Humans playing VR game Beat Saber move faster than what Steam thought was ‘humanly possible’

“Some people move so fast when they play the VR game Beat Saber, a rhythm game that’s basically Dance Dance Revolution with lightsabers, that Valve developers have had to issue a fix. People were moving so fast that Steam VR couldn’t track their movements.

“[D]evelopers noted that they had to ‘Increase limits of what we thought was humanly possible for controller motion.’”

Administrivia

- Assignments:
  - Reading Check 7 due @ 3:30 pm tomorrow (2/21)
  - Controlling Elli [submit] due tomorrow (2/21)
  - Word Guessing [checkoff/submit] due Saturday (2/23)
  - Living Computers Museum Report due Tuesday (2/26)

- “Big Ideas” lecture: CS and Ethics
Review: Rectangle Detection

if( (mouseX >= x) &&
    (mouseX <= x + w) &&
    (mouseY >= y) &&
    (mouseY <= y + h) )
Review: Rectangle Detection

```java
if ( (mouseX >= x) && (mouseX <= x + w) &&
     (mouseY >= y) && (mouseY <= y + h) ) {
    // do something
}
```

- **Potential Uses:**
  - To detect on *every frame*, place in `draw()` or function called by `draw()`
    - e.g. hover detection – change color when mouse is over rectangle
  - To detect on a mouse click, place in `mousePressed()`
    - e.g. a button that the user can click on

---

*see input/output lecture*
Circle Detection

- A circle is defined as all points in a 2-D plane that are equidistant from a center point
  - In mathematical terms, the set of all points \((x, y)\) that satisfy:
    \[
    (x - centerX)^2 + (y - centerY)^2 = radius^2
    \]

- To detect the mouse being \textit{inside} the circle, this becomes an inequality
  - \[
  (mouseX - centerX)^2 + (mouseY - centerY)^2 \leq radius^2
  \]

- In Processing:
  ```java
  if( (mouseX-x)*(mouseX-x)+(mouseY-y)*(mouseY-y) <= r*r ) {
    // do something
  }
  ```
Creating a Button

- The button needs to be visible to the user
  - Use `rect()` or `ellipse()` to draw on your canvas

- Generally, the user should know what the button is for
  - Use `text()` to either label the button or put directions somewhere else on screen
    - Often, `textAlign(CENTER)` makes finding appropriate coordinates easier
Button Demo

- Create a “Clear” button for phone or calculator
  - Use a 500 x 500 canvas
  - The button should be of size 46 x 46 and white at position (152, 302)
  - Labeled button with a red “C” text roughly centered
    - Hints: use `textSize(40)` and `textAlign(CENTER)`
  - When the mouse is hovering over the button, it should turn yellow: `color(255, 255, 98)`
    - Requires Active Mode
  - When the button is clicked, it should print "Cleared!" to the console
Grids and Boards

- Grids can be created using nested for-loops

- **Example**: numeric keypad
Grid Demo

- Grids can be created using nested for-loops
Your Board “State”

- The **state** of your board indicates its current configuration
  - In some applications, this never changes
    - *e.g.* numeric keypad
  - In other applications, this will change over time
    - *e.g.* tic-tac-toe

- Board state is typically represented via an **array**
  - Naturally ties a numeric location on your grid to the symbol/value currently associated with that cell
  - Similar to **pixels[]** holding the color “state” of your drawing canvas
Labeling Our Grid

- For the numeric keypad, the board state is the set of (ordered) key labels:
  - `char[] keypad = {'1', '2', '3', ..., '*', '0', '#'};`
Grid Detection

- Detection of mouse location within a grid

```c
int x_index, y_index;
if ((mouseX >= x) && (mouseX <= x+w) &&
    (mouseY >= y) && (mouseY <= y + h) ) {
    x_index = int( (mouseX-x)/(w/2) );
    y_index = int( (mouseY-y)/(h/2) );
}
```
Keypad Grid Click Detection

- Clicking on the keypad should add to the phone number you are trying to dial
  - Use a `String` to store and display on the canvas

```java
String dialed = " ";
```
Clear Functionality

- Our phone number should “reset” or “clear” when we click the clear button
  - Currently, it prints "Cleared!" to the console

```javascript
instead of set
dialed = " ";
```
Summary

- Sketched the idea on paper
- Planned out coding representations
- Built on previous work by adding one function or idea at a time
- Ran the program after *every* improvement to make sure that it worked correctly
  - Unit and integration testing!!!