Scrabble

Assignment 2
CSSE1001/7030
Semester 2, 2017

Version 1.0.1

20 marks : 10%
Due Friday 22 September, 2017, 21:30

1. Introduction

For this assignment, you will be writing code that supports a simple Scrabble game.

Rather than using functions, like assignment 1, you will be using Object-Oriented Programming (OOP). Further, this assignment has been designed according to the Model-View-Controller (MVC) design pattern. Your task is to write most of the Model. The View, Controller, and remainder of the Model are provided, along with some support code.

As is typical with projects where more than one person is responsible for writing code, there needs to be a way of describing how the various components interact. This is achieved by defining an Application Programming Interface (API). Your classes must be implemented according to the API that has been specified, which will ensure that your code will interact properly with the supplied View/Controller code.

One benefit to adhering to MVC is that the model can be developed and tested independently of the view or controller. It is recommended that you follow this approach. This means testing your model iteratively as you develop your code.

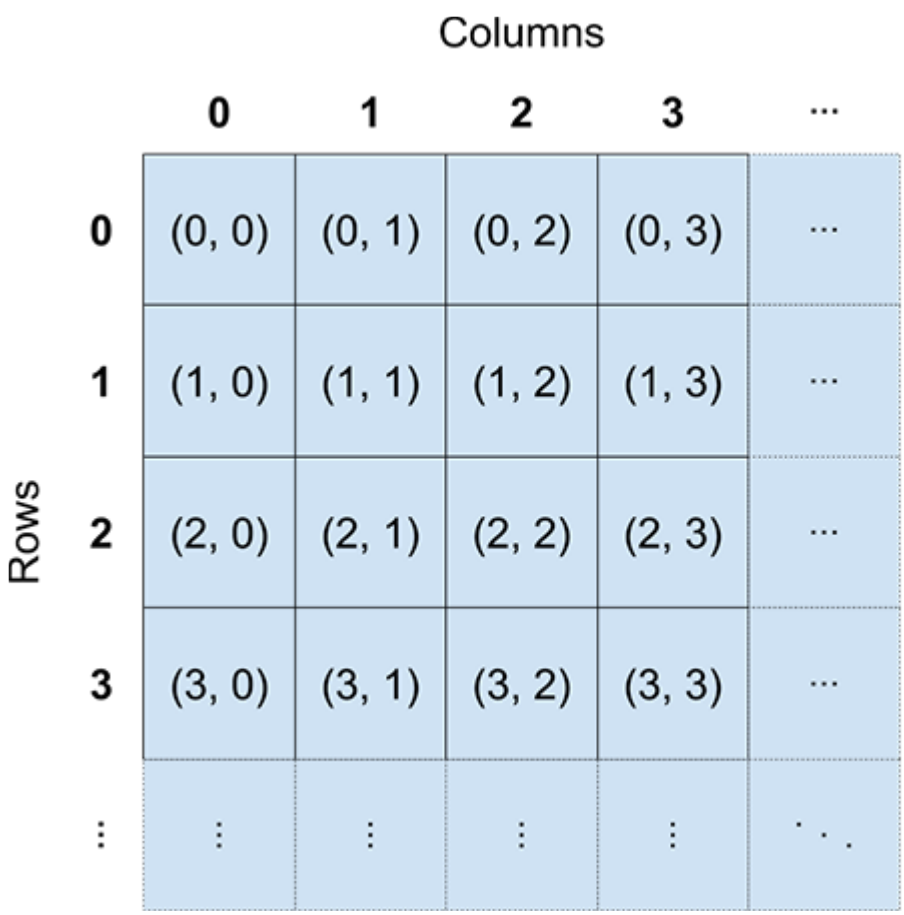
1.1. Rules & Important Definitions

Scrabble is a game where players take turns using the tiles in their racks to form words on the game board. If you are unfamiliar with this game, it is recommended that you research the basic gameplay.

This assignment will follow the standard rules of Scrabble, with the following additions:

- The game will also end if all players skip their turns consecutively
- All legal words are contained in the file `words_alpha.txt`
- The starting player is assigned randomly

The concept of a position is used throughout this assignment, including in the provided support code. A position is a (row, column) pair, with `(0, 0)` being the top left corner. This is demonstrated below.



Scrabble Board Positions

2. Getting Started

2.1. Files

Before beginning work on the assignment you must download the relevant files from the course website:

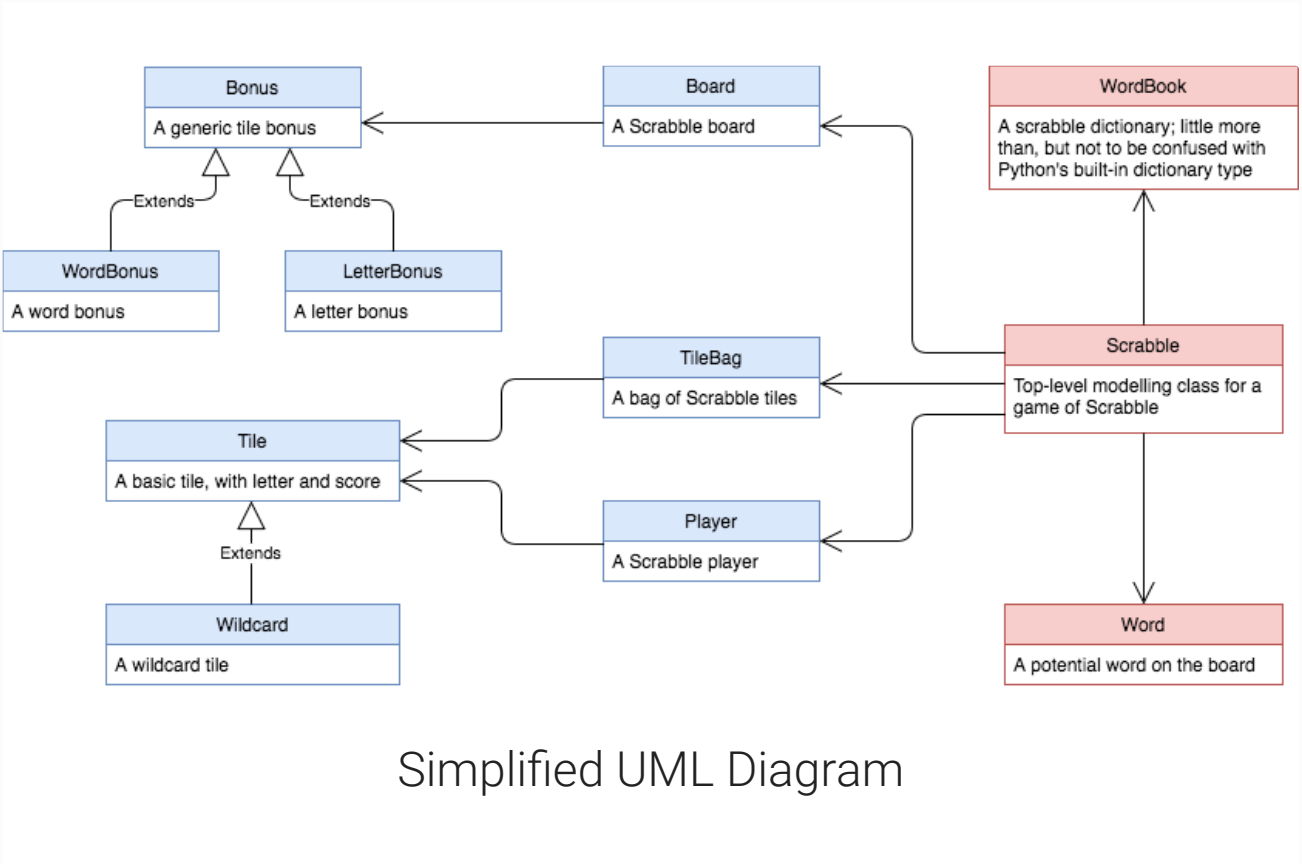
- `a2.py` is the template file for your submission; you should write your code here and **must not** modify any of the other files
- `a2_files.zip` contains:
  - `a2_support.py`: Contains useful support code
  - `letters.txt`: Contains letter frequency & score
  - *`words_alpha.txt`: A list of all valid Scrabble words

- `*scrabble.py`: Supplied modelling classes
- `*scrabble_gui.py`: A GUI for Scrabble that will work if your assignment is implemented according to the API outlined in this document
- `*ee.py`: A third-party library required by the GUI

> **Note:** Files marked with `*` do not need to be understood to complete this assignment

Do not begin writing code until you have reviewed `a2_support.py`. Be sure to review 8. Assignment Submission & 9. Assessment and Marking Criteria

2.2. Classes



Simplified UML Diagram

Above is a simplified UML diagram of the class structure of this assignment. There is a box for each class in the Model, with a brief description.

Here, a plain arrow from `A` to `B` indicates that `A` has a property of type `B` or a property that is a sequence whose elements are of type `B`. For example, the `Scrabble` class has a `Board` and a list of `Players`, respectively.

Further, an "Extends" arrow from `A` to `B` indicates that `A` inherits from `B`.

The blue coloured classes are the assignment tasks. Red classes have been provided for you, for simplicity.

These classes may be implemented in any order, but it is recommended to follow the order in which they are listed.

The parameter & return types for most methods can be inferred from the example code. Where this is not clear, annotations are included with the method description.

2.3. Help

If you are ever unsure or stuck, the best way to seek help is by attending a practical session and asking a tutor. You can also post on the course discussion board, even outside of class time. Be sure to clarify any potential ambiguities that you encounter.

Shortly after release, a video overview of the assignment will be posted on the assignment page of the course website.

3. Tiles

The most fundamental piece of a Scrabble game is the tile. A Scrabble tile has a letter and a corresponding base score.

3.1. Tile

The `Tile` class is used to represent a regular Scrabble tile.

Instances of `Tile` should be initialized with `Tile(letter, score)`. The following methods must be implemented:

get_letter(self): Returns the letter of the tile

get_score(self): Returns the base score of the tile

__str__(self): Returns a human readable string, of the form `{letter}:{score}`

__repr__(self): Same as `__str__`

reset(self): Does nothing (this method will be overridden in the following subclass)

3.2. Wildcard

The `Wildcard` class is used to represent a wildcard Scrabble tile. The user can choose the letter this tile represents when they play it on the board.

`Wildcard` inherits from `Tile` and should be initialized with `Wildcard(score)`. The following methods must be implemented:

set_letter(self, letter): Sets the letter of the tile

reset(self): Resets this tile back to its wildcard state (i.e. unsets the letter)

3.3. Examples

```
>>> tile1 = Tile('m', 3)
>>> tile2 = Tile('d', 2)
>>> tile1.get_letter()
'm'
>>> tile1.get_score()
3
>>> str(tile2)
'd:2'
>>> print(tile2)
d:2
>>> wild = Wildcard(0)
>>> wild
?:0
>>> wild.set_letter('r')
>>> wild
r:0
>>> wild.reset()
>>> wild
?:0
```

4. Bonuses

Bonuses are a critical part of a Scrabble game. They allow the score of a letter or a word to be doubled or tripled.

`Bonus` is a very simple superclass that is used to represent a generic bonus. `WordBonus` & `LetterBonus` inherit from it.

While not intended to be initialized directly, the constructor for `Bonus` should take a single argument, `value`, which is the value of this bonus. The reason for doing this is so that each type of bonus can be represented by using a subclass with minimal additional functionality. Although this case is quite simple, in general, this concept is very powerful.

The following method must be implemented on `Bonus`:

get_value(self): Returns the value of this bonus

The following method must be implemented on both `WordBonus` & `LetterBonus`:

__str__(self): Returns a human readable string, of the form `{type}{value}`, where `type` is `W` for `WordBonus` & `L` for `LetterBonus`

4.1. Examples

```
>>> double_word = WordBonus(2)
>>> double_word.get_value()
2
>>> print(double_word)
W2
>>> triple_letter = LetterBonus(3)
>>> triple_letter.get_value()
3
>>> print(triple_letter)
L3
```

5. Player

How could we play Scrabble without something to represent a player? The `Player` class represents a player and their rack of tiles.

The constructor for `Player` should take a single argument, the player's `name`. The following methods must be implemented:

get_name(self): Return's the player's name

add_tile(self, tile): Adds a tile to the player's rack

remove_tile(self, index): Removes and returns the tile at `index` from the player's rack

get_tiles(self): Returns a list of all tiles in the player's rack (Return type: `list<Tile>`)

get_score(self): Return's the player's score

add_score(self, score): Adds `score` to the player's total score

get_rack_score(self): Returns the total score of all letters in the player's rack

reset(self): Resets the player for a new game, emptying their rack and clearing their score

__contains__(self, tile): Returns `True` iff the player has `tile` in their rack

__len__(self): Returns the number of letters in the player's rack

__str__(self): Returns a string representation of this player and their rack, of the form `{name}:{score}\n{tiles}`, where `tiles` is a comma (&space) separated list of all the tiles in the player's rack, in order

5.1. Examples

```
>>> player = Player("Michael Scott")
>>> tiles = [Tile('t', 1), Tile('w', 4), Wildcard(0),
Tile('s', 1), Tile('s', 1)]
>>> for tile in tiles: player.add_tile(tile)

>>> print(player)
Michael Scott:0
t:1, w:4, ?:0, s:1, s:1
>>> player.add_score(50)
>>> player.get_rack_score()
7
>>> player.remove_tile(1)
w:4
>>> player.get_rack_score()
3
>>> len(player)
4
>>> tile  # Note that this is the last tile, which has the
same values as the second last, but is not the same object
s:1
>>> tile in player
True
>>> player.remove_tile(3)
s:1
>>> tile in player
False
>>> print(player)
Michael Scott:50
t:1, ?:0, s:1
```

6. TileBag

Where will we pickup new tiles after we've made a play? The `TileBag` class is used to hold Scrabble tiles.

`TileBag`'s constructor takes a single argument, a `data` dictionary whose keys are letters (lowercase) and whose values are pairs of `(count, score)`, where count is the number of tiles to create with this letter, and score is the tile's score. See the examples below. The following methods must be implemented:

__len__(self): Returns the number of tiles remaining in the bag

__str__(self): Returns a human readable string, of each tile joined by a comma and a space; i.e. "b:3, o:1, o:1, m:3" — the order the tiles are displayed does not matter

draw(self): Draws and returns a random tile from the bag

drop(self, tile): Drops a tile into the bag

shuffle(self): Shuffles the bag

reset(self): Refills the bag and shuffles it, ready for a new game

> **Note:** For efficiency, the bag should only be automatically shuffled when initialized or reset. You may assume that the user will shuffle the bag manually after dropping some tiles. The reason for this is that when dropping multiple tiles, shuffling only needs to occur after the last tile is dropped. Shuffling after each drop would be inefficient. The difference is not noticeable with this small amount of data.

6.1. Examples

Since the bag is shuffled randomly when initialized, this order of letters in each line of output will differ each time.

```
>>> data = {'b': (1, 5), 'z': (2, 8), 'e': (5, 1)}
>>> bag = TileBag(data)
>>> print(bag)
z:8, e:1, e:1, e:1, e:1, b:5, z:8, e:1
>>> for i in range(3): print(bag.draw())
e:1
z:8
b:5
>>> len(bag)
5
>>> bag.drop(Wildcard(0))
>>> len(bag)
6
```

```
>>> print(bag)
z:8, e:1, e:1, e:1, e:1, ?:0
>>> bag.shuffle()
>>> print(bag)
z:8, e:1, ?:0, e:1, e:1, e:1
>>> bag.reset()
>>> print(bag)
e:1, e:1, z:8, z:8, e:1, e:1, b:5, e:1
```

7. Board

And how could we play without somewhere to arrange tiles? The Scrabble tiles can be played on the `Board` class. It also keeps track of which cells have bonuses.

A `Board` should be initialized with `Board(size, word_bonuses, letter_bonuses, start)`, where:

- `size` is the number of rows/columns on the board (i.e. 15)
- `word_bonuses` is a dictionary with scale of word bonuses as the key, and a list of positions where this scale occurs (see the examples below)
- `letter_bonuses` as with `word_bonuses` above, but with letter bonuses instead
- `start` is the (row, column) position of the starting cell

get_start(self): Returns the starting position

get_size(self): Returns the number of (rows, columns) on the board

is_position_valid(self, position): Returns True iff the position is valid (i.e. it is on the board)

get_bonus(self, position): Returns the bonus for a position on the board, else None if there is no bonus (Return type: `Bonus`)

get_all_bonuses(self): Returns a dictionary of all bonuses, keys being positions and values being bonuses (Return type: `dict<tuple<int, int>, Bonus>`). Since `Bonus` is not intended to be instantiated directly, these values should be instances of `Bonus`' subclasses.

get_tile(self, position): Returns the tile at position, else None if no tile has been placed there yet (Return type: `Tile`)

place_tile(self, position, tile): Places a tile at position; raises an IndexError if position is invalid

__str__: Returns a human readable representation of the game board as shown in the examples below. The `Tile`'s string should be aligned to the left, four characters wide. The `Bonus`'s string should be aligned to the right, three characters wide. These two should have a single space in between. See the examples for what to use if no `Tile`/`Bonus` exists at a position. Assume that tiles have a maximum score of 99 and that a bonus' value will never exceed 9.

reset(self): Resets the board for a new game

7.1. Examples

```
>>> word_bonuses = {2: [(2,2)], 3: [(0, 0), (0, 4), (4, 0),
(4, 4)]}
letter_bonuses = {2: [(0, 3), (4, 1)], 3: [(1, 0), (3, 4)]}
>>> board = Board(5, word_bonuses, letter_bonuses, (2, 2))
>>> board.get_size()
(5, 5)
>>> board.get_start()
(2, 2)
>>> board.is_position_valid((2, 1))
True
>>> board.is_position_valid((2, 8))
False
>>> board.get_bonus((2, 2))  # the 0x... part will differ
<WordBonus object at 0x10590acf8>
>>> print(board.get_bonus((2, 2)))
W2
>>> print(board.get_bonus((3, 4)))
L3
>>> print(board.get_bonus((1, 1)))
None
>>> board.get_tile((2, 3))
>>> board.place_tile((2, 1), Tile('B', 3))
>>> board.place_tile((2, 2), Tile('A', 1))
>>> board.place_tile((2, 3), Tile('Z', 10))
>>> board.get_tile((2, 3))
Z:10
>>> print(board)
-------------------------------------------------
| None W3 | None    | None    | None L2 | None W3 |
-------------------------------------------------
```

```
| None L3 | None     | None     | None     | None     |
-----------------------------------------------------
| None    | B:3      | A:1  W2  | Z:10     | None     |
-----------------------------------------------------
| None    | None     | None     | None     | None L3  |
-----------------------------------------------------
| None W3 | None L2  | None     | None     | None W3  |
-----------------------------------------------------
>>> type(board.get_tile((2, 3)))
<class 'Tile'>
>>> board.place_tile((2, 3), Tile('E', 1))
>>> board.get_tile((2, 3))
E:1
```

8. Assignment Submission

Your assignment must be submitted via the assignment three submission link on Blackboard. You must submit `a2.py` **only**. Do not submit any of the support files or sample tests.

Late submission of the assignment will **not** be accepted. In the event of exceptional circumstances, you may submit a request for an extension.

All requests for extension must be submitted on the UQ Application for Extension of Progressive Assessment form: http://www.uq.edu.au/myadvisor/forms/exams/progressive-assessment-extension.pdf **at least 48 hours prior** to the submission deadline. The application and supporting documentation must be submitted to the ITEE Coursework Studies office (78-425) or by email to enquiries@itee.uq.edu.au.

9. Assessment and Marking Criteria

Criteria	Mark
Functionality	**16**
Tile classes	1
Bonus classes	1

Criteria	Mark
Player	4
TileBag	3
Board	7
Style[1] & Documentation[2]	**4**
Program is well structured and readable	1
Variable and function names are meaningful	1
Entire program is documented clearly and concisely, without excessive or extraneous comments	2
Total	**/20**

> [1]In order to be eligible for the marks for Programming Constructs & Documentation, you must have made a reasonable attempt at implementing most of the required classes.
>
> [2]See the course notes on Style & Commenting.

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in the week following the assignment submission deadline, in the practical session in which you are enrolled. **You must attend that session in order to obtain marks for the assignment.**

Change Log

☐ Toggle Change Highlighting

Any changes to this document will be listed here.

Version 1.0.1 - September 17

The purpose of this update is to improve clarity & reduce ambiguity. There are no breaking changes.

- Fixed typographical errors

- Fixed error handling in `scrabble_gui.py`
- Explicitly specified ambiguous return types for some methods on 5. Player & 7. Board
- Converted bonus positions in `a2_support.py` to lists (from sets)

Version 1.0.0 - September 11

- Removed redundant `position` parameter for `get_all_bonuses` & clarified return type
- Clarified `Board.__str__` & added example of non-empty `Board`
- `scrabble_gui.py`: Fixed bug & refactored
- Standardized import statements in `scrabble.py, scrabble_gui.py`
- Corrected typographical error in Programming Constructs & Documentation footnote, in 9. Assessment and Marking Criteria
- Clarified `get_tiles` & `remove_tile` on 5. Player
- Clarified 6. TileBag