Membros do grupo:
- Karen Miyuki Massuda (Nro. USP: 14585868 / Turma: 94)
- Vítor Benvenisti Laguna Navarenho (Nro. USP: 14749425 / Turma: 94)
- Henrique Naoki Teruya (Nro. USP: 14578324 / Turma: 94)
- Vivian Ayumi Miamura (Nro. USP: 14835140 / Turma: 04)

Relatório – Projeto COO (2024)

Críticas ao código original do jogo.

O programa apresenta muito código duplicado.

O gerenciamento das diferentes entidades é realizado através de arrays estático.

Muitas variáveis são criadas para gerenciar uma única entidade, mesmo quando todas poderiam ser unificadas em um único objeto para facilitar sua manipulação e encapsulação de suas funcionalidades.

O programa original viola todo e qualquer princípio SOLID existente, não havendo nenhuma divisão de responsabilidade entre os diferentes aspectos do funcionamento do jogo.

Descrição e justificativa para a nova estrutura de classes/interfaces adotada.

Primeiro abstraímos as entidades do jogo em duas classes que contém as funcionalidades delas esperadas: Enemy e Player.

Enemy é abstrato, para que possa ser devidamente especificado pelas classes que herdam dele.

Para gerenciar as diversas instâncias de Enemy no Código, criamos EnemyManager.

EnemyManager também é abstrato, e é especificada conforme o tipo de inimigo que ela deve gerenciar.

As entidades disparam projéteis, portanto a classe abstrata Projectile foi criada, juntamente com suas especificações PlayerProjectile e EnemyProjectile.

Para gerenciar todos os projéteis disparados criamos a classe ProjectileManager e suas especificações: EnemyProjectileManager e PlayerProjectileManager.

Para que as funcionalidades relacionadas a Enemy pudessem ser centralizadas (EnemyManager e EnemyProjectileManager), a classe EnemyController foi criada.

Quanto a lógica geral do jogo, dividimos as funcionalidades nas sequintes classes: InputManager, StateManager, BackgroundManager e CollisionManager.

InputManager gerencia as entradas do usuário.
BackgroundManager gerencia o plano de fundo.
CollisionManager monitora as colisões que acontecem.
StateManager controla os estados em que as entidades se encontram.

Com essa nova estrutura de classes a responsabilidade pelas diferentes funcionalidades do jogo foi dividida entre os objetos apropriados.

Descrição de como as coleções Java foram utilizadas para substituir o uso de arrays.

No lugar de Arrays estáticos, quando apropriado, foram implementados ArrayLists. Nos outros casos a necessidade de arrays foi tornada obsoleta pela reestruturação do código.

Para garantir que vazamentos de memória não ocorram devido ao uso de ArrayLists, criamos a classe GarbageController e adicionamos métodos dump() nas classes que utilizam essas estruturas dinâmicas.

O método dump() remove das ArrayLists todas as entidades cujo estado é State.INACTIVE continuamente ao longo de todas as iterações do programa.

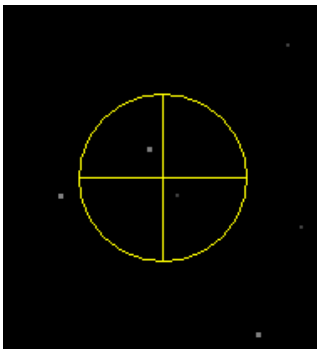
Descrição de como as novas funcionalidades foram implementadas e como o código orientado a objetos ajudou neste sentido.

A reestruturação do Código utilizando as boas práticas da programação orienta à objectos simplificou imensamente a implementação das novas funcionalidades.

Quando adicionávamos novas funcionalidades precisávamos apenas estender as classes certas e adicionar a nova classe nos locais apropriados, ou até mesmo apenas adicionar um método em uma classe já existente, e com isso a funcionalidade já ficava rapidamente pronta.

Implementamos as seguintes funcionalidades no jogo:

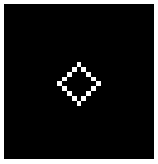
- *Adicionamos um novo tipo de inimigo*



Conheça o SlasherMcDasher, ele é uma bola amarela determinada a esmagar a nave do Player.

Para implementá-lo estendemos a classe Enemy, e criamos também um gerenciador que estende a classe EnemyManager, assim rapidamente adicionando essa entidade no jogo graças à estrutura orientada a objetos.

- *Adicionamos um power-up*



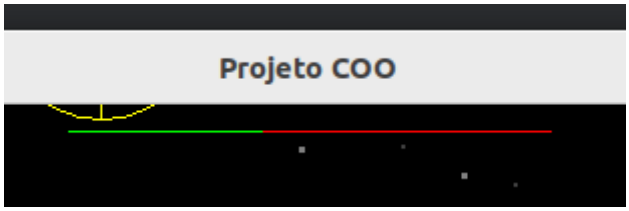
Difícil de pegar: ele é rápido, pequeno e aparece com uma baixa frequência. O único aliado do Player, mesmo que um tanto quanto desinteressado em ajudar.

Para obtê-lo o Player precisa enconstar nele com a sua nave, mas ele vai fugir! Atirar nele o destrói.

- *Adicionamos pontos de vida para o jogador*

Agora o jogador começa com 10 pontos de vida, ou seja, ele deve ser atingido 10 vezes por um projétil ou uma colisão para que ele exploda.

Adicionamos também uma barra de vida, que pode ser vista na imagem abaixo. Ela atualiza conforme o jogador perde pontos de vida.



Ela foi implementada através do método drawHealthBar() na classe Player, e é desenhada toda vez que o método draw() do Player é chamado.

==========

Membros do grupo:
- Karen Miyuki Massuda (Nro. USP: 14585868 / Turma: 94)
- Vítor Benvenisti Laguna Navarenho (Nro. USP: 14749425 / Turma: 94)
- Henrique Naoki Teruya (Nro. USP: 14578324 / Turma: 94)
- Vivian Ayumi Miamura (Nro. USP: 14835140 / Turma: 04)