A Mobile ITS for Invented Spelling Practice

No Author Given

No Institute Given

Abstract. Invented spelling is a common exercise administered in kindergarten classrooms to bolster students' phonological awareness and other early literacy skills in preparation for regular reading and spelling instruction. A correct invented spelling is composed of phonetically appropriate (but not necessarily correct) letters. For instance, "jyraf" is a phonetically correct invented spelling for "giraffe". Prior work has demonstrated that invented spelling can have a significant positive impact on students' future reading and writing. However, it is not an exercise that is always feasible to facilitate effectively at scale since it typically is most effective when children receive individualized feedback on their spellings. In this work we present a mobile application that robustly provides feedback for invented spelling practice on nearly any target word. We outline the technical details of this intelligent tutoring system, including our methods for implementing incremental feedback, the sounding-out of invented spellings, and outer-loop adaptivity. Additionally we discuss several interface design considerations that make our application user-friendly and engaging to 4- to 6-year-old learners and share some initial user testing results.

Keywords: Invented Spelling · Mobile App · Intelligent Tutoring System

1 Introduction

Invented spelling is an exercise where students are encouraged to try spelling words based on their current letter knowledge. Invented spelling practice engages important prerequisite skills for reading and writing such as knowledge of grapheme-phoneme (i.e. letter-to-sound) mappings and phonological awareness—the ability to identify the individual sounds that make up words. Prior work has demonstrated the importance of invented spelling as an early literacy skill. For instance, Ouellete and Sénéchal [7] found that invented spelling mediates the relationship between phonological awareness and word reading in kindergarten and that invented spelling predicts word reading and spelling abilities in first grade students.

Unlike traditional spelling exercises that have a single correct answer, invented spelling lends itself to a broad spectrum of potentially correct answers, and many degrees to which an answer can be incorrect. For instance, "trubl" is a phonetically correct spelling for "trouble", since all of the required sounds "T-R-UH-B-L" are present. Although "trbr" and "trbl" are neither correct nor phonetically correct, "trbl" is closer to correct since only the vowel "UH" phoneme is missing. Invented spelling gives students an emotionally low-risk way to engage in spelling, and provides tutors opportunities to provide fine-grained feedback, and celebrate students' competencies. By lowering the bar for what counts as correct, students can build confidence and feel empowered as spellers [10]. For instance, Clarke [2] found that students whose teachers encouraged invented spelling exercises tended to write longer stories with a greater variety of words, than students expected to produce correct spellings.

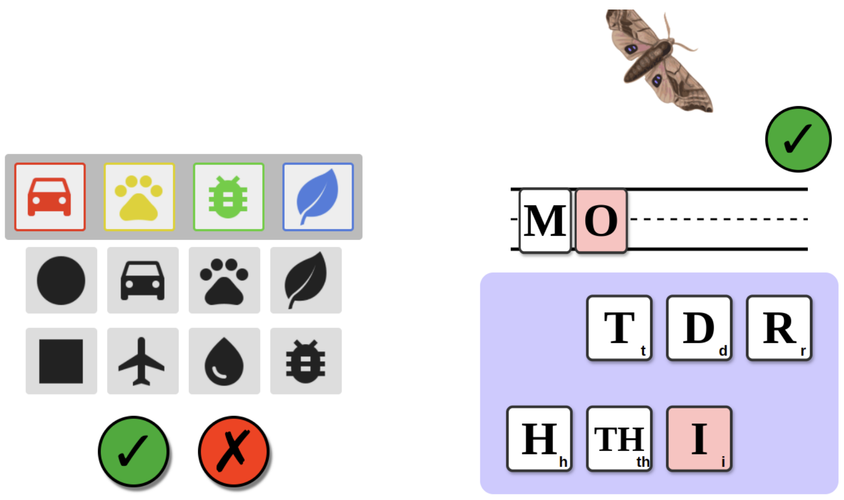
Invented spelling practice represents a potentially high-impact domain for implementing an intelligent tutoring system since facilitating invented spelling requires considerable one-on-one feedback from adults. It is, however, a technically challenging domain to produce automated feedback for, since there are many ways that invented spellings can be correct and many degrees to which they can be incorrect. Moreover, invented spelling presents several design challenges since ideally feedback must be conveyed in a manner that is understandable and engaging to preliterate users, friendly enough to avoid emotional upsets, and effective at facilitating learning.

Prior research on invented spelling provides two key considerations for effectively facilitating invented spelling practice. Firstly, Levin and Aram [4] find that invented spelling works best when the tutor sounds out the phonemes associated with letters instead of only naming the letters so as to provide students with the correct phoneme-grapheme (i.e. sound-to-letter(s)) relationship. Secondly, prior research [6, 9, 10] shows that the most effective mode of feedback to give in invented spelling exercises is incremental: removing, correcting, or adding a single letter in children's invented spellings to make them more phonetically correct. One explanation for this effect is that by providing just a single correction at a time the tutors can minimize the cognitive load a student experiences in trying to determine why corrections to their spelling are appropriate.

In this work we present a mobile intelligent tutoring system for invented spelling practice based on the incremental feedback procedure outlined by Ouellette and Sénéchal [6]. We begin by presenting an overview of our app and discuss several design decisions we've made toward making the app usable and engaging for kindergarteners. Next, we present our algorithm for producing feedback—a dynamic programming based method that produces an optimal correction sequence for transforming phonetically incorrect invented spellings to phonetically correct ones. This implementation draws inspiration from solutions to the related string-to-string correction problem [12]. We additionally demonstrate how this algorithm can be extended to find optimal sounding-outs (i.e. phoneme sequences) for phonetically incorrect invented spellings. Finally, we discuss our BKT [3] based implementation of outer-loop adaptivity [11] with a simple next-problem selection policy that selects words that students are likely to get wrong by a single letter.

2 An Invented Spelling Mobile ITS

Our mobile intelligent tutoring system is a cross platform application built with ReactNative that can be used with Android or iPad tablets or in any modern browser. The application is designed to be used by 4- to 6-year-old children across several sessions. It is designed from login to signout to be used by preliterate children independently, without the need for adult intervention. After a student account is made they are given a personal password consisting of a string of easily identifiable non-alphanumeric symbols. When the correct sequence of symbols is entered, students immediately begin invented spelling practice, which involves dragging available letter tiles to spell a target word.



(a) Login screen. (b) Invented spelling mobile ITS.

Fig. 1

When an invented spelling problem begins, the target word is presented with the audio: “Let’s spell [word] [word-slow] [word]” (where [word] is the target word), accompanied by an image depicting the target word. The target word is spoken three times (slowly the second time), in order to ensure that students clearly hear it, and to give them ample opportunity to shift their attention to the app. If students miss or forget the word that they need to spell, they can press on the image associated with the target word to have the audio repeated. In order to reinforce the idea that students should attempt to spell words even if they don’t know how to spell the word, we added an additional prompt triggered by inactivity. Every 17 seconds if a student hasn’t begun interacting with the app then the letter tiles will hop and wiggle to indicate that they are interactive and one of the following audio prompts is randomly played:

– “Try out dragging some letters.”
– “Your spelling doesn’t have to be perfect, just try something out!”
– “Go ahead try some letters.”

The set of letter tiles are randomly arranged and consist of all of the letters needed to spell the target word correctly, in addition to several additional random

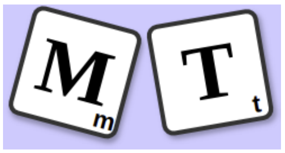


Fig. 2: Wiggling tiles.

letters. In early user testing we found that presenting the entire alphabet led students to spend a considerable portion of time searching for letters, but found that an exhaustible set of 8-10 total letter tiles limited search time but was still sufficiently challenging. Each letter tile contains a single grapheme—for example, a single letter or a digraph such as "TH" or "CH". To further reduce search time, consonant tiles always precede vowels. In the spirit of Meyer's multimedia learning principles [5], we've opted to keep the design of our tiles simple to avoid extraneous visual detail (like for instance varied font colors). Every grapheme is written in large black capital letters in a bold serif font accompanied by its lowercase form placed smaller in the bottom right corner. As a visual aid, vowel graphemes have a pale red background while consonants have white one.

To spell the target word students drag letter tiles onto a line resembling the blank lines of worksheets (Fig. 1.b) often used in kindergarten classrooms for spelling practice. When placed, the letters move into place adjacent to each other and can be rearranged by dragging. The student indicates that their spelling is correct by pressing a green checkmark button. If any tiles have been placed and no tiles have been interacted with for 10 seconds then the checkmark button will hop and wiggle accompanied by the audio prompt: "when you're done press the green checkmark". To prevent patterns of behavior where students just place all of the tiles, and reinforce the objective of picking a subset of the available letters, only two more tiles than the length the target word can be placed. Any tiles that students try to place beyond this will not snap to the spelling line.

After the green checkmark button is pressed the feedback phase begins, signified by one of three audio prompts praising the student for their effort: "Great job!", "Cool spelling!" or "Nice work!". At the start of the feedback phase the best correction sequence between the target word and the student's spelling is computed in preparation of giving feedback (Section 3). First the application sounds out the student's spelling which is initiated by the prompt: "let's sound out what you spelled". To find an appropriate sounding-out the best sequence of phonemes is found (using the correction sequence, Section 3.3) for the placed tiles. This is necessary to determine which phoneme should be spoken when a grapheme could produce multiple possible phonemes, and in cases where a pair of graphemes make one sound. For instance in "giraffe", "g" makes the "J" sound, and "ffe" collectively make the "F" sound. As each phoneme is spoken by the app, the tiles associated with this phoneme expand and retract their borders as a visual aid.

Next, if the student's spelling is not phonetically correct, the first item in the correction sequence is utilized to give incremental feedback that removes, replaces, or adds a tile. To facilitate comparison the student's tiles remain unmoved during this process. First copies of the student's tiles fall in from the top of the screen and rest above the placed tiles, followed by the audio prompt:

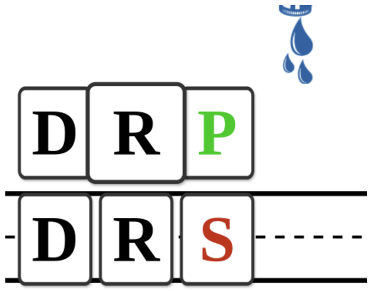


Fig. 3: An instance of incremental feedback on the target word ”drip” for spelling ”drs”. ”s” is replaced with ”p”. The ”R” tile is expanded because the ”R” phoneme of the corrected spelling is mid-way being sounded out.

“Let’s see if we can make your spelling a little closer to [word]”. Audio prompts accompanied by animations narrate the feedback. For instance if the target word “drip” was spelled “drs”, then the app would narrate “we’ll replace, ‘s’ with ‘p’ to make the ‘P sound”, as the copy of the “s” ascends off the screen and is replaced by a green colored “p” [1] Meanwhile the original “s”, which has remained in place, is colored red. Finally the improved spelling is sounded out, followed by the prompt: “This spellings sounds more like [word]”. If this corrected spelling is phonetically correct or totally correct then the audio prompt states this. If the student’s original spelling was phonetically or totally correct, they receive an audio prompt indicating this. However, in the event that a spelling is phonetically, but not totally correct then tiles spelling out the correct spelling fall in from the top of the screen accompanied by the prompt: “the correct spelling for [word] is”, followed by audio naming each letter in order. Consequently, the app facilitates correct spelling instruction only after the student can provide a phonetically correct spelling.

Finally a screen is displayed with several outlines of silver stars followed by the outline of a gold star, and simultaneously one of the three praise prompts is played again. The stars are filled in in accordance with the number that the student has earned based off of their spelling. Each silver star is earned for correct phonemes, calculated as the total phonemes in the word minus the length of the calculated correction sequence. The gold star is earned for producing the correct spelling for the word. At the same time a count of the total number of stars of each type earned (per session) tick up. After the star screen is finished animating, a fresh problem is started. If the word was not spelled phonetically correct then the same word is played again (up to a total of two consecutive times), otherwise a new word is selected using the outer loop adaptivity algorithm (Section 4).

[1] Note in this example that “p” is incorporated instead of “i”. The calculated correction sequences favor fixing missing or incorrect consonants (see Section 3.4)

3 Finding Correction Sequences for Invented Spellings

3.1 Backgound: The Wagner-Fisher Correction Sequence Algorithm

The string-to-string correction sequence problem [12] is to find the minimum cost sequence of edit operations needed to change one string into another. The Wagner-Fischer algorithm is a classic dynamic programming approach to solving this problem that recursively breaks down the problem into simpler subproblems. The algorithm proceeds by calculating the levenshtein distance between the two strings, defined as the minimum number of edit operations needed to transform a string A into a string B. This is achieved by iteratively filling a matrix D with the levenshtein distance between each prefix of A and B. The matrix can be filled one row or column at a time using previously computed values.

Algorithm 1 Compute Levenshtein (for Wagner-Fisher algorithm)

```
    Input(A : String, B : String)
 1: $D[0..,0] := 0$
 2: for $i := 1$ to $|A|$ do $D[i,0] := i$
 3: for $j := 1$ to $|B|$ do $D[0,j] := j$
 4: for $i := 1$ to $|A|$ do
 5:     for $j := 1$ to $|B|$ do
 6:         $substitutionCost = 1$ if $A[i] = B[j]$ else 0
 7:         $m_{substitute} := D[i-1,j-1] + substitutionCost;$
 8:         $m_{insert} := D[i,j-1] + 1;$
 9:         $m_{delete} := D[i-1,j] + 1;$
10:         $D[i,j] := min(m_{substitute}, m_{insert}, m_{delete});$
11: return D
```

After the levenshtein distance matrix D has been computed, the final element of the matrix holds the levenshtein distance between the input strings. The edit sequence between the strings can be obtained by tracing back from the final element in the matrix back to the first element along the path with the minimum levenshtein distance values. Diagonal steps that have a change in distance represent substitutions, diagonal steps with no change in cost represent no-edit, steps along the first dimension represent deletions, and along the second represent insertions. Alternatively the edit sequence can be maintained in a forward fashion by filling an equal sized matrix M with the edit sequence so far for each prefix pair. This method is typically not used since it is somewhat less memory efficient (an edit sequence must be stored for $O(|A||B|)$ prefix pairs), but serves as a good analogy to the method we describe in the following section for finding correction sequences for invented spellings.

3.2 An algorithm for producing invented spelling correction sequences.

Finding correction sequences for invented spellings poses additional challenges on top of the classic string-to-string correction sequence problem. Firstly, two invented spellings can be phonetically equivalent even if their characters are not equivalent, so we cannot rely on simple character equality checks to determine whether a substitution or no-edit operation is appropriate. Second, the goal sequence is a sequence of phonemes and the input sequence is a set of letters which need to be broken up into graphemes that can produce the target phonemes. Since we are not dealing with a simple character to character mapping it is necessary to sometimes make jumps of multiple characters to handle multi-character graphemes. Third, the sequence of letters from the input spelling can have multiple feasible pronunciations. Consider for instance a mispronunciation of the character sequence "foothill", that pronounces the "t" and "h" separately instead of as the digraph "th". Fourth, there are rare cases where a single character can produce multiple phonemes such as the letter "x" which can produce the phoneme sequence "K,S".

Some raw English language data is necessary as input to this algorithm. We utilize the Carnegie Mellon University Pronouncing Dictionary to find phoneme sequences for over 134,000 English words encoded using the ARPAbet phoneme set [1]. We have also mined, and hand cleaned this data to produce an additional dictionary that maps phonemes to lists of possible graphemes (i.e. letter sequences). For instance the hard "c" sound represented as "K" maps to the list ["c", "ck", "k"]. In order to align feedback with phoneme to grapheme relationships that kindergarteners would likely encounter, some infrequent graphemes have been removed from this set, including for instance those that only appear in French words (like "oux" in "roux").

Several subroutines in our algorithm depend on a PartialSpelling data structure which has the following properties:

```
class PartialSpelling:
    s_prefix_graphemes : string[]
    s_postfix_letters : string[]
    s_prefix_phonemes : string[]
    w_prefix_phonemes : string[]
    w_postfix_phonemes : string[]
    lev_dist : float
    parent : PartialSpelling
    correction : SpellingCorrection
```

Analogous to the Wagner-Fisher algorithm we use dynamic programming to incrementally solve the correction sequence problem. But instead of filling a matrix D with levenshtein distances we fill a matrix M with PartialSpellings. The first dimension of the matrix M represents prefixes of the characters of the input spelling, and the second dimension represents prefixes of the phonemes sequence for the target word. Each PartialSpelling object is analogous to a prefix pair in the Wagner-Fisher algorithm, but while matrix elements in the Wagner-Fisher algorithm encode the Levenstein distances between prefixes, in our algorithm each matrix element holds a PartialSpelling object.

A PartialSpelling holds the sequence of phonemes and graphemes for a spelling prefix (s_prefix_phonemes and s_prefix_graphemes) and the remaining letters in the spelling (s_postfix_letters), in addition to a prefix for the sequence of satisfied target phonemes (w_prefix_phonemes) and the remaining phonemes (w_postfix_phonemes). PartialSpellings also hold the phonetic levenshtein distance between the committed s_prefix_phonemes and the target word's phoneme sequence (lev_dist). Additionally, the PartialSpelling contains a reference to its parent PartialSpelling (parent), and a data structure called a SpellingCorrection (correction) that encodes information about the insertion, deletion, replacement, or no-edit operation that produced the PartialSpelling.

PartialSpellings have several methods that produce new child PartialSpellings to iteratively build out the matrix M. next_matchings() looks for graphemes that that appear next in s_postfix_letters and can be pronounced as the next phoneme in w_postfix_phonemes. For each such match next_matchings() returns a new PartialSpelling where the found phoneme-grapheme pair is appended to the prefix of the current PartialSpelling and removed from the postfix. Every match found by next_matchings() constitutes a viable no-edit operation. insert() returns a new PartialSpelling where the next phoneme-grapheme pair at the head of the word postfix is removed and added to the current spelling prefix. delete() returns a new partial spelling where the next letter from s_postfix_letters is removed. Finally, substitute() creates a new PartialSpelling that combines the changes made by insert() and delete—one letter from s_postfix_letters is removed and the next phoneme-grapheme pair is moved to the prefix. insert() and substitute() both take the breakdown of phonemes and graphemes from the target word as input in order to get the grapheme associated with the phoneme at the head of w_postfix_phonemes.

Algorithm 2 shows pseudocode for our invented spelling correction sequence algorithm which takes a spelling S and a target word W and outputs a correction sequence to edit S such that it is phonetically equivalent to W. First the phoneme sequence P for word W is found from the CMU Pronouncing Dictionary (line 1). Then a simple greedy search process finds the best breakdown B of letters to grapheme-phoneme pairs in the target word (line 2). First a starting partial spelling is placed at $M[0,0]$ with s_postfix_letters equal to the input spelling and w_postfix_phonemes equal to the phonemes for the target word (line 4). Then the delete() and insert() commands are called to build out the base cases in the first column and row (lines 5-6). Next, the matrix M is looped through along its two dimensions, and on each iteration each partial spellings in $M[i-1, j-1]$ is checked with next_matchings() to produce a series of partial spellings where the next letters in the partial spelling's s_letter_postfix can be moved to their prefixes with a no-edit operation (lines 9-14). Phoneme-grapheme pairs associated with these matchings can each have lengths equal to or greater than one, and are placed in the appropriate elements of the matrix M. At this stage in the body of the loop if $M[i][j]$ is empty then it is filled with substitutions, insertions, or deletions from the PartialSpellings in $M[i-1][j-1]$, $M[i][j-1]$, or $M[i-1][j]$, respectively depending on which one contains the PartialSpelling with the minimum lev_dist (lines 15-27). Finally the correction sequence is produced by starting with the PartialSpelling in the final element of the matrix $M[|S|, |P|]$, and extracting

Algorithm 2 Compute Correction Sequence

```
    Input(W : String, S : String)
 1: $P =: word\_phonemes(W)$
 2: $B =: best\_break\_down(W, P)$
 3: $M =:$ new empty Matrix of size $(|S|, |P|)$
 4: $M[0,0] =: PartialSpelling(s\_postfix\_letters =: S, w\_postfix\_phonemes =: P)$
 5: for $i := 1$ to $|S|$ do $M[i,0] := M[i-1,0].delete()$
 6: for $j := 1$ to $|P|$ do $M[0,j] := M[0,j-1].insert()$
 7: for $i := 1$ to $|S|$ do
 8:     for $j := 1$ to $|P|$ do
 9:         if $M[i-1][j-1]$ is not empty then
10:             for each $match$ in $M[i-1][j-1].next\_matchings()$ do
11:                 $k =: i - match.phoneme\_length - 1$
12:                 $r =: j + match.grapheme\_length - 1$
13:                 if $M[k][r]$ is empty or $M[k][r].lev\_dist > match.lev\_dist$ then
14:                     $M[k][r] =: match$
15:         if $M[i][j]$ is empty then
16:             $m_{substitute} =: M[i-1][j-1].lev\_dist$
17:             $m_{insert} =: M[i][j-1].lev\_dist$
18:             $m_{delete} =: M[i-1][j].lev\_dist$
19:             $m =: min(m_{substitute}, m_{insert}, m_{delete})$
20:             if $m = m_{substitute}$ then
21:                 $ps =: M[i-1][j-1].substitute(B)$
22:             else if $m = m_{insert}$ then
23:                 $ps =: M[i][j-1].insert(B)$
24:             else
25:                 $ps =: M[i-1][j].delete()$
26:             if $M[i][j]$ empty or $M[i][j].lev\_dist > m + 1$ then
27:                 $M[i][j] =: ps$
28: $ps =: M[|S|, |P|]$
29: $corr\_seq =: []$
30: while $ps.parent \neq null$ do
31:     if $ps.correction \neq null$ then
32:         $corr\_seq =: [ps.correction, \ldots corr\_seq]$
33:     $ps =: ps.parent$
34: return $favor\_consonant\_corrections(corr\_seq)$
```

the corrections from each partial spelling (lines 28-32). A final post-processing method described in Section 3.4 modifies the correction sequence before it is returned (line 34).

Table 1 shows a few examples of target-words, spellings, correction sequences, and the final spelling after the correction. Note that after the correction sequence is applied the final spelling is phonetically correct, and retains all phonetically correct elements of the source spelling like "K" instead of "C" to spell "SKRAP".

Target Word	Spelling	Correction Sequence	Final Spelling
SCRAP	SMKRMOP	remove(1,1), replace(3,1,A), remove(4,1)	SKRAP
TRIP	PTRD	remove(0,1), replace(2,1,P), insert(2,I)	TRIP
PITTSBURGH	BITSBG	replace(0,1,P), insert(5,UR)	PITSBURG

Table 1: Application of correction sequences on phonetically incorrect spellings. Correction sequence items take arguments of the form (index, length, new value).

3.3 Finding Optimal Sounding-Out

Finding a correct sounding-out for an incorrect invented spelling is nontrivial because it can have incorrect or missing graphemes. However, the phoneme-grapheme breakdown of the target word of the spelling can help disambiguate which phonemes should be used. In the process of finding the optimal correction sequence, an optimal alignment between the source spelling and the phonemes of the target word was found. So we can simply draw from phoneme-grapheme breakdown found in the final PartialSpelling in the matrix M. For extra graphemes in the input spelling that would be deleted by the correction sequence we can simply guess the pronunciation by picking out a hard-coded default phoneme for each grapheme. For sequences of multiple letters that would be deleted, we use a greedy matching process that favors breakdowns with fewer longer graphemes. For instance an extraneous ”ee” in a spelling would be pronounced ”IY” instead of ”EH,EH”.

3.4 Reordering Correction Sequences to Favor Consonant Corrections.

Early in the process of learning to spell, students often spell words only using consonants, for example, the spelling “grs” for “grass” [8]. Common abbreviations like “hwy” for “highway” or “mgmt” for “management” shed some light on this phonomemna—it is all too easy to impute the sound of a vowel while pronouncing the individual sounds of consonants. We have no evidence to suggest that learning to spell with consonants prior to introducing vowels is of any particular learning benefit, but in the interest of adapting to the typical pattern of spelling that we’ve observed in our interactions with kindergarteners, we’ve introduced a subroutine favor_consonant_corrections() to our adaptive feedback to favor the insertion of missing consonants over vowels. For instance in the second example in Table 1, the original correction sequence was [remove(0,1), replace(2,1,I), insert(3,P)], but after rearrangement it becomes [remove(0,1), replace(2,1,P), insert(2,I)]. Note that the replace() correction is transferred from the “I” to the “P”, demoting the I to an insert() correction.

4 Outerloop Adaptivity with BKT

We use an implementation of Bayesian Knoweldge Tracing (BKT) [3] to provide outerloop adaptivity for selecting next problems. For an initial knowledge component model we track one knowledge component for each letter in the alphabet plus the three diagraphs “th” “ch” and “sh”. Additionally we track knowledge components for 19 different two letter blends (like ”sp”, ”fr”, ”gl”, or ”nt”). To limit the initial letterset that students are presented, we break knowledge components into 13 levels where each level introduces a few more letters or blends, and with them additional words. Each new level is added into the total available word pool when BKT predicts that the student knows every available word knowledge component with at least 60% probability, and at least half of the available blend knowledge components with at least 60% probability.

When a new problem is requested it is sampled randomly from a distribution of all unlocked words. The unnormalized probability of selecting each word is computed as the absolute difference in the BKT prediction between the knowledge components with the lowest and second lowest BKT prediction probabilities:

$$P(word) = |\min_{k_1 \in K}(BKT(k_1)) - \min_{k_2 \in K,\ k_2 \neq k_1}(BKT(k_2))|$$

We’ve chosen this method of scoring next words because it roughly favors words where all but one letter or blend has been mastered by the student.

5 Pilot Testing

5.1 Methods

We pilot tested the version of our app presented here with five kindergarteners from a local lab school. We asked our participants to spend 10 minutes playing with our app. We introduced the participants to the app with a brief verbal introduction, and followed up after the 10 minute session with a visual likert scale assessment using sad through happy faces. The 1/sad-face indicated that they ”definitely wouldn’t want to play this game again”, and the 5/happy-face indicated that they ”definitely would want to play this game again”. We also asked the participants for their general impression of the app.

5.2 Results

All five participants completed the 10 minute session. None of the participants had trouble completing problems, and the participants exhibited a wide range of invented spelling capability. The participants generally understood and followed the audio prompts, with few exceptions. We observed a mix of behaviors that included, for instance, randomly placing tiles, spelling words completely correct, repeating the same incorrect spelling on the second attempt, and improving spellings in response to the given feedback on the second attempt. We observed a general pattern of students trying new spellings, or following suggestions from the feedback as they used the app more. Several participants verbally indicated that they enjoyed seeing the animation for earning stars at the end of each problem. On the happy/sad-face likert scale, three students rated the app with a 5, one rated it a 4, and one rated it 1 . The most positive verbal impression we recieved was that ”the app helped me spell a lot of new words that I didn’t know". The most negative verbal impression indicated that there was a different educational app that the participant preferred instead.

5.3 Discussion

Our impression from this small pilot study was that the app was usable by the participants, fairly engaging, and enjoyable. In future work we hope to gain a quantitative measure of the learning benefits of the app. However, our initial observations are promising since we found that the app often succeeded at providing feedback that students incorporated into their invented spellings, and that users of the app showed an interesting in improving their spellings.

6 Conclusion

Invented spelling practice is a potentially high-impact domain since it is effective at building core prerequisite skills for reading and writing, but requires nuanced feedback typically only possible in a one-on-one tutoring setting. In this work we've presented a robust algorithmic method for providing automated adaptive invented spelling feedback embedded in a kindergartener friendly app. Our initial pilot testing yields promising initial results for future work assessing the efficacy of this innovative intelligent tutoring system.

References

1. The cmu pronouncing dictionary, http://www.speech.cs.cmu.edu/cgi-bin/cmudict
2. Clarke, L.K.: Invented versus traditional spelling in first graders' writings: Effects on learning to spell and read. Research in the Teaching of English pp. 281–309 (1988)
3. Corbett, A.T., Anderson, J.R., O'Brien, A.T.: Student modeling in the act programming tutor. Cognitively Diagnostic Assessment pp. 19–41 (1995)
4. Levin, I., Aram, D.: Promoting early literacy via practicing invented spelling: A comparison of different mediation routines. Reading Research Quarterly **48**(3), 221–236 (2013)
5. Mayer, R.E.: Multimedia learning. In: Psychology of learning and motivation, vol. 41, pp. 85–139. Elsevier (2002)
6. Ouellette, G., Sénéchal, M.: Pathways to literacy: A study of invented spelling and its role in learning to read. Child development **79**(4), 899–913 (2008)
7. Ouellette, G., Sénéchal, M.: Invented spelling in kindergarten as a predictor of reading and spelling in grade 1: A new pathway to literacy, or just the same road, less known? Developmental psychology **53**(1), 77 (2017)
8. Paul, R.: Invented spelling in kindergarten. Young Children pp. 195–200 (1976)
9. Pulido, L., Morin, M.F.: Invented spelling: what is the best way to improve literacy skills in kindergarten? Educational Psychology **38**(8), 980–996 (2018)
10. Treiman, R.: Beginning to spell in english. Reading and spelling: Development and disorders pp. 371–393 (1998)
11. VanLehn, K., Ohlsson, S., Nason, R.: Applications of Simulated Students: An Exploration. Journal of Artificial Intelligence in Education **5**(2), 1–42 (1994)
12. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM (JACM) **21**(1), 168–173 (1974)