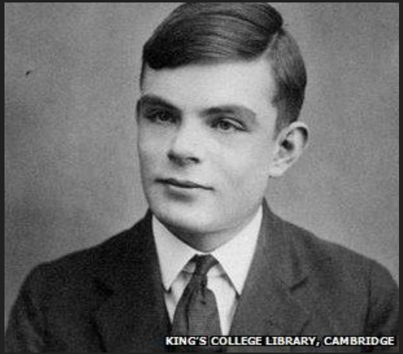
OQaml

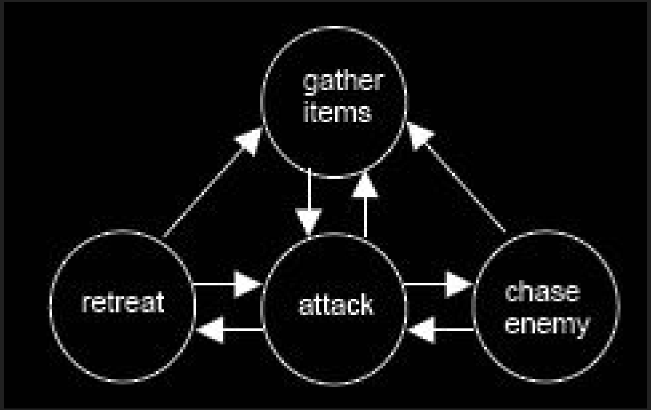
A OCaml-based QASM

Johannes Otterbach

Turing Machine



- Alan Turing (1936)
- Led to the definition of *Computability*
- A program is representable by
  - a finite set of states
  - a set of transitions
  - a set of instructions
  - an initial state



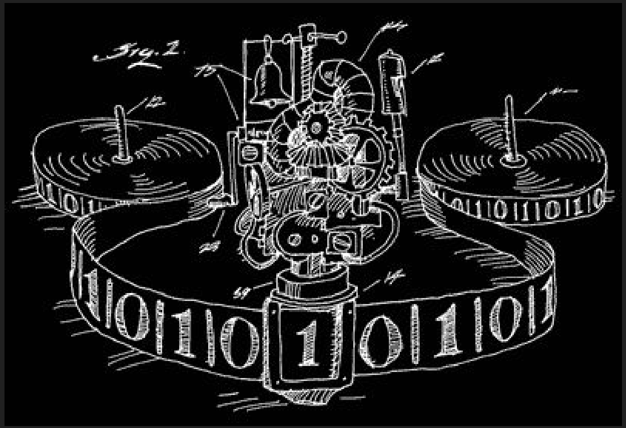
https://www.researchgate.net/publication/228920087_Evolutionary_neural_networks_applied_in_first_person_shooters

FSM Representations

- Encode information in bits 0, 1

- Boolean Logic: Operations on bits
  - NOT : bit -> bit
  - OR : bit -> bit -> bit
  - AND : bit -> bit -> bit
  - XOR : bit -> bit -> bit
  - …

- Universal gate sets:
  - NOT and AND
  - NOT and OR
  - AND or XOR
  - ...



Classic to Quantum

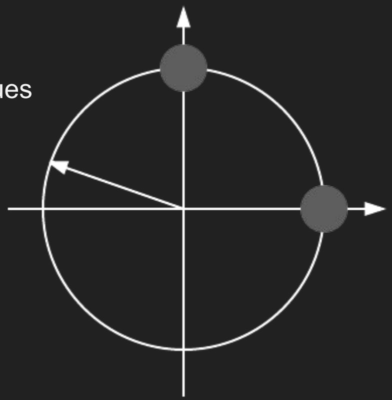
- Classical we can have only one state at a time:

$$0 \text{ XOR } 1$$

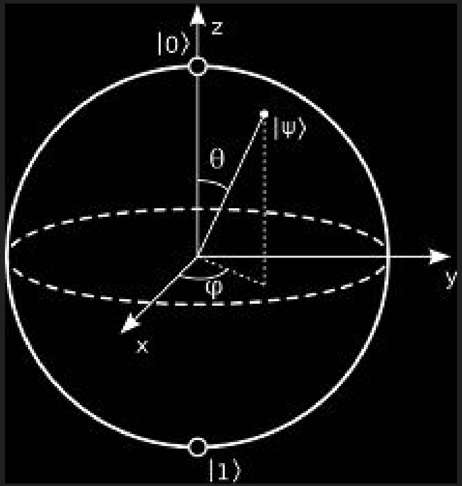
- Quantum Mechanics:

$$\alpha|0\rangle + \beta|1\rangle$$

Real values



Complex values



State Transformation

- QM states connected by gates

$$|\Psi_f\rangle = U|\Psi_i\rangle$$

- Computational basis

$$|\Psi\rangle = |b_0 b_1 ... b_{n-1}\rangle = |b_0\rangle \otimes |b_1\rangle \otimes ... \otimes |b_{n-1}\rangle$$

- Series of gates is a circuit

$$|\Psi_f\rangle = U_n U_{n-1} \ldots U_1 |\Psi_i\rangle$$

- “Time” flows from right to left

OQaml

- OCaml based implementation of Quil
- Statically typed, functional programming language
- Let's you program with "mathematical" notation

```ocaml
(** Gate operations on a qvm containing a classical bit register and a quantum
    state both indexed by integers. *)
type gate =
  | I of int
  | X of int
  | Y of int
  | Z of int
  | H of int
  | PHASE of float
  | RX of float * int
  | RY of float * int
  | RZ of float * int
  | CNOT of int * int
  | SWAP of int * int
  | CIRCUIT of gate list
  | MEASURE of int
  | NOT of int
  | AND of int * int
  | OR of int * int
  | XOR of int * int

(** The actual QVM type as a record *)
type qvm =
  { num_qubits: int;
    wf: V.vec;
    reg: int array;
  }

(** Initializes a QVM with a classical register of [reg_size] bist and [int]
    qubits in their ground-states*)
val init_qvm : ?reg_size:int -> int -> qvm

(** Applies [gate] to a [qvm] resulting in a new [qvm] state *)
val apply : gate -> qvm -> qvm
```

Evaluating small circuits

- Structural similarity between CIRCUIT and GATE

$$|\Psi_f\rangle = U|\Psi_i\rangle$$

$$|\Psi_f\rangle = U_n U_{n-1} \ldots U_1 |\Psi_i\rangle$$

- OQaml: Circuits are Gates!

- Example: 1 Qubit gates

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \qquad \alpha = \beta = \delta = \frac{\pi}{4}, \ \gamma = 0$$

OQaml:

```
let pg idx = Q.CIRCUIT [Q.PHASE (pi4); Q.RZ(pi4, idx); Q.RY (0.0, idx); Q.RZ (pi4, idx)];;
```

Demo

More examples

- 2 Qubit gate

$$\mathrm{SWAP}[i,j] = \mathrm{CNOT}[i,j] \otimes \mathrm{CNOT}[j,i] \otimes \mathrm{CNOT}[i,j]$$

- OQaml:

```
let swap i j = Q.CIRCUIT [Q.CNOT (i,j); Q.CNOT (j,i); Q.CNOT (i,j)];;
```

- Assert we are correct:

```
let tqvm = Q.apply (Q.X 0) (Q.init_qvm 2);;
Q.apply (swap 0 1) tqvm = Q.apply (Q.SWAP (0,1)) tqvm;;
```

Demo