The Minutovka – a Word Typing Web Game for Obtaining Typos to Create an Error Corpus

Štefan Toth, Michal Ďuračík, Patrik Hrkút, Matej Meško
University of Žilina
Žilina, Slovakia
{stefan.toth, michal.duracik, patrik.hrkut, matej.mesko}@fri.uniza.sk

***Abstract*— The paper describes the web application of a simple typing game, which we designed and implemented mainly to obtain typos and create an error corpus. We created the game for the Slovak language, as we lacked such data. However, it could be used for languages other than Slovak. We called it Minutovka (in English "One minute game") because we set the maximum duration of the game time to one minute. During this time, a player must rewrite the words from the randomly displayed sentences as quickly as possible and with as few errors as possible. On the one hand, the player is motivated to get the highest score and beat other players in the daily, weekly, or overall rankings, while improving his typing on the keyboard. On the other hand, our motivation was to obtain data on how users write and especially what typos and errors they make during the transcription of the text. In this paper, we describe this game from the design to the implementation, including issues that have occurred during the operation of the game. We also designed fraud protection and implemented it with the proposed hacker score. At the end, we evaluate and visualize all the obtained data, which we store in the document-based database MongoDB.**

I. Introduction

People make various mistakes when typing text on a computer, which is mainly due to inattention, fast typing on a keyboard or even ignorance of certain spelling rules in the given language. Currently, most common text editors include a spell checker that can help the user find and correct certain errors. For the world's most widely used languages, such as English, these editors and other text-correction tools are very well-developed. Unfortunately, for the Slovak language spoken by 7.23 million people in the world [1], spell checking is not as perfect as for the English language. The reason is the slight complexity of Slovak grammar, many exceptions to the rules of spelling and a little attention from scientists and programmers. Therefore, currently editors such as Microsoft Word or OpenOffice / LibreOffice still contain only a simple spelling checker based on a dictionary in Slovak.

While dealing with problems in the field of natural language processing, we found that it would be appropriate to have an error corpus. Thanks to it, we could generate various real user errors for input text data for training machine learning algorithms and artificial intelligence. For example, in the case of a text correction problem, it would be possible to train the system with such data to suggest corrections of erroneous words. In the case of other tasks, it would be possible to train the system to ignore such erroneous data and to be able to process it correctly.

Since we lacked such typos data in the Slovak language, we decided to obtain them in our own way by creating a simple typing game based on web technologies, which we will focus on in this article. We called it by the Slovak abbreviated name Minutovka, which means one minute game. In this following chapters, we describe it from design to implementation, along with the data we obtained using it.

II. Current state and related work

During analysis of the current state, we found that the idea of creating a game to obtain typos was dealt with by several authors, but neither publication has been focused on the Slovak language. Rodrigues and Rytting published their article describing the typing race game as a method to create spelling error corpora [2]. The game was played by 251 participating players. Each of the players had to retype one of 100 words which have been shown sequentially. Overall, authors analyzed 15,300 words from 153 native English speaker participants. 4.65% of words contained mistakes.

Another word typing games were created by Tachibana and Komachi [3]. They developed two word-typing games for analysis of English spelling mistakes. The first game (common word-typing game) was played by 7 and the second game (correctable word-typing game) by 19 students of computer science. Authors collected and analyzed 26 192 words.

Other games have been created for fun or learning, such as typing game for Tibetan words by Ga and Jun [4], who implemented a learning tool for Tibetan learners as well as a teaching tool for teachers. Also, for Taiwanese students learning English it was proposed game-based learning framework for typing games by Wang et al. [5]. Henze et al. [6] developed a typing game for observational and experimental investigation of typing behavior using virtual keyboards on smartphones. Another game for mobile devices equipped with touch screens is TypeJump video game by Costagliola et al. [7], which helps users to learn the KeyScretch text entry method.

There are several typing games on the Internet, especially for people to improve their keyboard typing or for having fun. The first example of the typing game is the Typing of the Dead, which is a 3D videogame produced by Sega [8]. A new version of this game called the Typing of the Dead: Overkill game is currently available in [9] and [10] released in 2013. The main goal of the game is to type correct words to kill zombie hordes. Another game is the TypeRacer [11], which is a simple multiplayer typing game released in 2008. Other example of the modern game is ZType [12], which is a galactic keyboarding adventure game. More typing free games can be found on the websites [13], [14], [15] and [16]. For example, the website [13]

contains beautiful simple 2D HTML games such as the Keyboard Jump, Keyboard Ninja, Zombie Defender, and many others.

III. IDEA AND DESIGN OF THE GAME

We tried to design the web application of the game as simply as possible, as can be seen in Fig. 1. We have placed brief information about the game and the rules on the main page. Immediately below it is the dominant part containing the text composed of randomly generated sentences, which must be sequentially rewritten word by word into the input text field within one minute. The time starts to count down as soon as the player presses the first letter on the keyboard. If the player does not like the generated sentences or starts to make a lot of mistakes, he has the option to reset the game by pressing the Try again button, which will generate new sentences and set the timer to the original 60 seconds.

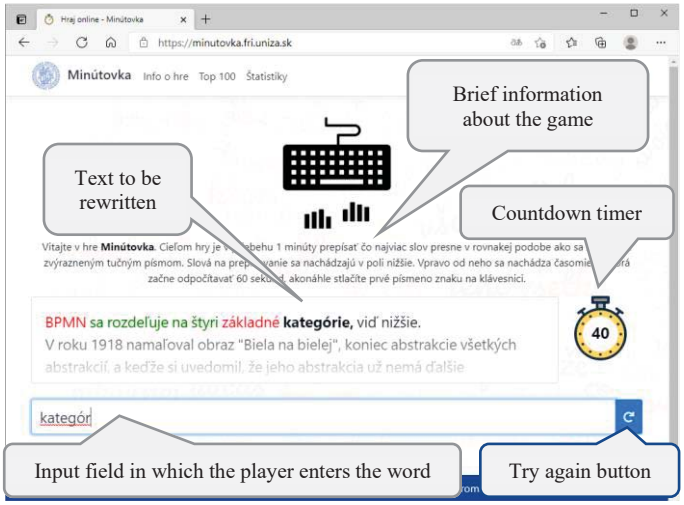

Fig. 1. The main web page of the Minutovka game

During the game, the current word is highlighted in bold, and if it is rewritten correctly by the player, it appears in green color. If it was rewritten incorrectly, it is displayed in red. To correct a currently written word, the player can only overwrite it without pressing the Space bar or the Enter key. If he presses these keys, the typed word in the input field is automatically recorded, and he is moved to the next word.

To make the game not too long and boring, we chose a time of one minute (60 seconds) during which it is necessary to rewrite the highlighted text exactly as it is appeared, i.e., in uppercase or lowercase, accented and punctuated. The goal for the player is to get the highest score – the more correct word he correctly rewrites, the higher score he gets.

At the end of the game, the player receives information about the overall score, the number of correctly and incorrectly transcribed words, and he can see, where he made mistakes. Also, he has the option to save his score to the Top 100 Players list. He needs only to enter a nickname and press the Save button. Besides, we offer him the opportunity to fill in a very brief questionnaire.

To support competition among players, we also implement a ranking of the top 100 players by the day, week, and overall (Fig. 2). In addition to the global scoreboard, we also provide the option to play a game in a group that displays the score in a separate table. We assume that this could stimulate competition. This prevents players to get demotivated if they do not score in the top 100 players. He can invite his teammates and compare himself with them by playing the game in a group. For this purpose, we create an option to join a group game with a unique name. If any other player joins this group, his score will be displayed in this local group ranking in addition to the global score ranking.

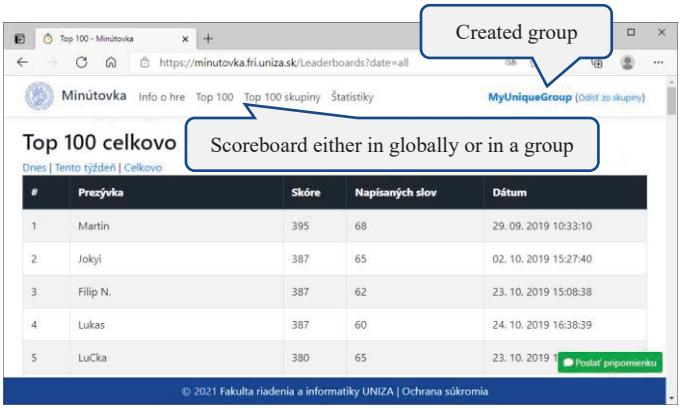

Fig. 2. Global scoreboard (rank, nickname, scores, transcribed word count, date and time)

The created game is publicly available via the URL address https://minutovka.fri.uniza.sk. We let the game be played by our students at the faculty and by many people as part of promotional events at our university. Thanks to this, we gradually obtained the data that we describe at the end of this article.

IV. IMPLEMENTATION

The game was implemented as a web application in C # using the ASP.NET Core framework. The application consists of the following parts:

- Database
- Application server
- Front-end module

The individual parts and their interconnection will be described in the following chapters. A docker is used for application deployment.

A. Database layer

We considered various types of databases, from the most used SQL [17] to NoSQL [18] databases. Finally, we decided to use the NoSQL document-based database MongoDB, which we use as a database server. Among its main advantages, compared to commonly used relational databases in web applications, is the simple possibility of storing structured documents.

The application itself does not need to store many entities for its operation. The application uses the following document collections:

- *GameResult*
- *Score*
- *GameToken*
- *CampaignVisitor*

The *CampaignVisitor* entity (Fig. 3) is used to track visitors based on campaigns. The application allows us to create a special URL for sharing, e.g., on social networks. After visiting the site via this URL, a unique *visitor identifier* is generated, which is then stored in cookies. This identifier is then stored together with the results. The structure of this entity is as follows:

KEY	COUNT	%
analyzed documents	207	100.0
_id -> *binData*	207	100.0
Campaign -> *string*	207	100.0
PersonIdentification -> *string*	207	100.0
Visit -> *date*	207	100.0

Fig. 3. The structure of the CampaignVisitor collection

The *GameToken* entity (Fig. 4) is used to uniquely identify the game. When text for the game is created, it is saved together with the timestamp of the game in the database. This token is only intended to protect against gaming cheating, which will be described in the following chapters. After the end of the game, it is removed from the database. The structure of the entity is shown in the Fig. 4:

KEY	COUNT	%
analyzed documents	10529	100.0
_id -> *binData*	10530	100.0
Created -> *date*	10530	100.0
IPAddress -> *string*	10530	100.0
PersonIdentification -> *string*	10530	100.0
Words -> *array*	3681	35.0
└ *[array item]* -> *string*	73620	

Fig. 4. The structure of the GameToken collection

These two entities are only auxiliary, but the most important are the *Score* (Fig. 5) and *GameResult* (Fig. 6) entities. In fact, these entities could be merged into one, but since we also needed to collect data from failed attempts, the *GameResult* entity contains the experiment data used to evaluate the typos. The *Score* entity is created only after the end of the experiment. Its structure is shown in the Fig. 5. As we can see, the entity *Score* contains the reference to *GameResult* entity (GameId), which represents the document ID from the *GameResult* collection. It can also contain a player - if he fills in at least the nickname. If it fills in the contact email address or whether it uses typewriting, it is also stored in the Player object. It also contains calculated statistics, such as the number of words per minute, respectively the value of the score, IP address, date, and time of the game. The last information we can find here is the output of the anti-cheater analysis.

From the point of view of typo analysis, the most important entity is *GameResult* shown in Fig. 6. The entity contains an ID, a player identification, and a list of player answers. These answers are a collection of words that the player has transcribed. For each word, we record the original word - what the player had to write (Word), what the player wrote (PlayerWord) and a list of actions that the player took when writing the word (AnswerParts). This list of actions contains information about the last state of the written word (Word), the time in milliseconds since the beginning of the spelling of the word (TimeStamp), the type of event (Trigger) and the data associated with the event (Data). A more detailed description of the event will be given in the following chapters.

KEY	COUNT	%
analyzed documents	7257	100.0
_id -> *binData*	7257	100.0
GameId -> *binData*	7257	100.0
GameKey	7257	100.0
-> *null*	7136	98.3
-> *string*	121	1.7
HackerReason	6096	84.0
-> *null*	5237	85.9
-> *string*	859	14.1
IpAddress -> *string*	6936	95.6
PlayedAt -> *date*	6936	95.6
Player	7256	100.0
-> *null*	5401	74.4
-> *object*	1855	25.6
Email	1855	100.0
-> *null*	1761	94.9
-> *string*	94	5.1
Nickname -> *string*	1855	100.0
Typewriting	1578	85.1
-> *null*	534	33.8
-> *bool*	1044	66.2
PossibleHacker -> *bool*	6096	84.0
UserAgent -> *string*	6585	90.7
Value -> *int*	7257	100.0
Visible -> *bool*	6933	95.5
Wpm -> *int*	7257	100.0
WpmCorrect -> *int*	7257	100.0

Fig. 5. The structure of the Score collection

KEY	COUNT	%
analyzed documents	13592	100.0
_id -> *binData*	13592	100.0
Answers -> *array*	13592	100.0
[array item] -> *object*	141846	
AnswerParts -> *array*	141846	100.0
[array item] -> *object*	942637	
Data	942637	100.0
-> *null*	414	0.0
-> *string*	942223	100.0
TimeStamp -> *int*	942637	100.0
Trigger -> *string*	942637	100.0
Word -> *string*	942637	100.0
Duration -> *int*	141846	100.0
PlayerWord -> *string*	141846	100.0
Word -> *string*	141846	100.0
PersonIdentification	12547	92.3
-> *null*	52	0.4
-> *string*	12495	99.6

Fig. 6. The structure of the GameResult collection

Based on this record, we can easily determine whether the player spelled the word correctly, we can determine how long it took him to write the word and whether he was wrong in writing.

B. Application server

The application server is created in the ASP.NET Core framework. We use Razor pages technology for individual pages in the application ("About", "Top 100", "Stats", "Privacy"). The "About" and "Privacy" pages are completely static and contain basic information.

The "Top 100" (Fig. 2) and "Stats" pages (shown in Fig. 7) dynamically retrieve information from the database. The "Top 100" page displays a list of the best players based on the score for the last day, week, or all time. The Stats page shows an overview of the number of games played and various statistics in the form of charts.

In addition to these basic pages, the application also has a page for administrator login and a page for creating group games. The administrator can delete entries in the list of the best players in the event in case of cheating by the players.

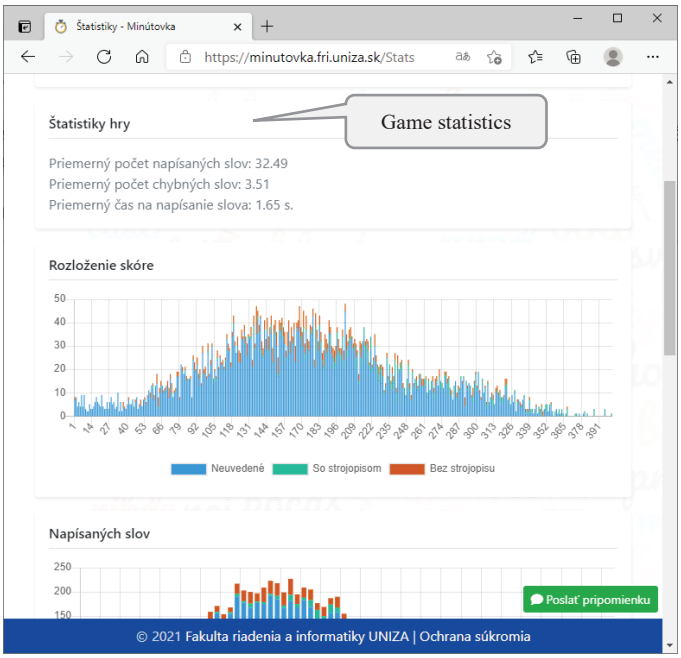

Fig. 7. Stats page of the Minutovka game

Another part of the application server is the API for playing the game. This API provides operations:

- Get text to transcribe
- Save game results
- Score calculation + storing a player information
- Get game score
- Data export

1) Getting text to transcribe: The texts used by the application for transcription were obtained from the Slovak Wikipedia from [19]. These texts were processed in several steps:

- Transformation to plain text using the WikiExtractor Python script [20]
- Text cleanup using own C# scripts
- Tokenization
- Sentense retrieval
- Removing nonsensical sentences - sentences that contain too many numbers or begin with a number have proven unsuitable for the purpose of transcribe words on time.

The result of this process was a set of sentences. The set of sentences from which the application generates the assignment has 134.4 MB in plain text. It consists of 1,287,680 sentences and 19,131,460 words. These sentences are used as input to an algorithm to generate texts for transcription. The algorithm randomly selects sentences from this list until it has the minimum length of the text which is 2000 characters. It does not make sense to generate more than 2000-character texts, as we assume that 2000 or more characters cannot be transcribed in one minute.

The text prepared in this way is stored in the database in the *GameToken* collection and together with the *GameToken* ID is returned from the appropriate API endpoint (Fig. 8).

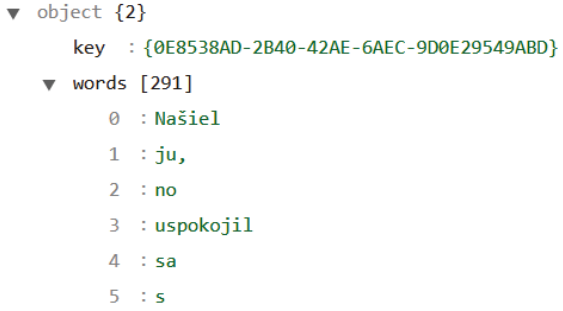

Fig. 8. Example of the game data (the first six Slovak words shown in the words)

2) Save game results: It is a simple method that takes output from a game (regardless of whether it ended successfully or prematurely) and saves it, along with other information, to the *GameResult* collection. As input, it receives the *Answers* field, which it saves without any transformation. This field is the same as presented in the database collection structure.

3) Score calculation: The calculation of the score was solved on the client in the initial version of the game, but after several attempts at cheating were detected, it was transferred to the server. Upon successful completion of the game, the client sends information about the completed game and asks the server to quantify the score of the attempt (Fig. 9).

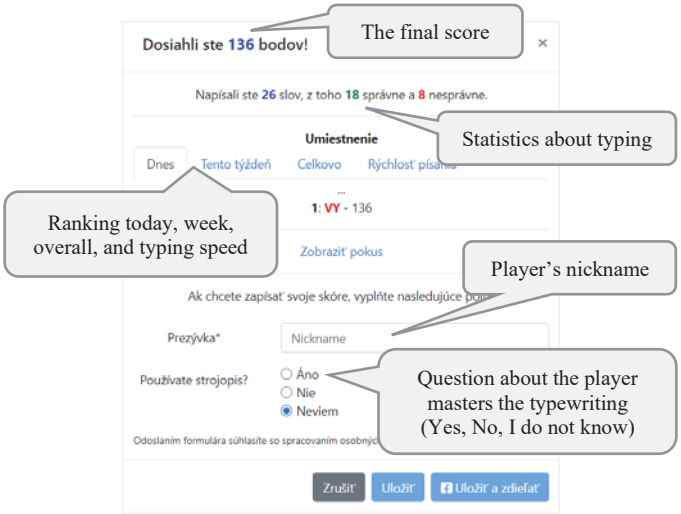

Fig. 9. Example of game score report

The server proceeds as follows to calculate the score:

- Load results for given game (*GameResult*)
- Load game assignment (*GameToken*)
- Calculates the score based on the input and the result of the game
- Calculate game statistics and typing speed chart

The base score and the number of words per minute are calculated from the list of answers stored in the game result (*GameResult* entity).

Words per minute (WPM) is the number of all words spelled out by a player, as one game lasts exactly 1 minute, so we do not need any other calculations. The number of correct and incorrect words is calculated based on the match between the written word and the sample word.

The Levenshtein distance metric is used to calculate the game score. The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by $lev_{a,b}(|a|, |b|)$ where

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

In the previous formula $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise, and $lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b. i and j are 1-based indices.

The total score of the game is then calculated from the submitted words as follows:

$$\sum_{\substack{submitted \\ words}} \min\left(0, len(originalWord) - 2 \times lev(originalWord, playerWord)\right)$$

This score calculation includes two aspects. The more words a player writes, the more points he can get. On the other hand, it pushes him to make as few mistakes as possible, as he gets penalized for wrong characters. The minimum scores a player can get per word is 0.

After calculating the achieved score, the algorithm finds the player's position in the top rankings. In addition to the score, the application offers the player an overview of the typing speed throughout the attempt. In this report, we divided 1 minute into twelve 5-second sections, in which we show the number of typed characters without spaces and line breaks. The application then displays this report in the form of a chart (Fig. 10).

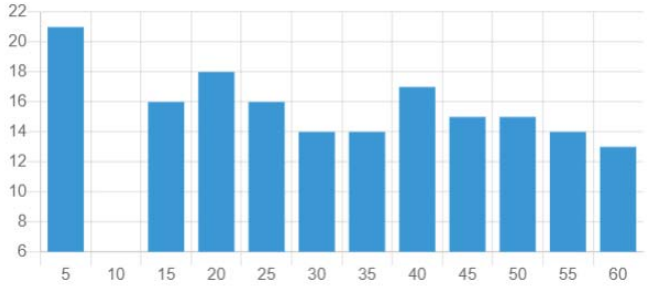

Fig. 10. Example of typing speed visualization (number of characters during the game in five-second bins)

In addition to the score itself and the typing speed report, the application also displays the typed text and highlights the errors (in red) made by the player as we can see in Fig. 11.

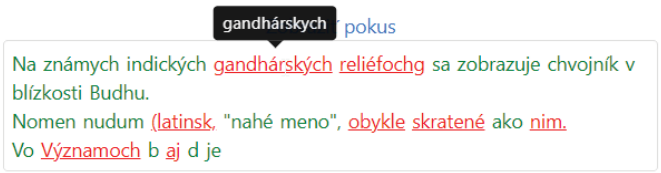

Fig. 11. The game summary

4) Detecting cheating attempts and "hacker score"

In most multiplayer games, players try to find the weakness of the game and use it to their advantage. In the original version of the game, the complete score calculation was performed on the client and the cheating was very simple.

Some of the cheating attempts were prevented by moving the game evaluation to the server, but this approach also proved insufficient. After moving the score calculation to the server, it was no longer easy to upload the adjusted score to the statistics, but it was necessary to create a larger payload, which contained a record of a correctly executed game with high typing speed.

The initial attempts to overcome this protection were such that the fraudster sent a record of the game to the server, but it was not valid.

Apparently, the score counts as the number of correct words based on user input. The cheaters prepared the payload (Fig. 12), which seemed to rewrite very quickly one and the same word. Thanks to this, they were able to get unlimited high scores.

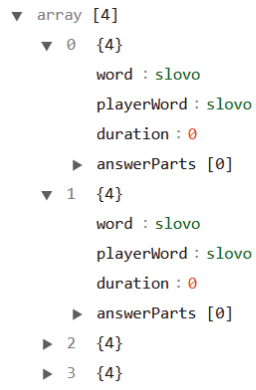

Fig. 12. Example cheated game result payload

We treated this problem by using a game token. The server saves the required text before the game starts, and the player must send the identifier of this game token along with the answers so that the server can verify that it is a valid attempt. Once the words sent by the player are the same as the words recorded in the game token, the player gets a zero score. The game token can only be used once and is only valid for a short time. This means that if you try to cheat, it can no longer be done manually and requires the need for automated tools. The tool must read real text and generate a payload, which passes the check, within one minute.

In addition, it is clear that it is not possible to write those 2000 characters per minute, so sending an attempt in less than 1 minute is also considered fraud. As a result, the automated tool

must also wait before submitting the score 1 minute after retrieving the text.

After checking the general rules, the written words are checked. When the player is typing, we record for each duration of the word and the list of all keys pressed. Ultimately, the sum of all words typing time must be less than 1 minute.

For each word, we then check whether the player pressed at least as many keys as there are characters in the typed word. We check whether the sum of the times for typing each character is the same as the time for typing the whole word. We also check individual times for negative, if the time for typing a word / character is less than or equal to 0. Then we consider the whole attempt to be a cheating.

In addition to these clear signs of cheating, we were forced to invent heuristics based on typing speed, because the methods mentioned so far could be deceived by a sophisticated algorithm.

Our heuristics compute the so-called "hacker score" and when exceeding the value of 1, the attempt is considered a cheating. In heuristics, we calculate the following characteristics:

- The word typing speed must be at least 100 ms (milliseconds) per character
- The keystroke length must be at least 100 ms
- If the player typed the word correctly but pressed more or less keys than the word length

For each word, which was written faster than $100\,ms \times world\ length$ we added „hacker score" according to the formula:

$$0.2 \times Min\left(1,1 - \frac{\frac{duration}{word\ length} - 40}{60}\right)$$

For each word that was typed quicker that $100\,ms \times key\ strokes$ we added „hacker score" according to the formula:

$$0.15 \times Min\left(1,1 - \frac{\frac{duration}{key\ strokes} - 40}{60}\right)$$

If the first condition is met, the second no longer applies. Both conditions seek to ensure that a player cannot write a maximum of 5 words in less than 100ms per character in order not to be labeled as cheater.

If the number of keys does not match the word length, we only check the keys with characters. If, the player presses more keys than the length of the given word and does not press any control key (e.g. Backspace, Delete, ...), then we add a hacker score of 0.3 for the given word.

Overall, the heuristics of limiting the maximum possible score to a little over 600, which is an acceptable upper limit. So far the best results we have obtained are around 390. It is possible that the 100 ms limit may be too high for professionals, so for professional purposes it would be appropriate to lower this limit.

C. Typing app

The last part of the application is the front-end part, which implements the very dynamics of the game. This part is implemented in the JavaScript programming language using the JQuery framework.

The application contains a minimalist user interface (Fig. 13). The potential player will be greeted on the home page by a text box displaying the text to be transcribed. In addition, it contains a graphic stopwatch that shows the remaining time. Below this is a large text box. Next to it, we can find a button to load other text. The game starts after the player type the first character of the sample text.

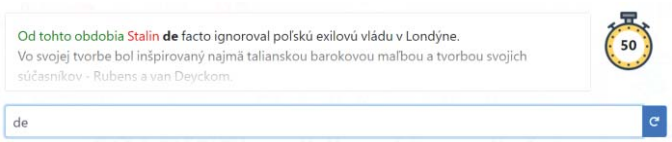

Fig. 13. Typing app frontend

The box with the displayed text always displays 3 lines of text. The first line is the line that the player is currently transcribing, and below are the following lines. While rewriting the text, the player is highlighted with the current word (marked with a bold) and the status of the words written so far - green are correctly spelled words and red are words in which the player made a mistake. After typing a line, the text automatically moves so that it always sees the currently transcribing words in the first displayed line. The text is copy-protected by the CSS property *user-select: none* and is overlaid by another element that also prevents copying of the sample text. This additionally adds some blur to the text at the bottom of the box.

The main input monitors user input. The countdown is started after first player input. This input field has a protection that is used to block text input using the Ctrl+V keyboard shortcut (Paste command). Each keystroke is recorded in the background. After pressing the "Space" and "Enter" key, the typed word is sent for processing and the transcription of the new word begins. The typed word is removed in the input - the user input always contains only one currently transcribed word. Multiple keystrokes have been protected against "sending" a word if the typed word is less than 50% of the length of the original word.

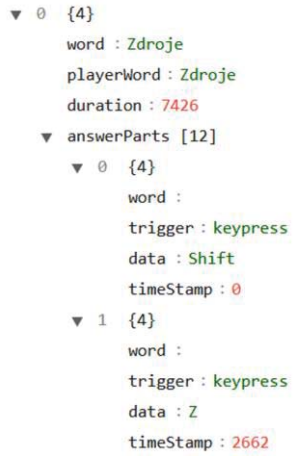

Fig. 14. Example data collected during word typing

After the timeout (1 minute), all acquired data are collected and sent to the server, which evaluates it. After the evaluation, the results of his game are graphically presented to the user (Fig. 9).

The data that the application collects at runtime was shown in part of the database layer. In this section, we will describe in detail one record of typing a word.

In Fig. 14 we can see part of the structure when writing the Slovak word "Zdroje" (in English: "Sources"). As we can see from the attached example, the player first pressed the Shift key and wrote the letter "Z" by up to 2662 ms. The total time to type this word was 7426 ms, while the player pressed 12 keys. The application also records repair attempts, which can be seen in the following image (Fig. 15):

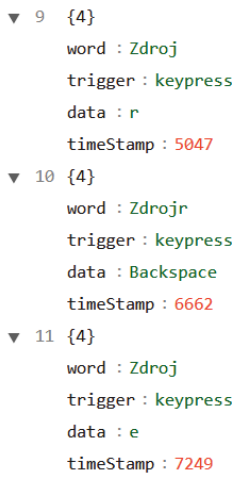

Fig. 15. Example shows fixing typing mistake

We can see that the player made a typo while typing the last character. He wrote the letter "r" instead of the letter "e". We can see on the record that the player corrected this by pressing the Backspace key.

In addition to typos, such data will allow us to analyze the way we write. On the Slovak keyboard there are some letters with accents directly, or they can be typed using two characters (for example a caron "ˇ" with a letter).

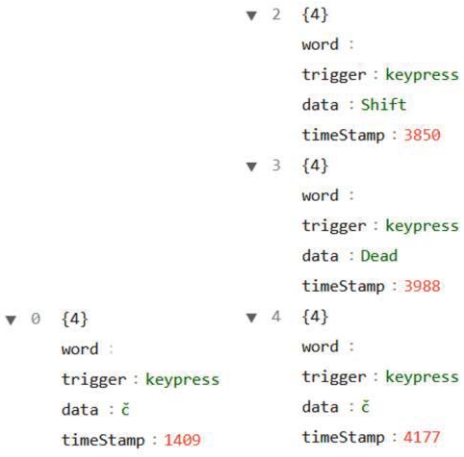

Fig. 16. Comparison of writing a letter č

Fig. 16 shows a comparison of the two records. In the first case (left) we see the typing of a letter "č" using a key on the keyboard. In the second example, we see the typing of the same letter "č" using the keyboard shortcut "Shift+ˇ+ c".

V. Results and Statistics

The application also contains some statistics from the game, which we made available to the public. The following statistics are available since the application was launched as shown in Table I.

TABLE I. Overall statistics since the game started

Number of played games	6,834
Number of players	1,772
Number of written words	221,114
Number of misspelled words	23,895
Number of corrected words during typing	6,952
Average number of written words	32.50
Average number of misspelled words	3.51
Average time to write a word (seconds)	1.65 s

Using the app, we managed to collect typos and mistakes that players made while playing the game. First, we excluded words where punctuation marks (dot, comma after a word) were missing only, then words where the words differed only by missing diacritics. The Levenshtein distance was the last criterion we used to filter the data. If the distance was bigger than the count of word characters divided by 2, we omitted the word from the misspelled list because this word was too different from the original. In total, we received from players 23,895 words with some mistakes. Finally, 14,903 words remained after applying the mentioned filters. After that, we analyzed the individual mistakes and found the following mistake categories that could be generalized (shown in the Table II).

TABLE II. Analyzed collected words from the game

Mistake category	**Count**	**%**	**Lev(a,b)**	**D(a,b)**
One incorrect letter	5042	33.83	1	0
One incorrect letter added	2896	19.43	1	1
One letter dropped	3028	20.32	1	-1
Two incorrect letters	592	3.97	2	0
Two letters swapped	1189	7.98	2	0
One incorrect, one letter added	514	3.45	2	1
One incorrect, one letter dropped	362	2.43	2	-1
Two incorrect letters added	209	1.40	2	2
Two incorrect letters dropped	240	1.61	2	-2
Combination of three errors	593	3.98	3	-3..3
More than 4 errors >= 4	238	1.60	4+	N/A

The meaning of the columns in the Table II. is as follows:

- *Column 1 (Mistake category)* – the type of mistake
- *Column 2 (Count)* – number of mistakes of the given type (out of all mistakes)
- *Column 3 (%)* – percentage of mistakes from total

- *Column 4 (Lev$_{(a,b)}$)* – Levenshtein distance between original and misspelled word in the given category
- *Column 5 (D$_{(a,b)}$)* – the difference in the number of characters between the misspelled and the original word in the given category

The following figures (Fig. 17 – 20) visualize the data we obtained from the game.

The graph in Fig. 17 shows the score distribution, where the x-axis represents the score and the y-axis the total number of games in which the players obtained the given score. We display the minimum score from 1 to 395, which is the highest score achieved so far. Of the total number of games for a given score, we display games in which the players indicate that they do not master the typewriting (red color), master the typewriting (green color) or did not provide this information at all (blue color). In this picture we can see that most players did not state whether they master the typewriting. In addition, we can see the almost normal distribution of scores.

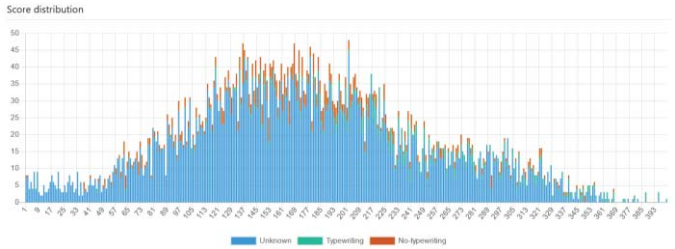

Fig. 17. Distribution of the obtained score and a corresponding number of games

The following graph (Fig. 18) shows the distribution of written words, where the x-axis represents the number of words and the y-axis the total number of games during which the players rewrite the given number of words within one minute. Again, it is possible to see an almost normal distribution, with the average number of words written by all players being 32.50.

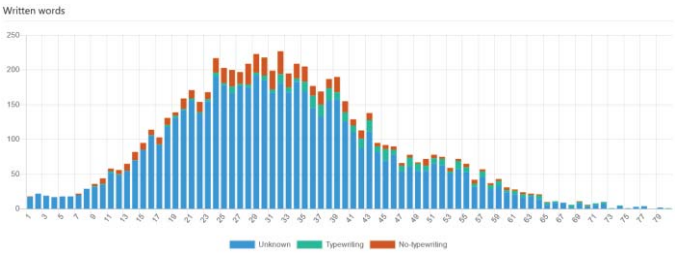

Fig. 18. Distribution of written words and a corresponding number of games

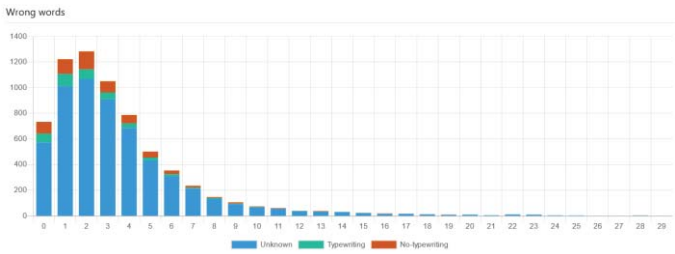

Fig. 19. Distribution of wrong words and a corresponding number of games

Fig. 19 shows the graph containing a number of erroneous words (x-axis) that players made when transcribing the text (the y-axis represents the total number of games played). We can see that most often 2 wrong words were transcribed in one minute.

The last graph (Fig. 20) shows the total number of games (y-axis) in which the player wrote a word with the specified number of characters (x-axis). We could see that the players most often wrote two-character and one-character words, while in Slovak the most common occurrence is a 7.5-character word. This means that players most often mistyped one-character and two-character words.

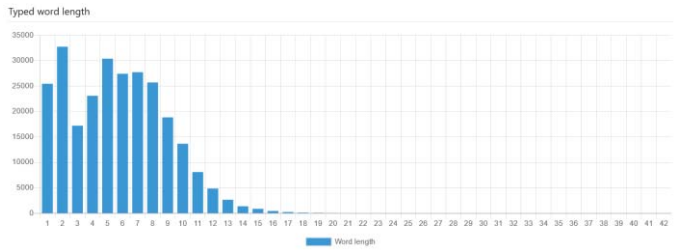

Fig. 20. Distribution of typed word length and a corresponding number of games

VI. Conclusion

In this article, we dealt with the design and development of the web word-typing game called Minutovka. The main goal of the game was to get typing errors in the Slovak language. In addition, we used it to get information how people write and how often they make typos. Thanks to this data, we can then generate a larger set of misspelt words (error corpus) and use them as inputs into various machine learning algorithms for natural language processing, especially with focusing on the Slovak language.

We would like to further expand the game and use it in secondary schools, where typewriting is taught. In addition to the generated texts, teachers could use their own texts and set a maximum transcription time. Students could rewrite these texts, and thus learn and practice typewriting, get feedback, and compare themselves with classmates to motivate them to improve. In this way, we could get a lot more data.

In the future, it could also be interesting to create other types of games, e.g., 2D or 3D games that could be even more interesting with a longer player's attention. In addition, it would be appropriate to obtain data in other ways than just transcribing the displayed texts. People often make mistakes also due to ignorance of spelling or misunderstanding of the word heard. In this case, it would also be possible to obtain data through dictation and thus transcription of words by players from recordings they would hear.

Acknowledgment

Acknowledgement: This work was supported by Grant System of University of Zilina No. KOR/1122/2020.

References

[1] „WolframAlpha computational intelligence,“ Wolfram Alpha LLC, [Online]. Available: https://www.wolframalpha.com/input/?i=total+number+of+Slovak+speakers. [Cit. March 2021].

[2] P. Rodrigues a C. A. Rytting, „Typing Race Games as a Method to Create Spelling Error Corpora," rev. *International Conference on Language Resources and Evaluation (LREC)*, Istanbul, 2012.

[3] R. Tachibana a M. Komachi, „Analysis of English Spelling Errors in a Word-Typing Game," rev. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia, 2016.

[4] Z. Ga a J. Jun, „Design and implement a typing game for commonly used Tibetan words based on the Macromedia Flash," rev. *2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation (IMSNA)*, Toronto, 2013.

[5] T.-L. Wang, T.-K. Chen a Y.-F. Tseng, „An leaner-centred, game-based, learning framework for typing games in English course," rev. *2010 International Symposium on Computer, Communication, Control and Automation (3CA)*, Tainan, Taiwan, 2010.

[6] N. Henze, E. Rukzio a S. Boll, „Observational and Experimental Investigation of Typing," rev. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[7] G. Costagliola, M. De Rosa, V. Fuccella a F. Torre, „TypeJump: A typing game for KeyScretch," rev. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Innsbruck, 2012.

[8] The International Arcade Museum, „The Typing Of The Dead," WebMagic Ventures, [Online]. Available: https://www.arcade-museum.com/game_detail.php?game_id=10244. [Cit. 27 September 2019].

[9] „Typing of the Dead: Overkill," SEGA, 2013. [Online]. Available: https://www.sega.com/games/typing-dead-overkill-0. [Cit. March 2021].

[10] „The Typing of The Dead: Overkill on Steam," October 2013. [Online]. Available:https://store.steampowered.com/app/246580/The_Typing_of_The_Dead_Overkill/. [Cit. March 2021].

[11] „TypeRacer," [Online]. Available: https://data.typeracer.com/misc/about. [Cit. 27 September 2019].

[12] Szablewski D. (PhobosLab), „ZType - Typing Game - Type to Shoot," [Online]. Available: https://zty.pe/. [Cit. 27 September 2019].

[13] Typing.com, „Typing Games," [Online]. Available: https://www.typing.com/student/games. [Cit. 27 September 2019].

[14] „Typing Games Zone - Play 131 Fun Keyboard Games Online," TypingMaster, Inc., [Online]. Available: https://www.typinggames.zone/. [Cit. March 2021].

[15] „Free typing games, typing lessons and typing tests. | FreeTypingGame.net," [Online]. Available: https://www.freetypinggame.net/. [Cit. March 2021].

[16] „Free Online Typing Games for Kids | Education.com," [Online]. Available: https://www.education.com/games/typing/. [Cit. March 2021].

[17] M. Kvet, E. Kršák a K. Matiaško, „Study on effective temporal data retrieval leveraging complex indexed architecture," *Applied Sciences*, zv. 11, %1. vyd.ISSN 2076-3417, 2021.

[18] Z. Parker, S. Poe a V. S. Vrbsky, „Comparing NoSQL MongoDB to an SQL DB," rev. *Association for Computing Machinery*, New York, NY, USA, 2013.

[19] Wikimedia Downloads, „skwiki dump," [Online]. Available: https://dumps.wikimedia.org/skwiki/latest/.

[20] G. Attardi, „WikiExtractor," 2009. [Online]. Available: https://github.com/attardi/wikiextractor. [Cit. March 2021].