1 The Universe Teachpack: Designing Worlds

The goal of this morning is to learn to design interactive programs using the *universe* teachpack in DrScheme. You may want to read through the lecture notes for this morning, examine the sample code provided here, but should make sure that you design at least one simple game yourself.

Examples of possible games are:

- **Space invaders:** An UFO falling from the sky, defending platform moves left and right, fires a shot when space bar if hit. Game ends when UFO is hit or when UFO safely lands on Earth
- **Rat Race:** A hungry rat controlled by arrow keys looks for cheese. If found, it gets fatter; otherwise it gets hungrier on each tick and dies. A new cheese appears after the old one is eaten.
- **Manna from Heaven:** Manna is falling from the sky (one piece, or many pieces). Control the basket on the ground that catches the fallen manna. Stop if no more manna is falling. Or stop when you have collected enough for your clan.
- **Random Blob:** A blob moves randomly on each tick some limited distance. Control the blob so it stays within the bounds, and guide it to the center of the universe where it will be devoured and killed.
- **Snake Game:** You know what to do - just do not let the snake eat itself or hit the wall.
- **Rat Invasion:** Rats appear at random, use the hammer, move it using arrow keys, kill the rat with space bar. Too bad - a new rat will replace the dead one. Stop once you have killed ten rats.

Two words of advice before you begin. First, KISS! keep it simple, do not try to make the fanciest game you can think of — you do not have enough time in one lab. Second, focus on one concept at a time: first make sure you can handle the drawing of shapes/images, then handle the ticks, then handle the key events. One task, one function. Run the test cases before you even try to run the game. You do not have to grok all the intricacies of dealing with shapes and images - just make sure you have a shape or two and they move as desired. You can deal with bells and whistles when you get home and are preparing a demo to impress your students.

Finally, have fun :)

1.1 Working with Images and Shapes

- Look at the file `animations.ss`. Make a picture of a flower and place the bug on it. Or paint a launching pad for the rocket. Or combine several snowflakes on a blue background to make a doily.
- Open the file `sample-images.html` in a web browser. Design an animation that will fly an airplane across the sky.
- Pretend the *Blue sky* image is really ocean water and animate a fish or a shark swimming across.

Summary of available functions

Drawing shapes:

Available modes: `'solid 'outline`
Available colors: `'red 'blue 'green 'yellow 'black 'white`
Available colors: `"red" "blue" "green" "yellow" "black" "white"`

User defined colors: `(make-color r g b)`
where `r`, `g`, and `b` are in the range `[0 to 255]`.

Shape drawing functions:
Note: All shapes come with a pinhole in the center, except as noted.

```
(circle radius mode color)
(rectangle width height mode color) ;; pinhole in the center
(ellipse width height mode color)
(triangle length mode color)
(star points inner-radius outer-radius mode color)
(regular-polygon sides radius mode color [angle])
(line x y color) ;; from (0 0)
(text a-string font-size color)

(add-line image x1 y1 x2 y2 color)
(nw:rectangle width height mode color) ;; pinhole at NW corner
```

Manipulating shapes:

```
;; produce the basic empty scene - a blank canvas
(empty-scene width height)
```

```
;; add the given image to the given scene
;; with the image pinhole at the given (x y)
(place-image image x y scene)  ;; produces a new scene

;; define a function of time that for each time tick
;; produces the desired scene
;; time-fun: NaturalNumber -> Scene
(define (time-fun t)
  (produce the scene at the time t)

;; run the simulation for time ticks 1, 2, ...
(run-simulation time-fun)
```

1.2 Key Events, Mouse Events, and Tick Events

The *World* is everything you want it to be. It is a data item that describes all the information that is relevant and necessary to define your world.

For a simple blob world we may only want to know where the blob is. For the snake game we need to know where every part of the snake is, in what direction is the snake heading, and where is the food the snake is looking for.

To design the world you need to identify the data that will represent the world at any moment. You then need to design the functions that will transform the world and display the state of the world to the user.

For example in our *Blob World* the world consists of one `blob`, where the `blob` has a location (a `posn` and a color. It moves at random on each tick and moves to the right when the right arrow key is pressed.

Functions that transform and display the world:

```
;; produce the world as it looks after one tick
;; tick-fun: World -> World
(define (tick-fun w) ...)

;; A KeyEvent is a String
;; for example "x" "left" "\t" "control" " " ...

;; produce the world after the given key has been hit
;; key-fun: World KeyEvent -> World
(define (key-fun w ke) ...)
```

```
;; A MouseEvent is one of:
;; "button-down" "button-up" "drag" "move" "enter" "leave"

;; produce the world after the given mouse event occurred
;; with the mouse at location (x y)
;; mouse-fun: World Number Number MouseEvent -> World
(define (mouse-fun w x y me) ...)

;; stop the world when the given condition is satisfied
;; last-world?: World -> Boolean
(define (last-world w) ...)
```

To run the world invoke the big bang that starts the world running:

```
(big-bang w
    (on-tick tick-fun)
    (on-tick tick-fun [rate-expr]) ;; optional tick rate
    (on-key key-fun)
    (on-mouse mouse-fun)
    (on-draw draw-fun)
    (on-draw draw-fun [width height]) ;; optional canvas size
    (stop-when stop-fun))
```

1.3 On Your Own

Design the game *Catch the butterfly*. A butterfly flutters around, moving a random distance from its current location on each tick. You have a net that you control with the arrow keys. The game ends when you catch the butterfly.

- Look at the code in the `blobworld.ss` and see how the world is designed.
- Design the data representation for a the butterfly game and define the function that will draw the state of the game. Try not to be too cute — there is much work to be done.
- Design the function `on-tick-event` that consumes the `World` and just moves the butterfly to the next position.
- Design the function `next-move` that consumes the `ButterflyWorld` and a `KeyEvent` and moves the net accordingly.

- Design the function `caught?` that consumed the `ButterflyWorld` and determines whether the butterfly got caught in the net.
- You can now play the game. Try it.