Generation of the optimal keyboard layout for programmers using RL methods

by Vsevolod Klyushev, Dmitry Beresnev, Ivan Inchin

Team

Vsevolod Klyushev

v.klyushev@innopolis.university

Dmitry Beresnev

d.beresnev@innopolis.university

Ivan Inchin

i.inchin@innopolis.university

Problem description

Goal

To find more optimal keyboard layout for code writing purposes

Problem

A unique keyboard layout might be generated by $\mathbf{5 \cdot 10^{207}}$ ways.
The modern computer performs about 10^8 simple operations over 1 second.
If we assume that the metric calculation for an arbitrary keyboard layout requires just one simple operation, then it would take more than $\mathbf{1.5 \cdot 10^{192}}$ **years** to check every possible combination.

Therefore we need more smart approach for solving this type of problem rather than random generation.

Dataset

We use Kaggle dataset with 120 000 different C/C++ programs.

Cleaning data steps

- Remove non-ASCII characters
- Concatenate programs into one line
- Substitute special symbols, for example spaces, tabs and new line characters
- Encode each character as integer
- Pad each data point with spaces at the end, to preserver line length of 400

Environment description

Since actual keyboard layout doesn't have rectangular shape we made following simplifications:

- We use 14 keys in first 4 rows and 11 in last one
- We use only 102 unique symbols for layout (check next slide)

Therefore, we use 134 cells (positions) as **state** representation for our environment

Actions are the following

- Swap two keys within low layout
- Swap two keys within high layout (available after pressing SHIFT)
- Swap two keys between low and high layouts, where first key is from low layout and second - from high layout

As a result, environment has **8911 actions** in total

QWERTY layout example

Environment description

Reward or score is calculated in the following way:

$$\lambda_{row}|x_1 - x_2|^2 + \lambda_{col} \cdot |y_1 - y_2|^2 + P$$

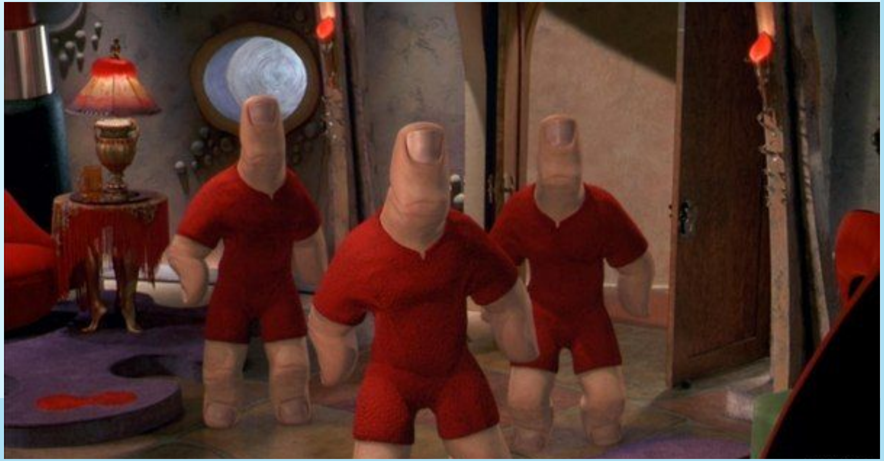
where

- x_1, y_1 – coordinates of current finger position
- x_2, y_2 – coordinates of target cell/key
- $\begin{cases} P = 1, \text{ if } |y_1 - y_2| > 3 \\ P = 0 \text{ otherwise} \end{cases}$
- λ_{row} – penalty multiplier for moving between rows
- λ_{col} – penalty multiplier for moving between columns

We take $\lambda_{row} = 1$ and $\lambda_{col} = 1.2$ because moving between rows is easier in terms of fingers displacement than moving between columns. We choose such P because moving further than 2 rows usually requires significant displacement of the wrist.

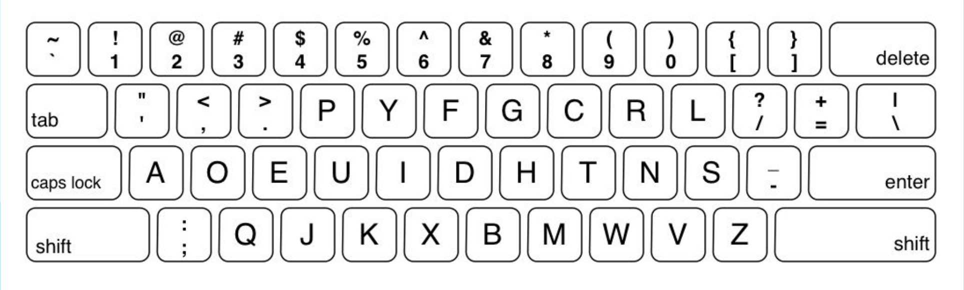
Finger class

- Tracks position of the single finger
- Returns to default position after several ticks (keys typed)
- Change position of the single finger
- Returns distance that finger has to overcome during movement process



KeyboardLayout class

- Contains lowercase and uppercase layouts as well as dictionaries with coordinates for each symbol
- Contains 10 fingers and operates with them
- Accumulates total distance that all fingers passed
- Has the ability to reach symbol from uppercase layout via combination `shift+key`
- Can swap different symbols in both lowercase and uppercase layout



Solution architecture

Policy NN

Action decision NN
Input size – 134
Output size – 8911

Softmax layer on output in order to decide which action to take on current state we performed sampling using the distribution based on Policy network output

State value NN

Neural network for state score estimation
Input size – 134
Output size – 1

MSE loss

Train process

Policy NN was trained on first 10 programs from our dataset. We initialized Policy NN with SGD optimizer with 1e-3 learning rate.

We used **QWERTY layout as starting state**. Then we conducted several episodes of training, in each episode we made **100 swaps maximum**.

Stopping criteria for each episode were either repetition of state within episode or exceeding the QWERTY score by more than 1000.

As a **reward** we use difference between current score and QWERTY score on our dataset.

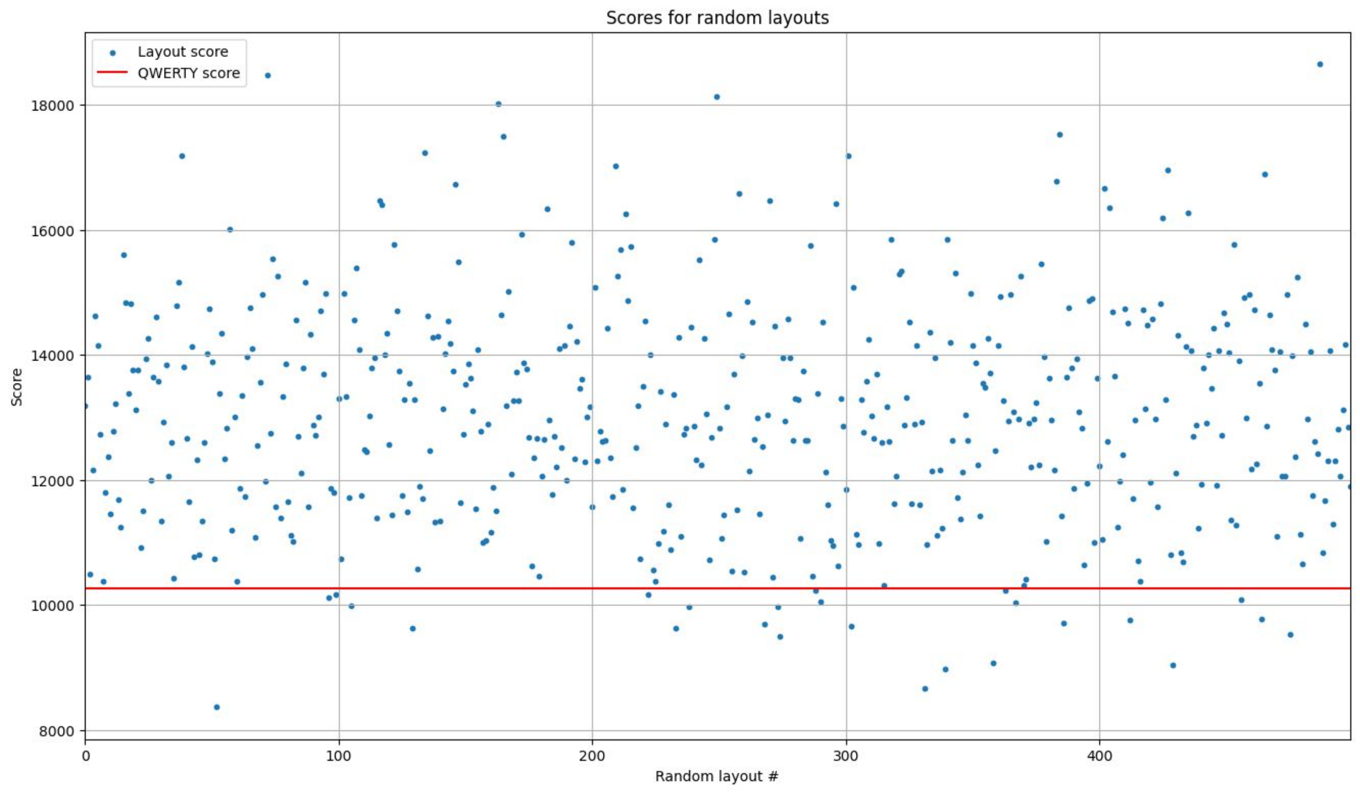
Loss was multiplication of reward and probability of selected action.

Baseline

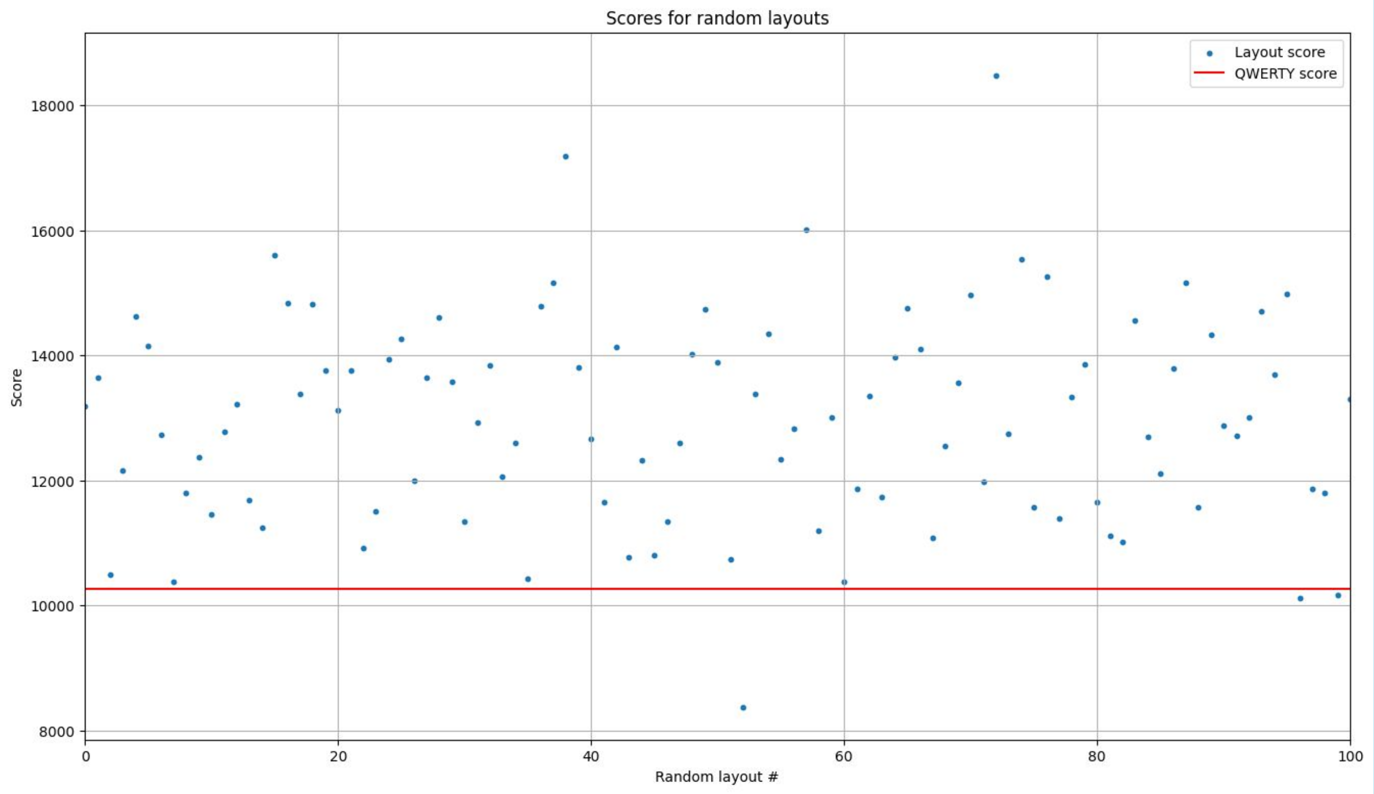
We generated 500 random layouts

Best one achieved score **8369**

Baseline


Scores for random layouts
Layout score
QWERTY score
Score
Random layout #
8000
10000
12000
14000
16000
18000
0
100
200
300
400

Baseline



Scores for random layouts
Layout score
QWERTY score
Score
Random layout #
18000
16000
14000
12000
10000
8000
0
20
40
60
80
100

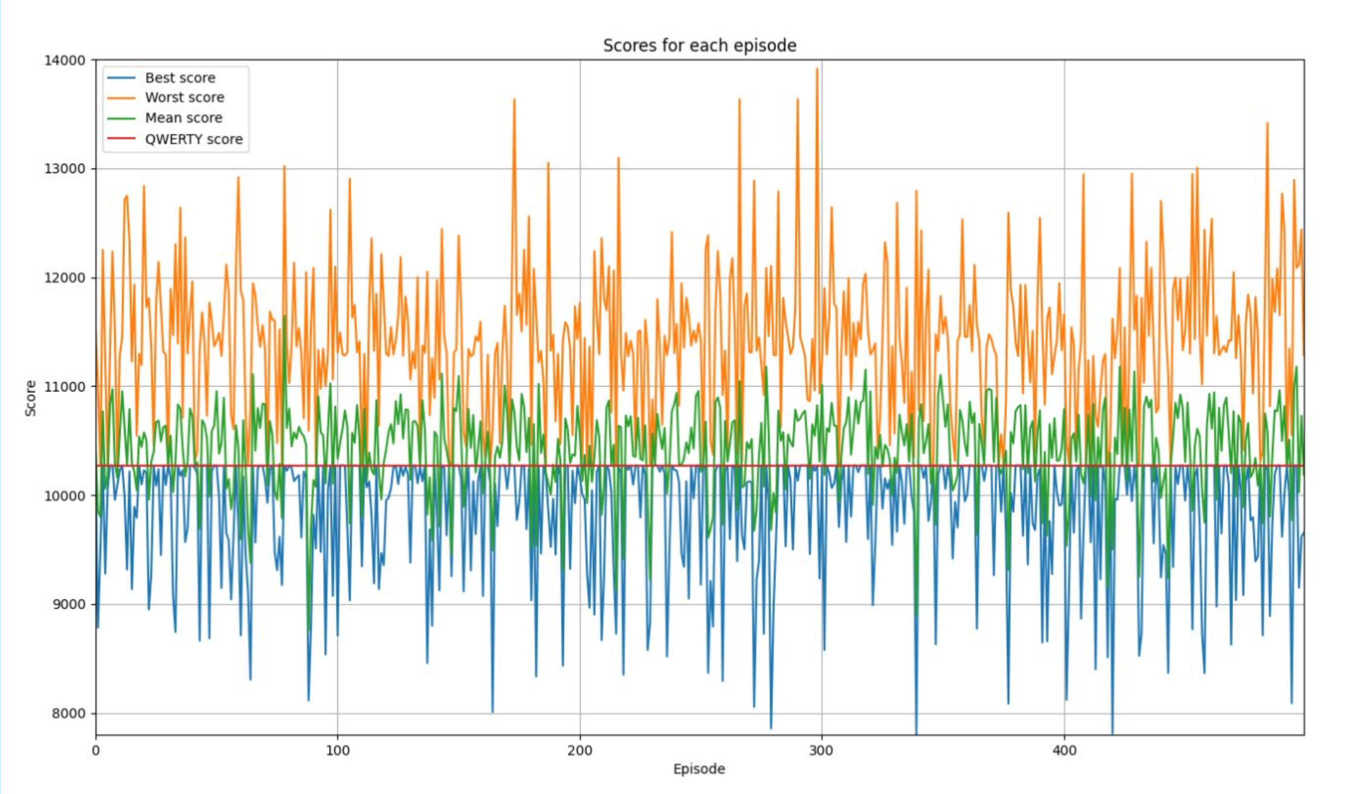
Results

QWERTY score on first 10 programs from our dataset was **10272**

Best found layout from Policy NN layout found after 500 episodes has score equal to **7800**

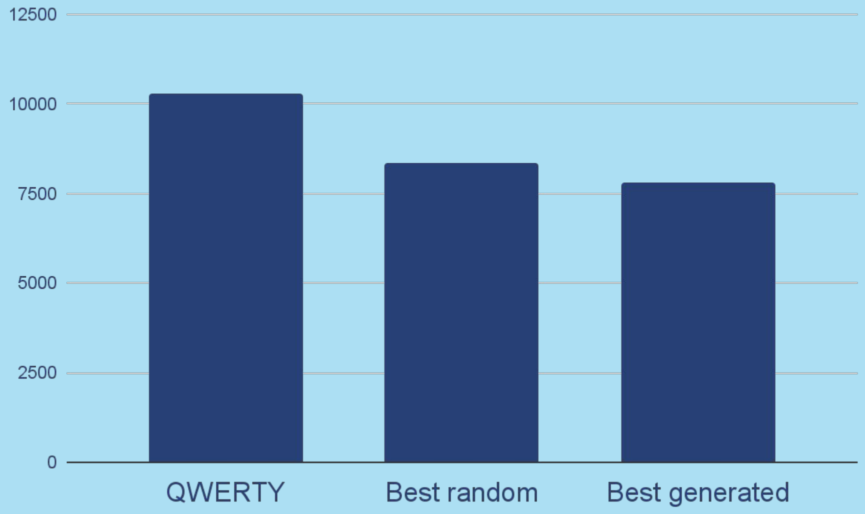
It is more than **24%** improvement
It took approximately 1 hour to train our Policy NN

Results



Scores for each episode
Best score
Worst score
Mean score
QWERTY score
Score
Episode
14000
13000
12000
11000
10000
9000
8000
0
100
200
300
400

Results analysis



24%

Improvement in comparison with QWERTY layout

Insights

- Both keyboards tends to place enter, space and shift near one of fingers positions
- For real usage both keyboards won’t be useful, since numbers are located in strange order
- For different datasets optimal keyboards would be also different

References

- A. Goldie and A. Mirhoseini, “Placement Optimization with Deep Reinforcement Learning,” arXiv.org, Mar. 18, 2020. https://arxiv.org/abs/2003.08445

- A. Mirhoseini et al., “Chip Placement with Deep Reinforcement Learning,” arXiv:2004.10746 [cs], Apr. 2020, Available: https://arxiv.org/abs/2004.10746

- Y. Matsuo et al., “Deep learning, reinforcement learning, and world models,” Neural Networks, Apr. 2022, doi: https://doi.org/10.1016/j.neunet.2022.03.037.

- Actor-Critic examples: https://github.com/chengxi600/RLStuff/tree/master/Actor-Critic

Thanks!

Do you have any questions?

v.klyushev@innopolis.university