Universidade Federal do ABC

Haskell Breakout

Paradigmas de Programação, A Noturno, Santo André.

Luiz Felipe Leal Gomes Azzolini RA: 11048715

Introdução.

Breakout foi um famoso jogo da plataforma *Atari* lançado no ano de 1976, suas mecânicas são básicas, o jogo consiste em basicamente três elementos, uma bolinha, tijolos e uma raquete. O objetivo do jogo é que o jogador quebre todos os tijolos com a bolinha, sem deixa-la cair.

O projeto visa implementar um clone do jogo de modo a explorar conceitos de programação funcional e específicos da linguagem Haskell, dita como puramente funcional

Estrutura do projeto.

O projeto foi dividido em vários arquivos de modo a manter sua organização, são eles:

1. **Main.hs**: responsável por centralizar renderização e carregamente de assets
2. **Breakout.hs**: mantém a estrutura de mundo, seus parâmetros gerais e valores iniciais do jogo
3. **KeysController.hs**: Define teclas de controle do jogo
4. **PaddleController.hs**: controla a movimentção da raquete
5. **TileModel.hs**: guarda estrutura dos tijolos no jogo e a programção em relação ao estado e escolha da figura de cada um dos tijolos em jogo
6. **BallController.hs**: arquivo que gerencia toda a interação da bolinha com o ambiente, incluindo colisões e movimentação e decremento de estado dos tijolos

Conceitos utilizados.

Foram utilizados vários conceitos de programação funcional e haskell, listados abaixo

1. **Pattern Matching**: característica da linguagem em que pode-se definir um comportamento específico para uma função a partir de sua entrada
2. **data types**: em haskell é possível definir estruturas de dados parecidas com objetos, de modo que esses tipos podem
3. **Imutabilidade**: no paradigma funcional isso quer dizer que a partir do momento que um valor é atribuido a um espaço de memória ele não pode ser mais ser mudado, porém ainda pode ser transformado por uma aplicação de função
4. **High order functions**: funções de alta ordem, nada mais são que funções que recebem outras funções como parâmetro, como exemplos mais clássicos podemos pensar no *filter* ou no *map* do haskell
5. **Paralelismo**: para que o programa se tornasse mais eficiente algumas computações foram feitas paralelamente ao programa principal, desse modo é possível que *threads* diferentes façam o processo paralelamente, assim a execuçãose torna mais eficiente.

Como utilizar.

Para utilizar o programa basta executar a linha de comando abaixo, dentro da pasta raiz do projeto:

```
stack run
```

Para a execução no WSL2 usando a distro **Ubuntu18.04** foi necessário instalar os pacotes freeglut3 freeglut3-dev com o comando:

```
sudo apt install freeglut3 freeglut3-dev
```

Dificuldades.

Durante a implementação algumas dificuldades foram encontradas. Uma delas foi a introdução de bitmaps externos no programa, como haskell é uma linguagem que visa evitar efeitos colaterais, entender como é feita a leitura de arquivos externos é um desafio, porém utilizando de modo correto sua implementação e utilização da forma como o programa foi montado é bastante simples.

Outro ponto de dificuldade foi arquitetar o sistema de modo legível e de fácil alteração. O problema foi resolvido após bastante esforço de modo que agora é possível definir facilmente um vetor de *Tile* definindo sua posição e estado, de modo que o estado exprime o numero de batidas necessárias para se quebrar o tijolo.

Um *bug* que não foi resolvido é que dada uma colisão o programa interpreta como duas de uma vez, dessa forma os tijolos se quebram mais facilmente do que deveriam, porém nos testes feitos, não atrapalharam a execução do programa.

Pontos de melhoria.

Além do *bug* citado anteriormente, faltaram pontos desejados a serem implementados, como *"poderes"* que cairiam dos tijolos ao serem quebrados, um deles já foi implementado e é acessível através da tecla *espaço* para testes, ao aperta-la no próximo contato que a bolinha tiver com a raquete, ela será grudada e pode ser movida com a mesma, para solta-la basta apertar a mesma tecla novamente.

Além deste foram planejados mais 3 poderes, desaceleração da velocidade da bolinha, triplicar o numero de bolinha em jogo e bolinha com maior dano aos tijolos e seriam implementadas da seguinte forma:

1. **velocidade da bolinha**: já existe um parâmetro reservado para este propósito dentro do tipo *Game*, dessa forma ao dar esse poder bastaria altera-lo em tempo de execução.

2. **triplicar o numero de bolinhas**: seria implementado um vetor de bolinhas dentro do tipo *Game*, de forma similar a que foi feita com os tijolos, de modo que cada uma teria sua própria posição em tela, colocando assim parâmetros como *ballPos*, que estão em *Game* atualmente, dentro de um outro tipo chamado *ball*.

3. **bolinha pesada**: esse poder seria implementado de forma parecida a anterior um parâmetro dentro do tipo *ball* seria utilizado como o dano causado ao estado do *Tile*, dessa forma decrementando de 2 em 2 ou mais por colisão.