SVKM's NMIMS

Mukesh Patel School of Technology Management & Engineering

Computer Engineering Department

Program: BTech Integrated, Semester IV

A

Project Report

On

CRUMBS - A Word Guessing Game

Project Report submitted for

Masters of Computer Applications

Submitted by

Aryan Srivastava

under the guidance of

Prof. Sofia Francis

YEAR: 2024-2025

Introduction

Crumbs is an offline word puzzle game inspired by the popular Wordle, offering players the chance to enjoy unlimited word challenges. While Wordle restricts players to a single puzzle per day, Crumbs allows continuous gameplay with an infinite number of words to guess, making it ideal for those who want to play at their own pace without waiting for the next puzzle. Like Wordle, Crumbs requires players to guess a five-letter word within a limited number of attempts, with color-coded feedback for each guess. The game provides a similar engaging experience but adds the freedom of offline play and limitless puzzles, giving users more flexibility and fun.

Algorithms Used

1. List

- **Usage**:

Lists are used to store ordered collections of items, such as words and letters in the game. In Crumbs, a List is employed to manage the game board, where each word and its corresponding letters are stored.

- **Location**:

```
final List<Word> _board = List.generate(
    6, (_) => Word(letters: List.generate(5, (_) => Letter.empty())));
```

This particular List is a collection of six `Word` objects, representing the rows on the game board, with each row containing a five-letter word. The `letters` field in each `Word` object is also a List, initialized with empty `Letter` objects. Lists provide a straightforward way to store and manipulate ordered elements, making them perfect for the grid-based structure of the game.

2. Set

- **Usage:**

A Set is used to store unique items, in this case, the letters that have been pressed on the keyboard. Sets automatically ensure that duplicate letters are not stored, making them ideal for tracking distinct keypresses.

- **Location:**

```
final Set<Letter> _keyboardLetters = {};
```

The Set `_keyboardLetters` holds `Letter` objects, which represent the letters players have pressed during gameplay. Since each letter only needs to be registered once, a Set is efficient in avoiding duplicates and ensuring that only unique letters are tracked.

3. Enum

- Usage:

Enums (Enumerations) are used to define a set of named constants that represent different states or categories in the game. In Crumbs, an enum is used to manage the different game states, such as playing, submitting a guess, or determining whether the player has won or lost.

- Location:

```
enum GameStatus { playing, submitting, lost, won }
```

The `GameStatus` enum provides a clear, structured way to manage the game's state. Each named constant corresponds to a specific phase in the game, making it easy to control and respond to different actions based on the current status, such as allowing new guesses only when the status is `playing`.

4. Map

- Usage:

Maps are used to associate keys with values, allowing for indexed access. In Crumbs, Maps are utilized to convert Lists into Maps, enabling you to access elements based on their index, particularly when rendering the board and the individual letters within it.

- Location:

```
return Column(
  children: board.asMap().map((i, word) => MapEntry(
    i,
    Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: word.letters.asMap().map((j, letter) => MapEntry(
        j,
        FlipCard(
          key: FlipCardKeys[i][j],
          // Other properties...
        ),
      )).values.toList(),
    ),
  )).values.toList(),
);
```

Here, `asMap()` is used to convert a List into a Map, where the indices of the List become keys. This allows you to map each word and letter to its corresponding position in the game board. The

indexed access provided by the Map is essential for operations like rendering the rows and columns of letters in the user interface, ensuring that each element is positioned correctly.

Flow

1. User Launches App:The user taps the app icon on their device.
2. App Initialization:The app displays a splash screen while loading.
3. Main Screen:The user is presented with the main screen of the app, which is the WordleScreen.
4. User Interaction:The user interacts with the app, playing the Wordle game.
5. Feedback and Results:The app provides feedback on the user's guesses and displays results.
6. End of Game:The user can choose to start a new game or exit the app.

Code

Crumbs/models
Letter_model.dart

```
import 'package:equatable/equatable.dart';
import 'package:flutter/material.dart';
import 'package:wordleforgf/app_colors.dart';

enum LetterStatus{initial, notinword,inword,correct}
class Letter extends Equatable{
  const Letter({required this.val,
this.status=LetterStatus.initial});

  factory Letter.empty() =>const Letter(val:"");
  final String val;

  final LetterStatus status;

  Color get backgroundColor {
    switch (status) {
      case LetterStatus.initial:
        return Colors.transparent;
      case LetterStatus.notinword:
        return notinwordcolor;
      case LetterStatus.inword:
        return inwordcolor;
      case LetterStatus.correct:
        return correctcolor;
    }
  }

  Color get borderColor {
    switch (status) {
      case LetterStatus.initial:
        return Colors.grey;
      default:
        return Colors.transparent;
    }
  }
```

```
  Letter copyWith({
    String? val,
    LetterStatus? status,
  }) {
    return Letter(
      val: val ?? this.val,
      status: status ?? this.status,
    );
  }

  @override
  List<Object?> get props => [val,status];

}
```

Word_model.dart

```
import 'package:equatable/equatable.dart';

import 'letter_model.dart';

class Word extends Equatable{
  const Word({required this.letters});

  factory Word.fromString(String word)=>
  Word(letters: word.split('').map((e)=>Letter(val: e)).toList());
  final List<Letter> letters;

  String get wordString => letters.map((e)=>e.val).join();

  void addLetter(String val) {
    final currentIndex = letters.indexWhere((e) => e.val.isEmpty);
    if (currentIndex != -1) {
      letters[currentIndex] = Letter(val: val);
    }
  }

  void removeLetter(){
    final recentLetterIndex = letters.lastIndexWhere((e)=>e.val.isNotEmpty);
    if(recentLetterIndex != -1){
      letters[recentLetterIndex]=Letter.empty();
```

```
    }
  }

  @override
  List<Object?> get props => [letters];

}
```

Crumbs/views
Crumb_screen.dart

```
import 'dart:math';

import 'package:flip_card/flip_card.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:wordleforgf/app_colors.dart';
import 'package:wordleforgf/wordle/model/letter_model.dart';
import 'package:wordleforgf/wordle/widgets/board.dart';
import 'package:wordleforgf/wordle/widgets/keyboard.dart';
import 'package:confetti/confetti.dart';
import '../data/word_list.dart';
import '../model/word_model.dart';

enum GameStatus { playing, submitting, lost, won }

class WordleScreen extends StatefulWidget {
  const WordleScreen({super.key});

  @override
  State<WordleScreen> createState() => _WordleScreenState();
}

class _WordleScreenState extends State<WordleScreen> {
  GameStatus _gameStatus = GameStatus.playing;
  final List<Word> _board = List.generate(
      6, (_) => Word(letters: List.generate(5, (_) => Letter.empty())));
  final List<List<GlobalKey<FlipCardState>>> _flipCardKeys = List.generate(6,
      (_) => List.generate(5,
        (_) => GlobalKey<FlipCardState>(),
```

```
  ),
 );
 int _currentWordIndex = 0;

 Word? get _currentWord =>
    _currentWordIndex < _board.length ? _board[_currentWordIndex] : null;

 Word _solution = Word.fromString(
    fiveLetterWords[Random().nextInt(fiveLetterWords.length)].toUpperCase());
final Set<Letter> _keyboardLetters={};
      final ConfettiController _confettiController = ConfettiController(duration: const Duration(seconds: 1)); // Create a ConfettiController

 @override
 void dispose() {
  _confettiController.dispose(); // Dispose the controller
  super.dispose();
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title:
      Text("CRUMBS",
        style: TextStyle(
        fontSize: 24,
        fontWeight: FontWeight.bold,
        letterSpacing: 3
      ),),
      centerTitle: true,
      backgroundColor: Colors.transparent,
      elevation: 0,

    ),
    body: Stack(
      children: [
        Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
```

```
          SizedBox(height: 4,),
          Board(board: _board, FlipCardKeys: _flipCardKeys,),
          Spacer(),
          Keyboard(
            onKeyTapped: _onKeyTapped,
            onDeleteTapped: _onDeleteTapped,
            onEnterTapped: _onEnterTapped,
            letters: _keyboardLetters,
          )
        ],
      ),
      Align(
        alignment: Alignment.topCenter,
        child: ConfettiWidget(
          confettiController: _confettiController,
          blastDirection: pi / 2,
          shouldLoop: false,
          numberOfParticles: 100,
          colors: const [Color(0xff065E39),
          Color(0xff179769),
          Color(0xff62FFC7),
          Color(0xff30CA89),
          Color(0xff217803),
          Color(0xff308B39),
          Color(0xffA1FA82),
          Color(0xffA1FA82),
            Color(0xff6ACD07),
            Color(0xff5AAF45),
            Color(0xff30CA89),


          ],
        ),
      ),
    ],
  ),
 );
}

void _onKeyTapped(String val) {
```

```
  if (_gameStatus == GameStatus.playing) {
    setState(() {
      _currentWord?.addLetter(val);
    });
  }
}

void _onDeleteTapped() {
  if (_gameStatus == GameStatus.playing) {
    setState(() {
      _currentWord?.removeLetter();
    });
  }
}

Future<void> _onEnterTapped() async{
  if (_gameStatus == GameStatus.playing &&
      _currentWord != null &&
      !_currentWord!.letters.contains(Letter.empty())) {
    _gameStatus = GameStatus.submitting;

    for (var i = 0; i < _currentWord!.letters.length; i++) {
      final currentWordLetter = _currentWord!.letters[i];
      final currentSolutionLetter = _solution.letters[i];

      setState(() {
        if (currentWordLetter == currentSolutionLetter) {
          _currentWord!.letters[i] =
              currentWordLetter.copyWith(status: LetterStatus.correct);
        } else if (_solution.letters.contains(currentWordLetter)) {
          _currentWord!.letters[i] =
              currentWordLetter.copyWith(status: LetterStatus.inword);
        } else {
          _currentWord!.letters[i] =
              currentWordLetter.copyWith(status: LetterStatus.notinword);
        }
      });
      final letter = _keyboardLetters.firstWhere(
          (e) => e.val == currentWordLetter.val,
        orElse: () => Letter.empty(),
```

```
      );

      if (letter.status != LetterStatus.correct) {
        _keyboardLetters.removeWhere((e) => e.val == currentWordLetter.val);
        _keyboardLetters.add(_currentWord!.letters[i]);
      }
      await Future.delayed(
        const Duration(milliseconds: 300),
          ()=> _flipCardKeys[_currentWordIndex][i].currentState?.toggleCard(),
      );
    }
    _checkIfWinOrLoss();
  }
}

void _checkIfWinOrLoss() {
  if (_currentWord!.wordString == _solution.wordString) {
    _gameStatus = GameStatus.won;
    _confettiController.play(); // Trigger the confetti animation
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        dismissDirection: DismissDirection.none,
        duration: const Duration(days: 1),
        backgroundColor: correctcolor,
        content: const Text(
          'You won!',
          style: TextStyle(color: Colors.white),
        ),
        action: SnackBarAction(
          onPressed: _restart,
          textColor: Colors.white,
          label: 'New Game',
        ),
      ),
    );
  } else if (_currentWordIndex + 1 >= _board.length) {
    _gameStatus = GameStatus.lost;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        dismissDirection: DismissDirection.none,
```

```
        duration: const Duration(days: 1),
        backgroundColor: Colors.redAccent[200],
        content: Text(
          'You lost! Solution: ${_solution.wordString}',
          style: const TextStyle(color: Colors.white),
        ),
        action: SnackBarAction(
          onPressed: _restart,
          textColor: Colors.white,
          label: 'New Game',
        ),
      ),
    );
  } else {
    _gameStatus = GameStatus.playing;
  }
  _currentWordIndex += 1;
}

void _restart() {
  setState(() {
    _gameStatus = GameStatus.playing;
    _currentWordIndex = 0;

    _board.clear();
    _board.addAll(
      List.generate(
        6,
        (_) => Word(letters: List.generate(5, (_) => Letter.empty())),
      ),
    );

    _solution = Word.fromString(
      fiveLetterWords[Random().nextInt(fiveLetterWords.length)].toUpperCase(),
    );
                                                    _flipCardKeys..clear()..addAll(List.generate(6,
(_)=>List.generate(5,(_)=>GlobalKey<FlipCardState>())));
    _keyboardLetters.clear();
  });
}
```

}

Crumbs/data
Word_list.dart

```
const List<String> fiveLetterWords=
["About","Alert","Argue","Beach","Above","Alike","Arise","Began","Abuse",
  "Alive","Array","Begin","Actor","Allow","Aside","Begun","Acute","Alone",
  "Asset","Being","Admit","Along","Audio","Below","Adopt","Alter","Audit",
  "Bench","Adult","Among",]
```

Crumbs/widgets
Board.dart

```
import 'package:flip_card/flip_card.dart';
import 'package:flutter/cupertino.dart';
import 'package:wordleforgf/wordle/model/word_model.dart';

import '../model/letter_model.dart';
import 'board_title.dart';

class Board extends StatelessWidget {
  final List<Word> board;
  final List<List<GlobalKey<FlipCardState>>> FlipCardKeys;

  const Board({Key? key,
  required this.board,required this.FlipCardKeys}):super(key: key);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: board.asMap().map((i, word) => MapEntry(
        i,
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: word.letters.asMap().map((j, letter) => MapEntry(
            j,
            FlipCard(
              key: FlipCardKeys[i][j],
              flipOnTouch: false,
```

```
            direction: FlipDirection.VERTICAL,
            front: Boardtitle(
              letter: Letter(
                val: letter.val,
                status: LetterStatus.initial,
              ), // Letter
            ), // BoardTile
            back: Boardtitle(letter: letter),
          ), // FlipCard
        )).values.toList(),
      ),
    )).values.toList(),
  );
 }
}
```

Board_title.dart

```
import 'package:flutter/cupertino.dart';
import 'package:wordleforgf/wordle/model/letter_model.dart';

class Boardtitle extends StatelessWidget {
 final Letter letter;
 const Boardtitle({super.key, required this.letter});

 @override
 Widget build(BuildContext context) {
  return Container(
    margin: EdgeInsets.all(4),
    height: 60,
    width: 60,
    alignment: Alignment.center,
    decoration: BoxDecoration(
      color: letter.backgroundColor,
      borderRadius: BorderRadius.circular(4),
      border: Border.all(color: letter.borderColor),
    ),
    child: Text(
      letter.val,
      style: TextStyle(
        fontWeight: FontWeight.bold,
```

```
          fontSize: 32,
        ),
      ),
    );
  }
}
```

Keyboard.dart

```
import 'package:flutter/material.dart';

import '../model/letter_model.dart';

const _qwerty=[
  ["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"],
  [ "A", "S", "D", "F", "G", "H", "J", "K", "L"],
  ["ENTER","Z", "X", "C", "V", "B", "N", "M", "DEL"],
];
class Keyboard extends StatelessWidget {
  final void Function(String) onKeyTapped;
  final VoidCallback onDeleteTapped;
  final VoidCallback onEnterTapped;
  final Set<Letter> letters;


  const Keyboard({super.key,
  required this.onKeyTapped,
    required this.onDeleteTapped,
    required this.onEnterTapped,
    required this.letters,
  });

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: _qwerty.map((keyRow) => Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: keyRow.map((letter) {
          if(letter == "DEL"){
            return _KeyboardButton.delete(
```

```
            onTap: onDeleteTapped,
          );
        }
        else if(letter == "ENTER"){
          return _KeyboardButton.enter(onTap:onEnterTapped);
        }
        final letterKey = letters.firstWhere(
            (e)=> e.val == letter,
          orElse: () => Letter.empty()
        );
        return  _KeyboardButton(
          onTap:()=>onKeyTapped(letter),
          letter: letter,
          backgroundColor: letterKey != Letter.empty()
              ?letterKey.backgroundColor
              :Colors.grey,
        );
      }).toList(), // Convert Iterable to List
    )).toList(),
   );
  }
}

class _KeyboardButton extends StatelessWidget {
  final double height;
  final double width;
  final VoidCallback onTap;
  final Color backgroundColor;
  final String letter;

  const _KeyboardButton({super.key,
  this.height=52,
  this.width=32,
  required this.onTap,
  required this.backgroundColor,
  required this.letter,
  });

  factory _KeyboardButton.delete({
    required VoidCallback onTap,
```

```
}) =>
    _KeyboardButton(
      width: 56,
      onTap: onTap,
      backgroundColor: Colors.grey,
      letter: 'DEL',
    );

factory _KeyboardButton.enter({
  required VoidCallback onTap,
}) =>
    _KeyboardButton(
      width: 56,
      onTap: onTap,
      backgroundColor: Colors.grey,
      letter: 'ENTER',
    );


@override
Widget build(BuildContext context) {
  return Padding(padding: EdgeInsets.symmetric(
    vertical: 3,horizontal: 2
  ),
    child: Material(
      color: backgroundColor,
      borderRadius: BorderRadius.circular(4),
      child: InkWell(
        onTap: onTap,
        child: Container(
          height: height,
          width: width,
          alignment: Alignment.center,
          child: Text(
            letter,
            style: const TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.w600,
            ),
          ),
```

```
            ),
          ),
        ),
      );
    }
}
```

Output

In Crumbs, the use of color is essential for providing feedback to players based on their guesses, making the game both intuitive and engaging. Each color has a specific meaning, helping players understand the correctness and positioning of their guesses:

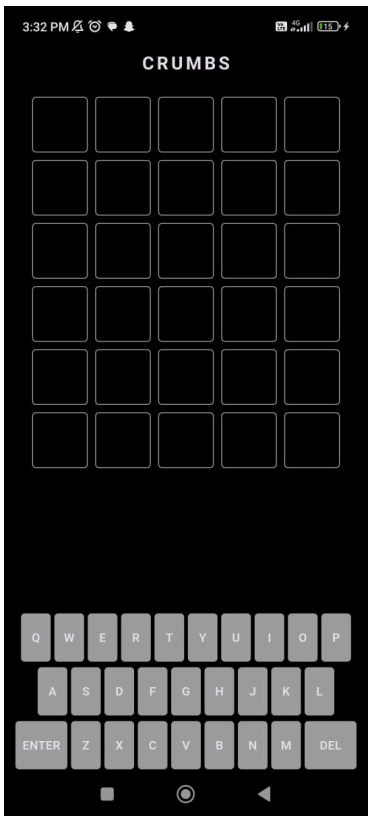
1. Green :
   - Meaning : A green tile indicates that the guessed letter is both correct and in the correct position within the word. This feedback confirms to the player that the letter is exactly where it should be.
   - Usage : Green tiles appear after a guess when a letter matches both in identity and placement with the target word.

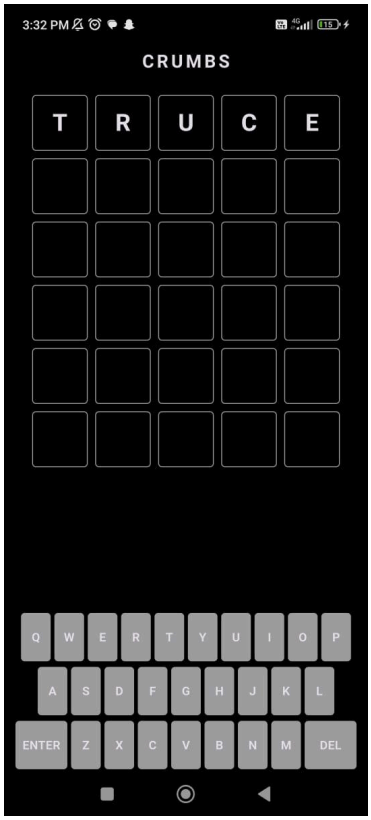
2. Yellow :
   - Meaning : A yellow tile signifies that the guessed letter is correct but in the wrong position within the word. While the letter exists in the word, it is not placed where the player guessed it.
   - Usage : This color helps players refine their guesses, as it confirms the letter's presence but suggests it needs to be moved to a different position.

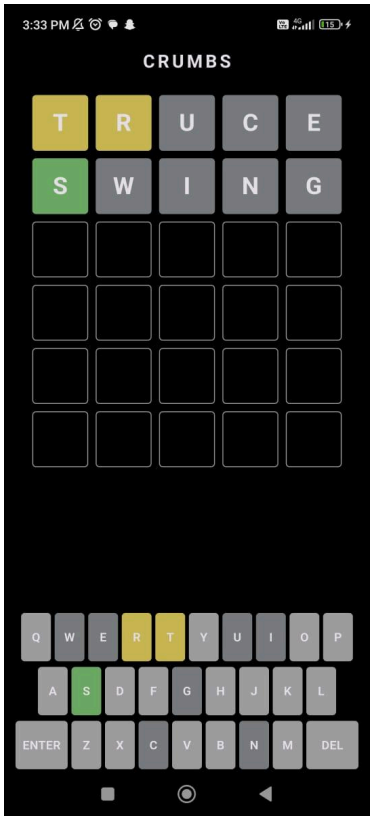
3. Greyed-out :
   - Meaning : A greyed-out tile indicates that the guessed letter does not exist anywhere in the target word. This feedback helps players eliminate letters that are not part of the solution, guiding their future guesses.
   - Usage : Grey tiles appear after a guess when a letter is not found in the word at all, providing clear elimination feedback.

4. Light Grey (or default):
   - Meaning : Light grey tiles represent letters that have not yet been guessed. These serve as placeholders for future guesses, helping players visualize which letters they still need to consider.
   - Usage : At the beginning of each round, the board consists of light grey tiles, indicating empty slots where guesses will be placed.

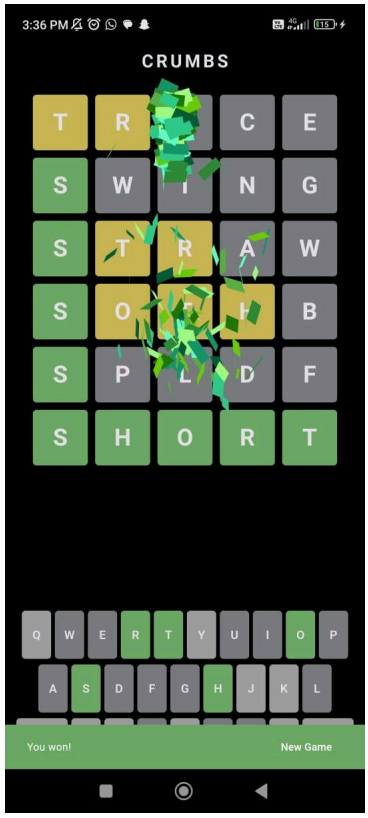
These colors are crucial for guiding players through the game, offering visual clues that progressively narrow down the possible correct word while also making the experience more engaging and intuitive.

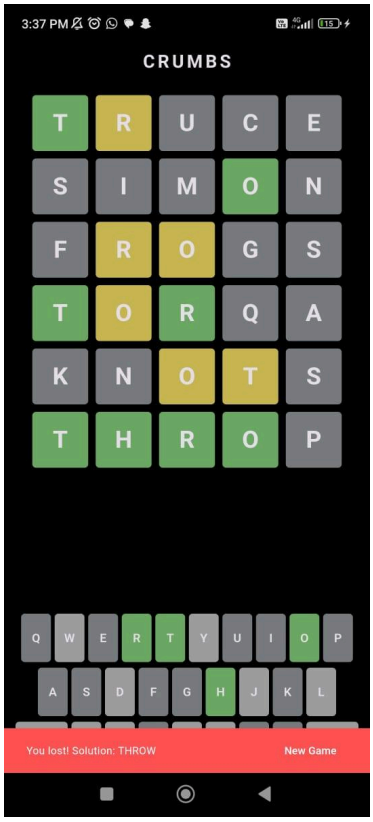

3:32 PM
CRUMBS
Q W E R T Y U I O P
A S D F G H J K L
ENTER Z X C V B N M DEL


3:32 PM
CRUMBS
T R U C E
Q W E R T Y U I O P
A S D F G H J K L
ENTER Z X C V B N M DEL


3:33 PM
CRUMBS
T R U C E
S W I N G
Q W E R T Y U I O P
A S D F G H J K L
ENTER Z X C V B N M DEL


3:33 PM
CRUMBS
T R U C E
S W I N G
S T R A W
Q W E R T Y U I O P
A S D F G H J K L
ENTER Z X C V B N M DEL


3:36 PM
CRUMBS
T R U C E
S W I N G
S T R A W
S O U N B
S P L A D F
S H O R T
Q W E R T Y U I O P
A S D F G H J K L
You won!
New Game


3:37 PM
CRUMBS
T R U C E
S I M O N
F R O G S
T O R Q A
K N O T S
T H R O P
Q W E R T Y U I O P
A S D F G H J K L
You lost! Solution: THROW
New Game