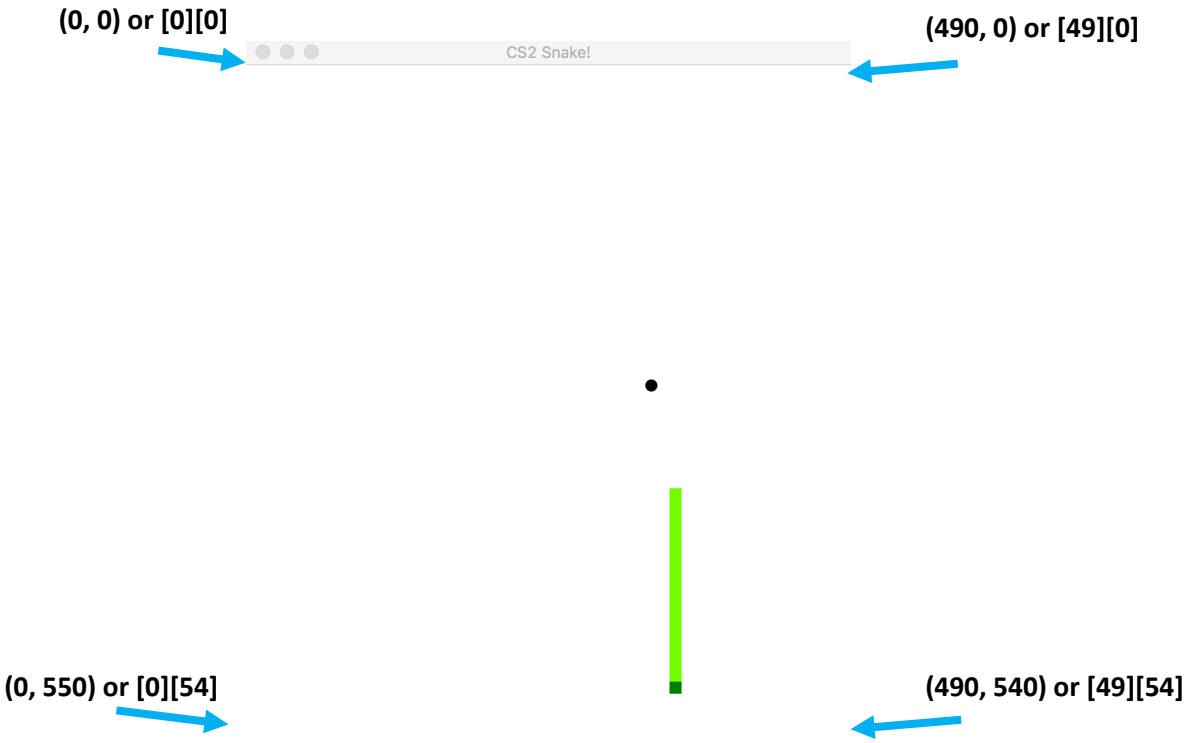
Snake Supplemental Instructions

Computer Graphics 101

One of the important things you'll notice while working on this game is that computer graphics applications employ a 2-dimensional coordinate system that is a little different that what you may be used to in mathematics.

The first thing you'll likely notice is that in computer graphics, the origin of a space starts at the top-left corner of the screen/window. Values of x get larger towards the right of the screen/window (columns), and y-values get larger as you approach the bottom of the screen/window (rows). Negative positions for x and y are allowed, but those objects would appear logically outside the viewable area of the window. Here's an annotated picture:



Normally you would place objects at their top and left coordinates. We have simplified this slightly as shown in the second part of the coordinate descriptions above. Each set of actual pixels will instead be grouped together to be considered as a smaller and more manageable number. While in actuality each part of the snake is 10 pixels on either side, each part will instead be represented by a single value in the 2-dimensional array map that is generated by the Level class. The circular Item is positioned based on the x and y coordinates for its center. Detecting if the head of the Snake has collided with the Item, a wall, or part of itself will be using the 2-dimensional array map values rather than the pixel values themselves to help ease you into collision detection algorithms. The map should be in row -> column order.

The Snake as a 2-dimensional array

The Snake itself will also be a 2-dimensional array. It will have a default max size that it should be initialized with and a length value that should keep track of the actual size of the snake (this could instead have been done by using a dynamic data type, but we will not discuss those until the second half of the course). The Snake should be in the format x, y. So the head of the Snake should have its x-value located at position [0][0] in the 2-dimensional array, while the y-value should be at position [0][1]. Similarly, as you go down the snake it should have the nth x-value in position [n-1][0] and the nth y-value in [n-1][1].

Something that is important to keep in mind is that this **will be opposite** from the map 2-dimensional array that uses the graphical system discussed prior. The X value is equal to the column value which is the second value in map and vice-versa for the Y value. **So in the snake 2-dimensional array, [0][0] is a column value in the map and [0][1] is a row value in the map.**

Enumerated Types

As you start looking through the starter project for this assignment you'll probably notice something that you may never have used before. Two of the Java files contain definitions for enumerated types and the UML stereotype `<<enumeration>>` is attached to their visual representations in the class diagram shown in the assignment.

Like classes, an enumerated type defines a new datatype in Java. In fact, they're just a very special kind of class definition. These definitions contain a series of related value names that are treated like distinct constant values in our programs. So, for example if we wanted to have a datatype for the days of the week we might define a type:

```
public enum DayOfWeek
{
     SUN, MON, TUES, WED, THURS, FRI, SAT
}
```

With a datatype defined, you can create variables of this type and use the values directly like other literal values. For example, in a main program you could say:

```
DayOfWeek aksarbenFarmersMarket = DayOfWeek.SUN;
```

Notice the datatype for the variable is the same as the enumerated type name, and the value is the fully-qualified value inside the enum (ie, it has the name of the enum out front followed by a dot).

An enumerated type name can be used anywhere a datatype can appear in a program (e.g., as a variable type, parameter type, or return type for a method).

Why Enums?

In previous programs you might have used a different strategy to represent the days of the week like assign int values 0, 1, 2, and so on to Sunday, Monday, Tuesday, … While this does work a potential issue arises when someone accidentally uses a valid int value, but one that is outside the logical range of days. For example, what day is -1 or 100?

Under the covers an enumerated type manages assignment of int constant values to each of these labels, and it prevents a programmer from ever specifying a non-existent value for a DayOfWeek variable.

Want to See More Examples?

For more details on enumerated types, see section 7.13 of your textbook (pg 386-390).