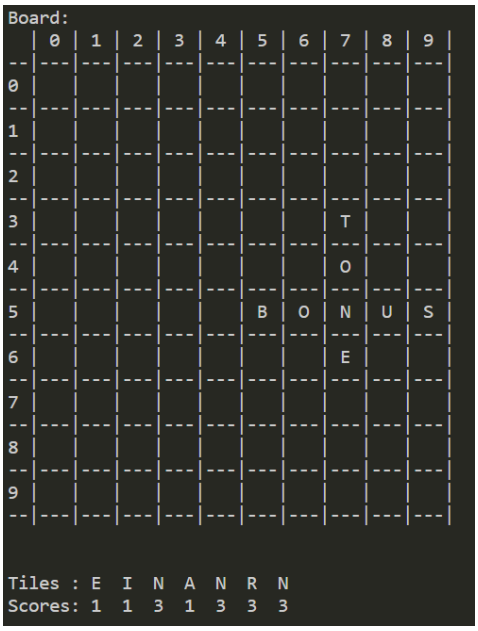
Efficient Approach: Improvised Brute Force

The approach that I had done for this assignment was an improvised form of brute force, where invalid situations are progressively excluded, leaving the remaining possible solutions to be checked. This approach will be demonstrated for the move below.



```
Board:
   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
---|---|---|---|---|---|---|---|---|---|---|
0  |   |   |   |   |   |   |   |   |   |   |
---|---|---|---|---|---|---|---|---|---|---|
1  |   |   |   |   |   |   |   |   |   |   |
---|---|---|---|---|---|---|---|---|---|---|
2  |   |   |   |   |   |   |   |   |   |   |
---|---|---|---|---|---|---|---|---|---|---|
3  |   |   |   |   |   |   |   | T |   |   |
---|---|---|---|---|---|---|---|---|---|---|
4  |   |   |   |   |   |   |   | O |   |   |
---|---|---|---|---|---|---|---|---|---|---|
5  |   |   |   |   |   | B | O | N | U | S |
---|---|---|---|---|---|---|---|---|---|---|
6  |   |   |   |   |   |   |   | E |   |   |
---|---|---|---|---|---|---|---|---|---|---|
7  |   |   |   |   |   |   |   |   |   |   |
---|---|---|---|---|---|---|---|---|---|---|
8  |   |   |   |   |   |   |   |   |   |   |
---|---|---|---|---|---|---|---|---|---|---|
9  |   |   |   |   |   |   |   |   |   |   |
---|---|---|---|---|---|---|---|---|---|---|


Tiles : E  I  N  A  N  R  N
Scores: 1  1  3  1  3  3  3
```

Initially the pool of possible letters and pool of possible words are created. The pool of possible letters (named “letterPool” in scrabble2.py) is created by appending every letter in the tiles and every letter already on the Board. For this move, the letter pool would be [“E”,”I”,”N”,”A”,”N”,”R”,”N”,”T”,”O”,”B”,”O”,”N”,”U”,”S”,”E”]

The pool of possible words (named “wordPool" in scrabble2.py) is then created. Scanning through every word in the dictionary file, if the word is able to be made from the letter pool and the length of the word is less than or equal to the size of the board, then the word is added to the word pool. For this move, the word pool is pictured below. It can be observed that all of the words fit both of the conditions above.

```
['ABET', 'ABIES', 'ABIT', 'ABNET', 'ABOON', 'ABOUT', 'ABUSION', 'ABUT', 'AE', 'AENEOUS', 'AEON', 'AI', 'AINO', 'AIT', 'AN', 'ANES', 'A
NET', 'ANION', 'ANISE', 'ANITO', 'ANOINT', 'ANT', 'ANTE', 'ANTI', 'ANUBIS', 'ANUS', 'AS', 'ASTUN', 'AT', 'ATE', 'ATONES', 'AUBE', 'AUB
IN', 'AUNE', 'AUSTIN', 'BA', 'BAIN', 'BAIT', 'BAN', 'BANE', 'BANNS', 'BANTU', 'BASE', 'BASENET', 'BASIN', 'BASINET', 'BASION', 'BASNET
', 'BASON', 'BASTE', 'BASTION', 'BASTO', 'BASTON', 'BAT', 'BATE', 'BATON', 'BATOON', 'BE', 'BEAN', 'BEAT', 'BEE', 'BEEN', 'BEET', 'BEN
', 'BENE', 'BENET', 'BENISON', 'BENNE', 'BENNET', 'BENT', 'BESAINT', 'BESANT', 'BESIT', 'BESOT', 'BEST', 'BESTAIN', 'BET', 'BETA', 'BE
TAINE', 'BETE', 'BETON', 'BETSO', 'BIAS', 'BIN', 'BINATE', 'BINE', 'BINOUS', 'BION', 'BIS', 'BISE', 'BIT', 'BITE', 'BO', 'BOA', 'BOAST
', 'BOAT', 'BOATION', 'BOEOTIAN', 'BOES', 'BOIST', 'BON', 'BONE', 'BONESET', 'BONNE', 'BONNET', 'BONNIE', 'BONUS', 'BOON', 'BOOSE', 'B
OOST', 'BOOT', 'BOOTEE', 'BOOTS', 'BOS', 'BOSA', 'BOSON', 'BOSTON', 'BOT', 'BOTS', 'BOUN', 'BOUSE', 'BUAT', 'BUNION', 'BUNN', 'BUNT',
'BUS', 'BUSTO', 'BUT', 'BUTANE', 'EAN', 'EAST', 'EBON', 'EBONIST', 'EBONITE', 'EEN', 'EET', 'EN', 'ENATE', 'ENATION', 'ENNUI', 'ENOINT
', 'ENS', 'ENSATE', 'ENSEINT', 'ENSUE', 'ENTUNE', 'EOS', 'EOSIN', 'ESE', 'EST', 'ETESIAN', 'ETNA', 'ETNEAN', 'ETUI', 'EU', 'EUSEBIAN',
 'IN', 'INANE', 'INEE', 'INN', 'INNATE', 'INNE', 'INSET', 'INSUE', 'INTO', 'INTONE', 'INTUNE', 'INTUSE', 'INUST', 'IO', 'ION', 'IS', '
IT', 'ITS', 'NA', 'NAB', 'NABIT', 'NAIS', 'NAN', 'NAOS', 'NAS', 'NASION', 'NAT', 'NATES', 'NATION', 'NE', 'NEAT', 'NEB', 'NEE', 'NENIA
', 'NESE', 'NEST', 'NET', 'NIAS', 'NIB', 'NIN', 'NINE', 'NINUT', 'NIOBATE', 'NIOBE', 'NIS', 'NISAN', 'NISTE', 'NIT', 'NO', 'NOB', 'NOE
TIAN', 'NOIE', 'NOINT', 'NOIOUS', 'NOISE', 'NON', 'NONE', 'NONES', 'NONIUS', 'NONSUIT', 'NOON', 'NOOSE', 'NOOT', 'NOSE', 'NOST', 'NOT'
, 'NOTE', 'NOTUS', 'NOUN', 'NOUS', 'NUB', 'NUBIA', 'NUBIAN', 'NUT', 'OAST', 'OBE', 'OBEISANT', 'OBESE', 'OBOE', 'OBOIST', 'OBTENSION',
 'OBTUSE', 'OE', 'OINT', 'ON', 'ONE', 'ONES', 'ONION', 'ONTO', 'ONUS', 'OO', 'OON', 'OONES', 'OS', 'OSANNE', 'OST', 'OSTEIN', 'OTIOSE'
, 'OTIS', 'OUSE', 'OUST', 'OUT', 'OUTNOISE', 'OUTSEE', 'SABINE', 'SABOT', 'SAI', 'SAIN', 'SAINT', 'SANBENITO', 'SANTON', 'SAO', 'SAT',
 'SATE', 'SATEEN', 'SATIN', 'SATION', 'SAUTE', 'SEA', 'SEAN', 'SEAT', 'SEBAT', 'SEBATE', 'SEEN', 'SEET', 'SEINE', 'SEINT', 'SEN', 'SEN
ATE', 'SENNA', 'SENNET', 'SENNIT', 'SENT', 'SENTINE', 'SET', 'SETA', 'SETEN', 'SETON', 'SI', 'SIB', 'SIENNA', 'SIN', 'SINE', 'SINNET',
 'SINUATE', 'SIT', 'SNEB', 'SNET', 'SNIB', 'SNITE', 'SNOB', 'SNOT', 'SNOUT', 'SNUB', 'SO', 'SOB', 'SOE', 'SON', 'SONANT', 'SONNET', 'S
ONNITE', 'SOON', 'SOONEE', 'SOOT', 'SOT', 'SOTE', 'SOU', 'SOUN', 'SOUNE', 'SOUT', 'SOUTANE', 'STAB', 'STAIN', 'STANE', 'STEAN', 'STEE'
, 'STEEN', 'STEIN', 'STIAN', 'STONE', 'STUN', 'SUANT', 'SUB', 'SUBITO', 'SUE', 'SUENT', 'SUET', 'SUINE', 'SUINT', 'SUIT', 'SUITE', 'SU
N', 'SUNN', 'SUNNA', 'SUNNITE', 'TA', 'TABES', 'TABOO', 'TABU', 'TAI', 'TAIN', 'TAN', 'TAS', 'TAU', 'TEA', 'TEASE', 'TEE', 'TEEN', 'TE
ENS', 'TEINE', 'TEN', 'TENE', 'TENIA', 'TENNE', 'TENNIS', 'TENNO', 'TENNU', 'TENON', 'TENSE', 'TENSION', 'TENUIS', 'TIE', 'TIN', 'TINE
', 'TINEA', 'TINEAN', 'TISANE', 'TOBIE', 'TOBINE', 'TOE', 'TOISE', 'TOISON', 'TON', 'TONE', 'TONNE', 'TONOUS', 'TONUS', 'TOON', 'TOSE'
, 'TOUSE', 'TSEBE', 'TUB', 'TUBE', 'TUE', 'TUN', 'TUNA', 'UNBE', 'UNBEAT', 'UNBIAS', 'UNBIT', 'UNBOOT', 'UNEASE', 'UNIO', 'UNION', 'UN
ISON', 'UNIT', 'UNITE', 'UNNEST', 'UNSAINT', 'UNSET', 'UNSIN', 'UNSOOT', 'UNTIE', 'UNTO', 'US', 'USANT', 'USE', 'USNEA', 'USTION', 'UT
', 'UTAS', 'UTES', 'UTIA', 'UTIS']
```

Next, the algorithm will compute the indexes of the rows and columns that contain a letter in it (known as “rowsFilled” and “columnsFilled” respectively). For this move, the rows filled would be 3,4,5,6 and the columns filled would be 5,6,7,8,9. This is to be used in later on.

For the **first move** the algorithm considers only two possible locations, with both rows and columns in the middle of the Board and either “H” – for horizontal placement or “V”- for vertical placement.

The algorithm goes through every word in the word pool, if the word is able to be placed in the middle of the Board horizontally (i.e. not extending beyond the Board and also being able to be made out of the tiles given) and can be made out of the tiles, then the score is checked and compared against the current highest score. If the score is higher than the current high score, then the word at the location will be the new best move.

Once the algorithm checks for horizontal placement of the word, it then checks the same word (with a vertical placement) for the same requirements as previously mentioned. The algorithm will check for all words in the word pool with each word either being horizontally or vertically placed, leaving the best score at the end of the scan.

Beyond the first move, the algorithm will check every word with varying locations based on the rows and columns that contain a letter, obtained from the previous lists. Similar to the procedure in the first move, each different location for each different word here is checked to see if they can be made from the tiles, if they fit the Board and also if they use one or more tiles already on the board.

In this move, the word will be tested at the first row that contains a letter (3:0:H) followed by (3:1:H) and so on until the word can no longer fit on the Board. It then moves to the next row that contains a letter (4:0:H → 4:1:H, etc.) and so on until all rows that contain a letter are scanned through.

The vertical placement is then considered, starting at the first column (column 5), progressing down the board (5:0:V → 5:1:V → 5:2:V), shifting it by one each time until the word no longer fits the board.

Justification for the improvised brute force approach

This approach takes on similar characteristics of a brute force approach, in which words are still checked at every location, constantly replacing the best move when a better move is found, until the end of its search range.

The improvement lies in eliminating invalid possibilities to greatly reduce the time taken. The construction of the letter pool and word pool allows the number of scanned words to reduce from 80986 words (the number of words in dictionary.txt that the brute force approach would go through) to only a few hundred (see picture above).

The time taken is improved further through only considering the words at locations with rows and columns already containing a letter. Due to the nature of the scrabble game, past the first move, it is impossible for a word to be placed where its row and columns do not contain a letter due to not coinciding with any letters already on the board. For example, in the move above, no word can ever start horizontally at rows 0,1,2,7,8,9 and hence they are not considered. This results in having to scan through only 10% to 90% of what the brute force approach would have.

Furthermore, the validity of each word/location here is still checked, having to go through the same conditions of being able to be made from the tiles, fitting between the Boards dimensions and only supplanting the previous best move if its score was higher to ensure that the resulting best move is guaranteed to be correct.

Taking all of the factors into account and measuring the time taken, this approach leads to a **maximum** of 60 seconds for a 15x15 board with all rows and columns filled with at least one letter.