Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 2048

0.1.1 Teacher's guide

Emne rekursion, grafik og winforms

Sværhedsgrad Middel

0.1.2 Introduction

2048 is a popular solitaire board game available, e.g., online on `https://2048.io/`. In this exercise, you are to implement a version of this game in F# and using `Canvas`. The rules, you are to implement are as follows:

0.1.2.1: The board is a square board with 3×3 field.

0.1.2.2: The pieces are colored squares: red, green, blue, yellow, black corresponding to the values 2, 4, 8, 16, 32.

0.1.2.3: There can at most be one piece per field on the board.

0.1.2.4: The initial conditions is a red and a blue piece placed on the board.

0.1.2.5: The game can be tilted left, right, up, down by pressing the corresponding arrow keys.

0.1.2.6: When the game is tilted, then all the pieces are to be moved to the corresponding side of the board.

0.1.2.7: If two pieces of the same color touch after being tilted, then they are replaced by a single piece of double the value. For example, the board in **??** is tilted to the right, and the two red pieces are replaced with green piece.

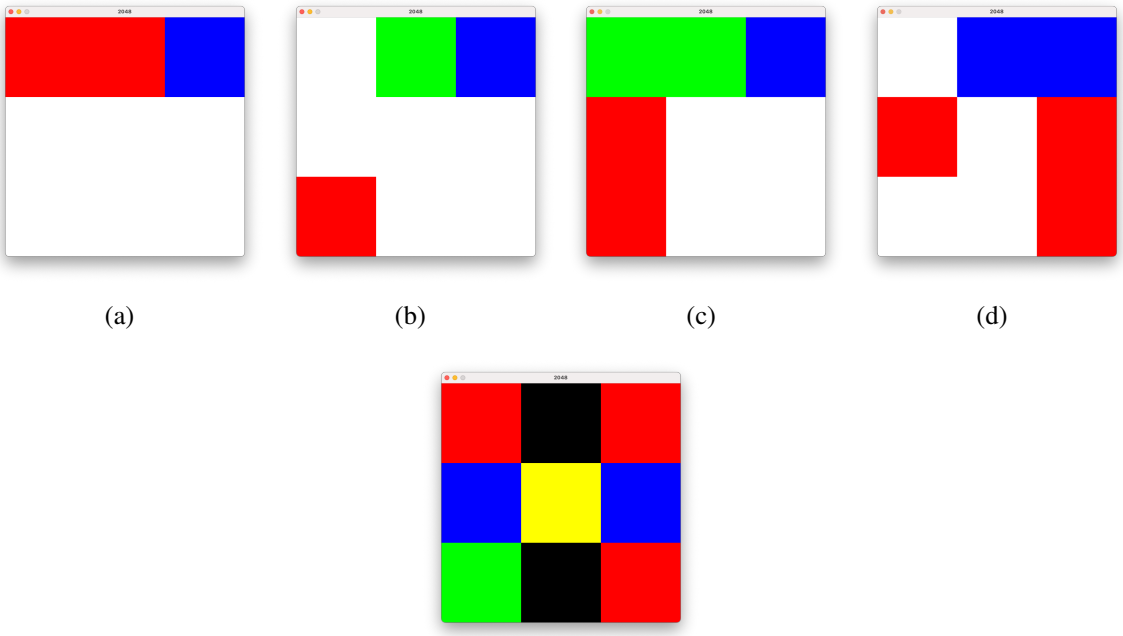
0.1.2.8: Identical pieces are combined in the order of the direction, they are tilted, e.g., if the board is tilted left, then identical pieces are identified from the left to right. Matching colors that touch in the direction of the tilt after a combination of pieces are not combined. E.g., the board in **??** is tilted to the right combining the two green to a blue, but the resulting two blues are not combined.

0.1.2.9: Two black pieces are combined into one black piece.

0.1.2.10: After each turn, a new red piece is to be placed randomly on an available field on the board.

0.1.2.11: The game ends when there are no possible moves and no empty locations for a new piece to spawn.

In your solution, you are to represented a board with its pieces as a list of pieces, where each piece has a color and a position. This is captured by the following type abbreviations:



(a) (b) (c) (d)

(e)

Figure 1: Some examples

```
type pos = int*int // A 2-dimensional vector in board-coordinats (not
   pixels)
type value = Red | Green | Blue | Yellow | Black // piece values
type piece = value*pos //
type state = piece list // the board is a set of randomly organized
   pieces
```

0.1.3 Exercise(s)

0.1.3.1: Make a library consisting of a signature and an implementation file. The library must contain the following functions

```
// convert a 2048-value v to a canvas color
fromValue: v: value -> Canvas.color
// give the 2048-value which is the next in order from c
nextColor: c: value -> value
// return the list of pieces on a row r on board s
filter: r: int -> s: state -> state
// tilt all pieces on the board s to the left
shiftLeft: s: state -> state
// flip the board s such that all pieces position change as
   (i,j) -> (N-1-i,j)
fliplr: s: state -> state
// transpose the pieces on the board s such all piece positiosn
   change as (i,j) -> (j,i)
transpose: s: state -> state
// find the list of empty positions on the board s
empty: s: state -> pos list
```

```
// randomly place a new piece of color c on an empty position on
   the board s
addRandom: c: value -> s: state -> state option
```

With these functions and Canvas it is possible to program the game in a few lines. Add the following to your library:

(a) Write a canvas draw function

```
draw: w: int -> h: int -> s: state -> canvas
```

which makes a new canvas and draws the board in s.

(b) Write a canvas react function

```
react: s: state -> k: key -> state option
```

which titles the board base according to the arrow-key, the user presses. Note that tilt left is given by the `shiftLeft` function. Tilt right can be accomplished by `fliplr >> shiftLeft >> fliplr`, and tilt up and down can likewise be accomplished with the additional use of `transpose`.

Finally, make an application program, which calls `runApp "2048" 600 600 draw react board`.

All above mentioned functions are to be documented using the XML-standard, and simple test examples are to be made for each function showing that it likely works.