Advanced Programming Paradigms

Assignment3

Project Topic: Snake (Video Game)
Language: racket
Aim: Build a program which has simple GUI, player need use "↑","↓","←" and "→" which are "UP","DOWN","LEFT" and "RIGHT" on the keyboard respectively to control a square-made "Snake". Snake attempts to eat "Apple" by running into them with the head of the snake. Each apple eaten makes the snake longer.
Rules:

1. The edges of the windows are untouchable;
2. Any part of snake is untouchable;
3. Reverse operations are prohibited;
4. Above all cases will cause failure;

Pre-define:

1. The GUI windows is 20*20 by default, the frame size is 10 by default;

```
;window's height
(define WINDOW-HEIGHT 20)
;window's width
(define WINDOW-WIDTH 20)
;frame size
(define WINDOW-SIZE 10)
```

2. The label of the frame is "Snake";

```
(define frame (new frame%
  [label "Snake"]
  [width (* WINDOW-WIDTH WINDOW-SIZE)]
  [height (* WINDOW-HEIGHT WINDOW-SIZE)]))
```

3. Background is white;

```
(send dc set-brush "white" 'solid)
```

4. Snake is black;
5. Apple is red;

```
(draw-block dc (list-ref apple 0) (list-ref apple 1) "red")
```

6. Snake, apple and score have initial case, but when game start, apple's position is random;

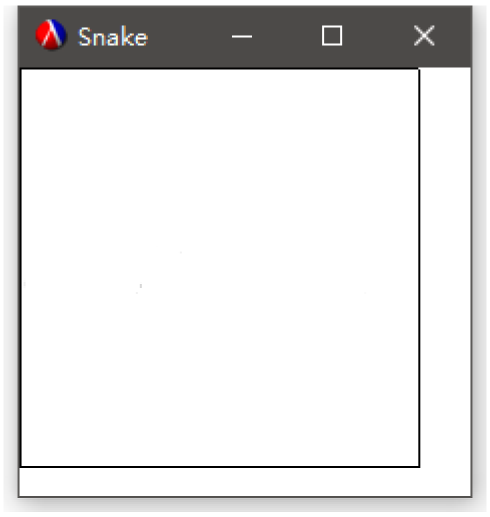
```
;Initial cases
(define snake (list (list 2 1) (list 1 1)));Initial snake position
(define apple (list 5 5));Initial apple position
(define direction 'r);Initial direction of snake's head
(define score 0);Initial score board
```

7. Restart case is the same with initial case;

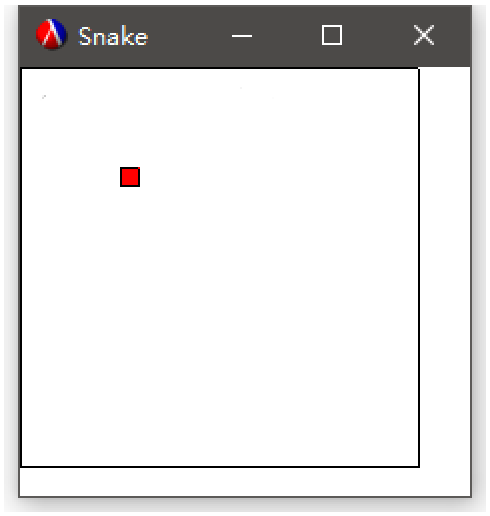
```
(define restart (lambda()
  (set! snake (list (list 2 1) (list 1 1)))
  (set! apple (list 5 5))
  (set! direction 'r)
  (set! score 0)
))
```

Details:

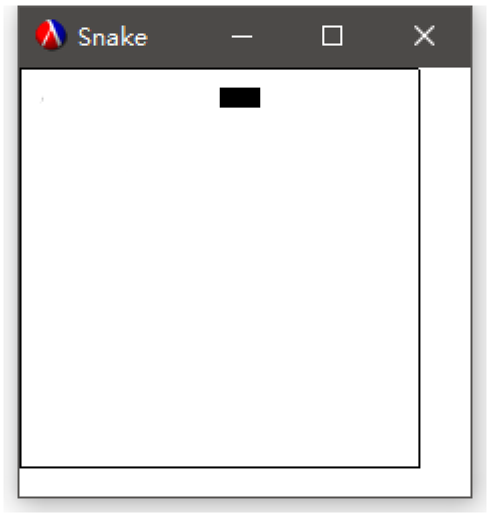
1. Set the background colour;



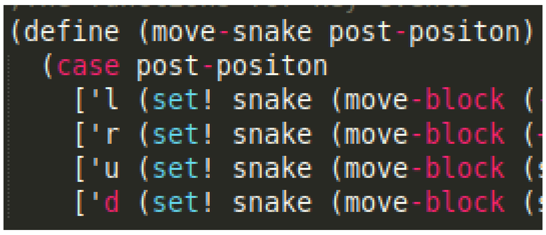
2. Apple's look like;



3. Snake's look like;

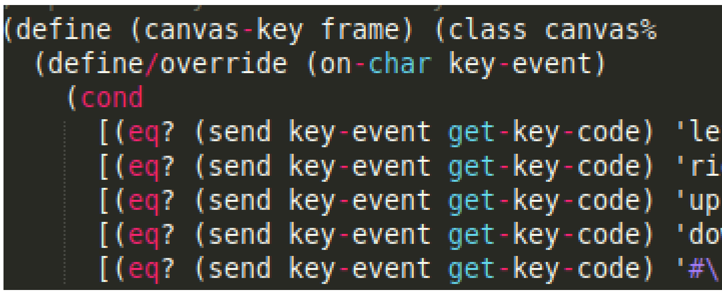


4. Use X & Y represent the coordinates of snake's head;
5. Define blocks' operations (how to move and transform);
6. Define snake's operations;



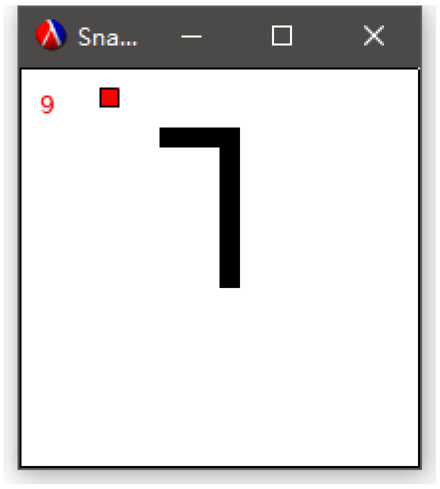
```
(define (move-snake post-positon)
  (case post-positon
    ['l (set! snake (move-block (
    ['r (set! snake (move-block (
    ['u (set! snake (move-block (
    ['d (set! snake (move-block (
```

7. Map key events into snake's operations;

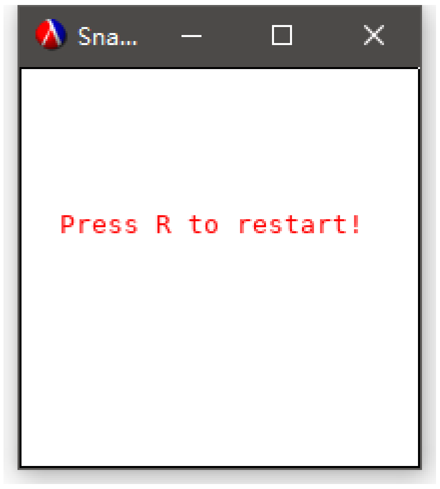


```
(define (canvas-key frame) (class canvas%
  (define/override (on-char key-event)
    (cond
      [(eq? (send key-event get-key-code) 'le
      [(eq? (send key-event get-key-code) 'ri
      [(eq? (send key-event get-key-code) 'up
      [(eq? (send key-event get-key-code) 'do
      [(eq? (send key-event get-key-code) '#\
```

8. Here is the game play;



9. When game is end, show information;



Press R to restart!