Dynamic Taint Analysis for Nodejs Applications

Siddharth Subramaniyam
Texas A & M University
College Station, TX
subsid@tamu.edu

Jeff Huang
Texas A & M University
College Station, TX
jeffhuang@tamu.edu

ABSTRACT

In this work, we develop a babel-plugin for adding domain specific instrumentation for Nodejs applications, that will help prevent malicious server side attacks such as SQL-injection, URL interpretation. The tool takes a rule-config file for a specific application and generates instrumented source code that can prevent tainted information flow through the application, at a domain level. The rule-engine supports adding rules by constraining code paths, validating inputs in a regular expression style language.

INTRODUCTION

Javascript has always been the goto language for frontend development. But since the release of Node.js in 2009, it has been rapidly gaining popularity in backend development, especially web servers. Its cross-platform ability, dynamic typing and rapid development combined with the fact that developers can write both backend and frontend in the same language has made it grow exponentially. But this also means its becoming more vulnerable and prone to attacks.

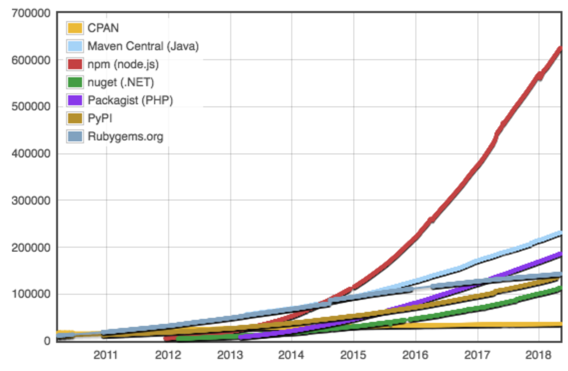
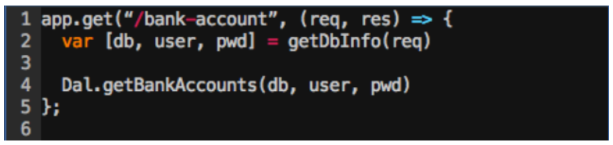


Figure 1. Rapid growth of Nodejs modules (https://modulecounts.com)

In this work, we develop a babel plugin that instruments javascript source based on domain specific rules. While we believe this can be used with any javascript backend, our initial version is geared towards web-servers. Since javascript is weakly typed, its hard to perform sound **static analysis**. On the other hand, performing **dynamic analysis** without any rules, will bloat the server code significantly, causing significant slowdown of the service. Our babel-plugin instruments javascript source code based on a rule-file, that a developer can create for each type of application. Our tool takes this rule file, and instruments certain code-paths in the source code *statically*, during the build phase.

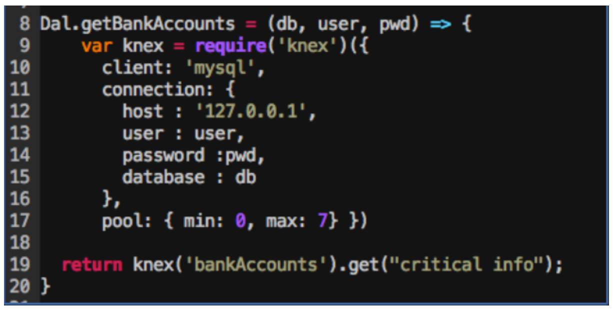


```
app.get(“/bank-account”, (req, res) => {
  var [db, user, pwd] = getDbInfo(req)

  Dal.getBankAccounts(db, user, pwd)
};

```

Figure 2. A simple API endpoint that returns some critical information



```
Dal.getBankAccounts = (db, user, pwd) => {
    var knex = require('knex')({
      client: 'mysql',
      connection: {
        host : '127.0.0.1',
        user : user,
        password :pwd,
        database : db
      },
      pool: { min: 0, max: 7} })

  return knex('bankAccounts').get("critical info");
}
```

Figure 3. A dataacess function that makes a database call

As an example, consider a simple webserver consisting of an *api endpoint* called */bank-account* (Fig.2), that makes a call to the database using a functions defined in the *Data Access Layer (DAL) called Dal.getBankAccounts* (Fig.3). Since this endpoint can return some critical information, it is useful to constrain the functions that can make calls to this endpoint. For example, if some other end-point is compromised, we can prevent that from making calls to this function. This can be done using a simple constraining rule as shown in Fig.4.

Our initial idea was to implement our tool using Jalangi2. Jalangi2 uses a techniques for shadowing various calls/variables in the code and perform checks on these shadow objects, this causes a 20-30x slowdown. Its hard to selectively instrument code using jalangi. Jalangi2 (the newer release by samsung) is also not actively maintained (Last commit at the time of writing this paper was April 2017, more than an year old). These reasons make it hard to integrate a plugin using Jalangi, into existing continuous deployment pipelines. Also, testing the instrumented code becomes hard. For our

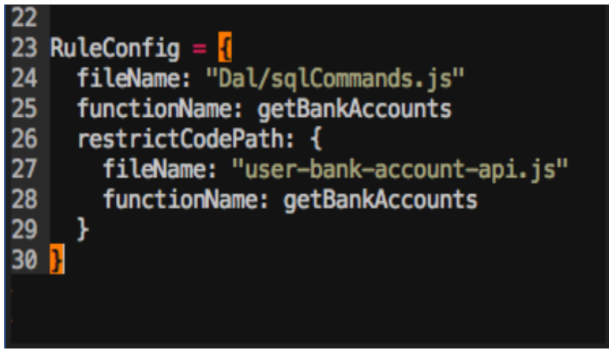

```
RuleConfig = {
  fileName: "Dal/sqlCommands.js"
  functionName: getBankAccounts
  restrictCodePath: {
    fileName: "user-bank-account-api.js"
    functionName: getBankAccounts
  }
}
```

Figure 4. A simple constraining rule that prevents malicious calls to the above mentioned DAL function

tool, we decided to implement it as a babel plugin. The reason we chose to implement the tool as a babel-plugin was the prevalence of babel in the ecosystem. **Babel** is a javascript compiler, that allows developers to use modern javascript capabilities, without worrying about runtime (Nodejs) or browser compatibilty issues. Babel has support for the latest version of JavaScript through syntax transformers. These transformenrs are written as plugins and allow developers to use new syntax, without waiting for browser support. Due to its popularity, it has great documentation and active support. Babel uses a javascript parser called *babylon* and makes it easy to modify the syntax tree for instrumentation.

RELATED WORK

The main idea of this work is the gamification of the learning task. Gamification is the process of adding game elements to a non-game scenario to make it more interactive and appealing. Gamification of learning techniques are quite popular and have been extensively studied in the past [1, 2, 3, 4]. [5] mentions about the history of the gamification process. [2] contrasts the difference between gamification and game based learning and also provides an analysis of gamification of serious games. [3] mentions relatively recent approaches followed in the gamification areas. Also, much research is being done in the gamification areas including [6, 7] indicate the significance of gamification for learning tasks. In [8], the authors propose teaching a new characters from a new language through a Sudoku gaming setup on a sketch based platform. All these works have motivated us to gamify the whole new character teaching process and engage the user through a gaming strategy. This also encouraged us to make the interface as user driven as possible. Some other works that investigate gamification in a sketch based domains are [9, 10].

As a part of the game design we had to develop an optimal algorithm that performs the task of multi-stroke sketch matching. For this we have gone through the literature for algorithms being implemented in this domain. One more thing we had to consider for choosing an optimal algorithm was its time complexity. As we were planning to develop a game that in future includes a lot of data when symbols from other languages will be considered, we had to choose an algorithm that had less time complexity. The $-family algorithms are well known for their simplicity and their ability to scale in real-time scenarios [11, 12, 13]. For these reasons, we have analyzed the $-family algorithms. While $1 and protractor algorithms mentioned in [12, 13] are good real-time algorithms, they can not handle multi-stroke sketches. On the other hand, $N and $N-Protractor algorithms mentioned in [11, 14] handle multi-stroke sketches but are computationally intense algorithms and hence may not be as scalable as required when the game considers multiple languages. With this in mind, the optimal algorithm we had obtained was the $P algorithm introduced in [15]. This is better than all other algorithms for the current task because it does not consider strokes as a gesture but considers them as a point cloud and hence when matching with a template it does not introduce complexity in matching each point of both the sketches.

INTERFACE

The user-interface of the project is browser based. The game is meant to be platform independent, so that it can be played on either a computer, tablet, phone or any pen enabled device. Web browsers, because of their ubiquity, make it an ideal choice for this game. The interface is built in javascript using Reactjs framework. The interface is designed to be responsive. i.e Adaptive to multiple screen sizes.

The application consists of 3 main components.

Config page

The config page is meant for configuration settings before playing the game. As shown in Fig.5, user can select the symbols to use for the next session of the game. Thus, a user can focus on mastering a few symbols, before proceeding further. Allowing a user to set their own target score gives them flexibility on how long they would like to play. Also, a user can set the minimum number of attempts for each symbol. This page is easily extendable to add more settings, such as speed/time control and selecting a different alphabet.

Game page

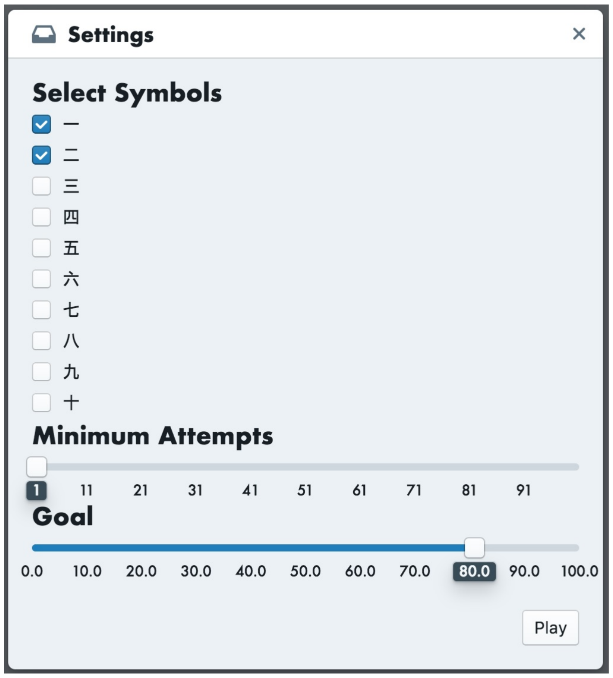
The game page (Fig.6) is where the game is played, it consists of a draw canvas, view canvas and info section. The view canvas is where symbols fall, that the user has to replicate in the draw canvas. Info section is used for displaying score and progress related information.

Done page

Done page is a simple end page, that acknowledges the user for successfully finishing the game. (Fig.7)This can later be extended to show user statistics such as comparison with previous attempts.

RECOGNITION

The recognition part of the game starts as the user ends drawing the template. We have used $P algorithm to match the user drawn sketch to the actual templates. $P recognizer represents a gesture as clouds of points. This alleviates the problem of complexity in comparison of multi-stroke gestures. All the gestures are represented as a set of points irrespective of the order and time at which they were drawn. This makes the recognizing part much simpler as the time factor now is not to be taken into consideration. This is similar to template matching in the field of image-processing where the


Settings

Select Symbols

- [✓] 一
- [✓] 二
- [ ] 三
- [ ] 四
- [ ] 五
- [ ] 六
- [ ] 七
- [ ] 八
- [ ] 九
- [ ] 十

Minimum Attempts: 1

Goal: 80.0

Figure 5. Configuration selection page

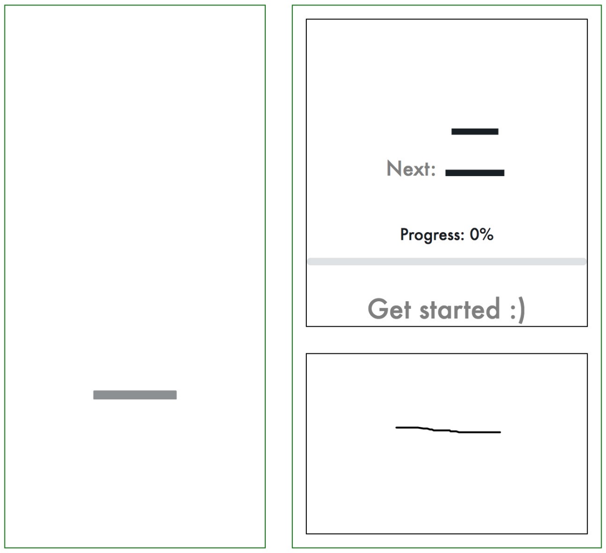
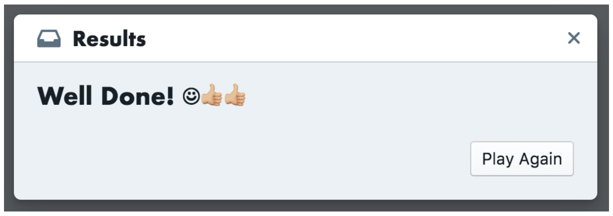


Figure 6. Game interface while sketching


Results

Well Done! ☺👍👍

Figure 7. Window after completing the task

the template is resized, and shifted to match to the given image. Similarly, the user-drawn sketch is resampled to same number of points, scaled and shifted to match to the templates. $P algorithm was chosen as we had to handle multiple strokes in a sketch as many numerals in Mandarin and other languages can not be drawn in one stroke. As mentioned in the introduction section, $1 algorithm was not used as it does not consider multi-stroke sketches. Also, $N algorithm needs to generate all possible permutations of the given multi-stroke sketch which causes higher memory usage and longer execution time. This was not ideal in the current scenario as we needed to implement it in real time for game designing. $P algorithm avoids the storage complexity of $N algorithm by representing gestures as "clouds of points" and thus ignoring behaviour in terms of stroke order and direction.

The input from the drawing canvas is stored as a .json file which is then parsed to get the list of points. These points are then pre-processed as mentioned in the following steps.

- **Resampling** The user drawn sketch is first resampled making sure that the number of points after resampling is same for every template and user-drawn sketch and is equal to a pre-defined value which in this work is chosen as 32. Standard resampling algorithm is used where the sampling distance is adjusted according to the required number of points after the operation.

- **Scaling** Points can be drawn on drawing canvas of different sizes. In order to compare two sets of points, the points clouds must be scaled to the same size and the corresponding points must be compared. The scaling function calculates x_{min}, x_{max}, y_{min} and y_{max} from the set of points. The new boundaries are given by x_{max}-x_{min} and y_{max} - y_{min}. Each point in the set of points is scaled to new boundaries computed as indicated in the following expression

$$p_{new} = (p.x - x_{min})/scale, (p.y - y_{min})/scale$$

- **Translate** This is done to translate the points to origin (0, 0). In case the sketch drawn by the user are referenced to some point other than origin, this function translates the points to new origin (0, 0).

The pseudo codes of the pre-processing steps are given below:-

After pre-processing, the user-drawn sketch is compared to all the templates which are pre-defined. This is done by a cloud-matching function which matches two clouds (points and template) by performing repeated alignments between their points (each new alignment starts with a different starting point index i). A parameter ε which lies between 0 and 1(usually kept to 0.5), controls the number of tested alignments. The cloud-matching function returns the minimum alignment cost.

For each point in the first cloud (C_i), we find the closest point from the second cloud which has not been matched yet. Once the point C_i is matched, we continue with C_{i+1} until all the points from C are matched (i = 1, 2, ... n - 1). This algorithm is run multiple times from different starting points and returned the minimum matching of all the runs. If C_k is the starting

Algorithm 1 Resample

```
1: procedure RESAMPLE(POINTS points, INT n )
2:     $I \leftarrow$ Path-Lengths$points/(n-1)$
3:     $D \leftarrow 0$
4:     $newpoints \leftarrow points_0$
5:     for each $p_i$ in points such that $i \geq 1$ do
6:         if $p_i.strokeId == p_{i-1}.strokeId$ then
7:             $d \leftarrow$ EUCLIDEAN-DISTANCE $(p_{i-1}, p_i)$
8:             if $D + d \geq I$ then
9:                 $q.x \leftarrow p_{i-1}.x + ((I-D)/d).(p_i.x - p_{i-1}.x)$
10:                $q.y \leftarrow p_{i-1}.y + ((I-D)/d).(p_i.y - p_{i-1}.y)$
11:                APPEND(newPoints,q)
12:                INSERT(points,i,q)
13:                $D \leftarrow 0$
14:            else$D \leftarrow D+d$
15:    return newPoints
```

Algorithm 2 Scale

```
1: procedure SCALE(POINTS POINTS)
2:     $x_{min} \leftarrow \infty, x_{max} \leftarrow 0, y_{min} \leftarrow \infty, y_{max} \leftarrow 0$
3:     for each $p$ in points do
4:         $x_{min} \leftarrow MIN(x_{min}, p.x)$
5:         $y_{min} \leftarrow MIN(y_{min}, p.y)$
6:         $x_{max} \leftarrow MAX(x_{max}, p.x)$
7:         $y_{max} \leftarrow MAX(x_{max}, p.y)$
8:     for each p in points do
9:         $p \leftarrow (p.x - x_{min})/scale), (p.y - y_{min})/scale)$
```

Algorithm 3 Translate-To-Origin

```
1: procedure TRANSLATE-TO-ORIGIN
2:     $c \leftarrow (0,0)$
3:     for each $p$ in points do
4:         $c \leftarrow (c.x + p.x, c.y + p.y)$
5:     $c \leftarrow (c.x/n, c.y/n)$
6:     for each $p$ in points do
7:         $p \leftarrow$ (p.x = c.x, p.y = c.y)
```

point then:

$$\sum w_i.||C_i - T_j|| = \sum_{i=k}^{n} ||C_i - T_j|| + \sum_{i=1}^{k-1} ||C_i - T_j||$$

where i goes circularly through all the points in C. Weights w_i encode the confidence in each pair (C_i, T_j) computed during the greedy run. The first match is weighted with score 1 as first point has all the data in order to make a decision for its match. As the algorithm progresses to subsequent points, few options remain for the rest of the points from the first cloud, when searching their closest pair into the second. Hence as the algorithm progresses, the confidence score decreases. A liner weighting scheme is used in which

$$w_i = (i-1)/n.$$

The result of the greedy function depends on the direction of the matching(either from C to T or vice-versa), hence we take the minimum of both the distances.

$$min(Greedy(C,T), Greedy(T,C))$$

As for sketchTutor game, we took 5 templates of each numeral from 4 participants, thus a total of 20 templates per numeral and total of 180 templates. Whenever the user is asked to draw or choose a character, what user intends to draw is already known to us. So the drawn sketch is compared to 20 templates of the character and if the similarity score is more than 0.6, we declare that as success.

RESULTS

We have let five users play this game and asked some subjective questions in order to obtain insights into the design evaluation of the interface. The key points suggested by some of the users were as follows. Two users felt that the interface was more user driven and suggested that the interface should be initialized better to have a good starting point rather than letting the user make all the decisions as the user does not have much information on how to choose a particular strategy initially. This led to users spending time before arriving at a particular strategy for learning or not following any particular strategy at all. This provoked us to work on a set of ideal initial point as the game gets complicated after addition of new languages and new symbols.

Apart from this, one key finding that was observed during these tests was that the user does not have a baseline measure which could guide him/her if the game were to be tested on multiple days which is similar to a real game scenario as it will be played many times. This problem can be solved by introducing a login page and collecting logs of each user so as to adapt the game whenever the user starts over the game in a later session.

Later, to identify which symbols of Mandarin were easily recognizable and which are not easily recognizable, we have tested the game on two users where each user is asked to sketch one symbol at a time for three times and the average scores of each symbol were noted down. Before starting this evaluation, the users were given five minutes each to explore the interface as it was observed that some user required some initial acquaintance with the game to avoid initial errors. So, to avoid any bias in order of choosing the symbols, we have let them explore it for five minutes. The results of accuracy obtained on each symbol are mentioned in Table.1.

Table 1. Average scores obtained from two users for each symbol

Symbol	Accuracy (in percent)
1	93
2	62
3	64
4	48
5	35
6	38
7	56
8	44
9	43

DISCUSSION

In sight of the above results from Table.1, the following insights were drawn into the interface. It has been observed that the accuracy was heavily dependent on the number of training samples. As we use two training examples for each symbol, if the sketch by the user does not match one of the particular templates, a low score is detected. Also, one drawback that was observed was that of choosing the beautified unicode symbols for displaying to the user in the game was not ideal as the training templates and the unicode symbols varied for certain symbols such as 4 and 8. Also, it has been observed that more training data is required for keeping up with the requirement of capturing the variabilities under the current approach if it were to be scaled up to include other languages and symbols. One set of training samples being used for this study can be seen in fig.8. One good strategy to alleviate the above issue of high dependency would be to choose the score by matching all the training templates with the user sketch and then choosing the maximum of these scores. Even in this scenario, care must be taken to carefully design the training set to capture most of the variability.

It was also observed that single stroke numerals performed significantly higher than multi-stroke numerals which can be explained intuitively as follows. The user does not scale two different strokes very well and hence the combination of strokes while scaling using $P algorithm gives very poor result. One way of tackling this issue can be to consider each stroke separately but this involves matching at strokes level and then matching using some simple algorithm. This work would be quite interesting.

FUTURE WORK

From our current development, we have identified a few areas that can assist the user better in the teaching process. Before sketching, we would like the user to be able to practice the symbols in a free to draw canvas, which will give accuracy values for the user's drawing. This will allow the user to get better, before playing the game. We would also like to add more levels to the game, such that it supports more complex learning methods. For example, a user can sketch by looking at symbols in a different language, forcing them to map it to the required symbol and then draw. This would enable more improvement in recall.

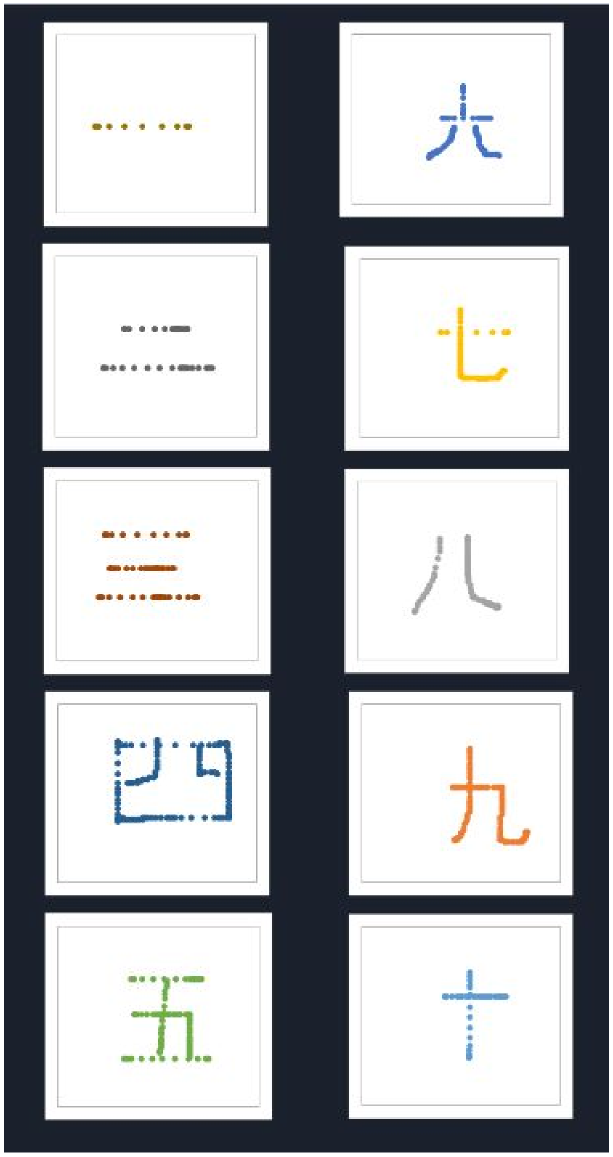


Figure 8. One set of training samples used in the current game

As it has been identified that the scoring is heavily dependent on the training samples as the training samples were very low, it would be worthwhile to consider more training examples for capturing all the variabilities of a symbol.

As mentioned in discussion section above, it would be an interesting study to compare both the $P implementation used in this work and the two stage recognition which matches strokes at first level and then points on the strokes matched. We would also like to improve our scoring metric, by including ordering of strokes. This will allow us to support more complex learning patterns.

CONCLUSION

In this work, we have developed a game for teaching symbols by playing. The current application supports Mandarin numerals from 1 to 10. The motivation for gamifying the language has been mentioned and the two main parts of the system are mentioned briefly describing data handling and user interface. The application is aimed at any browser enabled device, from a computer to a mobile phone.

The preferred algorithm for comparing sketches, in our implementation is $P (As opposed to other $Family algorithms) because of its speed, scale and rotational invariance. It also makes it easy to add more point-cloud templates, for better results. The algorithm runs realtime as a user sketches the given symbol and outputs a score, that is used for measuring the user's accuracy.

We aim to extend the application to support more symbols and increase the robustness of the recognition stage. Furthermore, we'd like to add more levels and other ui-features, as mentioned in future works, to make the experience more pleasant and fun.

REFERENCES

1. I. Caponetto, J. Earp, and M. Ott, "Gamification and education: A literature review," in *European Conference on Games Based Learning*, vol. 1, p. 50, Academic Conferences International Limited, 2014.
2. D. N. Karagiorgas and S. Niemann, "Gamification and game-based learning," *Journal of Educational Technology Systems*, vol. 45, no. 4, pp. 499–519, 2017.
3. P. Buckley and E. Doyle, "Gamification and student motivation," *Interactive Learning Environments*, vol. 24, no. 6, pp. 1162–1175, 2016.
4. T. J. Brigham, "An introduction to gamification: Adding game elements for engagement," *Medical Reference Services Quarterly*, vol. 34, no. 4, pp. 471–480, 2015. PMID: 26496401.
5. S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining "gamification"," in *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, (New York, NY, USA), pp. 9–15, ACM, 2011.
6. S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI'11 extended abstracts on human factors in computing systems*, pp. 2425–2428, ACM, 2011.
7. J. McGonigal, *Reality is broken: Why games make us better and how they can change the world*. Penguin, 2011.
8. C. D. Monteiro, M. Narayanan, S. Polsley, and T. Hammond, "A multilingual sketch-based sudoku game with real-time recognition," in *Frontiers in Pen and Touch*, pp. 187–196, Springer, 2017.
9. B. Paulson, B. Eoff, A. Wolin, J. Johnston, and T. Hammond, "Sketch-based educational games: "drawing" kids away from traditional interfaces," in *Proceedings of the 7th International Conference on Interaction Design and Children*, IDC '08, (New York, NY, USA), pp. 133–136, ACM, 2008.
10. G. Johnson and E. Y.-L. Do, "Games for sketch data collection," in *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, (New York, NY, USA), pp. 117–123, ACM, 2009.
11. L. Anthony and J. O. Wobbrock, "A lightweight multistroke recognizer for user interface prototypes," in *Proceedings of Graphics Interface 2010*, GI '10, (Toronto, Ont., Canada, Canada), pp. 245–252, Canadian Information Processing Society, 2010.
12. Y. Li, "Protractor: a fast and accurate gesture recognizer," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2169–2172, ACM, 2010.
13. J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes," in *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pp. 159–168, ACM, 2007.
14. L. Anthony and J. O. Wobbrock, "$n-protractor: A fast and accurate multistroke recognizer," in *Proceedings of Graphics Interface 2012*, pp. 117–120, Canadian Information Processing Society, 2012.
15. R.-D. Vatavu, L. Anthony, and J. O. Wobbrock, "Gestures as point clouds: a $p recognizer for user interface prototypes," in *Proceedings of the 14th ACM international conference on Multimodal interaction*, pp. 273–280, ACM, 2012.