(The reports have to be individual, but the information shared between us (answers for a and d) will be here and can be copied into your individual report.)

(Shared) A random board and its corresponding dictionary is initially created by having the user run the board\_genrator.exe application. Afterwards, the user runs the CAKEproj.asm program, which presents the user with the random board, found words list, current score. The program then waits for user input. If the user enters a 1, the board is shuffled and more input is asked for. If the user enters a 2, then program exits and displays the final score as well as a goodbye message. If the user enters a word, the word is checked by the program for length and used center word and other processing techniques. If the word passes validity checks, then the program compares the contents of reservedspace to user input to find the word. The program displays if the word was found and not used, found and used, or not found. As stated before, the program loops until a 2 is entered to exit the game.

(Shared) (Alastair talk about the algorithms and techniques in the board generator.) After the board and dictionary are generated, they are loaded input a space called reservedspace to be used later on. We used a series of syscalls to print the board and prompt user for input. During the initial printing of the board, score, and found words, the program jumps to a loop that counts the number of found words and divides that by game time in minutes to calculate the score. We made the score correspond with average words found per minute according to the user’s gameplay. Adding the $ra to the stack was an important technique for calculating the score. Found words were printed by looping through the reservedspace which stored the dictionary. The found words were the ones with a \* in front of them and a . indicated a not found word. A | or null character indicated end of dictionary was reached. After the input was received from user, processing of input began. (Phil talk about shuffle feature and validating input.) (Chris talk about dictionary check and printing messages based on 0,1,2. 0 = word found and not used yet, 1 = word found and already used, and 2 = word not found in dictionary.)

Chris Orton

Philip Katterjohn

Alistair Feille

Courtney Erbes

Command Flow

1. Prompt user for input
2. Receive & store user input
3. Check if word (alphabetical) or command (numerical) or N/A (anything else)
   1. If N/A - Return error and ask for proper input
   2. If command – switch to case and run subroutine
      1. Validate command
      2. Shuffle
      3. Exit
   3. If word – proceed
4. Check requirements
   1. Copy array for manipulation
   2. Letters contained in board
   3. 4-9 letters long
   4. Uses center letter
   5. Doesn’t use too many of each letter
5. Compare to word bank
   1. Found words
      1. Ex: .fade    = not found
      2. Ex: \*fade    = found
      3. f = end of dictionary reached
   2. Dictionary bank (unfound words)
6. Return response
   1. If valid
      1. Return success message
      2. Add to found words
      3. Update score
      4. Proceed to end turn
   2. If invalid
      1. Display error message
7. End turn
   1. Display
      1. Found words
      2. Rejected words
      3. Score
      4. Board