WordTwist Lab

Provided Code:

The WordTwistModel Interface – the model class will/must support the following functions:

Functions which get information (ie for use in the View):

char getUnusedLetter( int index ) – returns the letter at index (0-5) that hasn’t been used yet or

‘ ’ if that letter has already been used to form the current word

String getCurWord() – returns the current word being formed by the user

int getWordBankSize() – returns the number of words in the word bank

String getWordBankWord( int index ) – returns the word bank word at index

boolean isWordBankWordFound( int index ) – returns true if the word at index has been found

by the user, false otherwise

int getTime() – returns the amount of time left in the round in milliseconds

int getScore() – returns the current score

boolean roundOver() – returns true if the round is over (ie all words have been found or the

time limit for the round has elapsed)

boolean gameOver() – returns true if the game is over (ie the current round finished without the

user finding at least one 6 letter word).

Functions which change the state (ie for use by the Control):

void useLetter( int index ) – moves the letter at the given index from the unused letters to the

end of the current word

void unuseLetter( int index ) – moves the letter at the given index from the current word to the

first empty space in the unused letters

void submit() – submits the current word. If the word is in the word bank and hasn’t already

been found, the score should be increased (50/75/100/200 pts for words of length 3/4/5/6). Regardless, the current word is reset so that all tiles are unused

void reset() – resets all letters in the current word to the unused letters

void scrambleUnusedLetters() – scrambles all letters in the unused letters

void reduceTime( int dt ) – reduces the time left in the round by dt milliseconds

void startNewRound() – picks a new 6 letter word, gets a new word bank, resets the round time,

and, if the game was over, the score.

class WordTwistModelUtility

This class implements a static function which will be useful to the Model:

ArrayList<String> getPermutations( String s ) – returns a list of all the possible permutations of

the letters of s. For example, if s is “BAA”, getPermutations will return a list containing “B”, “A”, “BA”, “AB”, “AA”, “BAA”, “ABA”, and “AAB”.

And several functions which you need to implement:

ArrayList<String> readDictionary() – reads all the words from the file WordTwist Words.txt and

puts them in a list of Strings.

ArrayList<String> getSixLetterWords( ArrayList<String> dictionary ) – returns a list of all the six

letter words in the dictionary

String chooseRandomSixLetterWord() – returns a random 6 letter word from the list of six letter

words

ArrayList<String> getHiddenWords( String s ) – returns a list of the legal English words which can

be formed from s.

class WordTwistDummyModel implements WordTwistModel

This class implements all the methods of WordTwistModel… badly. The Control and View parts

can use an instance of this class to test against while the actual Model is being written.

class WordTwistTextView and WordTwistTextControl

These classes use the an instance of WordTwistModel to display and manipulate the state of the

game. The Model programmer can use this class to test his methods while the actual View and

Control classes are being written.

class WordTwist

This class contains the main method and four other methods:

testControlView – creates a dummy model, and a control and view object and a window and

runs them.

testModel – creates a model and a text-based control and view and runs them.

testModelUtility – tests the functions in WordTwistModelUtility

testViewInfo – tests the info functions on the view object

finalIntegration – creates a model, control, view, and window and runs the final program

The main function allows you to select which testing function you would like to run. When you are finished implementing both the C/V and M parts, you can rewrite the function so that it calls finalIntegration instead.

Tasks to complete:

Model

1. Implement the functions in WordTwistModelUtility.

The instructions for each function are in the comments of WordTwistModelUtility

As you complete each function, run the Model Utility tester to see if your implementation is correct.

1. Implement class WordTwistModelImpl

Add your private variables and a constructor. Your constructor should not take any arguments

Have the class implement interface WordTwistModel, and then implement each of the functions. Basic instructions for each function are on page 1 above and also in the in the comments for WordTwistModel.

After you’ve implemented the information getting functions, you should be able to run the model tester and see the basic layout. There are some instructions regarding uncommenting in the testModel function. After you follow them and run, you should see:

==========================================================

\_ \_ \_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

----------------------------------------------------------

MOBBED 0

==========================================================

Next, implement the functions useLetter and unuseLetter. Now, when you run the tester, you’ll be able to use and unused letters. To use a letter, type its index (0...5) and press enter. For example, after entering 0 (M) and 2 (B):

==========================================================

\_ \_ \_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

----------------------------------------------------------

MB

O BED 0

==========================================================

To unuse a letter, type –index. For example, after -1 (B):

==========================================================

\_ \_ \_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

----------------------------------------------------------

M

BO BED 0

==========================================================

Next, implement the reset function. To test reset, type ‘r’ in the test program. All used letters should revert to the unused line.

Next, implement the submit function. To test submit, use letters until you have a valid word and then type ‘su’. You should see the word appear in the word bank and all letters reset. For example, after using letters 0 (M), 1 (O), 2 (B) to form MOB, and typing ‘su’:

==========================================================

\_ \_ \_ \_ \_ \_

\_ \_ \_

M O B

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

----------------------------------------------------------

MOBBED 50

==========================================================

Also, make sure that the score is correct. Submit several words, including valid and invalid words. Also, submit the same word twice and make sure that it is only scored once.

Next, implement scrambleUnusedLetters. To scramble, type ‘sc’ in the test program. Try this with all of the letters unused, and also with some letters used and some unused:

==========================================================

\_ \_ \_ \_ \_ \_

\_ \_ \_

M O B

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_

\_ \_ \_ \_

\_ \_ \_

\_ \_ \_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

\_ \_ \_ \_

----------------------------------------------------------

BMEBOD 50

==========================================================

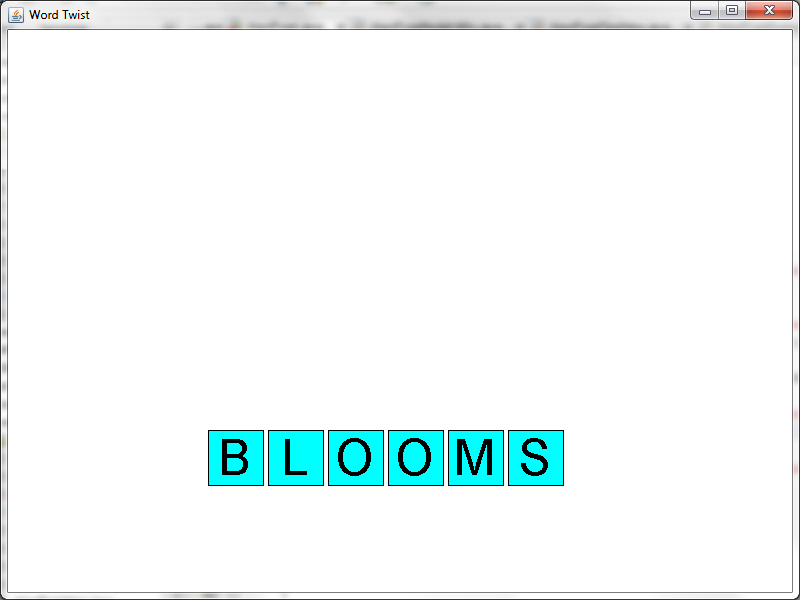
Unfortunately, the text view program can’t test the timer functions or the round/game over or the nextRound functions… you’ll have to wait for integration.

Control/View

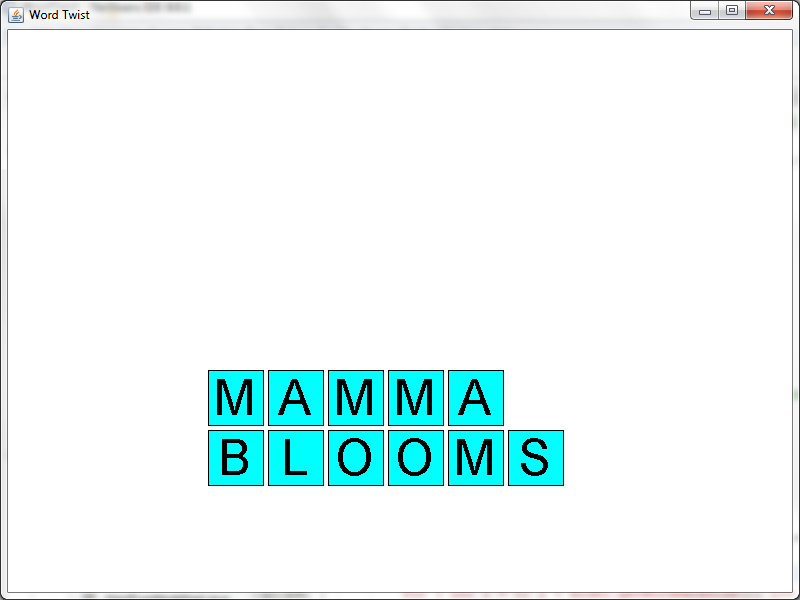
1. Begin by implementing the view functions of the View class, WordTwistGUIView. It needs a constructor which takes no arguments. You will need two variables of type Font: one to display large text (like the used and unused letters) and one to display small text (like the found words). You can obtain these objects by calling the getFont function in the GraphicsUtilityFunctions class. Your class should NOT have any other private (or public) variables.

WordTwistGUIView should implement AnimatedObject and its functions update and paint. The update function should do nothing. The paint method should use the functions that the model provides (see interface WordTwistModel) to get information about the current state of the program which should then be displayed.

Start with the unused letters. You can get them using the getUnusedLetter function. Each letter should be displayed in a square. To do this, you need to draw a square, and then use the drawStringWithFontInRectangle function to center text within the square. When you have finished this, you should run the tester. You’ll need to follow the instructions for uncommenting regarding the view class (leave the control stuff commented out). Run the tester and select (V)iew/Control. You should see this:

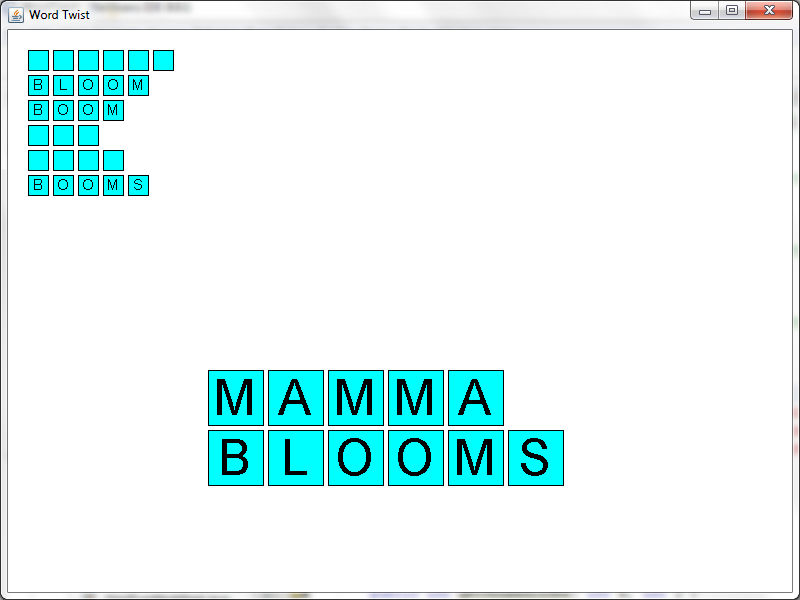


Next, display the used letters. You can obtain the current word from the model as a String. You’ll need to display each letter individually above the used letters. When you are done, test again and you should see:

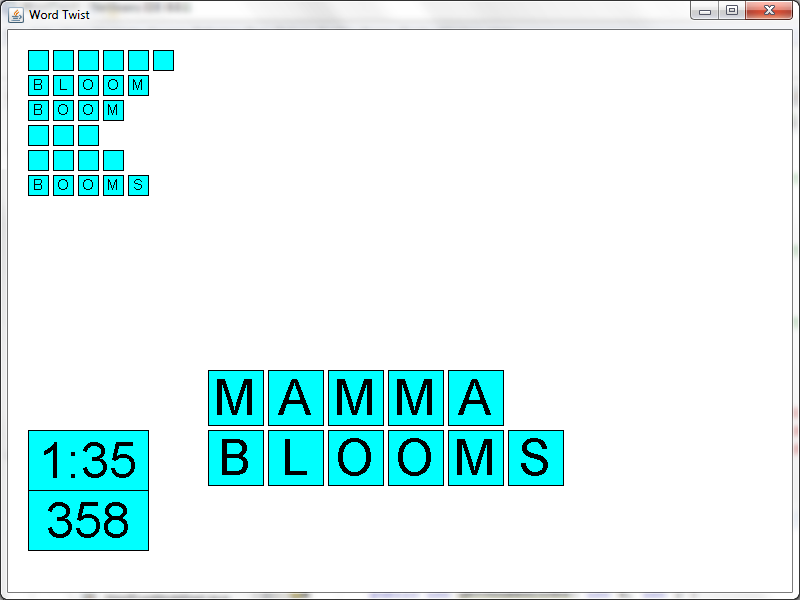


Note that this is an impossible state for the program… the model that you are coding against doesn’t really work, but it provides enough functionality for you to see if your view is working properly.

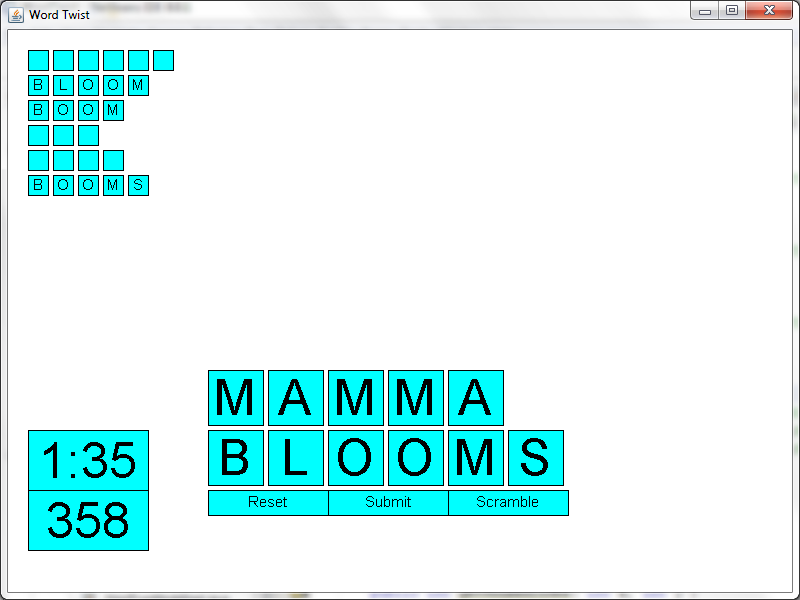
Next, use the model to get the word bank words. Some of them are hidden. You should only display blanks for them. Others are revealed. You should display these words. You can ask the model how many words there are, what each word is, and whether it has been found. When you are done, test and see this:



Next, display the score and time left. Again, you can obtain the raw information from the model and then display it:



Finally, draw the buttons. You shouldn’t need to talk to the model to do this:



1. Now you’re ready to implement some utility functions which detect where on the screen a point is. Each of these functions (getUsedIndex, getUnusedIndex etc) takes x and y coordinates representing a click on the screen. They return the index of the letter clicked or -1 if a letter is not clicked. The other functions test to see if the click is in one of the buttons. Implement each one. When you have done so, you can test them by following the uncomment instructions in the testViewInfo function. Run the tester, selecting View (I)nfo option to see if you’ve done it properly.
2. Next are the control functions.

The handleTimePassage method gets called to indicate that time is passing. Do something appropriate here. When you are ready to test, you’ll need to follow some uncomment instructions in testControlView. If you’ve done it properly, you’ll see a printed message indicating so.

The handleMouseClick method handles a mouse click. You’ll be passed a model object and a MouseEvent object which contains x and y coordinates of the click. You need to figure out what exactly (letter, button etc) was clicked on, and then call the appropriate function in the model. If you’ve done it correctly, then clicking will result in an appropriate message being printed to the console.

Shared Integration

Once you have implemented the control, the view and the model, you’re ready to integrate. There are some uncomment instructions in main. Follow them to get the whole program up and running. Test and fix any problems.

Also, note that you did not test the timer, round end, and new round functionality in the model or the buttons that correspond to these situations in the view. Make sure you work through these test cases.

Congrats! You have finished the WordTwist Lab!