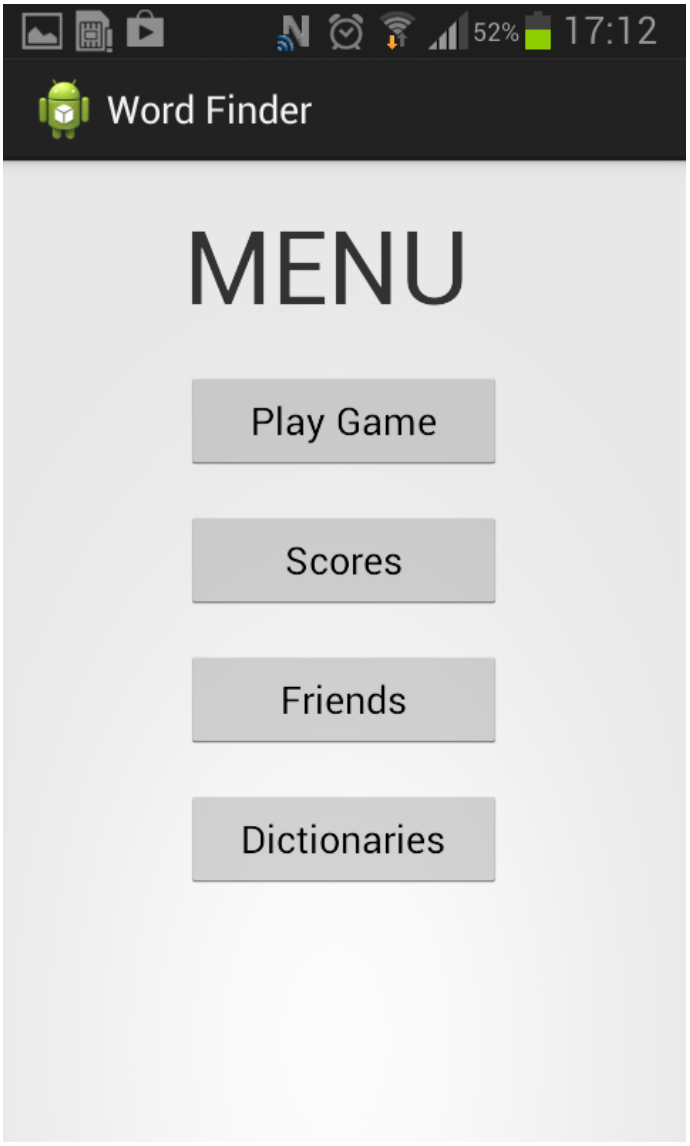
Review of Team 4

By Team 1



WordFinder game

ESE – HS2013

1.- Introduction and First Impressions

As one first opens the application, the first thing you see is the main menu, which contains its title and a series of buttons to the available features that the app offers.

The design itself doesn't seem very customized, since even the launcher icon itself is the default one. Some of the features remain only half implemented.

If you click on the score button, you get an empty page with 'Score' as its title. Even after you complete a game, this page remains empty, not showing you your previous scores.

The Friends feature is not only incomplete, but it makes the application crash whenever you try to add one. There is also a delete radio button, whose action is not very clear. After clicking it, the application also crashes.

The dictionary feature is a bit more complete, in the sense that there is some text and buttons available, but still, half implemented. The user is not able to add or delete neither regular nor custom dictionaries, or even to see the default or included ones.

However, the main and most important feature is fully functional, which is the word finder game itself. The timer works unsuccessfully together with the pause button, not allowing the user to take a break during a stressful situation (you have only a few seconds to beat your old high-score), or whenever he pleases. If you click the pause button, you get another screen with a message saying that the game is paused, but what a surprise you get when you click the resume button and return to your game to see that the timer really never stopped.

After the timer is up though, you still get to see what your score was, in a new screen with the title 'AfterGame'. A "return to main menu" button also appears, which is not misleading as the "Pause" button and really redirects you to the Main Menu.

2.- Design

1. Violation of MVC layers
   - We've not found any implementation of MVC
   - Views are handled by new activities
2. Usage of helper objects between view and model
   - Yes, for example in *DataBaseHelper*

3. Rich OO domain model
    - *MainMenu* controls navigation in program
    - Clear OO implementation of *MainMenu* by button-views
4. Clear responsibilities
    - *BoardFactory* controls creation of different types of boards
    - *SeedGenerator* generates seed from board
    - Point represents a letter coordinate on board
    - Clear responsibilities here
5. Sound invariants
    - In main-classes Board and *SeedGenerator*
    - Legitimate use of invariants
6. Overall code organization & reuse, e.g. views
    - Reuse of *CustomButton* in multiple places

3.- Coding style

1. Consistence
    - Yes, proper Java-OO style
2. Intention-revealing names
    - Intuitive and understandable naming
3. Do not repeat yourself
    - Nothing to complain here
4. Exception, testing null value
    - A lot of exceptions
    - Testing null value could be better (for example in *SeedGenerator* matrix)
5. Encapsulation
    - Clear encapsulations
6. Assertion, contracts, invariant checks
    - Use of assertions but no defined invariants
7. Utility methods
    - Yes, for example in *BoardFactory*

4.- Documentation

1. Understandable
    - Enough comments and @-Annotations where needed
2. Intention-revealing

- Documentation in understandable words, clear for readers which role documented code parts play in hole game

3. Describe responsibilities
   - Class, method, parameters and variables have clear self-describing names. Classes have clear responsibilities & seem quite smart
4. Match a consistent domain vocabulary
   - In consistent English, speaking words but not unnecessarily long

5.- Test

1. General Overview

There are currently three test classes in the test project, which all clearly state the main project class that they are testing, using the same name but appending the –Test at the end, making them clear and distinct.

In total, they account for 16 test methods, which all seem to be correct and free of any errors. Their main project consists of 40 classes, which means that tests provided account for 7.5% of the entire project.

The tests are well executed, and for the most part well written. I would advice to include a test case for every method in the tested class, and to broaden the coverage of tests to extend to the whole project.

2. PointTest class
   - Very well written test class. Everything is clear and concise, testing successfully all of the implemented methods in the Point class. The methods' names make them very easy to understand what is being tested.

3. BoardPointRetrieverTest class
   - All eight test methods are correct and test different scenarios, explained with the method name, using both of the methods included in the BoardPointRetriever class.

4. DatabaseHelperTest class
   - The class seems to have two test methods, which work all right, but both of them are exactly and identically the same, having

only different names. Just to point out, the methods used in the tested class DatabaseHelper are misspelled (getDitcionaryEntries()) which might result very confusing when trying to implement them.

6.- Conclusion

In general terms, the application seems a bit incomplete, but making good progress. Even though its main feature is functional, since the user does get a certain score for completing words, the experience you get while playing it is not the one you expect from a game of this kind. The rules for the game and the possible way to connect letters are also not clear. The design itself leaves a lot to be desired.

We think that the presented version isn’t good enough as to say that they really accomplished v1.0. The grounds for this statement are found throughout the whole application in the form of bugs or unsuccessfully implemented features.