Developer Guide for Fury Breakout 2020

A remake of Super Breakout (1978 Atari)

This document is meant to be a guide describing the source files, which means the roles of the main functions, how and why they are defined in a certain way, and how the combination of such functions allows the program to work as intended. This process will consist in highlighting the main parts of our code, going through it from top to bottom, to keep consistency in the development of our ideas.

The first thing worth mentioning is the choice of libraries. Since our project is fundamentally a video game, which means an interactive program, the use of the **2htdp/universe** library as our main resource is pretty self explanatory: the universe library allows us not only to render scenes with moving objects, but also to interact with them using mouse and keyboard. The rest of the libraries used are mainly auxiliary libraries that help us implement different functions to the program:

- the **2htdp/image** library is used to draw the scenes;
- the **2htdp/batch-io** library is used to mimic the original font of the arcade version of Super Breakout
- the **rsound** library allows us to implement sounds for certain events in the game

The first part of the actual code is simply a series of data types definitions needed to implement the different elements of the game (such as bricks, balls, paddles), followed by various constant definitions useful to make the code cleaner and easier to understand.

The next part consists of auxiliary functions which will be needed in the definition of the main functions that actually run the program. There are for example functions that translate a row or column number into x-y coordinates (**row->y**, **col->x**) as well as the functions that work as shortcuts to calculate the reflection angle of the collision of the balls (**reflect-h**, **reflect-v**).

Following the auxiliary functions that help us design the game, we finally find one of the main functions, probably the most important one, which is **update**. This function is the heart of the program, it is what handles the behaviour of the game one tick at a time. While it may look extremely compact, each sub-function called by **update** was carefully crafted to manage a different aspect of the game evolution.

The **update-balls** function is at the core of every action that involves balls, from ball movement to ball collisions. It takes the current state of each ball and updates it taking into consideration any possible obstacle or peculiar feature that the game may require.

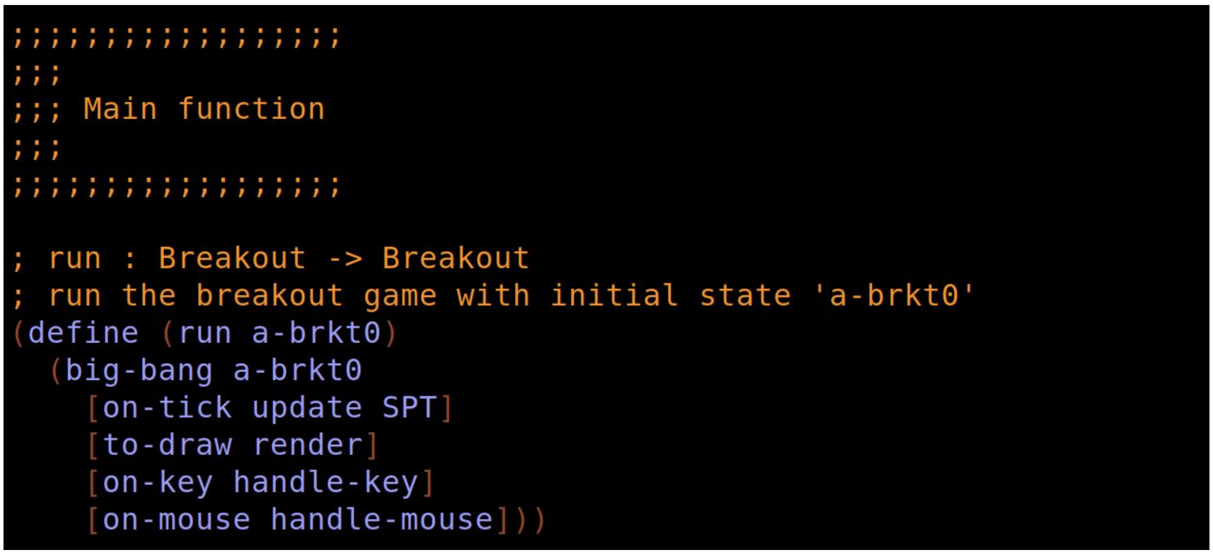
The three functions **update-attract**, **update-ready-to-play**, and **update-play**, are in charge of handling the possible game modes the program could be running. **update-attract** takes care of what we could call the main screen, which is the idle animation of the game playing itself while waiting for an action from the user. The second function, **update-ready-to-play**, is what manages the state of the game right before selecting a mode. The third and last **update-play** handles specific parameters of the mode currently being played (which can be progressive, doubles or cavity) such as scores or ending conditions.

Another important aspect of the game is of course the visuals, which in this case are handled by a main **render** function that contains the individual rendering functions for the necessary elements, such as **render-balls**, **render-bricks**, and **render-paddles**. The single images of the elements are put together to create the necessary scenes.

The last two functions that allow the program to become interactive are the functions controlling the input of the user:

- **handle-key** is the function that manages the keyboard inputs and the corresponding response. The precise actions that can be performed are the selection of the game mode, the selection of the number of players, the action of putting a coin in the machine, and the serve.
- **handle-mouse** is simply used to link the horizontal movement of the mouse to the movement of the paddle during the game.

As said at the start, the interactive nature of the program makes the universe library the perfect solution for our project, and this means that the main function that combines every step and every part of the development of the game is a big-bang function that allows the game to run properly.



```
;;;;;;;;;;;;;;;;;;
;;;
;;; Main function
;;;
;;;;;;;;;;;;;;;;;;

; run : Breakout -> Breakout
; run the breakout game with initial state 'a-brkt0'
(define (run a-brkt0)
  (big-bang a-brkt0
    [on-tick update SPT]
    [to-draw render]
    [on-key handle-key]
    [on-mouse handle-mouse]))
```