This document is designed to walk the reader through the process of creating a game using the EaselSL framework, it assumes a *basic* understanding of sequenceL and at least a read-through of the Easel overview. Moreover, this document assumes the user has SequenceL, Easel, and Visual Studio installed and working.

By the end of this walkthrough we are going to have implemented the game *Move the Circle*.

Move the Circle specification

Game Description:
There is a red circle with radius 5 in the center of the screen. The circle can move left, right, up, or down, and it can change colors.

Game State:
The state of the game consists of the circle's location, and color.

Initial State:
The circle begins in the center of the screen, and it is blue.

Player actions:
The player can press any of the following keys to interact with the game: 'W', 'A', 'S', 'D', 'X'. The player can also left-click inside the game window to interact with the game.

What actions do:
In each frame,

1. If 'A' is pressed, 'D' is not pressed, the circle moves to the left by 5 pixels, unless this would move its center outside of the game window.
2. If 'D' is pressed and 'A' is not pressed the circle moves to the right by 5 pixels unless this would move its center outside of the game window.
3. If 'W' is pressed, 'S' is not pressed the circle moves up by 5 pixels unless this would move its center outside of the game window.
4. If 'S' is pressed, 'W' is not pressed the circle moves down by 5 pixels, unless this would move its center outside of the game window.
5. If the player clicks on the screen, and the circle is blue, then the circle will become red.
6. If the player clicks on the screen, and the circle is red, then the circle will become blue.
7. If 'X' is pressed then the game returns to the initial state.

Move the Circle implementation

To begin, we will use the template and the first thing we need to change is the state function. The only thing we need to keep track of are the circle's position and its color.

We will use a point for the position, and a boolean to keep track of the color.

```
State ::= (circ: Point(0), colorToggle: bool(0));
```

We have now modified what the state can look like, now we are going to take the time to make a quick constructor-function for the state, *buildState*: *Point* x *Bool* -> *State*

```
buildState(p(0), toggle(0)) := (circ: p, colorToggle: toggle);
```

This won't save us much time in this case, but when the state starts to get hard to build this is good practice.

The next thing we need to do is describe what new states will look like. The action description we were provided tells us what we need to do, so let us look at the first line.

We want the box to move left by 5 pixels if it's center won't leave the game window. The box will be okay to move 5 to the left as long as it's x-position is greater than 0, so we will make a condition in the newState function to do just that.

We will then do the same for the remaining actions.

```
newState(I(0), S(0)) :=
    let
        xCoordinate := S.circ.x;
        yCoordinate := S.circ.y;
        newPoint := point(xCoordinate, yCoordinate + 5) when equalList(I.keys, "W")
            else
                point(xCoordinate - 5, yCoordinate) when equalList(I.keys, "A")
            else
                point(xCoordinate, yCoordinate - 5) when equalList(I.keys, "S")
            else
                point(xCoordinate + 5, yCoordinate) when equalList(I.keys, "D")
            else
                S.circ;
        newToggle := not(S.colorToggle) when I.iClick.clicked else S.colorToggle;
    in
        buildState(newPoint, newToggle);
```

Now that we have the actions taken care of, the only thing left to do is describe how the game should look to the player. We want to create a circle at the point we have stored, and we want it's color to be red or blue depending on our boolean.

```
images(S(0)) := [disc(S.circ, 5, dRed)] when S.colorToggle else [disc(S.circ, 5, dBlue)];
```