2020 CI401
Introduction to programming

Week 1.11
Simple Animation

Dr Roger Evans

Module leader

15th December 2020

Lecture recording and attendance

- This lecture will be recorded and published in the module area
- The focus of recording is on the lecturer, not the audience
- If you are particularly concerned not to be part of the recording, turn off your microphone and camera.

- In addition, lecture attendance is now being routinely recorded (in all modules) to help the School Office monitor engagement

- (This slide is really a reminder to me to start recording and record attendance!)

Module structure (version 3)

Semester 1

Week	Topic	Theme
1.01	Introduction / Hello World	Coding
1.02	Variables, loops and choices	Coding
1.03	Input, more loops and choices	Coding
1.04	Variables and expressions	Coding
1.05	Types, assignment and arrays	Data
1.06	Let's play Top Trumps!	Data
1.07	Objects and methods	OO
1.08	Working with numbers	Data
1.09	Simple Algorithms	Dvp
1.10	Introduction to JavaFX	Dvp
1.11	Simple Animation	Dvp
	Xmas vacation 21 Dec – 8 Jan	
1.12	GUIs using MVC	OO
1.13		

Semester 2

Week	Topic	Theme	Project
2.01	Project topics and assessment	Project	Set
2.02	Simple Inheritance	OO	Lab
2.03	Scope, Visibility and Encapsulation	OO	Lab
2.04	Testing - JUnit	Testing	Lab
2.05	Documentation - Javadoc	Doc	Study
2.06	Collections and generic types	Data	Study
2.07	IO: files and streams	Dvp	Study
	Easter Vacation 29 Mar – 16 Apr		
2.08	Numbers – the computer's view	Data	Study
2.09	Java vs Python		Submit?
2.10	More algorithms – search and sort	Dvp	
2.11	How fast is my code?	Dvp	
2.12	Java 'under the hood'		
2.13	Revision week		Exam ↓

JavaFX so far

Key parts of a JavaFX app

- Stage object which generally maps onto a window on the screen.
- Scene object which represents the view/management of the GUI in the window
- A layout manager, such as a GridPane, which holds the content of the GUI (individual controls or other layout managers).
- Controls, such as Labels, Buttons, Textfields etc. which provide main user interface functions
- Event handlers, methods which allow the app to 'do something' when a user event happens (eg a button press)

JavaFX controls

- JavaFX provides all the standard controls (as Java classes) expected in a modern GUIs
- We used Label and Button in last week's lecture
- We also used Label, Button, RadioButton, ToggleButton, CheckBox, and Slider in the ControlsDemo class in last week's lab.

JavaFX layout managers

- Layout managers look after sets of other GUI objects (such as controls)
- The objects a layout manager looks after are called its children, and it is their parent
- Layout managers are responsible for sizing their children and arranging them on screen.
- A Scene object requires a layout manager to provide the content for the scene it displays
- So the simplest application consists of a single layout manager containing a single child control – in our Hello world example, we had a GridPane with a single Label as a child

Images and pictures

Pictures and graphics – the Image and Canvas classes

- GUI controls are intended to implement specific standard user interface functions
- They can be styled in various ways (colour, font, size etc.) but they still have quite fixed forms
- Often in a GUI, you want to display other things, such as images, or pictures/diagrams you have drawn yourself
- The classes to support this are Image (and ImageView) and Canvas

A brief introduction to images

- The Image class allows you to load an image using a URL. For example:

```
Image i = new Image("http://example.com/myimage.png");
```

- If you give just a pathname, the image is found in the same way classes are found, so in the simplest case, the image should be in the project folder:

```
Image i = new Image("/myimage.png");
```

- Images themselves cannot be added as children to layout managers – to do that you need to use the ImageView class, giving the image object (i)as an argument when you create it:

```
ImageView iv = new ImageView(i);
```

- Or you can do both steps at once, by giving the image pathname which you create the ImageView:

```
ImageView iv = new ImageView("/myimage.png");
```

- Then you can add your ImageView to a GridPane eg:

```
grid.add(iv, 1, 1 );
```

The Canvas class

- Canvas is a class which creates a space that you can ‘paint’ on
- Each canvas object has a GraphicsContext object, which is a bit like a GUI paintbrush for painting on your canvas (as a programmer, not interactively)
- The GraphicsContext stores information about painting, such as the line colour and the fill colour, the thickness of lines etc..
- It also provides methods for drawing standard shapes such as lines, circles, rectangles, polygons etc.

Drawing a tree

DrawTree creates a GridPane and a Canvas

Then it draws a tree on the Canvas

Then it adds the Canvas to the GridPane

Then it make Scene using the Pane, attaches the Scene to the window and shows the window (as before)

Notice that we didn't set the size of the Scene – because the Canvas has a specific size, the Scene will just be 'big enough' for that

Notice also WIDTH and HEIGHT – examples of variables which are fixed (and have all-caps names)

```
16    // the width and height of the canvas
17    public static int WIDTH = 300;
18    public static int HEIGHT = 250;
19
20    // layout objects
21    public GridPane pane;
22
23    // control objects
24
25    // canvas objects
26    public Canvas treeCanvas;
27
28    // method to start the app
29    public void start(Stage window) {
30        window.setTitle("DrawTree");
31        pane = new GridPane();
32
33        // create the canvas of the requried width and height
34        treeCanvas = new Canvas(WIDTH, HEIGHT);
35
36        // draw a tree whose top is at 50,50, width is 60 and height is 100
37        drawTree(treeCanvas, 50, 50, 60, 100);
38
39        // add the canvas to the layout grid
40        pane.add(treeCanvas,0,0);
41
42        // create the scene from the grid, add to window and show the window
43        Scene scene = new Scene(pane);
44        window.setScene(scene);
45        window.show();
46    }
```

Drawing a tree

To draw the tree, we get the GraphicsContext object from the canvas (the 'paintbrush').

Set it to fill in green, and draw a triangle

Set it to fill in brown and draw a rectangle

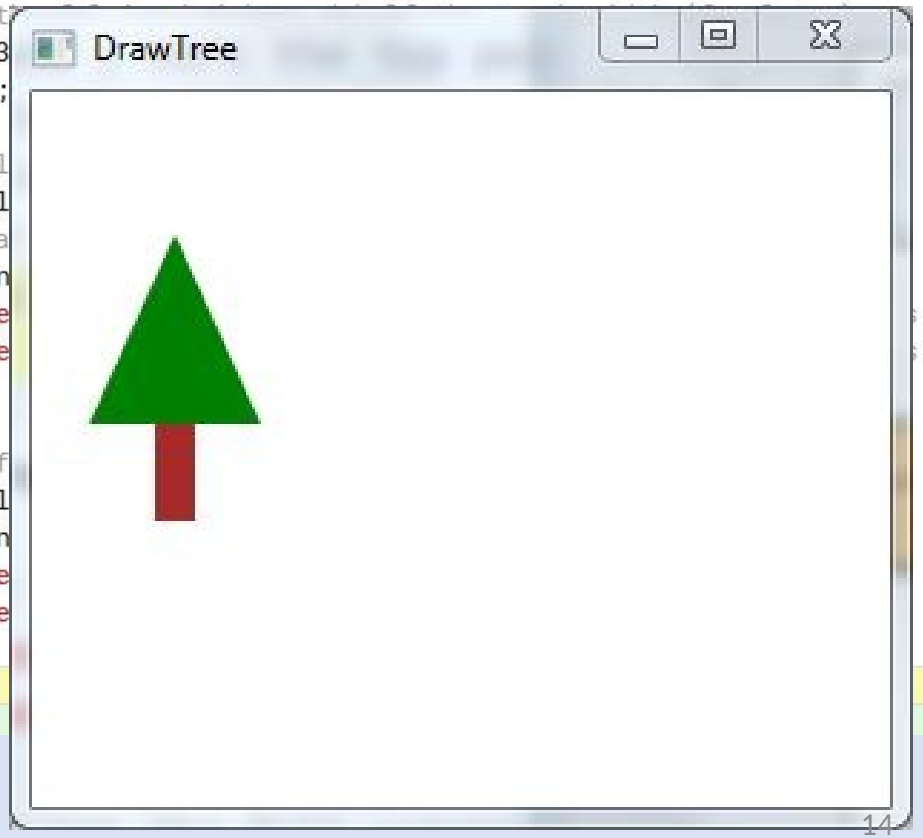
Notice the variables x, y, w, h passed as arguments, and fh and htw which are calculated from them

Notice also the rather weird way we specify the corners of the shapes – an array of x coordinates and an array of y coordinates (as doubles, not ints)

```
48      // method to draw a tree
49      // x and y mark the top point (as coordinates on the Canvas)
50      // w is the width of the foliage at the bottom
51      // h is the total height - the foliage is 2/3 an the trunk is 1/3
52      // the trunk width is 1/4 the total width.
53      public void drawTree(Canvas myCanvas, int x, int y, int w, int h)
54      {
55          // get the 'paintbrush' obejct for the canvas
56          GraphicsContext gc = myCanvas.getGraphicsContext2D();
57
58          // calculate the folaige height and half the trunk width (for later)
59          int fh = 2*h/3;     // foliage height - 2 thirds of total height
60          int htw = w/8;      // half trunk width
61
62          // set the fill colour in the 'brush' to green
63          gc.setFill(Color.GREEN);
64          // draw a triangle
65          gc.fillPolygon(
66              new double[] { x-w/2, x, x+w/2}, // the x coordiantes of the corners
67              new double[] { y+fh,   y, y+fh}, // the y coordinates of the corners
68              3);                              // the number of points
69
70          // similarly for the trunk
71          gc.setFill(Color.BROWN);
72          gc.fillPolygon(
73              new double[] { x-htw, x-htw,   x+htw, x+htw},
74              new double[] { y+fh,  y+h,     y+h, y+fh},
75              4);
76      }
77  }
```

Here's the result …

```
48    // method to draw a tree
49    // x and y mark the top point (as coordinates on the Canvas)
50    // w is the width of the foliage at the bottom
51    // h is the total height - the foliage is 2/3 an the trunk is 1/3
52    // the trunk width is 1/4 the total width.
53    public void drawTree(Canvas myCanvas, int x, int y, int w, int h)
54    {
55        // get the 'paintbrush' obejct for the canvas
56        GraphicsContext gc = myCanvas.getGraphicsContext2D();
57
58        // calculate t
59        int fh = 2*h/3
60        int htw = w/8;
61
62        // set the fil
63        gc.setFill(Col
64        // draw a tria
65        gc.fillPolygon
66            new double
67            new double
68            3);
69
70        // similarly f
71        gc.setFill(Col
72        gc.fillPolygon
73            new double
74            new double
75            4);
76    }
77 }
```



Simple animation

Second law: if we get to animation by the end of term 1, we **must** animate a snow scene.

Animation – how to make something move

- So far, all our programs have just given a set of instructions that the computer just does – usually so fast that it looks instantaneous, except when it is waiting for input from us.
- We often want elements in our GUI to move ‘on their own’ (ie without the user doing anything)
- Sometimes this is part of the interface – eg progress bars, or ‘buffering’ indicators
- Sometimes it’s a visual effect eg for decoration, or as part of a game

Animation – two issues

The basic way to achieve animation is to change what is displayed on the screen, frame by frame. We could just write code to do this. BUT:

- Computers run so fast that you need to make then wait between frames (and for the computer, that means waiting a long time – a computer running at 2GHz can do 33 million things between two frames being displayed at 60 frames a second)
- Our ‘event driven programming’ model means your program mustn’t actually wait itself – event handlers have to give control back quickly so that other tasks can also happen

Simple animation in JavaFX

- Let’s consider a simple animation task – we want an image (or a piece of text) to move across the screen
- We use a control for the thing that is going to move, and we need a layout manager which lets us position it wherever we like (unlike the GridPane, which locks controls into a grid)
- Then we need to specify the path we want the object to take, and how quickly we want it to travel.
- Then we pass all that information over to JavaFX and it runs the animation for us (our code doesn’t need to do anything ese)

Animation in JavaFX

So simple animation in JavaFX has four components:

1. A layout manager that lets us position an object anywhere (using coordinates) – eg the Pane class
2. An object to move in the animation (eg a Label control)
3. A Path object to specify where the object moves during the animation
4. A PathTransition object which manages the movement in the background, with the timing etc.

Animating a snowflake

- Use a Pane layout manager
- Create a Label containing an asterisk for the snowflake (you could use an ImageView object, or anything)
- Create a Path object which specifies the path the snowflake will take (straight down the screen)
- Create a PathTransition object which ‘runs’ the animation – moves the snowflake label along the path taking a specified duration

Animation example – standard JavaFX set-up

Pane layout manager

setID for CSS control (see Extra Notes)

All the action is in snowflake method

Set the scene up (adding a CSS stylesheet)

```
15 // Simple animation - the first snowflake of winter
16 public class Animation extends Application
17 {
18     // Main method for freestanding use (not BlueJ)
19     public static void main(String[] args) {
20         launch(args);
21     }
22
23     // the width and height of the canvas
24     public static int WIDTH = 300;
25     public static int HEIGHT = 200;
26
27     // layout objects
28     public Pane pane;
29
30     // method to start the app
31     public void start(Stage window) {
32         window.setTitle("Animation");
33         // create a simple layout manager (that doesn't control position - our snow
34         // give it a CSS Id so we can style it
35         pane = new Pane();
36         pane.setId("SnowStorm");
37
38         snowflake(150,0,10000);
39
40         // create the scene link to css, add to window and show the window
41         Scene scene = new Scene(pane, WIDTH, HEIGHT, Color.BLACK);
42         scene.getStylesheets().add("snowstorm.css");
43         window.setScene(scene);
44         window.show();
45     }
```

Snowflake animation

When we call snowflake we pass three arguments, the column, the delay before starting, and the duration of the fall

Make a snowflake Label, add to the pane, but 'translated' off the visible screen)

Make a Path, starting by moving to the top of the column, col, and then drawing a line to the bottom

PathTransition – run the path over duration dur, but also delay its start by delay (milliseconds)

Set the snowflake as object to move, and play animation

```
48    // Animate a snowflake at a given col, with start delay and duration
49    public void snowflake(int col, int delay, int dur) {
50        // a snowflake is just a text label
51        // create it, and set it off the screen initially, then add it to the pane
52        Label snowflake = new Label("*");
53        snowflake.setTranslateX(col);
54        snowflake.setTranslateX(-30);
55        pane.getChildren().add(snowflake);
56        // create a path that moves first to the top of the column and then traverses to the bottom
57        Path path = new Path();
58        path.getElements().add(new MoveTo(col,0));
59        path.getElements().add(new LineTo(col,HEIGHT));
60        // create a PathTransition object which waits for the delay time, and then traverses the
61        // path in the given duration.
62        PathTransition pt = new PathTransition();
63        pt.setDuration(Duration.millis(dur));
64        pt.setPath(path);
65        pt.setDelay(Duration.millis(delay));
66        // attach the snowflake to the transition, and then run the transition
67        // (NB: it runs in the background, and this method returns straight away)
68        pt.setNode(snowflake);
69        pt.play();
70    }
```

Class compiled - no syntax errors

How JavaFX runs the animation

- The method call ‘play’ returns immediately – so that your method call can return immediately (as it should)
- JavaFX use timer events to step through your animation – eg at sixty events per second
- Each event is detected by the secret event loop, and an event handler runs to do an update to your object’s position
- So if you specified a duration of 5 seconds, then each update needs to move 1/300th of the distance along the path (5 seconds times 60 frames = 300 updates)

SnowStorm

- Example class in project file
- Brings together the tree drawing and the snowflake animation into a little scene
- Basis for creating an animated Greetings Card
- Try and make your own!

JavaFX – more documentation

- I have deliberately kept our use of JavaFX quite simple, and the slides and labs include commented example code which teach you all you need to know for CI401, and will let you play around quite a lot.
- If you want to explore more on your own, some places to look include
  - https://docs.oracle.com/javafx/2/ - this has articles on many JavaFX topics which are not too difficult to follow, although they use Java code that is more advanced than we have seen so far
  - https://docs.oracle.com/javase/8/javafx/api/ - the detailed references for all the JavaFX classes. Far too much to read, but as you become familiar with using classes, you can sometimes look here at an individual class to find out how to do more
- But be aware that these go way beyond what you need to understand for CI401.

Extra notes

Additional JavaFX material (optional)

More JavaFX controls

JavaFX controls

- JavaFX provides all the standard controls (as Java classes) expected in a modern GUIs
- We used Label and Button in last week's lecture
- We also used Label, Button, RadioButton, ToggleButton, CheckBox, and Slider in the ControlsDemo class in last week's lab.
- Additional controls include MenuBar, Menu, ScrollBar, ScrollPane, ComboBox, ProgressIndicator, Accordion, and ListView (Google for 'javafx ScrollBar' (etc) to find out more).

More layout managers

More layout managers

- The only layout managers we have seen are GridPane and Pane
- Here are a few more useful layout managers
- All of these manage children, and those children can be controls, canvases, imageViews etc.
- But layout managers can also be children of other layout managers, to create more complex structure
- Children can be laid out on top of each other – if a child has a transparent background, the content underneath will show through
- A child's coordinate system operates relative to the parent's coordinate system (in general)

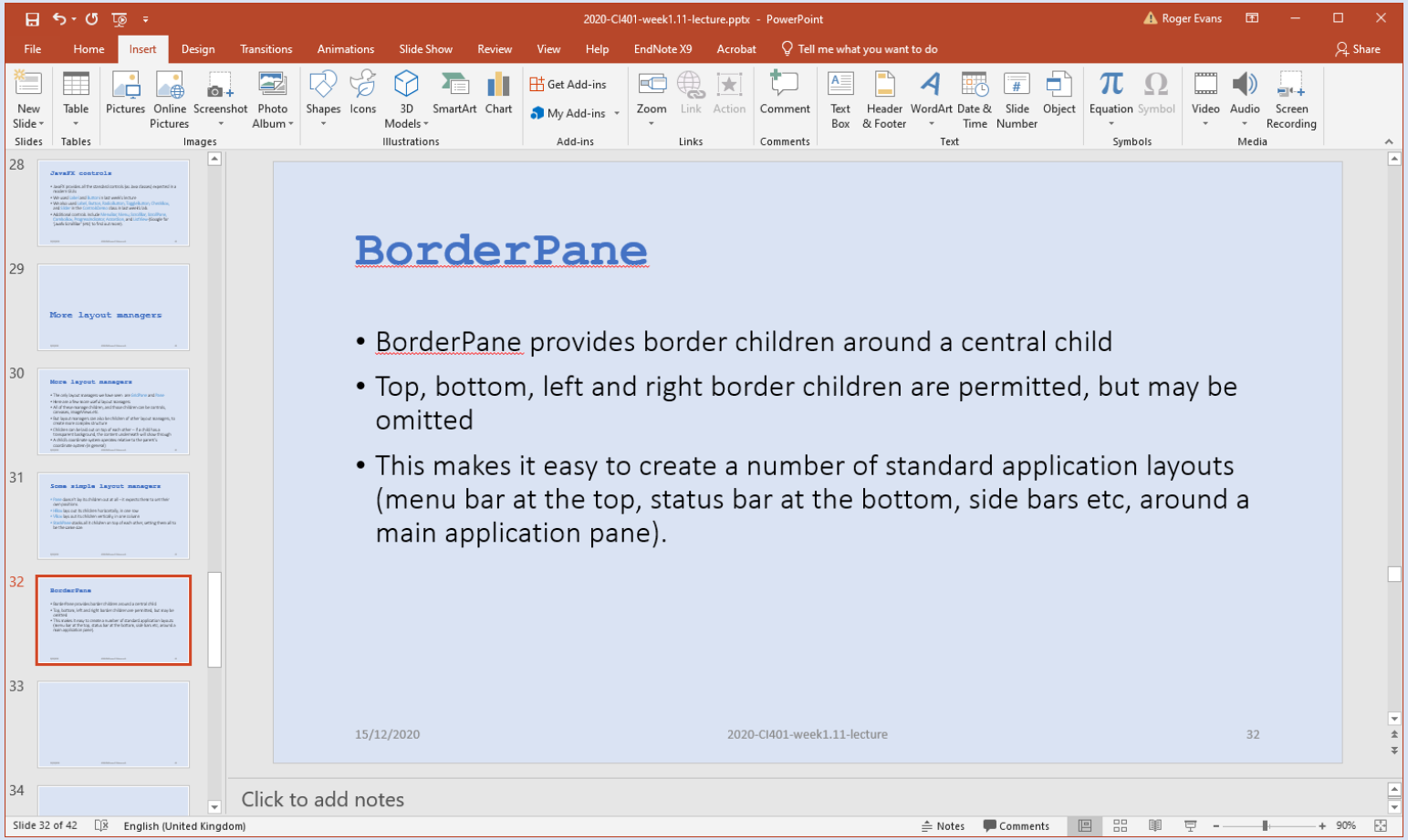
Some simple layout managers

- Pane doesn’t lay its children out at all – it expects them to set their own positions
- HBox lays out its children horizontally, in one row
- VBox lays out its children vertically, in one column
- StackPane stacks all it children on top of each other, setting them all to be the same size.

BorderPane

- BorderPane provides border children around a central child
- Top, bottom, left and right border children are permitted, but may be omitted
- This makes it easy to create a number of standard application layouts (menu bar at the top, status bar at the bottom, side bars etc, around a main application pane).

Example of complex layout management



More drawing methods

Other things you can do with GraphicsContexts (in Canvas objects)

- Other filled shapes:
  - fillRect(x,y,w,h);
  - fillRoundRect(x,y,w,h,aw,ah)
  - fillOval(x,y,w,h);
  - fillArc(x,y,w,h,s,e,c)
  - fillText(s,x,y);
- x,y are position
- w,h are size
- aw, ah are size of rounded corner
- s, e are start angle and extent angle
- Stroke versions – border lines without filling
  - strokeRect(x,y,w,h);
  - strokeRoundRect(x,y,w,h,aw,ah)
  - strokeOval(x,y,w,h);
  - strokeArc(x,y,w,h,s,e,c)
  - strokeText(s,x,y);
- Other
  - drawImage(img,x,y,w,h)
  - clearRect(x,y,w,h);
- Colours
  - Provided by the Color class

JavaFX and CSS

JavaFX and CSS

- JavaFX interfaces can be styled using Cascading Style Sheets (CSS), as used for styling web pages
- CSS allows us to control colours, fonts and font sizes, and spacing, among other things
- We are not going to learn CSS in detail here, but just show some simple examples, applied to the Snowflake example from the labs

Making JavaFX use a CSS file

- To use a CSS file (or several) with a JavaFX GUI, you have to attach the file to the Scene object for the GUI
- In Snowflake we say:

```
Scene scene = new Scene(pane, WIDTH, HEIGHT);
// add a CSS file to the scene
scene.getStylesheets().add("snowstorm.css");
```

- This adds snowstorm.css to the list of stylesheets the GUI uses
- (You can add further stylesheets if appropriate for your application)

Referencing GUI elements from the CSS file

- GUI elements can be referenced by (CSS) class, or by id
- A CSS class is assigned to common GUI controls, such as Label and Button. In these cases the class is the Java class name all lower case. So the CSS selector is .label or .button
- You can define an individual id to any GUI object using setId as follows:

```
pane = new Pane();
pane.setId("SnowStorm");
```

- This would allow the use of the CSS selector #Snowstorm to reference this element
- You can assign your own class to an instance by adding to the StyleClass list:

```
Label snowflake = new Label("*");
snowflake.getStyleClass().add("snowflake");
```

Using CSS styling

Here is snowstorm.css

The first definition matches anything with id SnowStorm (ie the Pane object), and sets its background black

The second matches anything with class snowflake (ie the snowflake Label object(s)), and sets the font, size and colour

```
1  /* CSS settings for SnowStorm demo */
2
3  #SnowStorm {
4      -fx-background-color: black;
5  }
6
7  .snowflake {
8      -fx-font: bold 20pt "courier new";
9      -fx-text-fill: white;
10 }
```

Lab exercises Week 1.11

Lab exercises – Week 1.11

- The DrawTree, SnowFlake and Snowstorm examples are provided for you to look at
- Lab1 asks you to add more trees etc to DrawTree (a bit like what we did in week 9 with print statements)
- Lab2 suggests things you could try and add to Snowstorm to make your own winter animation