Clone do jogo Asteróides

Técnicas de Programação 1

- Brenno Pereira Cordeiro - 190127465
- Joao Vitor Maia Ferreira - 190110007

1. Descrição de Problema

Neste projeto, desenvolvemos um clone do jogo Asteroids usando a linguagem de programação Java e os conceitos de programação orientada a objetos.

Asteroids é um jogo do tipo arcade, lançado em Novembro de 1979 pela Atari, Inc. O jogador controla uma nave espacial num campo de asteróides, e deve atirar e destruir os asteróides enquanto evita colidir com os mesmos.

Figure 1: Screenshot do Jogo

Nosso objetivo é implementar um jogo similar, com uma nave espacial se movendo livremente pelo tela do jogo, e asteróides errantes que ameaçam colidir com o jogador. No entanto, nosso objetivo é bem mais simples, pensando apenas em implementar o fluxo básico do jogo, para desenvolver a prática dos conceitos de orientação à objeto.

2. Regras de Negócio

1. O *jogador* pode se mover livremente pela tela do jogo, usando as teclas `a` e `d` para rotacionar a nava, e a tecla `w` para adicionar propulsão.
2. Ao sair da tela do jogo, o **fim do jogo** deve ocorrer.
3. Durante o jogo, *asteróides* aparecem em posições aleatórias.
4. Um *asteróide* deverá ser destruído ao sair da região da tela.
5. O *jogador* é capaz de atirar um número ilimitado de *balas* com uma certa cadência.

6. Uma *bala* tem uma direção inicial, e se move nessa direção com velocidade constante.
7. Uma *bala* deverá ser destruída ao sair da região da tela.
8. Um *asteróide* é destruído ao colidir com uma *bala*.
9. Uma *bala* é destruída ao colidir com um *asteróide*.
10. A *pontuação* é incrementada quando um *asteróide* colide com uma *bala*.
11. O **fim do jogo** ocorre quando um *asteróide* colide com o *jogador*.
12. Ao final do jogo as pontuações mais altas ficam salvas em um arquivo.

3. Diagram de Classes

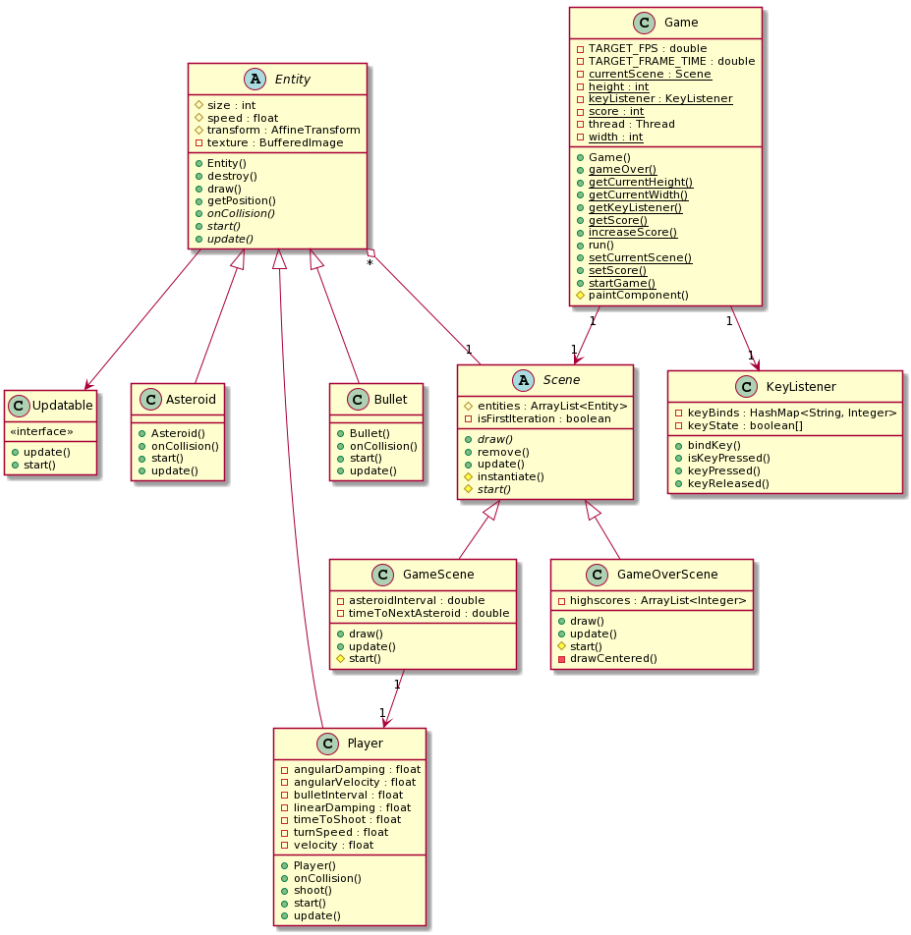

Figure 2: Diagrama de Classes

Game

A classe Game é responsável por selecionar a *cena* atual e criar o *loop de jogo*. O loop de jogo é um loop infinito, que guarda o tempo que levou para completar. Isso é importante para que nossa simulação de física precisa levar em conta o tempo nos cálculos.

Cada iteração do loop de jogo vamos chamar de *frame*, e o tempo levado por um *frame* vamos chamar de *delta time* ou `dt`. A cada frame a classe Game chama o método `update` da cena atual, informando o delta time para o método. A implementação do loop de jogo é baseada no artigo Fix Your Timestep!.

Scene

A classe Scene, ou cena, representa uma composição de *entidades*. Essa classe é responsável por instanciar as entidades, atualizá-las, desenhá-las na tela e destruí-las.

Cada cena do jogo herda dessa classe, e implementa sua própria lógica para interagir com o ciclo de vida.

Entity

A classe abstrata Entity, ou *entidade*, é uma abstração de todos os elementos que serão desenhados na tela. Uma entidade possui tamanho, velocidade, um Transform, a cena que o criou e uma textura.

A cada frame, a textura da entidade é desenhada na tela, na posição e rotação definidas pelo seu Transform.

A entidade implementa os métodos `destroy`, para sinalizar a sua cena que deve ser destruída, e os métodos abstratos `update`, chamada a cada frame com o delta time, e `onCollision`, chamado quando ocorre uma colisão envolvendo aquela entidade.

Asteroid

Asteroid herda da classe Entity. No seu método `update`, se move em uma direção com velocidade constante e verifica se sua posição saiu da região da tela.

Bullet

Bullet herda da classe Entity. Em seu método `onCollision` chama seu método `destroy()` caso tenha uma colisão com uma *Entity* do tipo *Asteroid*, isso foi feito para que um *Bullet* não anule um outro do mesmo tipo.

Player

Player herda da classe Entity. Em seu método `onCollision` chama o método `Game.gameOver()` caso o objeto colidido seja do tipo *Asteroid*.

Possui também o método `shoot()`, responsável por instanciar um novo objeto do tipo *Bullet* na *Scene*.

Scene

Scene é uma classe abstrata. Seu método principal, o `update(double dt)`, é responsável por fazer as atualizações na tela de acordo com as ações do jogo e com o tempo *dt* passado, nesse método chama o update de toda sua lista de entidades, checa por colisões, e ao fim adiciona/remove entidades que colidiram ou foram criadas.

Game

Game herda da classe JPanel, é responsável por inicializar os principais componentes como tela, *timer* e *KeyListener*. Também cria a instância da classe GameScene.

GameScene

GameScene herda da classe Scene. Em seu método `start()` realiza o binding dos códigos de cada tecla para uma string, facilitando a leitura do código em outras classes, também é responsável por controlar o tempo em que novos `Asteroids` são criados já que também é responsável pela criação dos mesmos.

GameOverScene

Essa classe representa a tela de **fim de jogo**. Ela é responsável por mostrar a mensagem de **fim de jogo** ao usuário, assim como ler e escrever no arquivo de pontuações, com as 5 maiores ponturações.

KeyListener

Essa classe guarda o estado dos 256 códigos do teclado. Durante o `update` das entidades, é possível obter o estado de uma determinada tecla para executar uma determinada ação. Também é possível associar um nome a um código, para manter o código mais legível.