projeto de Laboratórios de Informática 1
Grupo 145

Ricardo Petronilho (81744) Joaquim Simões (77653)

3 de Janeiro de 2017

Resumo

Nesta unidade curricular foi proposta a resolução de um projeto informático com dimensão média utilizando a linguagem de programação funcional *Haskell*. O objetivo do projeto é a recriação do jogo *Bomberman*. Ao longo do projeto deparamo-nos com vários desafios, alguns mais complexos, no entanto através de muito treino, estudo e, por vezes, mesmo tentativa erro, conseguimos superar todos esses desafios.

Neste relatório é apresentado a organização e estruturação do projeto, bem como alguma funções importantes para a concretização do mesmo.

Conteúdo

1 Introdução

O objetivo do projeto é a criação do jogo Bomberman, para tal efeito o grupo, inicialmente, constituiu uma metodologia necessária para a simplificação da realização do mesmo, em sintonia com a estrutura cronológica predefinida pelos professores.

A metodologia utilizada neste projeto é a divisão do mesmo em seis tarefas cada uma com um objetivo concreto. Maior parte das tarefas são independentes das restantes a nível de programação no entanto existe uma sequência ideal para uma abordagem consciente e facilitadora do objetivo de cada uma.

A sequência cronológica de desenvolvimento do projeto foi desde a tarefa 1 ate a 4 por ordem crescente, no entanto o grupo decidiu desenvolver a quinta tarefa anteriormente à sexta uma vez que facilita a criação da sexta tarefa havendo um feedback instantâneo pois existe a possibilidade de verificação do comportamento do bot (criado na sexta tarefa) através da quinta tarefa.

Esta sequência de desenvolvimento do projeto não é necessariamente obrigatória e por isso não é de aplicação universal contudo o grupo concluiu que seria a melhor organização e estruturação do mesmo.

Sendo o grupo constituído por dois elementos, houve uma distribuição das tarefas pelos membros com o propósito de economizar tempo sem prejudicar o conhecimento de cada membro sobre

as tarefas não responsáveis por estes, pois os dois membros tiveram sempre acesso ao trabalho de cada um e mantiveram-se actualizados.

2 Descrição do Problema

O objetivo final do projeto é, como já foi dito anteriormente, a criação do jogo clássico *Bomberman*. No entanto devido a metodologia utilizada cada tarefa tem um objetivo de seguida apresentados.

- A primeira tarefa implementa um mecanismo de geração de mapas, isto é, a estrutura de ambiente de jogo;
- A segunda tarefa determina o efeito de um comando (recebido pelo teclado do computador) num estado de jogo genérico;
- A terceira tarefa implementa um mecanismo de compressão/ descompressão de um estado de jogo com o objetivo de poupar espaço em disco quando o jogo é gravado;
- A quarta tarefa determina o efeito da passagem de um instante de tempo num estado de jogo genérico;
- A quinta tarefa implementa o desenvolvimento gráfico do ambiente de jogo;
- A sexta tarefa implementa um bot que jogue *Bomberman* automaticamente.

3 Concepção da Solução

3.1 Estruturas de Dados

Ao longo do desenvolvimento das tarefas o grupo procurou sempre organizar os passos utilizados para concretizar o objetivo final de forma a clarificar e facilitar o trabalho. O mecanismo utilizado consiste na divisão de cada tarefa em várias finalidades. Visto que o estado de jogo se fragmenta em quatro partes - tabuleiro de jogo, estado dos power ups, estado das bombas e estado dos jogadores - foi intuitivo separar cada tarefa em funções com o propósito de modificar cada fragmento de jogo. Na próxima secção demonstramos de que forma adoptamos este mecanismo ao longo das seis tarefas.

3.2 Implementação

A primeira tarefa não foi totalmente idealizada em sintonia com o mecanismo descrito uma vez que é a tarefa de ponto de partida do projeto (ainda não estava definido um estado de jogo possível de ser fragmentado em quatro finalidades) no entanto é notável uma semelhança ao mecanismo exposto pois as funções principais desta tarefa estão organizadas da seguinte forma:

- formação/ geração do tabuleiro do jogo (*drawMap e drawMap2*);
- geração do estado dos power ups (*findFlames e findBombs*);

Visto que esta tarefa não implica a existência de jogadores ou de bombas plantadas o mecanismo descrito não é totalmente seguido.

A segunda tarefa foi realizada utilizando coerentemente o mecanismo pois o estado de jogo recebido por esta tarefa é genérico, ou seja todos os fragmentos do estado de jogo têm de ser consideradas (quer existam ou não). Esta tarefa esta organizada com a seguinte estrutura:

- recebendo um estado de jogo qualquer é apresentado apenas o tabuleiro do jogo (*justMap*);
- em função do comando recebido atualiza o estado dos power ups (*new Coordinates*);

- em função do atual estado de jogo atualiza o novo estado das bombas (*all-bombs-players-state*);
- em função do comando recebido atualiza o novo estado de jogadores (*all-players-state*);

Cada uma das funções em cima referidas utilizam diversas funções auxiliares, no entanto todas convergem nessas quatro funções principais.

A terceira tarefa ficou inacabada no entanto tem muitas informações e funções úteis que apesar de não terem finalidade concreta (devido a ter ficado truncada) devem ser apresentadas:

- recebendo um valor numérico (em string) codifica o mesmo economizando caracteres (*codex*);
- função inversa à codex, recebendo uma string converte a mesma num valor numérico (em string) (*convert*);
- recebendo um estado de jogo codifica o tabuleiro do jogo apenas, de forma a economizar caracteres (*encode*);
- função inversa à encode, recebendo a string (codificada) reverte a mesma numa string correspondente ao tabuleiro (*decode-only-map*);

A quarta tarefa é coerente ao mecanismo descrito em cima, no entanto foi provavelmente a tarefa mais importante para adquirir conhecimentos em relação a otimização de programação, pois o grupo implementou um novo sistema de manipulação do estado de jogo, utilizamos *Data* para facilitar a programação:

- Celulas = Pedra — Tijolo — Vazio deriving (Show,Eq);
- data Power-Up = PU-Bomba Coordenada — PU-Flame Coordenada deriving (Show,Eq);
- data Estado-da-Bomba = Bomba-P Quintuplo deriving (Show,Eq);
- data Jogador = J Int Coordenada PUs deriving (Show,Eq);

Após a criação dos *Data* definimos o novo estado de jogo através de novos *type*:

- type Mapa = [[Celulas]];
- type Power-Ups = [Power-Up];
- type Jogadores = [Jogador];
- type Estado-de-jogo = (Mapa, Power-Ups, [Estado-da-Bomba], Jogadores);

Desenvolvendo-se o estado de jogo toda a manipulação tornou-se mais concreta e objetiva, sendo as principais funções as seguintes:

- recebendo o estado de jogo em lista de strings converte o mesmo no novo estado de jogo (em *Data*) utilizado nesta tarefa (*estado-de-jogo*);
- função inversa à estado-de-jogo, recebendo o novo estado de jogo (em *Data*) reverte o mesmo no estado de jogo em lista de strings (*reverte-Estado-de-jogo*);
- recebendo o estado de jogo apresenta o tabuleiro atualizado, eliminando os tijolos que foram atingidos por uma bomba (*destapa-tijolo-atingidos*)
- recbendo o estado de jogo determina o raio de explosão de todas as bombas que vão explodir numa lista de coordenadas (*raio-acao-de-todas-as-bombas*);

- recebendo o estado de jogo e o raio de explosão de todas as bombas que vão explodir atualiza o estado de todas as bombas: reduz o tempo a todas as bombas, elimina as que explodirão, provoca a explosão forçada de outra bomba (*atualiza-Bombas*);
- recebendo o estado de jogo atualiza estado dos power ups, eliminando aqueles que são atingidos por uma bomba (*elimina-Power-Ups*);
- recebendo o estado de jogo atualiza o estado dos jogadores, eliminando aqueles que foram atingidos por uma bomba (*atualiza-Jogadores*);
- recebendo uma mapa em lista de strings coloca uma pedra na coordenada correta da espiral (*coloca-pedra*);
- recebendo um estado de jogo e a coordenada da ultima pedra colocada da espiral atualiza o estado de jogo totalmente, eliminando um power up ou uma bomba ou um jogador no caso da espiral atingir um destes (*retirar-PU-ou-B-ou-J*);

A quinta tarefa implementa um *record* que facilitou muito o uso de imagens ao longo do desenvolvimento da tarefa:

- data Figuras = Figuras tijolo :: Picture, pedra :: Picture, dirt :: Picture, jogador0 :: Picture, jogador1 :: Picture, jogador2 :: Picture, jogador3 :: Picture, bomba :: Picture, pubomba :: Picture, puflame :: Picture, aba :: Picture, logo :: Picture, relogio :: Picture, preto :: Picture, raio :: Picture ;

Foi, também, implementado um novo *type* que contém todas as informações necessárias à realização da quinta tarefa:

- type Estado = (Numero-jogador,Dimensao-do-mapa,Estado-de-jogo,Figuras,Time,Var)

Nesta tarefa a função mais útil para desenhar o estado de jogo é a *desenha-coord*, função que recebe uma coordenada e uma figura, colocando esta na coordenada correta do referencial. Criando esta função tudo se tornou mais simples pois para desenhar o estado de jogo basta evocar recursivamente esta função atualizando as coordenadas e figuras a serem desenhadas através das seguintes funções:

- desenha apenas o tabuleiro do mapa (Pedra, Tijolo e Vazio), faltam os jogadores, power ups e bombas plantadas: (*desenha-mapa*)
- desenha os power ups nas suas respectivas coordenadas no referencial (*aux-put-PU*);
- desenha os as bombas nas suas respectivas coordenadas no referencial (*aux-put-B*);
- desenha o jogadores power ups nas suas respectivas coordenadas no referencial (*aux-put-J*);
- desenha a aba lateral do mapa (*desenha-aba*);
- função na qual todas as funções em cima referidas convergem e produzem como resultado final o estado de jogo totalmente desenhado (*junta-tudo*);
- função que centra o mapa desenhado (*desenhaEstado*)

A função (*reageEvento*) e (*reageTempo*) foram mais simples de se desenvolver uma vez que apenas evocam funções da segunda tarefa e quarta tarefa respectivamente, no entanto salienta-se o uso de uma variável de controlo na função (*reageTempo*) para obter uma taxa de *frames per second* eficiente. Esta variável é acumulada e adiciona-se recursivamente um valor constante á mesma, quando a soma alcançar ou for superior a um, o tempo do estado de jogo reduz um segundo.

A sexta tarefa tem como único objetivo a implementação de um bot capaz de jogar de modo autónomo com o objetivo de ganhar aos seus oponentes. Utilizando a estrutura de jogo implementada na tarefa 4 e expandindo-a um pouco, o grupo conseguiu com sucesso programar um bot que se afasta do perigo, recolhe power ups e tenta eliminar os inimigos. A estratégia utilizada consistiu em prioritarizar o modo evasivo, isto é, programar um bot que apenas se dispõe a atacar se for totalmente seguro avançar com esse intuito. Caso contrário, o bot foge para um local seguro. Se o bot se encontrar em segurança, a sua próxima prioridade será recolher power ups e rebentar tijolos. Posteriormente, se o caso for tal que já não haja tijolos para rebentar, este toma uma estratégia ofensiva, ou antes, uma estratégia de longevidade. Visto ser impossível de prever o movimento dos outros bots e, consequentemente, muito difícil de conseguir com sucesso atacá-los premeditadamente, isto é, formular uma estratégia em que se consegue aniquilar o bot de uma certa maneira já concebida sem qualquer aleatoriedade - visto haver tais circunstâncias aleatórias, a estratégia adotada consiste em dirigir o bot para o meio do tabuleiro, sempre com a prioridade de escapar às bombas, e, uma vez lá, manter a posição, esperando que os adversários restantes sejam devorados pela espiral que se acerca quando o tempo começar a escassear. Para este efeito, foram concebidas pelo grupo as seguintes funções chave:

- a função que determina o nível de segurança de uma certa posição (*minSafetyLevel*)
- a função que comanda o bot para determinada coordenada, e que é utilizada por todas as mencionadas a seguir (*goTo*)
- a função que comanda o bot para destruír tijolos (*blowBricks*)
- a função que comanda o bot para fugir (*evade*)
- a função que comanda o bot para o meio do mapa (*goToMiddle*)
- a função que determina qual a prioridade do bot: recolher power ups, destruir tijolos ou dirigir-se para o meio (*attack*)
- a função que liga todas as anteriores e, consequentemente, todas as utilizadas por esta tarefa, fazendo um bot funcional (*act*)

Novos tipos de dados foram também criados, o mais notável sendo o tipo *Jogada* que é um par cuja primeira componente é do tipo *Char* e a segunda do tipo *Int*. Esta estrutura foi criada com o efeito de possibilitar listar as direcções possíveis (L,U,R,D) e para cada uma delas o seu nível de segurança, sendo 10 o mais alto e 1 o mais baixo. O nível 1 significa que naquela posição, no próximo instante, explodirá uma bomba.

3.3 Testes

Ao longo do desenvolvimento do projeto, o grupo criou vários testes de forma a testar as funcionalidades das soluções desenvolvidas, nomeadamente na segunda e quarta tarefa. De seguida esclarecemos dois testes que são fundamentais para a compreensão das funcionalidades de todo o projeto.

A figura 1 é o teste que o grupo considera mais importante uma vez que demonstra o efeito de todas as condicionantes envolvidas na tarefa 4. Quando o tempo avança 1 segundo é notável o efeito da reação à passagem do tempo no estado de jogo, uma vez que existem bombas que são forçadas a explodir, outras bombas não alcançam o seu raio de explosão total uma vez que são bloqueadas por power ups, tijolos ou mesmo pedras.

Os restantes exemplos também demonstram diversas funcionalidades das tarefas, no entanto são mais restritos pois não contém tantas condicionantes.

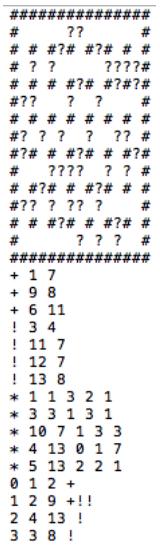

Figura 1: exemplo 8 no svn

4 Conclusões

Este projeto, realizado ao longo de três meses no âmbito da unidade curricular de LI1, permitiu em inúmeros aspetos aprofundar a nossa visão e conhecimento acerca da programação, adquirimos conhecimentos fulcrais acerca de estratégias de desenvolvimento de grandes projetos e também aprofundamos imensamente o uso da linguagem Haskell. Ao longo das 6 tarefas em que tivemos de utilizar diferentes estratagemas para resolver todas as etapas, fomos confrontados com o dilema universal da programação: criar código que subsista a ulteriores alterações e que para estas seja útil e não redundante, possibilitando assim a construção de programas não só úteis, mas alteráveis e recicláveis. Ao analisar o nosso código é visível que a estratégia foi mudando desde a primeira tarefa para as seguintes, e cada vez se afastou mais das bases da linguagem haskell para formar uma estrutura aplicada especificamente ao desenvolvimento do jogo *Bomberman*. Nas últimas tarefas os tipos são específicos ao funcionamento do jogo e não universais, sendo intuitivo para qualquer um que analise o código, sem grande esforço, compreender de que maneira funciona. Desde o desafio, novo na altura, de conseguir juntar várias funções básicas para fazer uma tarefa complexa nas primeiras tarefas; desde o processo de compactar o código necessário à tarefa 3 e através da tarefa de reagir ao tempo e de programar um bot funcional, inúmeros conhecimentos foram adquiridos e a implementação de uma componente gráfica permitiu-nos compreender de que maneira se relacionam diferentes aspetos de um programa, como adaptar um mecanismo a uma interface. Com estas bases, é-nos possível fazer qualquer tipo de programa numa linguagem funcional, e neste aspeto, o objetivo foi cumprido com sucesso.