Functional Programming: Labyrinth

Laurent Christophe, Wolfgang De Meuter

Programming Project: Assignment #3 (2016 - 2017)

1 Introduction

The final mark for the Functional Programming course is calculated as follows: (i) 12.5% for the first programming assignment (ii) 12.5% for the second programming assignment (iii) 25% for the third programming assignment (iv) 50% for the oral exam. All exam parts have to be done in order to obtain a final mark. This document describes the third programming assignment.

Submission is done by sending your raw code files to the teaching assistant: Laurent Christophe (`lachrist@vub.ac.be`). This third assignment is due 16th January 2017 (8AM). Notice that the execution of the project is strictly individual and no plagiarism shall be tolerated!

The project will be marked according to how well you fulfill the functional requirements and according to how well you apply the concepts explained during the lectures and the lab sessions. If you encounter any problem or if you have a precise question, feel free to contact the assistant at `lachrist@vub.ac.be`.

2 Labyrinth's Rules

The goal of this assignment is to implement the Labyrinth board game[1]. This game is designed for up to 4 players and its aim is to collect treasures in a dynamically changing maze. At the beginning of the game, treasure cards are distributed across players. The winner is the first player who collect all the treasures associated to its cards and return to its starting position. A turn consists of:

1. Shift a movable row of the maze using an extra tile, producing an novel extra tile.
2. Gather your treasures reachable from your pawn position.
3. Move your pawn to a reachable tile of your choice.

[1] `https://www.ravensburger.com/uk/games/family-games/labyrinth/index.html`

Figure 1: The board with only the fixed tiles and the board randomly filled.

Figure 2: The three kinds of tiles and their occurrences: corner (20), t-shaped (18) and line (12)

3 Implementation Details

You should develop a terminal-based application that enables 1 to 4 players to play labyrinth and support basic "artificial intelligence". At the beginning of the game the users specify how many real players will take part in the race and which will be controlled by the AI. Then the free tiles (16 corners, 6 t-shaped, 12 lines) are randomly placed between the fixed tiles with an extre free tile as shown in Figure 1. The kind and orientation of each tile should be clear from its ASCII representation. Finally, the 24 treasures are randomly placed on the tiles with maximum one treasure per tile and no treasure on the starting positions. Each one of these treasure is associated with a treasure card. These 24 treasure cards are randomly distributed across the players with a equal number of treasure cards for every player. Once the preparation phase is completed, players play each one in turn. A the beginning of its turn, a player has access to the following information:

1. The state of the board (the extra tile, placed tiles, treasures and pawns).
2. The player's own treasure cards
3. The number of treasure cards left for every other player.

If the player is not controlled by the AI, the user is invited to insert the free tile on the board to shift a movable row of the maze without making a pawn fall off the board. Then, all the player's treasure cards whose associated treasure is reachable from the player pawn are automatically revealed from the player's hand. The player can now move its pawn to a reachable tile of its choice (a tile may receive multiple pawns). If the player moves to its starting position without having any treasure left to collect he wins the game. Make sure that the next player does not see the treasure cards of the current player by using `System.Process.callCommand "clear"`. The turns of the players controlled by the AI should not be shown and the next human controlled player should be prompted to play (unless the AI won the game). The minimal requirement for the AI is that it should win the game alone in a finite number of turn for every possible starting configuration. Smart strategies will be considered as bonuses.

4 Serialization

A game of labyrinth may take a long time therefore you should enable saving and loading functionalities. Human players should be able to save and exit the game at the beginning of each turn. Your executable should accept an optional command-line argument to load a game from a file. The serialization format is specified in Table 1 and is mostly self-explaining. The order of the player's list indicates the turn order to resume the game. Tiles are filled from top to bottom and then left to right as shown in Table 2. ϵ denotes the empty string and NATURAL denotes natural numbers: `/[0-9]+/`. All tokens can be separated by multiple blank characters: `/[\n\t ]+/`.

LABYRINTH	:=	PLAYERS XTILE TILES
PLAYERS	:=	PLAYER PLAYERS
		ϵ
PLAYER	:=	COLOR CONTROL POSITION CARDS
COLOR	:=	yellow
	:=	red
	:=	blue
	:=	green
CONTROL	:=	human
		ai
POSITION	:=	NATURAL NATURAL
CARDS	:=	NATURAL CARDS
		ϵ
XTILE	:=	KIND TREASURE
KIND	:=	corner
	:=	tshape
	:=	line
TREASURE	:=	NATURAL
		ϵ
TILES	:=	TILE TILES
		ϵ
TILE	:=	KIND TREASURE DIRECTION
DIRECTION	:=	north
	:=	east
	:=	south
	:=	west

Table 1: Grammar for the serialization format of a labyrinth game.

01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

Table 2: Tiles filling order.