Processing Game Design Session 5 Teacher Outline

Goals

- Explore different end-of-screen options
  - bounce
  - wrap
- Discuss and create user controlled sprites
  - keypressed() vs keypressed
  - key variable, keyCode variable, and ASCII
  - review x and y axis on the screen
  - switch statements

End-of-Screen Options

The screen ends after passing 640 pixels to the right or 480 pixels down from the upper left corner of the running screen (this may be different depending on what you set the size of your window to). After your sprite passes that point, it is out in the unkown. There is no display. There is no background for the sprite to be drawn on to.

We have already figured out how to make the sprite “wrap around” the window. There is also another way to handle the sprite coming to the edge of the window. We can make it bounce off the edge. How do we make this happen?

Well let’s think about it like this. Imagine walking down a road. You start at 10th street, and start to walk north. You eventually hit 11th street and then 12th street. Once you make it to 15th street, there is a dead end. You don’t want to stop your walk so early. So, you head in the *opposite* direction. The streets you were walking on increased by 1. **So, when you walk in the opposite direction, they will decrease by 1.**

Instead of adding 1 to your position, you will subtract 1.

Code when walking originally:

```
ellipse(xposition += speed, yposition += speed, size, size);
```

Code when you went the opposite way:

```
ellipse(xposition -= speed, yposition -= speed, size, size);
```

Well how do we make this change? We can’t change the operator adding/subtracting speed and xposition. So, let’s keep it the original way. Instead of changing the operator, we can change the value of speed itself. How? We can multiply `speed` by `-1` when we hit an edge.

Wraping
`checkBoundsAndWrap()`

```
if(xposition >= width){
    xposition = 0;
}else if(xposition <= 0){
    xposition = width;
}

if(yposition >= height){
    yposition = 0;
}else if(yposition <= 0){
    yposition = height;
}
```

Bouncing
`checkBoundsAndBounce()`

```
if(xposition >= width){
    speed*=-1;
}else if(xposition <= 0){
    speed*=-1;
}

if(yposition >= height){
    speed*=-1;
}else if(yposition <= 0){
    speed*=-1;
}
```

When we mulitply by `-1` it's like we are changing directions instead of jumping to the other side of the window and moving in the same direction.

Wrapping	**Bouncing**
go to **opposite side**, then move in **same direction**	stay on **same side**, move in **opposite direction**

User Controlled Sprites

There are two ways you can control your sprites: by using the mouse or using the keyboard. For this, we're going to use the keyboard because it gives us many more options.

When you press a key on your keyboard, a signal is sent to your computer that says that specific key is pressed. Computers don't speak English, or any other human language; they speak binary. Binary is a different way of writing numbers. We write our numbers with 10 different symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Computers use two different symbols: 0 and 1. A combination of 0's and 1's are sent to your computer whenever a key is pressed. This combination tells the computer which key was pressed, and it is called ASCII (pronounced "as-key").

Luckily for us, we do not have to know what ASCII number is associated to each key because Processing allows us just to test with a single character. So when we test if a certain key is pressed, it will look this:

```
if(key == 'c'){
```

```
/* your code here*/

}
```

Okay here's the deal. when you press a key when your processing game is up and running, you automatically call a specific function called `keyPressed()`. Once the function is called you can then test to see what key has been pressed. Processing has also given us a built-in variable called `key` that has the most recently pressed key stored. So we can do this:

```
void keyPressed(){ //called everytime a key is pressed when game is running
    if(key == 'c'){

        /* your code here */

}
```

We actually have write in the `keyPressed()` function into our code to test for what key is pressed.

Here's the tricky part. If we want to move our character with the arrow keys (up, down , left, and right), we have to add in something else to our code. Arrow keys are not part of ASCII, meaning they don't have an ASCII number assigned to them. They have numbers that your cumputer knows about, but numbers that aren't easily accesible to us. So, Processing has given us `keyCode` and `CODED`, which when used in the `keyPessed()` function allows us to test for an arrow key being pressed. Here's how:

```java
void keyPressed(){

if(key == CODED){
    if(keyCode == UP){
        /*do something when up arrow key is pressed*/
    }
}

}
```