CS 154 Project Proposal
RackEnigma
Team Name- The Racketeers

Anuj Jitendra Diwan	Soumya Chatterjee	Yash Sharma
170070005	170070010	17D070059

1 Description

It's World War 2, except in the modern world. We are the head computer engineers for Nazi Germany, and the Axis powers want a way to secretly communicate with each other. Since Racket is the only hope against the Scheming Allied forces, we are commissioned to make the best encryption machine the world has ever seen.

- We propose to develop the famous Enigma machine from the WW2. Wikipedia page linked - The Enigma Machine
- The machine is based on dynamic state encryption that encrypts standard English characters to incomprehensible gibberish.
- The state/seed in the machine changes on every input.
- It is a self-reciprocal machine, meaning given the same initial state/seed on two Enigma machines i.e same encryption E, using the encrypted output obtained from one machine as input on the other, the original message is obtained i.e applying the encryption on the encrypted message gives the original message.

2 Design - Inputs and Output

2.1 Encryption Mode

- Input: A string of upper case alphabets (the message) to be encrypted
- Output: The encrypted message, seed and initial configuration of the rotors to be used for decryption The seed and initial configuration of rotors are randomly chosen inside the machine

2.2 Decryption Mode

- Input: The decryption seed, initial configuration of the rotors used during encryption and the encrypted message
- Output: The decrypted message

2.3 Brute Force Solver

- Input: The encrypted message, seed and a known prefix
- Output: Initial configuration of rotors used for encryption

3 Solution idea

- The encryption E can be modelled as a 26×26 permutation matrix (link here). The n length input message is expressed as a $26 \times n$ matrix where each column is each character encoded using a one-hot encoding.
- Since the encryption must be self-reciprocal, E must satisfy $E = E^{-1}$ so that if M is the original message, EM is the encrypted message, and $E(EM) = E^{-1}EM = M$ is the encryption of the encrypted message i.e. the original message.
- There are 3 rotors L, M and R, each rotor existing in one of 26 states. These rotors are also expressible as 26×26 permutation matrices, each expressible as

  $$\rho^n R \rho^{-n}$$

  where ρ is the cyclic permutation mapping A to B, B to C, and so forth. After each key press, the rotors turn, changing the transformation.

- The reflector U is a special self-reciprocal permutation matrix. We are using U as the char–>(char+13) transformation.
- The plugboard transformation is a user-controllable permutation matrix (mapping) that adds another layer of complexity to the machine.
- The Enigma transformation for each letter can be specified as a product of permutations. Let P denote the plugboard transformation, U denote that of the reflector, and L, M, R denote those of the left, middle and right rotors respectively. Then the encryption E can be expressed as

$$E = PRMLUL^{-1}M^{-1}R^{-1}P^{-1}$$

and finally the encryption transformation can then be described as

$$E = P\left(\rho^{n}R\rho^{-n}\right)\left(\rho^{j}M\rho^{-j}\right)\left(\rho^{k}L\rho^{-k}\right)U\left(\rho^{k}L^{-1}\rho^{-k}\right)\left(\rho^{j}M^{-1}\rho^{-j}\right)\left(\rho^{n}R^{-1}\rho^{-n}\right)P^{-1}$$

3.1 Sample Input and Output

1. Encryption

```
Input:
    HELLO WORLD
Output:
    279066841     (seed of random number generator)
    (N E S)       (initial rotor configuration)
    IXIKQ DNTEO   (encrypted message)
```

2. Decryption

```
Input:
    279066841
    NES
    IXIKQ DNTEO
Output:
    HELLO WORLD
```

4 Additional Features

- **Use of Macros**:
  - Used macro `m-ind` for `C++` style indexing into the matrices, mainly for debugging purposes.
  - Used list-comprehension in the brute force solver.
- **Object Oriented Design**: Program divided into many GUI-based classes for handling different things. eg Keyboard, Rotors(knobs), Control Buttons, Seed Dialog
- **Higher Order Functions**:
  - `set-enigma-mode!` is a procedure that returns an encryption/decryption procedure based on it's input.
  - Multiple level of abstractions used in the Brute Force Solver
- **Abstraction**
  - The mentioned matrix operations have been abstracted out into encryption and decryption.
  - Different Racket files handle different tasks. For example, the knobs.rkt handles rotor tasks. The turing.rkt handles the brute force solver.
- **Features not discussed in class**:

- Matrices in Racket
- Callback and events in GUI
- Random numbers in Racket

4.1 Software Packages and Tools

- racket/gui
- 2htdp/image
- math/matrix

5 Limitations

- The Brute Force Solver, despite the best of our efforts, takes time (about 10 minutes). However it does display an answer as soon as it finds one.
- Another drawback in the Solver is the need of the guess to be a prefix.
  The guess could be a word from in between but then the position will have to be guessed and hence the state of the machine at that position. This, although achievable, predictably would take undesirable amount of time and hence was dropped.