YEAH Hours: Enigma

11/6/18
Ryan Eberhardt
Objects

- Recall: arrays are ordered collections of things
- An object is an *unordered* collection of "key-value pairs"
  - "Key-value pair" is a fancy term for an idea you're already familiar with!
  - In a phonebook, each entry is a key-value pair. "Ryan Eberhardt" is a key and "(123) 456-7890" is a value
  - In the Explore Courses database, you might find a key-value pair for every class, where a key like "CS 106AJ" has the value "Programming Methodology in JavaScript"
- We will be talking much more about objects and how to use them in the next two weeks!
Creating objects

- Let's create an object to represent a point in space:
  
  ```javascript
  let point = {x: 5, y: 10};
  ```

- This object now contains two entries (a.k.a two key-value pairs)

- We can get the contents of the object like this:
  ```javascript
  console.log(point.x);
  console.log(point.y);
  ```
Creating objects v2

- Javascript also allows you to add onto objects at any time. That means you can also create an object by creating an empty one and then adding entries:

```javascript
let point = {};
point.x = 5;
point.y = 10;
```

- This produces an identical object to the previous slide. We can use it the same way:

```javascript
console.log(point.x);
console.log(point.y);
```
Objects in Enigma

- As it turns out, nearly *everything* in JavaScript is an object!
  - When you do `console.log("Hello world".length);`, you are taking a string object and looking up the value for the `length` key
  - When you create a `GRect`, you are creating an object with keys like `getWidth`, `getHeight`, `setColor`, etc., and values that are functions

```javascript
let rect = GRect(200, 100);
console.log(rect);
```

```javascript
Object {
  contains: contains(),
  getBounds: getBounds(),
  getCanvas: getCanvas(),
  getColor: getColor(),
  getFillColor: getFillColor(),
  getHeight: getHeight(),
  getLineWidth: getLineWidth(),
  getLocation: getLocation(),
  getSize: getSize(),
  getWidth: getWidth(), ...
}
```
Objects in Enigma

- As it turns out, nearly **everything** in JavaScript is an object!
  - When you do `console.log("Hello world").length);`, you are taking a string object and looking up the value for the `length` key.
  - When you create a GRect, you are creating an object with keys like `getWidth`, `getHeight`, `setColor`, etc., and values that are functions.
    - As such, you can attach extra properties and functions to a GRect, just like we saw on the last slide! I'll explain this technique in a few slides :(
Objects in Enigma

- As it turns out, nearly everything in JavaScript is an object!
  - When you do `console.log("Hello world").length`;, you are taking a string object and looking up the value for the `length` key
  - When you create a GRect, you are creating an object with keys like `getWidth`, `getHeight`, `setColor`, etc., and values that are functions
  - As such, you can attach extra properties and functions to a GRect, just like we saw on the last slide! I'll explain this technique in a few slides :)

- This assignment will require you to manipulate many objects, so make sure you have some basic idea of what they are!
Enigma

Interactive demo: http://web.stanford.edu/class/cs106aj/assignments/assign5-milestones/
Enigma logistics

- Due next Friday (Nov 16)
- Partner assignment
- Broken into milestones
  - Follow along with the interactive demos: [http://web.stanford.edu/class/cs106aj/assignments/assign5-milestones/](http://web.stanford.edu/class/cs106aj/assignments/assign5-milestones/)
This assignment emulates a real (complicated) machine!

- We have several handouts about the workings of the machine:
  - Slides from lecture on cryptography
  - Theory of the Enigma machine
  - Assignment handout
- The assignment handout is long, but worth paying attention to
- Make sure you understand what you're trying to do before starting to write code!
Milestone 1: Create the Keyboard

- Create a GCompound (with the reference point at the center) containing two GOvals and a GLabel
- Add the GCompound to the screen using coordinates from a KEY_LOCATIONS constant
  - From the handout: `KEY_LOCATIONS[ch.charCodeAt(0) - "A".charCodeAt(0)].x`
Milestone 2: Making keys interactive

- **Problem:** There are a lot of things on the screen that are supposed to do different things when I click them. If I write all the code in my click handler function, it's going to be really long and ugly!
- **Solution:** Event forwarding
  - Described in page 4 of handout, "Forwarding mouse events to graphical objects"
- **Basic idea:** Did the user click an object that is capable of handling clicks? If so, pass the event onto that object instead of handling it ourselves
  - Then, we'll modify all our GCompounds so that they know what to do when they are clicked on!
Milestone 2: Making keys interactive

- Simplified event listener:

```javascript
function mousedownAction(e) {
    let target = gw.getElementAt(e.getX(), e.getY());
    if (target knows how to handle events) {
        Tell the target that the mouse pressed down on it
    }
}
gw.addEventListener("mousedown", mousedownAction);
```
How do we create a GCompound that's capable of handling clicks on itself?

- Add a mousedownAction function to it!

```javascript
function makeKey(i) {
    let key = GCompound();
    let label = GLabel(figure out what goes here :)
    // add the shapes/label to the GCompound
    key.mouseDownAction = function() {
        label.setColor(KEY_DOWN_COLOR);
        // more code added in later milestones
    }
    return key;
}
```
Milestone 2: Making keys interactive

- Now, our main event listener can call this second listener:

```javascript
function mousedownAction(e) {
    let target = gw.getElementAt(e.getX(), e.getY());
    if (target.mousedownAction !== undefined) {
        target.mousedownAction();
    }
}

gw.addEventListener("mousedown", mousedownAction);
```
Milestone 3: Creating the lamp panel

- Create a GCompound for each lamp, similar to what was done for keys
  - There are only two parts to the GCompound instead of 3
  - No need to attach mousedownAction or mouseupAction methods (because lamps don't respond to clicks)
  - However, we do need to attach a label property so that the GLabel can be accessed in the future
function makeLamp(i) {
    let lamp = GCompound();
    let label = GLabel( figure out what goes here :) );
    // Do other stuff to make the lamp
    lamp.label = label;
    return lamp;
}
Milestone 4: Connect the keyboard to the lamp panel

- When we press one of the keys, the corresponding lamp should illuminate.
- In the `runEnigmaSimulation` (the main function), create an `enigma` object that contains an array of keys and an array of lamps.
- Modify the main `mousedownAction` and `mouseupAction` functions to pass this object when dispatching events.

```javascript
function mousedownAction(e) {
    let target = gw.getElementAt(e.getX(), e.getY());
    if (target.mousedownAction !== undefined) {
        target.mousedownAction(enigma);
    }
}
```
Milestone 4: Connect the keyboard to the lamp panel

- When we press one of the keys, the corresponding lamp should illuminate
- In the runEnigmaSimulation (the main function), create an enigma object that contains an array of keys and an array of lamps
- Modify the main mousedownAction and mouseupAction functions to pass this object when dispatching events
- The key's mousedownAction function can now receive the array of lamps via the enigma parameter, and can change the color of the label attached to the appropriate lamp
Milestone 5: Add rotors in their default positions

- Create a GCompound for each rotor; the handout details this
- Attach a string to each GCompound:
  ```python
  rotor.permutation = ROTOR_PERMUTATIONS[i];
  ```
  (where i is the index of the rotor)
  - We don't do anything with this string at this stage, but later on, this string dictates a substitution cipher that this rotor implements. The handout explains this.
Milestone 6: Making rotors clickable

- When a rotor is clicked, it needs to advance to the next letter!
- When creating a rotor object, add an `offset` property indicating its current position (the default position, showing "A", is an offset of 0)
- Need to add a `clickAction` method to the rotor object, similar to the keys
- When the rotor's `clickAction` method is called by a main dispatcher `clickAction` function, it should increment `offset` and update the `GLabel` to show the next letter in the alphabet
  - Make sure the rotor wraps from Z back to A when we click it 27 times!
Milestone 7: Implement one stage in the encryption

- From milestone 5, each rotor should have a permutation string attached to it. The first rotor's permutation looks like this:
  EKMFLGDQVZNTOWYHXUSPAIBRCJ
- That string gives us this input-to-output mapping:
- However, if the rotor is in offset 1 (instead of offset 0), we should encrypt A like this:
Milestone 7: Implement one stage in the encryption

- From milestone 5, each rotor should have a permutation string attached to it. The first rotor's permutation looks like this:
  EKMFLGDQVZNTOWYHXUSPAIBRCJ
- That string gives us this input-to-output mapping:
  \[
  \begin{array}{cccccccccccccccccccc}
  \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
  \end{array}
  \]
- If the rotor is in offset 2:
  \[
  \begin{array}{cccccccccccccccccccc}
  \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
  \end{array}
  \]
Milestone 7: Implement one stage in the encryption

- Your job is to implement an `applyPermutation` function that takes an index (in the alphabet), permutation string (like the one on the previous slide), and offset, and returns the index (in the alphabet) of the resulting character.
- Then, use this function to illuminate the correct lamp when a key is pressed.
Milestone 8: Implement the full encryption path

- After passing through all 3 rotors and the reflector, current flows backwards through the rotors:
Milestone 8: Implement the full encryption path

- You need to construct the *inverse* of a rotor's permutation string. This process is described fairly well in the handout.
- Implement an `invertKey` function to return an inverted permutation string:
  - Test it using `console.log` before you continue!!
  - If you call `invertKey(invertKey(permutation))`, you should get the same thing as the original permutation.
- Then, update the key's event handler functions to call `applyPermutation` 7 times (once on each of the rotor permutations, then on the reflector permutations, then on the inverted rotor permutations in reverse order).
Milestone 9: Implement rotor advance on pressing a key

- Whenever a key is pressed, the fast rotor should be advanced BEFORE doing the encryption.
- When the fast rotor rolls over from Z to A, the medium rotor should be incremented. When the medium rotor rolls over, the fast rotor should be incremented.
- This milestone isn't long, but it will be challenging, because it ties together all the previous milestones, and any latent mistakes you made will be exposed here!