CSC2021 Coursework
Scrabble Scoring Model
2020

Issued: 21 February 2020
Due: 15:00 (3pm), Friday 13 March 2020
Submission: via Ness

1. Aims

This coursework develops and assesses your skills in modelling using VDM-SL. It provides practice with reading and understanding a specification and with using a variety of modelling techniques and abstractions. The coursework is marked out of 16: it is worth 16% of the module mark for CSC2021.

The coursework consists of two major tasks. In the first, you will complete function definitions provided in the initial model. In the second, you will extend the model by adding additional features. It is possible to get a 2i mark (10/16) with a full solution to the first task.

2. Submission

Submit to Ness your final vdmsl specification file containing your final model.

- Name your file "scrabbleNNNNNNNN.vdmsl" where NNNNNNNN is replaced by your student number (i.e. if your student number is 12345678 your file will be "scrabble12345678.vdmsl").
- Include in a comment at the head of the file with your name and student number, for example
- `-- CSC2021 Coursework Semester 2 2020`
- `-- Name: .....`
- `-- StdNo: .....`

3. Scenario

The coursework concerns a model of the scoring system used in a game of Scrabble. In this word game, players have a number of tiles representing letters (A – Z, plus a "blank" tile) which they take it in turns to use to make words on a grid. Each letter has a point value. A player scores points for valid words added to the grid.

In this coursework, we are concerned only with the process of calculating a score for a given word. We are **not** considering:

- where the word has been placed (which would add modifiers such as double letter, triple word etc)
- what other word it has been attached to (which would affect the score for the attached tile)

A set of English-language Scrabble tiles consists of 100 tiles scored and distributed as follows:

- 0 points: Blank tile (2)

- 1 point: E (12), A(9), I(9), O(8), N(6), R(6), T(6), L(4), S(4), U(4)
- 2 points: D(4), G(3)
- 3 points: B(2), C(2), M(2), P(2)
- 4 points: F(2), H(2), V(2), W(2), Y(2)
- 5 points: K(1)
- 8 points: J(1), X(1)
- 10 points: Q(1), Z(1)

A score is awarded for a particular word by totalling the points scored for each letter. A blank tile can represent any letter, but has no point value. In order to be scored as a valid word, the word must:

- use only the available letters, including blank tiles to replace up to 2 of the letters
- be valid according to an agreed dictionary. A blank tile ("_") can represent any letter and so a word "EXA_PLE" would be valid in a dictionary lookup if the dictionary contains the word "EXAMPLE".

4. Initial model

The initial model is contained in a project zip file **scrabbleScore.zip** in Blackboard. You will need to import the project into Blackboard in the usual way: refer to the 2 tutorials for a reminder of how to do this.

The project contains the following two files. For reference these files are also provided in the appendix to this specification.

scrabble.vdmsl – initial types and definitions for the model. Your final submission will be an updated version of this file.

values.vdmsl – data to set up the tile scores and frequencies and a test dictionary. You should not modify this file.

The Scrabble model sets up datatypes to represent a letter **Tile**, represented as a character, a **Word** (represented as a sequence of **Tiles**), a **Dictionary** (set of **Words**) and a **LetterMap**. The **LetterMap** type, a mapping from **Tile** to natural number, will be used to represent two distinct maps: the points score for each tile; and the distribution of tiles.

The state for the model consists of the two letter maps **thePoints** and **theTiles,** and the **Dictionary**. The invariant defined for the state ensures that both maps have the same domain, defined as the tileSet variable mentioned below.

The values file defines variables as follows:

- **tileSet**: the full set of tiles a-z plus the special blank tile
- **pointsMap**: defines the points available for each letter tile
- **tilesLeft**: defines the distribution for each letter tile
- **smallWords**: a dictionary of words defining the full set of valid words. This is a short dictionary of 153 words consisting of the contents of /usr/share/dict/connectives on a standard unix distribution plus three extra words to give coverage of all letters available

In order to calculate the score for a given word, an operation ScoreWord has been provided:

```
ScoreWord: Word ==> nat
ScoreWord(word) ==
 return GetWordScore(word,thePoints)
pre EnoughTiles(word,theTiles) and ValidWord(word, theWords)
```

This operation will return the total points scored by the given word, according to the state variable **thePoints** mapping. This score is calculated by the Boolean function **GetWordScore**. **ScoreWord** can only calculate the score if there are enough tiles to form the word, and if the word is valid according to the dictionary. Both of these conditions are represented in the operation's precondition, as calls to Boolean functions **EnoughTiles** and **ValidWord** respectively.

Signatures for these Boolean functions are provided, but only the body of **ValidWord** has been defined. A further function **Occurrences** has also been provided and is described in Question 2 below.

5. Tasks

There are two major tasks: Task 1 (questions 1,2,3) is to complete the existing model. Task 2 (questions 4,5) extend the model to record scores and update the available tiles after each move.

In all questions, consider whether there are further restrictions to represent on functions and operations (as preconditions) or on datatypes (as invariants).

Task 1: Complete initial model

Question 1.

The function **GetWordScore** has the following signature

```
GetWordScore: Word * LetterMap -> nat
GetWordScore(word,points) == is not yet specified
```

The function is to take a word and a map giving the points for each letter (or blank) in the word, and return the total points score for the word. Complete the function definition. Hint: you may find a recursive function helpful. [3 marks]

Question 2.

The function **EnoughTiles** has the signature

```
EnoughTiles: Word * LetterMap -> bool
EnoughTiles(word, tiles) == is not yet specified
```

The function is to take a word and map giving the number of tiles available of each letter (e.g. 2 blanks, 12 'e' etc), and return true if there are enough tiles to form the word. You can make use of the provided function **Occurrences** which, given a tile (letter or blank) and a word, returns a count of the number of times that tile appears in the word. For example **Occurrences('p', "people")** would return 2. [3 marks]

Question 3.

The function **ValidWord** currently checks whether a word is playable by simply looking it up in the dictionary. However, this does not take account of words partly formed with blanks. Write a new version of **ValidWord** with the same signature (taking input word and dictionary), which returns true if there is a word in the dictionary which has either all the same letters as the input word, or any other letter where the input word has a blank. For example **ValidWord("a_", d)** would return true for dictionary **d** if that dictionary contains "at", "an" or any other 2 letter word beginning with 'a'.

[4 marks]

Task 2: Extend the model

This task is to add a record of player scores to the model, and reduce the number of tiles available after a word has been scored.

Question 4.

Add to the model:

- a type to represent a **Player**. Your chosen type should be sufficient to distinguish one player from another but does not need to record player name or any other data
- a **ScoreMap** mapping type to represent the numeric score recorded for each player
- Add a state variable to your state definition to keep track of the current player scores. Ensure this state variable is initialised to an empty mapping.
- Write a new operation **AddPlayer: Player ==> ()** to add a new player to the state variable above, setting their score to 0.

[3 marks]

Question 5.

Rewrite the **ScoreWord** operation to record scores and update the tiles mapping. The new signature should be:

```
ScoreWord: Player * Word ==> ()
```

The operation will use **GetWordScore(word,thePoints)** as before to calculate the score, but use this to update the score mapping for the given player. The tiles used in the word are then to be removed from the **theTiles** state variable. You can use a call to the operation **RemoveTiles**, given below, to achieve this last part.

```
RemoveTiles: Word ==> ()
RemoveTiles(word) ==
       theTiles := theTiles ++
         { t |-> theTiles(t) - Occurrences (t, word) | t in set dom theTiles
            & t in set elems word };
```

[3 marks]

Ensure you submit to Ness your final vdmsl specification file containing your final model, renamed following the file naming convention requested (see Section 2).

Appendix A. Initial model

```
types

Tile = char
inv t == t in set tileSet;

Word = seq of Tile;

Dictionary = set of Word;

LetterMap = map char to nat;

state Scorer of
thePoints: LetterMap -- points scored for each tile
theTiles: LetterMap -- initial distribution of tiles
theWords: Dictionary
inv mk_Scorer(thePoints,theTiles,theWords) ==
      dom thePoints = tileSet and dom theTiles = tileSet
      and theWords <> {}
init s ==
      s = mk_Scorer( pointsMap, tilesLeft, smallWords )
end;

functions
GetWordScore: Word * LetterMap -> nat
GetWordScore(word,points) == is not yet specified
;

Occurrences: Tile * Word -> nat
Occurrences(tile,word) ==
  len[word(i) | i in set inds word & word(i) = tile];

EnoughTiles: Word * LetterMap -> bool
EnoughTiles(word, tiles) == is not yet specified
;

ValidWord: Word * Dictionary -> bool
ValidWord(word, dictionary) ==
  word in set dictionary;

operations
ScoreWord: Word ==> nat
ScoreWord(word) ==
 return GetWordScore(word,thePoints)
pre EnoughTiles(word,theTiles) and ValidWord(word, theWords)
```

Appendix B. Data for initial model

```
-- Data for CSC2021 scrabble coursework Feb 2020.

-- Tiles include all letters of alphabet plus BLANK, represented by _

-- pointsMap gives points scored by each tile: use as pointsMap(letter)
--   to get points for letter. e.g. pointsMap('d') returns 2.

-- tilesLeft represents the initial letter distributions and is used in --
the same way as pointsMap, e.g. tilesLeft('d') returns 4.

-- smallWords is a set of words taken from /usr/share/dict/connectives
-- in standard unix distribution, with a couple of extra words to cover
-- all letters of alphabet.

-- all types are defined in the main scrabble specification file.

values
BLANK: char = '_';
tileSet: set of char = {BLANK} union elems "abcdefghijklmnopqrstuvwxyz";

pointsMap: LetterMap =
        {BLANK |-> 0, 'k' |-> 5}
        munion { c |-> 1 | c in set elems "eaotinrslu"}
            munion      { c |-> 2 | c in set elems "dg"}
            munion      { c |-> 3 | c in set elems "cmbp"}
            munion      { c |-> 4 | c in set elems "hfwyv"}
            munion      { c |-> 8 | c in set elems "jx"}
            munion      { c |-> 10 | c in set elems "qz"};

tilesLeft: LetterMap =
  { BLANK |-> 2, 'g' |-> 3, 'o' |-> 8, 'e' |-> 12}
      munion { c |-> 1 | c in set elems "kjxqz" }
      munion { c |-> 2 | c in set elems "bcmpfhvwy"}
      munion { c |-> 4 | c in set elems "lsud"}
      munion { c |-> 6 | c in set elems "nrt"}
      munion { c |-> 9 | c in set elems "ai"}


values

smallWords: Dictionary = {"quick", "zero", "onyx"} union {
"the", "of", "and", "to", "a", "in", "that", "is", "was", "he", "for",
"it", "with", "as", "his", "on", "be", "at", "by", "i", "this", "had",
"not", "are", "but", "from", "or", "have", "an", "they", "which",
"one", "you", "were", "her", "all", "she", "there", "would", "their",
"we", "him", "been", "has", "when", "who", "will", "more", "no", "if",
"out", "so", "said", "what", "up", "its", "about", "into", "than",
"them", "can", "only", "other", "new", "some", "could", "time",
"these", "two", "may", "then", "do", "first", "any", "my", "now",
"such", "like", "our", "over", "man", "me", "even", "most", "made",
"after", "also", "did", "many", "before", "must", "through", "back",
"years", "where", "much", "your", "way", "well", "down", "should",
"because", "each", "just", "those", "people", "mr", "how", "too",
"little", "state", "good", "very", "make", "world", "still", "own",
"see", "men", "work", "long", "get", "here", "between", "both",
"life", "being", "under", "never", "day", "same", "another", "know",
"while", "last", "might", "us", "great", "old", "year", "off", "come",
"since", "against", "go", "came", "right", "used", "take", "three"  };
```