Relatório de Desenvolvimento do Space Invaders –Programação Orientada a Objetos

Gabriela Panta Zorzo - 20280527-1

Morgana Luiza Weber - 20103601-9

Turma 128

- **Descrição do problema**

Complementar as funcionalidades do jogo Space Invaders conforme solicitadas pelo professor:

1. Devem existir diferentes tipos de personagens que são representados por uma hierarquia de herança. Cada tipo de personagem tem uma aparência e um comportamento diferente. Por exemplo, alguns apenas se movimentam lentamente da esquerda para a direita e da direita para a esquerda, e sempre que chegam numa extremidade eles descem. Alguns podem ter um movimento semelhante, mas também podem atirar. Outros também podem ser mais difíceis de eliminar, necessitando receber dois ou mais tiros para serem eliminados. A classe Ball serve apenas para demonstrar como um personagem pode ser implementado. Deve ser removida da implementação final. Por fim pelo menos um dos tipos de personagem deve prever comportamento de grupo. Comportamento de grupo significa que a lógica que movimenta os personagens faz com que eles se movimentem em conjunto e não isoladamente.

2. Os personagens devem, obrigatoriamente, serem derivados das classes fornecidas explorando adequadamente os conceitos de programação orientada a objetos.

3. A interface com o usuário deve ser aprimorada usando JavaFX (contagem de pontos, início e suspensão do jogo, configurações etc.).

4. O jogo deve ser capaz de manter a relação das 10 melhores pontuações em arquivo. Ao final de cada jogo deve ser apresentado o escore obtido pelo jogador e o ranking atual.

O Jogo deve seguir as seguintes restrições solicitadas:

1. Devem existir pelo menos 4 tipos de personagens invasores diferentes.

2. Pelo menos um dos tipos de invasores deve ser capaz de atirar contra o canhão.

3. Pelo menos um dos tipos de invasores deve prever comportamento de grupo.

4. O objetivo do jogo deve ser eliminar todos os invasores antes que eles cheguem na parte de baixo da tela.

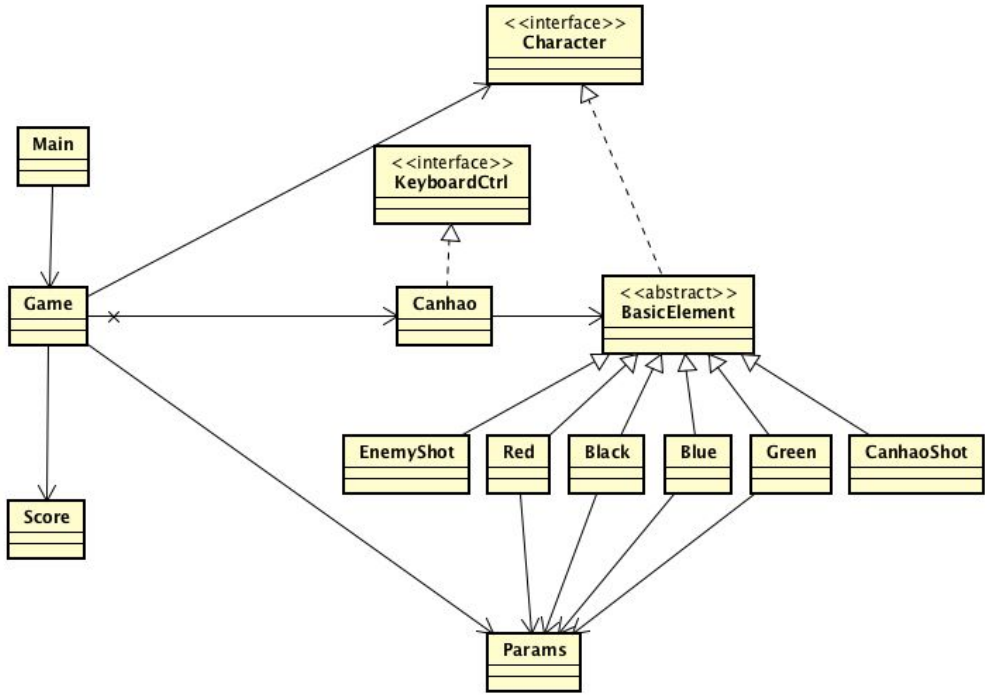
5. Conforme o usuário elimina invasores, novos tipos invasores vão surgindo. A cada nova “fase”, ou seja, quando surgem novos invasores, a velocidade deles deve aumentar, de maneira a aumentar complexidade do jogo. Cada invasor eliminado pontua conforme seu tipo.

6. Se o canhão é atingido ele é destruído. O jogo inicia com 2 canhões reserva. Se todos os canhões disponíveis forem destruídos o jogo acaba.

7. A implementação deve ser capaz de detectar o final do jogo: ou um invasor chega na parte de baixo da tela ou todos os canhões são destruídos. O objetivo é fazer o maior número de pontos antes que o jogo acabe.

- **Diagrama de classes do sistema**

O programa foi desenvolvido seguindo o diagrama de classes abaixo:



Originalmente o jogo contava com dois personagens derivados de BasicElement: Ball e Shot. Com os novos ajustes, Shot foi transformado em CanhaoShot e EnemyShot, enquanto Ball foi transformado em Red, Black, Blue e Green. Além disto, foi implementada uma classe Score, que armazena os pontos ao fim do jogo, grava esses dados em um arquivo, realiza a leitura do arquivo quando é inicializada novamente, além de ordenar a pontuação total em um ranking da maior para a menor.

- **Principais algoritmos desenvolvidos**

1. **testaColisão():** verifica se personagens invasores colidem com eles mesmos (Red, Blue, Green, Black), se colidem com um tiro do canhão, ou ainda, se personagens invasores colidem com os tiros do personagem invasor Red, ou se estes tiros colidem com o canhão.

2. **update():** método chamado de acordo com o deltaTime que é definido como (currentNanoTime - lastNanoTime). Este método é sobrescrito em cada um dos Characters de acordo com suas necessidades específicas, mas de forma geral, busca atualizar a pontuação do jogo, o número de vidas do jogador, o deslocamento dos personagens, se houve uma colisão ou não e verifica se o jogo acabou.
3. **activateEnemy():** método chamado a cada novo nível para ativar os invasores de acordo com o nível que eles devem ser inseridos. Ele utiliza um comando switch para validar o nível de cada inimigo e ativar eles adequadamente.
4. **testLevel():** método chamado a cada momento que um personagem é eliminado para verificar se todos os personagens daquele tipo já foram eliminados. Caso não tenha mais personagens daquele tipo, o jogador passa de nível e é chamado o método activateEnemy() para ativar novos invasores.
5. **incPontos():** método para incrementar os pontos do jogador a cada vez que um invasor é eliminado.
6. **update() [classe Score]:** método para atualizar o ranking de pontuação através do carregamento de um arquivo score.txt, buscar a pontuação final para ser atualizada, ordenar o ranking do maior para o menor e gravar em arquivo o novo ranking atualizado.
7. **imprime(): [classe Score]:** método para imprimir o ranking com os 10 primeiros colocados de acordo com as melhores pontuações.

- **Dificuldades e lições aprendidas**

Uma das principais dificuldades encontradas foi o entendimento do código desenvolvido por outras pessoas, levando em conta sua complexidade e extensão. Além disso, trabalhar com interfaces gráficas foi muito diferente de tudo que já havíamos visto até o momento. Acreditamos que teria sido interessante ter uma prática deste modelo anteriormente, não somente para a realização do trabalho final da disciplina.

Em virtude disso, algumas lições foram aprendidas, como que devemos sempre buscar compreender o código e seu funcionamento como um todo antes de começar a desenvolver e realizar as demandas solicitadas. Também, em alguns momentos será necessária a busca por conteúdos além da sala de aula, seja com o auxílio de outros colegas ou em bibliotecas de códigos.