A multimodal interface for chess

How we made people gesticulate and scream at their computers

SAPIENZA
UNIVERSITÀ DI ROMA

Giuseppina Iannotti, 1938436
Davide Marincione, 1927757

Sapienza, University of Rome

A. Y. 2023 - 2024

The Idea

Remember this?

Figure: Wizard's Chess, Harry Potter and the Philosopher's Stone

The Idea 2

Know this feeling?

Figure: Some stock image of an hand holding a chess piece.

Before that

We have to get from here...

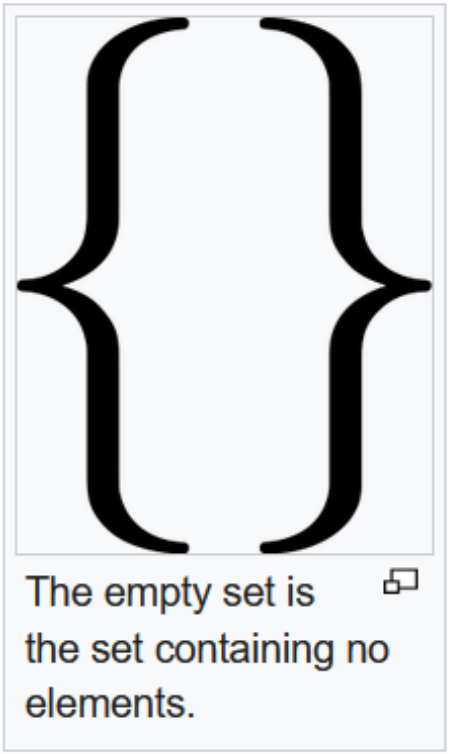


Figure: What we have.

To here!

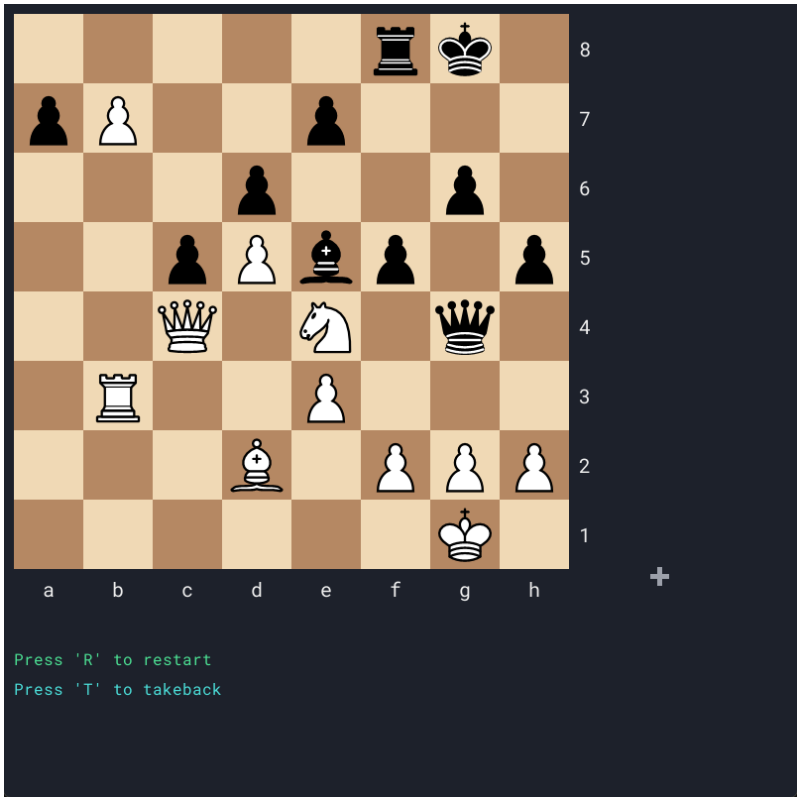


Figure: What we want.

We need some OOP

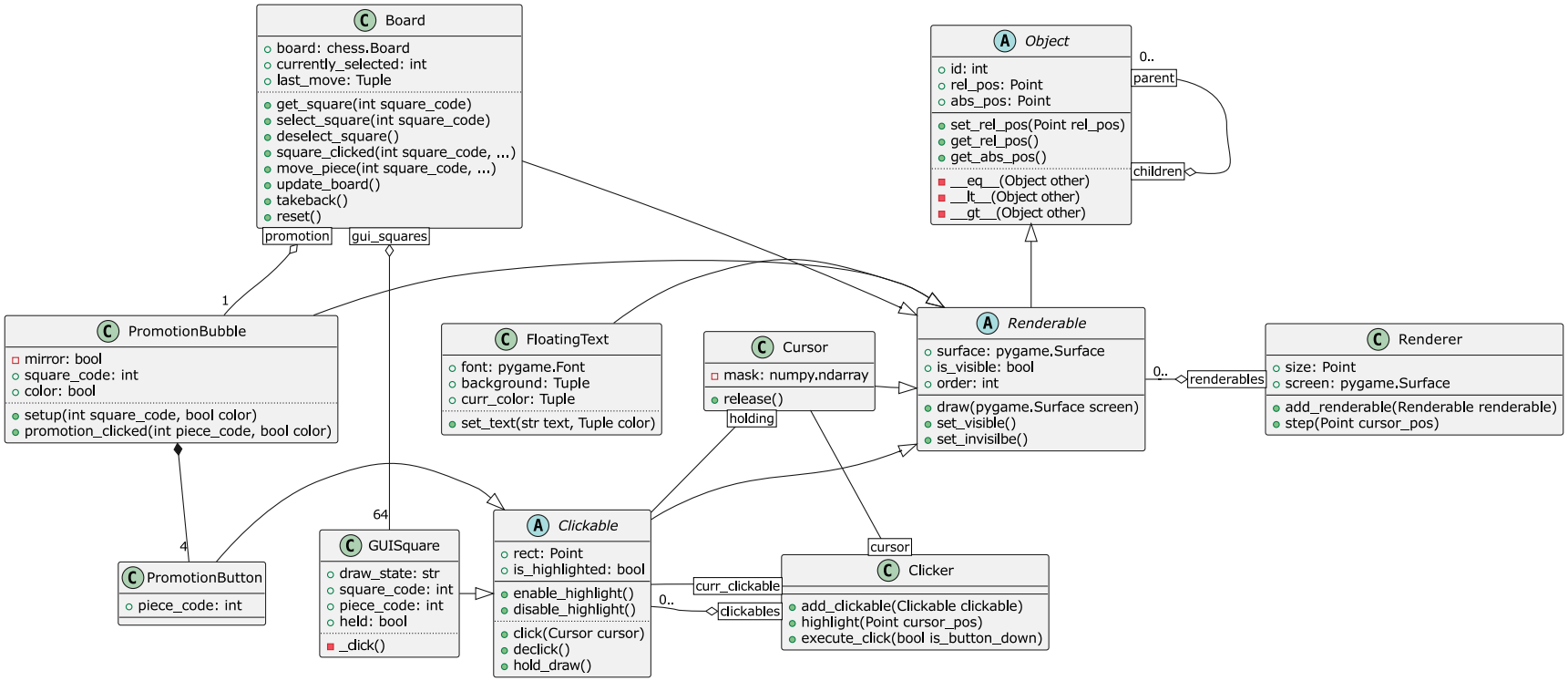


Figure: Class diagram of the game's elements

A bit in detail 1

The `Renderer`... draws `Renderables`!

1. Keeps track of them.
2. Draws them based on each object's `order` attribute.
3. Draws them only if they are set to visible.

The `Clicker`:

1. Keeps track of the `Clickables`.
2. Highlights the current `Clickable`, calls its click/declick method.
3. Drives hold/release with `Cursor`.

A bit in detail 2

Our `Cursor` is this neat thing:

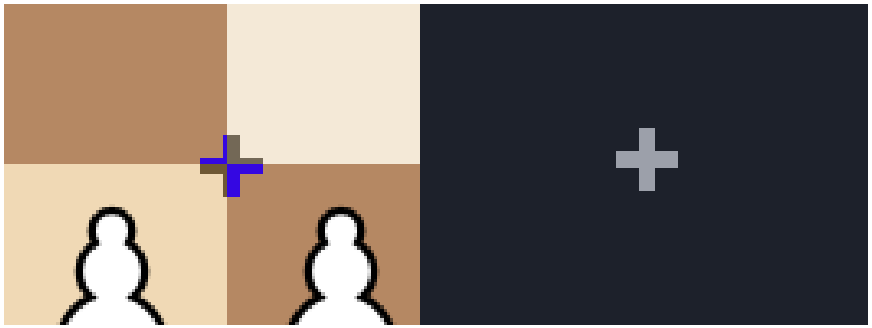


Figure: Our `Cursor`.

It is simple, but we are pretty happy about it:

1. It is extremely visible, because of the dynamic color

$$c^* = (c + 128) \bmod 256.$$

2. It can hold pieces.
3. Being stylistically different might have helped!

A bit in detail 3

The `Board`:

1. Wraps a `chess.Board` object (and all its complicated chess logic).
2. Handles the state of all the `GUISquare` and that of the `PromotionBubble`.
3. Plays audio when moves are done!

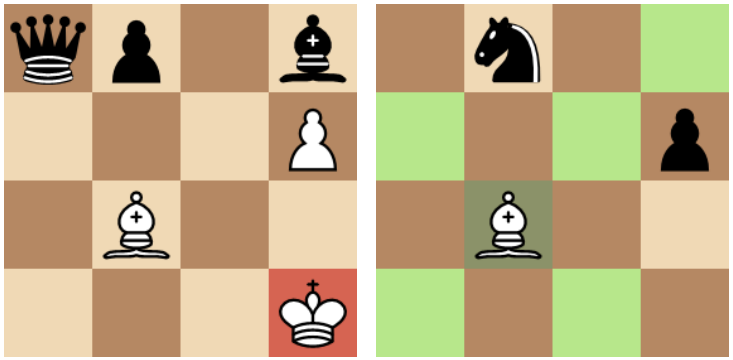


Figure: Examples of `GUISquare` states

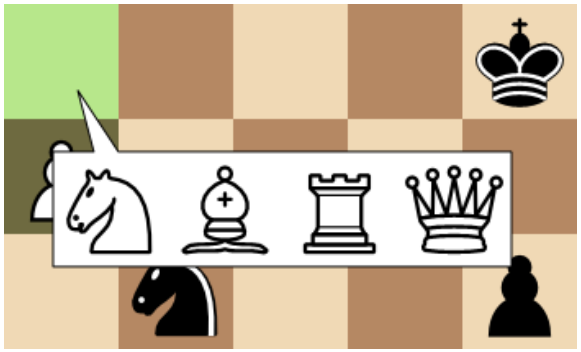


Figure: What `PromotionBubble` looks like

The main loop

All of this runs on the main thread, within the loop:

1. Update cursor with latest mouse or hand position.
2. `clicker.highlight(cursor_pos)`.
3. Resolve events, such as mouse clicks, key presses (quit game, takebacks), and moves done (for the AI).
4. Resolve voice commands.
5. `renderer.step()`.
6. Run metrics recorder.

Dragonfly? What's that?

Not this. . .

Figure: The insect.

This!

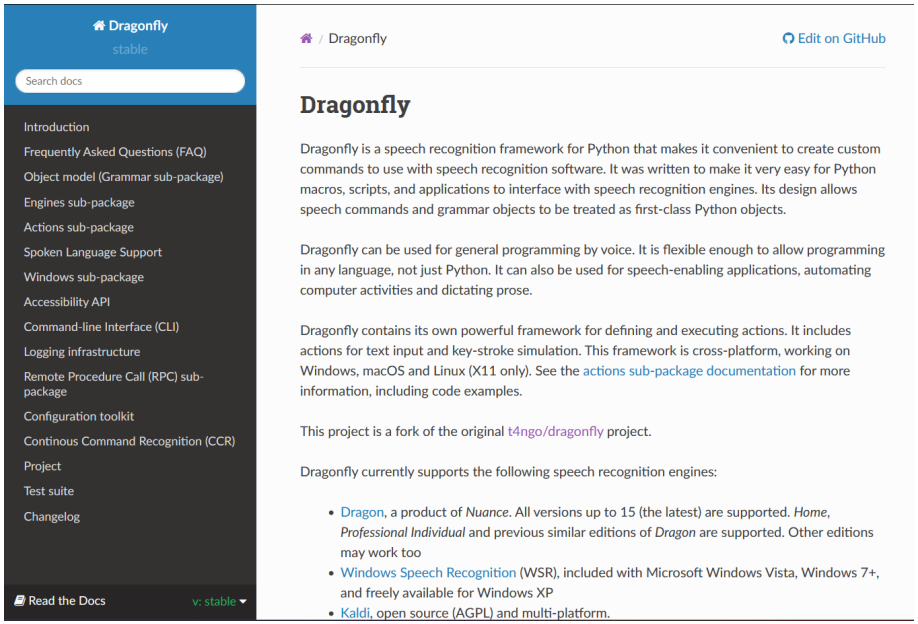


Figure: The package.

Hey, but what is a Rule?

```
class PromoteRule(CompoundRule):

    def __init__(self, manager):

        super().__init__()
        self.manager = manager

    spec = "promote [(<src_piece> | <src_square>)] to <prm_piece>"
    extras = [
        Choice("prep", prep_map),
        Choice("src_piece", piece),
        Choice("prm_piece", prm_piece),
        Compound( name = "src_square", spec = "<file> <rank>", extras = [ Choice("file", file_map), Choice("rank", rank_map)],
                  value_func = lambda node, extras: (extras["file"], extras["rank"]))
    ]
```

Figure: Example of Compound Rule

Each rule is instantiated as a `Compound Rule`, with parameters :

- `spec` : Compound specification for the rules root element
- `extras` : Extras elements referenced from the compound spec, like choices for prepositions, pieces, and squares.

Hey, but what is a Rule? - ALTERNATIVE

Each rule is instantiated as a `Compound Rule`, with parameters :

- `spec` : Compound specification for the rules root element
- `extras` : Extras elements referenced from the compound spec, like choices for prepositions, pieces, and squares.

It is characterized by a process recognition method that bla bla bla

Rules I

Table: Rules I

Rule Name	**Specification**
Move Rule	"move ([<src_piece>] | [<src_piece> [<prep> <src_square>] to <tgt_square>] | [[<prep>] <src_square> to <tgt_square>]) [and promote to <prm_piece>]"
Capture Rule	"capture (<tgt_piece> [<prep> <tgt_square>] | <tgt_square>) [with (<src_piece> [<prep> <src_square>] | <src_square>)] [and promote to <prm_piece>]"
Promote Rule	"promote [(<src_piece> | <src_square>)] to <prm_piece>"

Rules 2

Table: Rules II

Rule Name	**Specification**
Castle Rule	"(castle <special_direction> | <special_direction> castle)"
Piece Rule	"<src_piece> ([<prep>] <tgt_square> |in <src_square> <verb> [<prep>] (<tgt_square> | <tgt_piece> [in <tgt_square>]) | <verb> [<prep> <src_square>]([<prep>] <tgt_square> | <tgt_piece> [in <tgt_square>])) [and promote to <prm_piece>]"
Square Rule	"<src_square> <verb> ([<prep>] <tgt_square> | <tgt_piece> [<prep> <tgt_square>])"

Validating commands

Good ol' mediapipe

Of course, we use `mediapipe` for hand tracking.

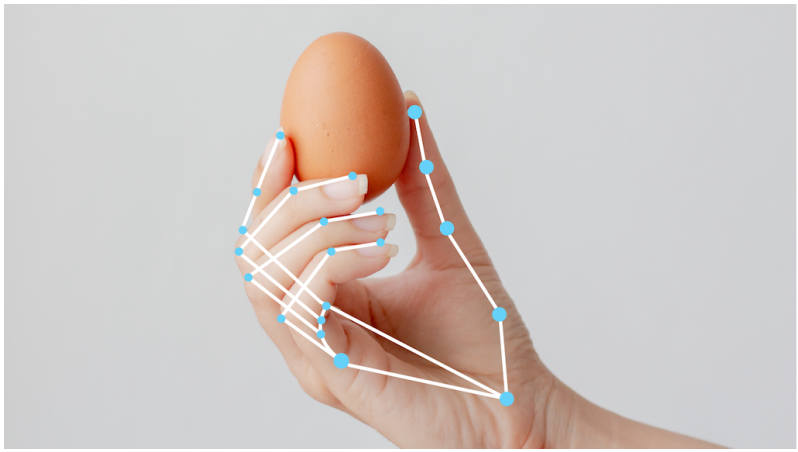


Figure: You know what this is.

1. We run it on a separate thread (`HandDetector`).
2. Process it for our needs.
3. Make it temporally coherent.

'Hand'made normalization 1

Get palm center:

$$p = \frac{\mathbf{L}_1}{2} + \frac{\mathbf{L}_6 + \mathbf{L}_{10} + \mathbf{L}_{14} + \mathbf{L}_{18}}{8}. \tag{1}$$

Its width

$$w = ||\mathbf{L}_6 - \mathbf{L}_{18}||_2, \tag{2}$$

and compute the relative position:

$$\bar{\mathbf{L}} = \frac{\mathbf{L} - p}{w}. \tag{3}$$

'Hand'made normalization 2

Find the palm's normal:

$$\mathbf{n}_{\text{palm}} = (1 - \mathbf{2}_{\text{left}}) \frac{(\mathbf{L}_6 - \mathbf{L}_1) \times (\mathbf{L}_{18} - \mathbf{L}_1)}{||(\mathbf{L}_6 - \mathbf{L}_1) \times (\mathbf{L}_{18} - \mathbf{L}_1)||_2}. \tag{4}$$

Then we get the pinky normal,

$$\mathbf{n}_{\text{pinky}} = (1 - \mathbf{2}_{\text{left}}) \frac{\mathbf{n}_{\text{palm}} \times (\mathbf{L}_{10} - \mathbf{L}_1)}{||\mathbf{n}_{\text{palm}} \times (\mathbf{L}_{10} - \mathbf{L}_1)||_2}. \tag{5}$$

And the fingers' normal,

$$\mathbf{n}_{\text{fingers}} = (1 - \mathbf{2}_{\text{left}}) \frac{\mathbf{n}_{\text{pinky}} \times \mathbf{n}_{\text{palm}}}{||\mathbf{n}_{\text{pinky}} \times \mathbf{n}_{\text{palm}}||_2}. \tag{6}$$

'Hand'made normalization 3

Finally,

$$\mathbf{L}^{\star} = \overline{\mathbf{L}}\left[\mathbf{n}_{\text{pinky}}, \mathbf{n}_{\text{fingers}}, \mathbf{n}_{\text{palm}}\right]^{T} \tag{7}$$

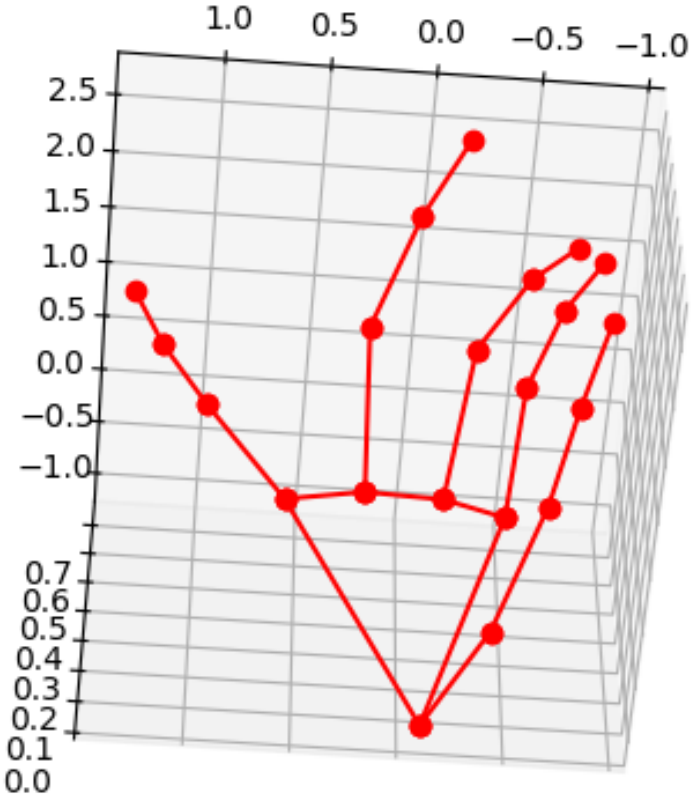


Figure: A right hand, normalized

Gesture recognition

At first, we wanted two gestures:

1. Tapping gesture, for "clicks". ☹
2. Grabbing gesture, to hold pieces. ☺

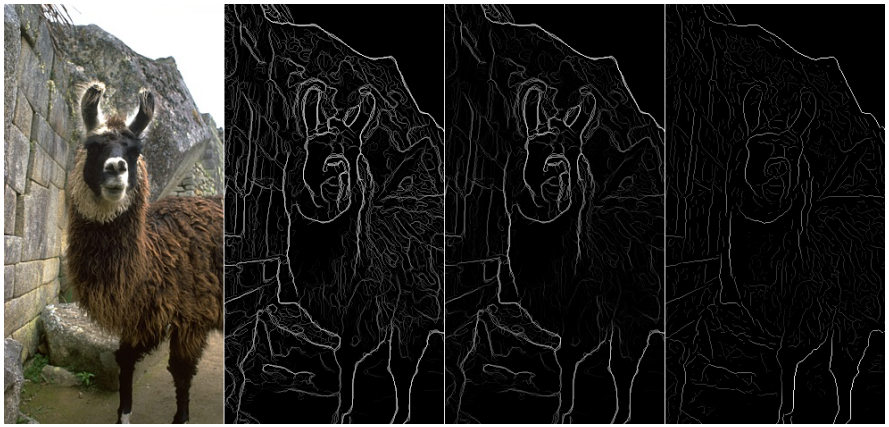
Figure: Frame from "A Guided Tour of Apple Vision Pro"

Recognizing grab

Very simple: hysteresis.

1: **input** `prev_click`
2: $m \leftarrow ||\mathbf{L}^{\star}_{\text{thumb}} - \mathbf{L}^{\star}_{\text{index}}||_2$
3: $d \leftarrow \frac{\mathbf{L}^{\star}_{\text{thumb}} \cdot \mathbf{L}^{\star}_{\text{index}}}{||\mathbf{L}^{\star}_{\text{thumb}}||_2 ||\mathbf{L}^{\star}_{\text{index}}||_2}$
4: **if** `prev_click` **then**
5: **return** $m < \alpha_\gamma \wedge d > \beta_\gamma$
6: **else**
7: **return** $m < \alpha \wedge d > \beta$
8: **end if**

Remember the Canny edge detector?



That uses hysteresis too!

Hand2Cursor mapping

The direct mapping is

$$r = \mathrm{clip}_{[m,M]}\left(\frac{p-m}{M-m}\right). \quad (8)$$

But we can't directly use r...
Let c be the internal cursor of `HandDetector`.

1. Noisy tracking: c moves at constant rate, either bilinear or linear interpolation, if r_{t-1} present.
2. Cursor moving when hand still: only update if distance is enough.
3. Random dropping of pieces: keep a list of most recent detections, if even one is a grab then output a grab.

Recording System I


```json
{
    "fen": "r2qk2r/ppp1bppp/2n1bn2/3pp3/4P3/3P1P2/PPP2KPP/RNB1QBNR",
    "player": [
        {
            "action_start": "2024-05-28 20:14:37.066157",
            "action_type": "mouse",
            "action_dist": 810.9136822128481,
            "down_button": 0,
            "up_button": 0,
            "moves": [],
            "action_end": "2024-05-28 20:14:42.439008"
        },
        {
            "action_start": "2024-05-28 20:14:42.439008",
            "action_type": "hand",
            "action_dist": 539.3721049295063,
            "down_button": 3,
            "up_button": 3,
            "moves": [],
            "action_end": "2024-05-28 20:15:06.577269"
        },
        {
            "action_start": "2024-05-28 20:15:06.577269",
            "action_type": "speech",
            "utterances": 7,
            "moves": [["pawn","a2","a3",null]],
            "action_end": "2024-05-28 20:15:31.592501"
        }
    ],
    "ai": [["pawn","e7","e5",null]]
}
```


Each recording is organized into two primary sections, namely **Player Actions** and **AI Moves**.

Recording System II

Each action performed contains information about :

- `Type` : Mouse, Hand or Voice
- `Start & End Time` : Time of Action Start and End
- `Moves` : (Source Piece, Source Square, Target Piece, Target Square, Promotion Piece)
- `Optional` : Utterances, Hand & Mouse Distance, Hand & Mouse Bottons Up and Down

Results, Total Time

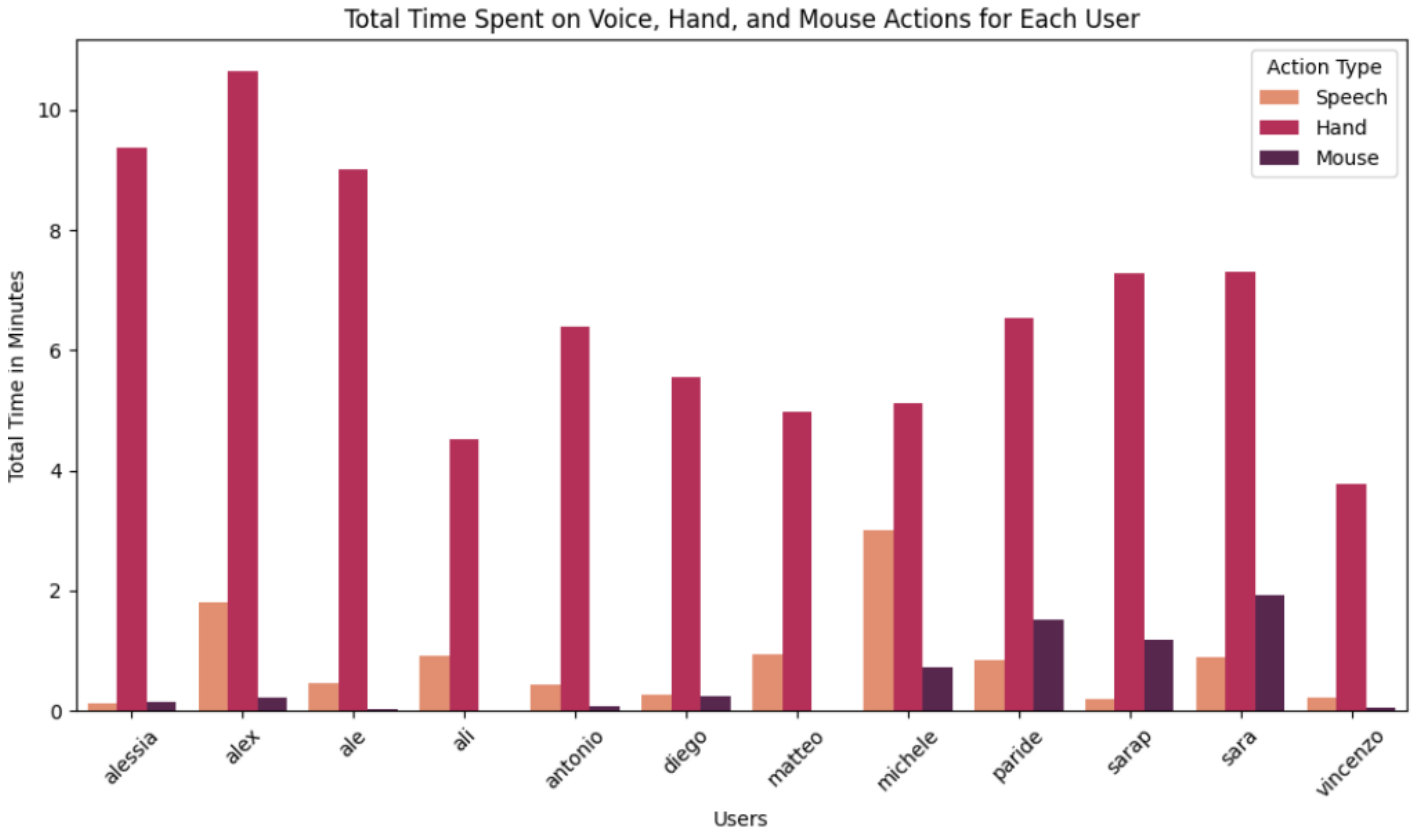


Figure: **Total Time** (seconds) on Voice, Hand and Mouse Actions per user

Results, APM

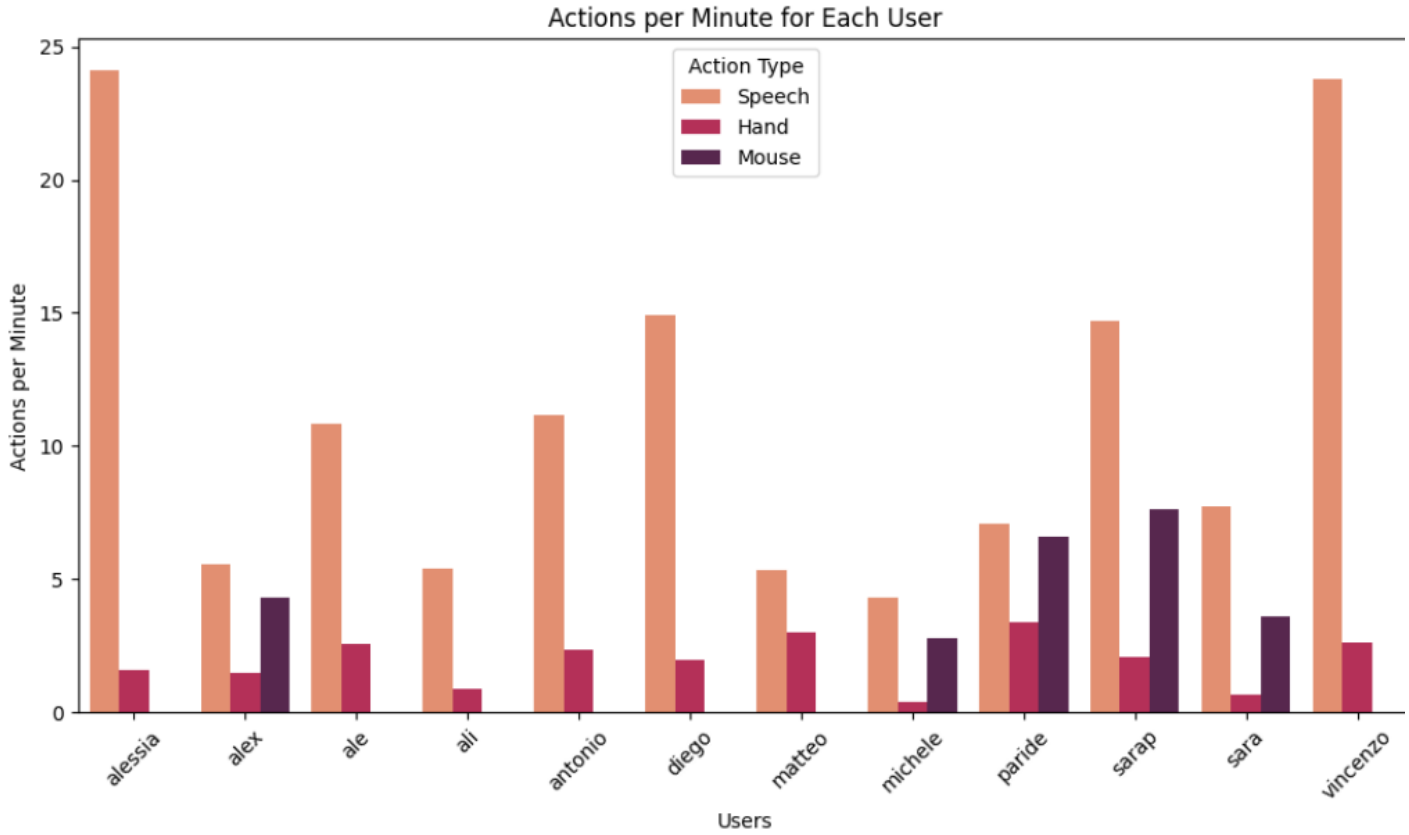


Figure: **Actions per Minute** per user

Results, Total Actions

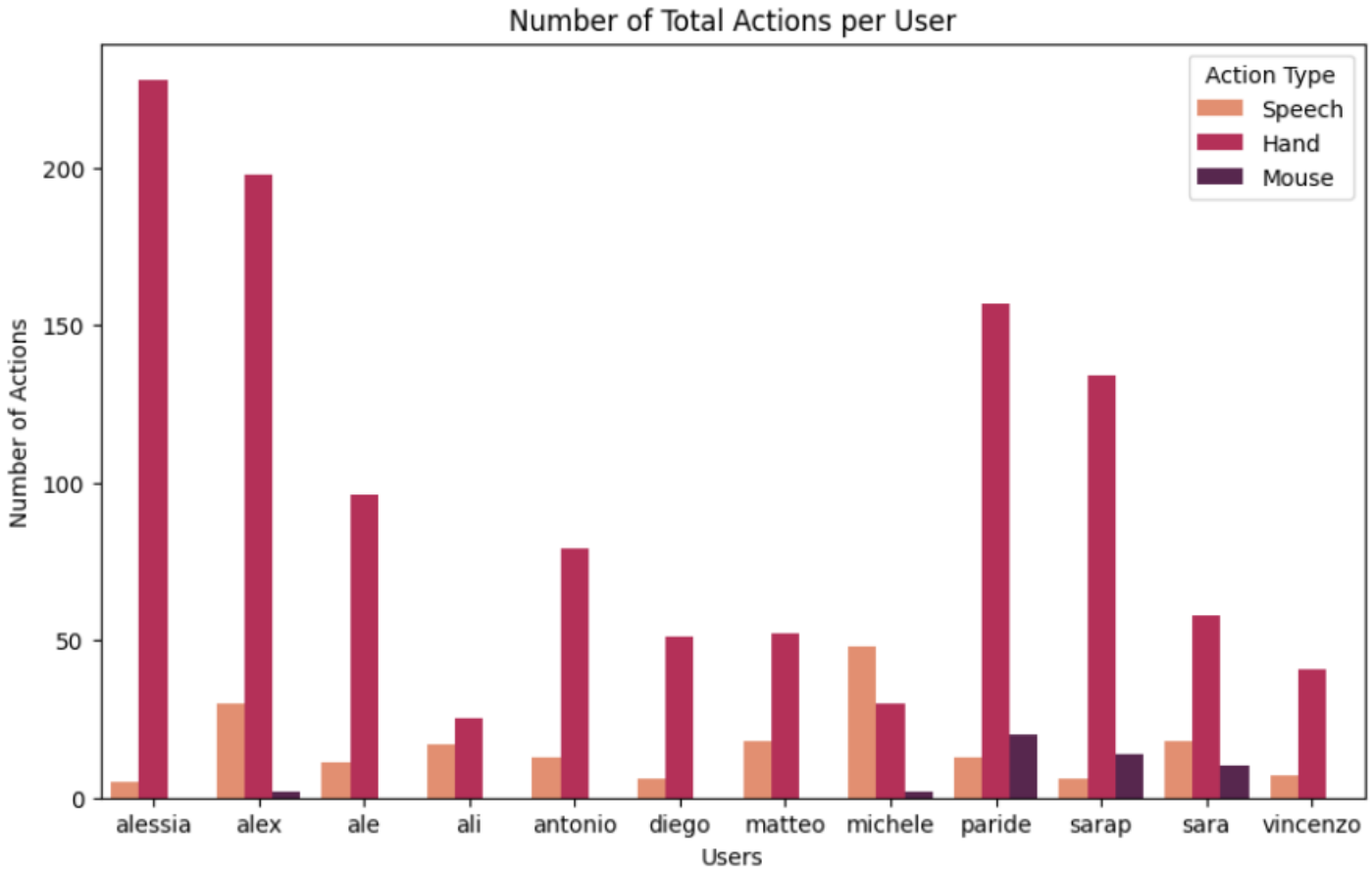


Figure: Number of **Total** Actions per user

Results, Legal Actions

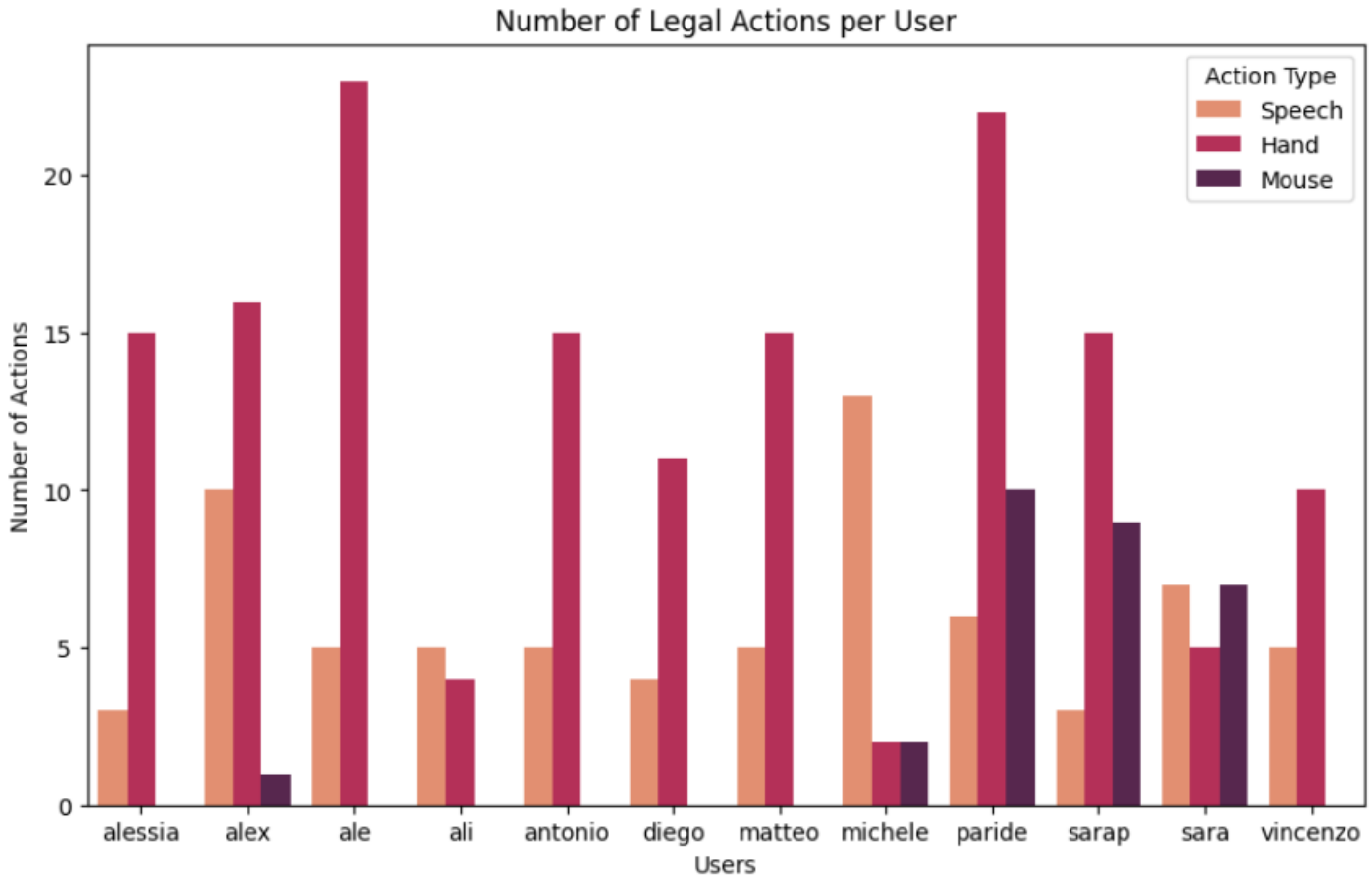


Figure: Number of **Legal** Actions per user

Results, Error rate hand

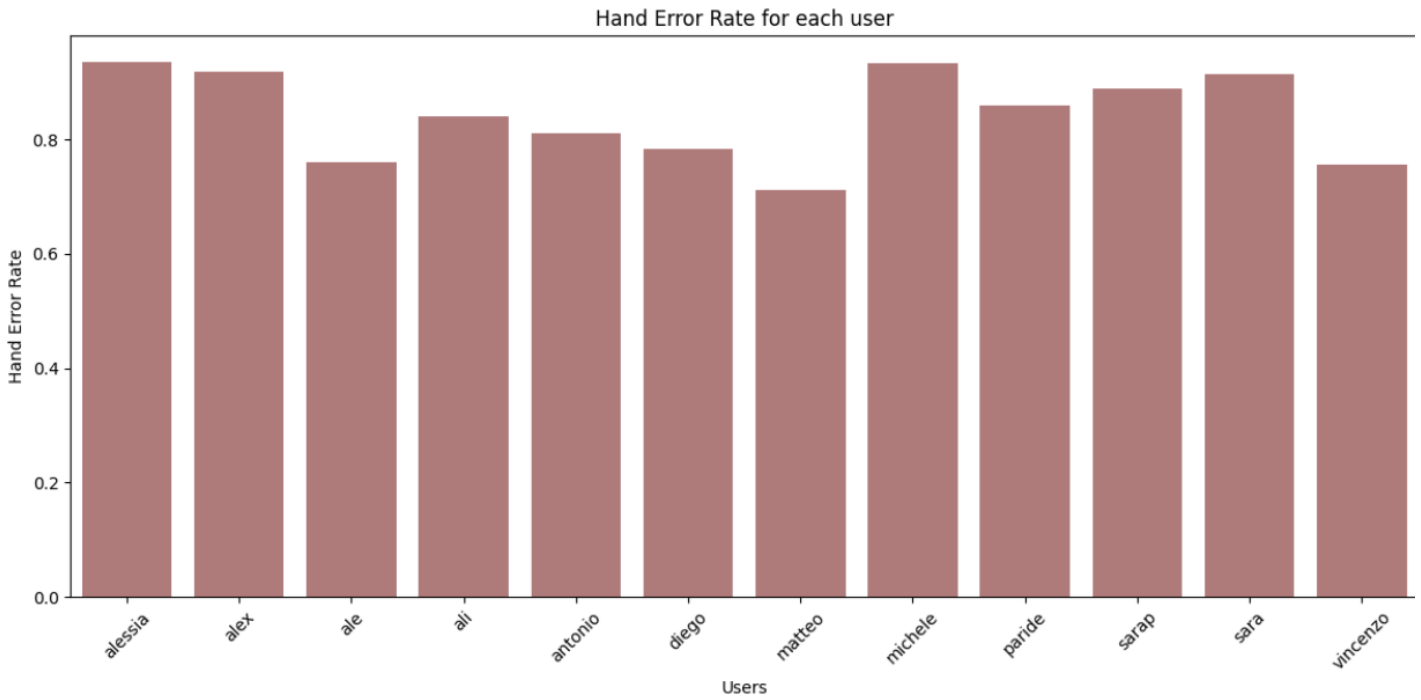


Figure: **Error rate** for **Hand** actions

Results, Error rate voice

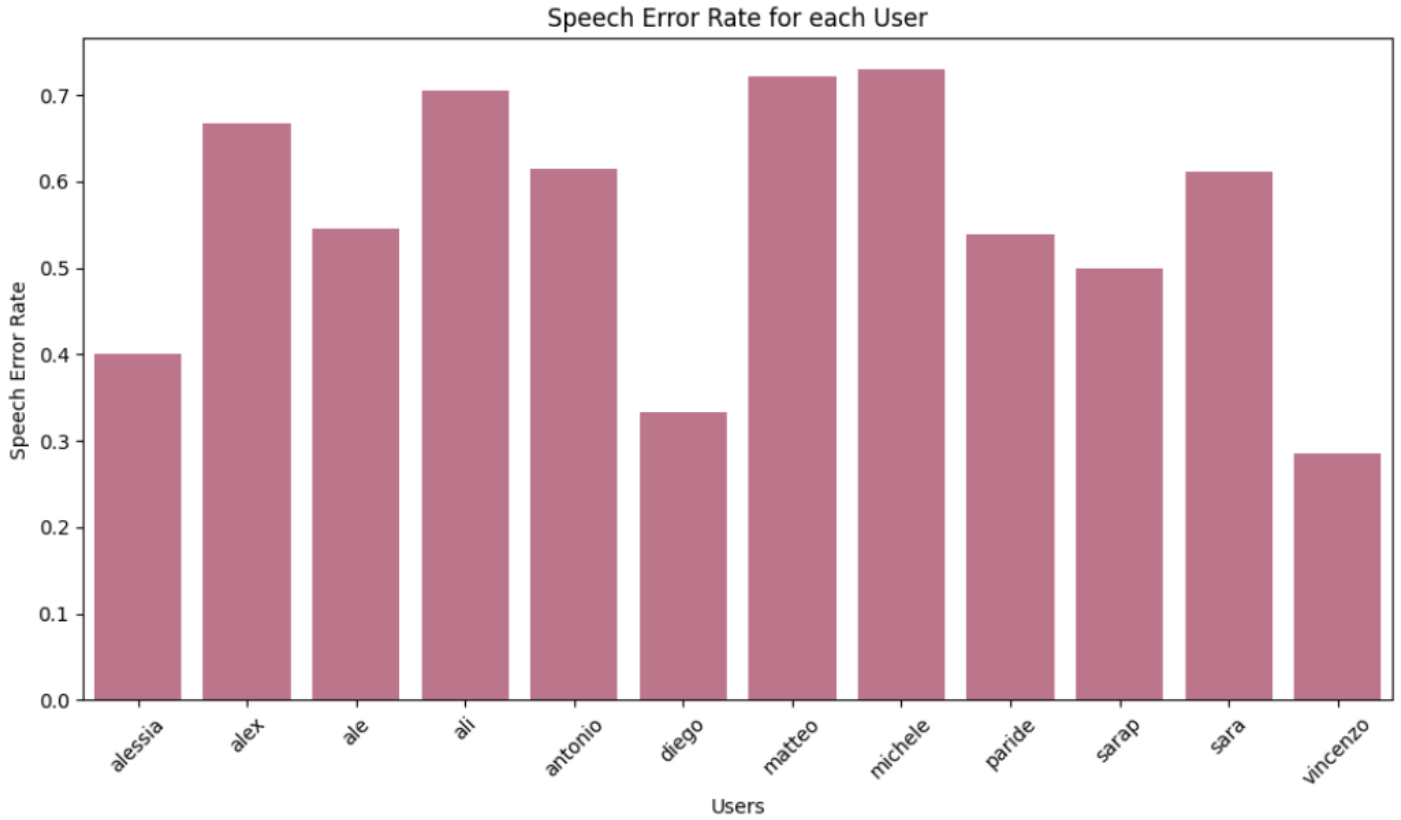


Figure: **Error rate** for **Voice** actions

Conclusions

Something something

Figure: Fin.