Shubhit Mohan Singh

shubhits

15-112 Term Project Assignment

**Design**

Introduction

My intention was to build a word-search game that would work effectively with Python and Tkinter. I looked into various pre-existing word-search games that existed online, and I came across this:

<http://games.yahoo.com/game/word-scramble-ii-flash.html>

However, I decided that the full algorithmic capabilities of python would not be effectively used in trying to solve a board with such few letters. As such, I looked towards more complicated word search games, and came across Boggle (explained here: <http://en.wikipedia.org/wiki/Boggle> ). Boggle is a physical game that is played using a 4 by 4 board of letters, where the player is tasked with finding as many words as possible on that board by connecting adjacent letters to one another. I decided to bring this game from the physical world to the virtual world.

My Design

I experimented with a few approaches to this game before deciding on one that I found best. First, I wrote the game such that once the user clicks start, the program randomly assigns letters to the board. When the user enters a word while playing the game, the program checks that:

1. The word is longer than 3 letters
2. The word has not been made before
3. The word exists on the board; every letter is adjacent to the previous letter (above, below, left, right or diagonally)
4. The word exists in the dictionary

This worked fine, but it had a number of disadvantages. Firstly, a lot of computation had to be done whenever the user entered a word. Secondly, there would be no way for the user to know, once the game has ended, what words he missed out or he could have made on the board.

As a result, my actual design adopts a slightly different approach. After randomly assigning letters to the board, the program automatically and recursively solves the board. It goes through every possible combination of letters that can be formed on the board as long as the string of letters is a prefix of another word that exists on the dictionary, and it checks if each combination of letters exists in the dictionary. As a result, before the user even begins playing the game, the program has a list of every possible solution on the board. Whenever a user enters in a word, the program just has to check whether or not that word is in the list of solutions.

This approach also helps with the end-of-game page. Once the game is over, the user is able to see a list of all the possible solutions that can be made on the board, with the solutions that he/she did make highlighted in green. During testing, I found this aspect to be highly beneficial for the overall user experience. The user gets the opportunity to reflect and review on his/her game performance, and learn to get better by playing more games.

My Code

My code is separated into two python documents: Graphics.py and Boggle.py.

Graphics.py lays the foundation for all the code in Boggle.py. It sets up the canvas and the various classes required for the creation of graphics and animation in Tkinter. Through Graphics.py, Boggle.py communicates with Tkinter.

Boggle.py contains the class BoggleEvents, which is a subclass of BoggleGraphics in Graphics.py. This is the main event-handler of the program. It is where mousePressed, keyPressed and timerFired are defined.

Boggle.py is separated into 5 sections:

* Overall functions
* Launch screen functions
* Loading screen functions
* Game screen functions
* End-of-game screen functions

The first section, Overall functions, distributes the events from the user to the various functions in each of the other 4 sections, depending on which state the game is in. The game has 4 states, separated into the other 4 sections, which are a reflection of the 4 main screens of the user interface (UI).

Each of the other 4 sections has an init(), mousePressed(), keyPressed() and timerFired() in order to handle events assigned to them in the first section of the code. They each also have a redrawAll() function under the BoggleGraphics class in the Graphics.py file.

Key functions:

* solveBoard and solvePiece: recursively solves 4 by 4 board before the game starts
* isPrefix: Instead of searching through every single possibility on the board, the above recursive function stops calling itself once it determines that the current string of letters is not a prefix for any other word in the English dictionary. By doing this, the function is saving time and improving efficiency without sacrificing any possible solutions on the board.
* keyPressedGame: Handles the key presses of the user when playing the game in order to determine and store each word that the user types in. It also scrolls up and down following inputs from the user. I created my own scroll function in order to implement the scroll feature.
* timerFiredGame: Handles the animation of the board that occurs every time a hint is given or a correct word is formed.
* createBoard: randomly creates a new board at the start of every game. The likelihood of each letter of the alphabet appearing is not the same. Rarer letters like Z and J are less likely to appear than letters like A and E. This feature was modeled after Scrabble, which has larger occurrences of letters like A and E and fewer occurrences of letters like Z and J.
* scrabblePoints and addPoints: These two functions dictate the scoring for the game. Each letter has a different number of points associated with it. Rare letters like Z and Qu are worth 10 points each, while frequent letters like E and I are worth 1 point each. To score each word, first, the function addPoints sums up the value of each letter in the point. Then depending on the length of the word, more bonus length points are awarded.
* isLegal: determines whether or not a particular word entered is legal. If it is, points will be added.
* generateHint: This function provides the user with a hint of a word that can be formed on the board every time the user hits ‘1’. The user has 3 hints in each game. In order to generate a hint, this function randomly selects a solution from the list of solutions.
* showPoints: at the end of the game, this function calculates the score for each solution that the user clicks on and displays that score.

In order to implement this game, I use classes extensively. Some important classes I create in order to design the game much more easily include MakeButton, Text, Solution and Location. These classes are defined in the file Graphics.py.

User Interface

The user interface consists of 4 main screens.

1. Launch Screen

This is the screen that appears when the program is run. It contains a background, a title, and a start button.

1. Loading Screen

This screen contains some basic instructions on what the game is, and how it is played. It consists of an 'initialize game’ button that that user clicks while reading the help information. This is to allow the user to have a basic idea of what to expect and how to go about playing the game while the game loads and the solvePiece function recursively solves the board. Once the game is ready, a ‘Start’ button appears.

1. Game screen

This is the screen where most of the gameplay occurs. The screen is divided into two main sections. On the left, the board is displayed. The pieces are drawn big in size because that is a focus of the game. The answers box is located on the right. This is where the user can type in his/her answers. Once the user hits enter, the word that was keyed in appears on the screen and gets added to the answers. If the user keys in something wrongly, he/she can hit ‘backspace’, and this will erase whatever he/she typed. The user can then re-type the answer and hit enter.

Below the 4 by 4 board, there are 4 game control buttons: ‘End Game’, ‘Restart’, ‘Pause’ and ‘Help’. By clicking ‘End Game’, the user can end the current game and go directly to the end of game screen. By clicking ‘Restart’, the user will be redirected to the loading page where a new game will load. By clicking ‘Help’, the game will be paused, and a popup screen will appear displaying basic instructions. By clicking ‘Help’ again, the popup will disappear and the game will continue. By clicking ‘Pause’, the game will be paused. By clicking ‘Pause’ again, the game will resume. During both ‘Pause’ and ‘Help’, I made it a point to cover up the board and the answers box, so that the user cannot just pause the game and take as much time as needed to determine all the solutions to the board before resuming again.

On the right of the buttons, there is a ‘Feedback’ panel that gives the user comments every time a correct/incorrect word is made. Below this, the time remaining and score are also displayed.

Below the answers box, there are two boxes that display the powerups. The first powerup is hint. The user has 3 hints. When the user presses ‘1’ on the keyboard, the board will show an animation of the hint of a word that can be made. The second powerup is a ‘Freeze’ feature. By pressing ‘2’ on the keyboard, the time will freeze and the user can continue to make as many words as possible.

1. End of Game screen

In this screen, the ‘Feedback’ panel states that the game is over. The final score is displayed on the score panel. In the answers box, all the possible solutions are displayed. The user can scroll through all the answers by pressing Up or Down on the keyboard. If the user clicks on a particular answer, the pieces, or tiles, that need to be used to form that answer will light up in green on the board or the left, so that the user knows how that word can be formed on the board. Also, the score of that particular answer is displayed to the right of that answer.

From this page, the user has the option to restart the game, which will take the user to the Loading screen where a new game will be initialized.

Further Enhancements

This game can be enhanced even further to make it 2-player. I feel that this will add a lot of fun to the game, as having two players compete against each other to find the most number of words on the same board in a limited time would be very entertaining. This is something I may look to implement in the future.