para a próxima fase, ou finalize o jogo caso não existam mais fases, por fim, fazemos as verificações de fim de jogo, ou seja, caso o player não tenha mais vidas restantes ou caso a tecla ESC tenha sido pressionada, o jogo é finalizado.