Features for Release C

Grouping Found Words by Regular vs Bonus:

To implement this feature, we will use separate displays for the list of regular words and the list of bonus words using the factory method. We will turn the WordHuntWordsFoundView class into a WordHuntWordsFoundViewFactory and add WordHuntRegularWordsFoundView and WordHuntBonusWordsFoundView classes as children, changing the few places where the WordHuntWordsFoundView class is used. This approach is easy to implement and gives a lot of flexibility for how we want to group words in the future.

Count Number of Words Left to be Found in each Category:

We will use the factory method again in order to split ScoreView and ScoreViewModel into versions for regular/bonus words. With this, only the factories must be changed in order to calculate and display the appropriate numbers. The same remarks about flexibility apply here, although it is a bit awkward to make so many factories with only two children each.

Showing Letters of not yet Found Words:

To implement this feature, we will create classes WordHuntHintsView and WordHuntHintsViewModel. The latter will keep track of all not yet found words along with, for each one, some of their letters which should be replaced by asterisks, with the former displaying this. When the user finds a word, it will be removed from consideration and a random letter that was previously replaced by an asterisk will be revealed, although we may change when letters

are revealed in a different version. This feature does not seem to be easily implemented using the current data structures for our program.

Accuracy:

This can be solved by changing the (regular) score view so it displays the accuracy and the (regular) score viewmodel so it calculates accuracy and sends to the view. This change is easy given our design.

Score Dependent on Length:

As a regular word is currently precisely a four letter word in our program, it would first be necessary to change when a word is viewed as regular. We will make it so that a word being regular is determined by rarity and not length, changing the word lists accordingly. With this out of the way, small changes to WordHuntScoreView and WordHuntScoreViewModel (and consequently to WordHuntBoardViewModel) can make it so that these methods factor word lengths into the score. While our decision to call regular words 4-letter words in WordHuntWords leads to some extra work when changing what a regular word is, our organization as a whole means these changes are not difficult to implement.

Rotating Boards:

The key step in allowing the user to rotate the board will be to use the State design pattern. In the model, we will add an interface WordHuntBoardState, a class WordHuntBoardContext and classes WordHuntBoard<Direction>State where direction is one of the four cardinal directions, with the direction displaying where an upward pointing arrow would point after rotation.

WordHuntBoardState will have abstract methods rotateClockwise and rotateCounterclockwise and concrete methods for getting and setting the values of the iterated array lists in wordHuntGame depending on the state. For example, if the state is East and one tries to get what is visually the element of the 1st row and the 3rd column, they will get what is internally the element of the 0th row and the 1st column. These new methods will replace the instances where the iterated array lists are directly modified in WordHuntGame. After this is added, it remains to create classes in the view and viewmodel to handle the buttons which allow the rotate the board clockwise/counterclockwise and connect the latter to WordHuntBoardState. In all, allowing the user to rotate the board requires a lot of changes in our program. This approach was chosen, however, because it is very modular, with WordHuntGame as the only previously designed class it interacts with.

Grids Where Some Cells may not Have a Tile:

On the internal side, we will add a variable excludedCells, which is an ArrayList of Pairs of Integers, to WordHuntGame. Following the design invariant that we will never pass indices of excludedCells to the viewmodel, we can modify findWordsHelper in WordHuntWords so it ignores words which include excluded tiles. WordHuntBoardView, however, will access excludedCells in order to gray out cells at these indices. This is all doable, but large additions would be needed to allow the user to select which cells they want to have tiles.