Test Plan

---

i. User interface: invalid special command
ii. Call parsing method with automated input instead of scanner input
iii. Input of “/foobar” to user-interface parsing method
iv. Expect “invalid command /foobar”

---

i. User interface: /quit command
ii. None
iii. Input “/quit”. This must be done manually rather than through junit tests.
iv. Expect program shutdown.

---

i. Word validity
ii. Call parsing method with automated input instead of scanner input
iii. Input of “about ab0ut” to user-interface parsing method
iv. Expect “no word ladder can be found between about and ab0ut”

---

i. Self-match
ii. Code block with automated input to bfs and dfs word ladder instead of scanner input
iii. Call BFS and DFS wordladder methods with “aorta” and “aorta” as input parameters, automated
iv. Expect :

>“a 0 rung word ladder exists between aorta and aorta”
>aorta
> aorta

---

i. BFS/DFS correctness test
ii. Code block with automated, valid input to bfs and dfs word ladder instead of scanner input
iii. Call both versions with "aorta” and “berts”
iv. Expect: Both should output “a <?> rung word ladder exists between aorta and berts”
Note: may need to be manually checked or analyzed between the dictionary set created and our implementation of the searches before fully automated.

---

i. No valid path
ii. Smaller subset of dictionary used to guarantee that a path does not exist
iii. Call both versions of wordladder with “aorta” and “ables”
iv. Expect “no word ladder can be found between aorta and ables” from both

---

Short test dictionary:

abbes abbey abled ables aorta
bella berta berts borta bongo
about

DFS Length-reducing method

Run DFS from start to end, and repeat from end to start. If two differing paths exist in a DFS solution, this method has a high chance of discovering them, and will pick the shortest path.

Our implementation treats "neighbors" as any word in the dictionary that differs by one letter. If we were to compare every single dfs path through the dictionary graph, we would increase the complexity of the algorithm by an order or n or more. This shortest path problem is highly susceptible to failing to detect long-term benefits over locally good choices. Thus, we chose this less robust but much simpler method in order to compromise between finding the shortest dfs path and computational complexity.

Team plan

Danny:
Implementation of BFS search
Implementation of DFS search (together)
Stress/general testing
Debug of BFS and DFS (together)

James:
Implementation of UI
Implementation of test suite
Implementation of DFS search (together)
Debug of BFS and DFS (together)

In general, Danny's focus was on the search algorithms and general manual testing. He set up the initial implementations of the BFS and DFS searches and the class structure, and Jimmy provided input and helped to debug.

In general, James focused on implementing the UI, designing the test suite, and completing the implementation of the DFS and BFS searches so that they ran under all conditions. Along with Danny, he helped to debug the final WordLadder project.