1. Introdução:

1.1) Orientação a Objetos

A programação orientada a objetos(POO) é um dos principais paradigmas da programação. A orientação a objetos permite que os códigos sejam encapsulados em “pacotes” – que são conhecidos como classes –, o que torna possível a abstração e também uma melhor reutilização do código. Os objetos, responsáveis pelo nome “Orientada a Objetos”, são instâncias de uma classe, que possui métodos e variáveis específicos, com um valor “único” para cada objeto. Com a utilização do POO é possível criar tipos mais complexos de objetos, quebrando a barreira “imposta” pelos tipos primitivos, além de tornar seu código mais estruturado. (Anotações de aula)

1.2) Polimorfismo e Herança

Quando uma classe estende outra, ela herda todos os métodos e atributos que estão na classe mãe (ou superclasse), organizando assim as classes de forma em que a superclasse é a classe mais geral, e conforme existem suas “filhas”, as classes passam a ser cada vez mais específicas.

É possível sobrescrever os métodos da classe mãe, fazendo assim cada classe ter um comportamento diferente. Por exemplo, cães e gatos “estendem” mamíferos e fazem um som diferente, como se o método “fazSom()” tivesse sido sobrescrito em cada um deles.

O polimorfismo também pode ser exemplificado como dito acima. Ambas as classes (cães e gatos) herdaram o método fazSom() da superclasse(mamíferos) e, o fato desse método ter um resultado diferente em cada um deles (um cachorro latir e o gato miar) “ocorre” por conta do polimorfismo. (Anotações de aula)

1.3) Multiprogramação

Embora pareça, o computador ainda não é capaz de fazer duas, ou mais, coisas “ao mesmo tempo”. O que ocorre é o chaveamento dos processamentos no processador de forma tão rápida que esse chaveamento é imperceptível.

Tal ato porém foi perceptível no desenvolvimento do jogo, uma vez que era necessário remover um Asteroide da lista, enquanto essa era percorrida para imprimir seus elementos. Ao tentar fazer isso o erro de Concurrent Modification aparecia. Para resolver tal problema foram criadas as seções críticas no código. Nessas seções o processador é impedido de chavear o processamento, ou seja, somente executa um processo por vez. (Anotações de aula)

2. Objetivos

Aprender a utilizar os conceitos que foram ensinados na sala de aula – POO, multiprogramação, herança e polimorfismo – de forma prática. Desenvolver o jogo Asteroids do atari.

3. Problemas

No desenvolvimento do jogo, foi possível perceber que além do “problema” da lógica, existe também o problema de que desenvolver um jogo não é somente saber programar ou alguma linguagem, mas também realizar algoritmos para: descrever a física do jogo, detectar colisões, e o que fazer em cada colisão, além da movimentação de cada elemento do jogo.

3.1) Física

Por se passar no universo não existe atrito, portanto se algum elemento é acelerado ele não para de se mover até que seja imposta um força de mesma intensidade no sentido oposto do movimento. Os elementos, quando ultrapassam a tela, aparecem no outro canto, dando a ideia de que o universo do jogo é infinito.

3.2) Colisões

Os algoritmos de colisões foram feitos da seguinte maneira. Para os Asteroides e os Misseis se a distância das coordenadas de seu centro forem menores que a soma dos raios, houve colisão, conforme mostrado na imagem 1. No caso da Nave, não existe colisão com o míssil, apenas com os Asteroides. A colisão da Nave com o Asteroide foi dividida em duas partes. A primeira realiza uma verificação igual à do Asteroide com a Nave, a Nave está circunscrita em um círculo, cujo o centro foi utilizado para a primeira etapa da verificação, caso ocorra a colisão de um Asteroide com o círculo, o qual a nave esta circunscrita, é efetuada uma segunda verificação que irá ser mais especifica. Nessa segunda etapa da verificação vemos se o Asteroide passou dentro da Nave. O algoritmo dessa parte foi construído da seguinte forma, foi calculado as equações das 3 retas da Nave para achar a região de “dentro” da Nave, caso o Asteroide possua algum ponto que esteja dentro dessa região houve colisão, conforme mostrado na imagem 2.

Imagem 1: Método testaColisão() em Missel

```
28      @Override
29      public boolean testaColisao(Elemento other) {
30          List<Elemento> lista = Universo.getInstance().listaDeElementos;
31          for (Elemento show : lista) {
32              if (show instanceof Asteroide) {
33                  double soma = Math.pow((show.x - this.x), 2) + Math.pow((show.y - this.y), 2);
34                  double d = Math.sqrt(soma);
35                  if (d <= (this.raio + show.raio)) {
36                      show.vivo = false;
37                      if(show.raio == 25){
38                          if(!Universo.getInstance().f){
39                              s.explosaoSound.play();
40                          }
41                          Universo.getInstance().score += 1;
42                          show.raio = 10;
43                          Asteroide asteroid = new Asteroide(show.x, show.y, -show.vx, -show.vy, 10, true);
44                          Universo.getInstance().adicionar.add(asteroid);
45                      }else{
46                          if(!Universo.getInstance().f){
47                              s.explosaoSound.play();
48                          }
49                          Universo.getInstance().score += 5;
50                          Universo.getInstance().remover.add(show);
51                          Universo.getInstance().criaAsteroid((int)(show.x*2.6), (int)(show.y*2.9));
52                      }
53                      Universo.getInstance().remover.add(this);
54                      return true;
55                  }
56              }
57          }
58          return false;
```

Foi necessário a importação da lista “listaDeElemtentos” pois o método “testaColisao()”, que foi herdado da classe Elemento, recebe apenas um elemento como parâmetro e é necessária a verificação de todos os Asteroides que estão no jogo.

Imagem 2: Método testaColisao() em Nave

```
168     @Override
        public boolean testaColisao(Elemento other) {
170         double soma = Math.pow((other.x - this.x), 2) + Math.pow((other.y - this.y), 2);
171         double d = Math.sqrt(soma);
172         if (f) {
173             return false;
174         }
175         if (other instanceof Missil) {
176             return false;
177         }
178         if (d <= (this.raio + other.raio)) {
179             double reta1 = -2.6 * ((other.x) - (this.x - 10)) + (this.y + 13);
180             double reta2 = 2.6 * ((other.x) - (this.x + 10)) - (this.y + 13);
181             double reta3 = 1 * (other.x - this.x) + (this.y + 13);
182             if (other.y > (int) reta1) {
183                 if (other.y < (int) -reta2) {
184                     if (other.y < (int) reta3) {
185                         return true;
186                     }
187                 }
188             }
189         }
190         return false;
191     }
```

3.3) Movimentação

Foi definida uma velocidade máxima para a Nave, portanto um vez que a Nave atinge essa velocidade a aceleração passa a ser 0. A seta para cima acelera a Nave na direção em que ela está apontada e a seta para baixo acelera a Nave na direção contrária. As setas para os lados giram a Nave e o espaço é utilizado para atirar.

3.4) Easter Egg

Foi introduzido também um Easter Egg no jogo. Quando apertada a tecla F o universo começa a piscar em cores aleatórias, a Nave começa girar de forma frenética e a atirar – os tiros saem de forma mais rápida nesse modo – para todos os lados enquanto gira, começa a tocar a música “Shooting Stars – Bag Raiders” de fundo e as colisões com a Nave são desativadas. É possível desativar apertando a tecla F novamente.

3.5) Score

Quando se destrói um Asteroide grande é somado 1 ao Score, e quando destruído um Asteroide pequeno é somado 5 ao Score. Os valores foram escolhidos de forma aleatória.

3.6) Respawn dos Asteroides

Quando um Asteroide grande é atingido ele se transforma em dois Asteroides pequenos, e sempre que um Asteroide pequeno é destruído, surge outro Asteroide em uma posição aleatória do Universo.

4. Conclusão

Após a finalização do projeto é notável minha melhoria em todos os conceitos que foram ensinados na sala de aula e que foram utilizados nesse projeto. Além dos conceitos ensinados na aula, foi necessária a pesquisa de algumas coisas na internet – como para colocar algo escrito no JPanel e alterar sua fonte – para fazer algo além do que foi pedido. Acredito que tal ato é importante para desenvolver a habilidade de entender o conceito por trás do código e o seu funcionamento sem a necessidade de um professor a todo instante explicando tudo, pois chegará um momento em que não terei um professor para explicar todas as minhas dúvidas (infelizmente hehe).

Foram realizados todos os desafios que foram impostos pelo projeto, entretanto vejo que algumas melhorias são possíveis, tais como: melhorar a performance, as vezes o jogo da uma travada ao executar pela primeira vez os sons; colisão, algumas vezes a colisão da Nave e do Asteroide não funcionam tão perfeitamente quanto eu gostaria; melhorar o código, é possível tornar o código mais legível e limpo; e por último, acrescentar mais funções, como um menu e a possibilidade de um reset após o Game Over.