Zig Zag Education

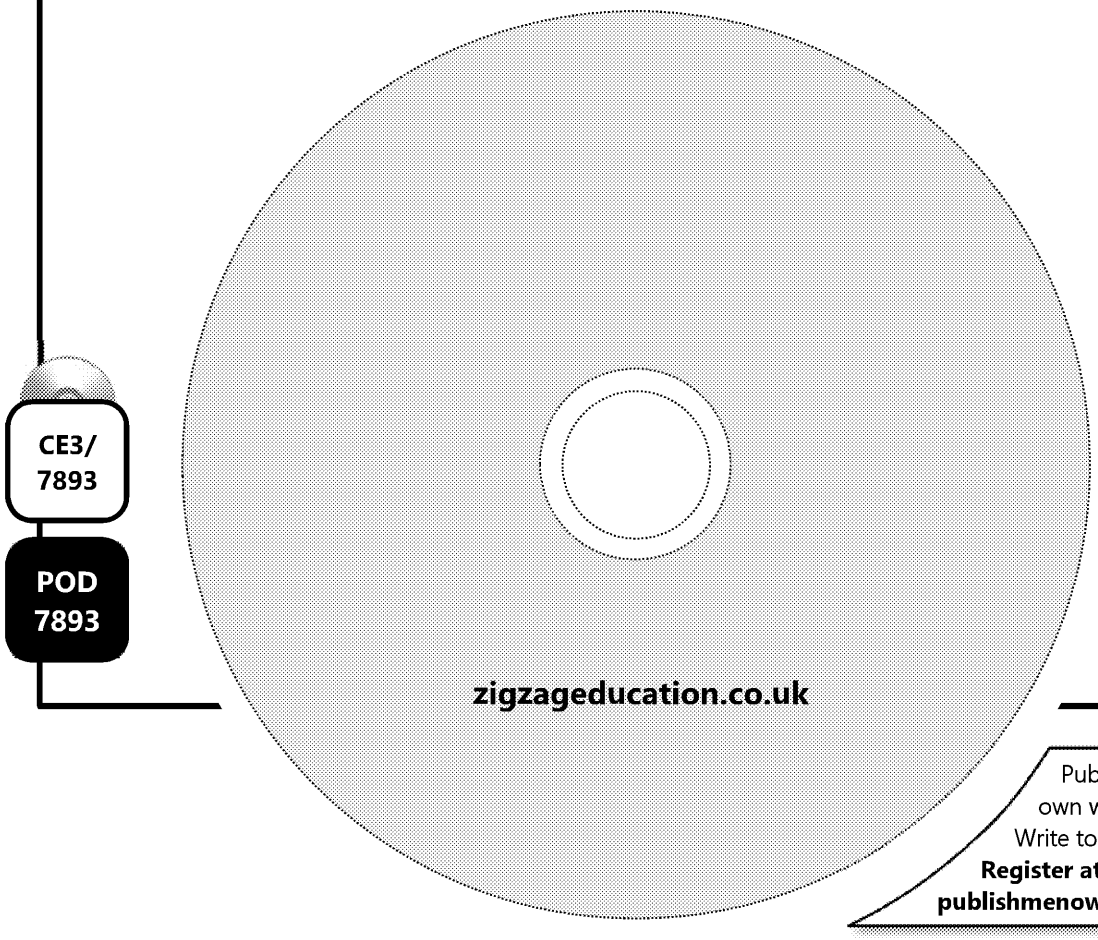
**Computer Science** A Level | AQA | 7517


# PAPER 1 EXAM RESOURCE PACK 2018

## WORDS WITH AQA

**PYTHON2**

for A Level AQA Computer Science



**CE3/ 7893**

**POD 7893**

**zigzageducation.co.uk**

# Contents

# Teacher's Introduction

This resource pack is designed to help you support your students taking the **A Level Computer Science Paper 1** examination. It is based on the **'Words with AQA'** preliminary material (Python2) – for examination June 2018.

**New Format:** The biggest improvement in this 2018 resource pack sees all content provided electronically* for the first time. On the CD, you will find the following files.

| | |
|---|---|
| `WordsWithAQA` | for student use – this folder contains all of the content, accessible via a HTML interface |
| `editable` | for teacher use – this folder contains ALL of the documents in editable (docx /pptx) formats |
| `Passwords.txt` | for teacher use – this file contains all of the passwords for the protected PDFs (also listed below) |

* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

**Installation:** Copy the entire `WordsWithAQA` folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

**Passwords:** All of the PDFs in the 'Answers & Solutions' HTML page (`answers.html`) are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

| | |
|---|---|
| `Commentary.pdf` | `1158` |
| `Diagram1Complete.pdf` | `4773` |
| `Diagram2Complete.pdf` | `5382` |
| `Diagram3Complete.pdf` | `3091` |
| `QuestionsMarkScheme.pdf` | `7642` |
| `TaskMarkScheme.pdf` | `2966` |

Should you wish to give students access to ALL protected-PDFs, the master password for all files is:

`zz2ghc4`

The resource pack consists of the following:

① **Pre-release Commentary**, consisting of two parts:

- A general walkthrough of the skeleton program; a written description, flowchart and an animated PowerPoint giving a visual demonstration of the game. It is non-technical in the sense that it doesn't reference or explain any actual code elements – only how the program works when it is run.
- A detailed, technical overview of the skeleton program, describing how all Python subroutines, classes and variables work, including the relationship between them.

**Note:** although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves. For this reason, this content has been placed on the 'Answers & Solutions' HTML page as a password-protected file, to allow you to control if/when students access it.

② **Structure Diagram Activities**

Three partially complete structure diagram activities for students to complete while getting to grips with the skeleton program. Any missing identifiers, data types, return values, directional arrows, etc. must be added to the diagram. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the 'Words with AQA' code, like Section C in the exam. These questions require access to the skeleton code, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on version are available format. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test, like Section D in the exam. Solutions are provided on the *Answers & Solutions* page as a protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

**Free Updates**

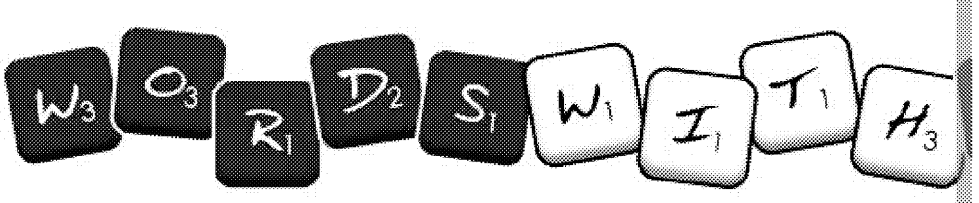
Register your email address to receive any future free minor updates made to this resource or other Computing resources your school has purchased, and details of any promotions for your subject.

** resulting from minor specification changes, suggestions from teachers and peer reviews, or occasional errors reported by customers*

**zzed.uk/freeupdates**

An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

This resource is intended to supplement your teaching only. **Please read full disclaimer (p. iv) before using it.**




# Introduction

*Words with AQA* is a game in which two human players take turns to make words from tiles that have been dealt to them.

When the game begins, a queue of tiles is created, in which 20 tiles are generated. Tiles are removed from the front of the queue and, once removed, are replaced by an identical tile at the end of the queue. The tile queue can be replenished any number of times, so the same tiles can be played.

When the game begins, Player One and Player Two are each assigned 15 tiles taken from the queue, their scores are set to 50. An array is also assembled from a text file, which contains the words that can be played. The players then take turns, with each turn following this format:

1. The player attempts to play a word using their tiles (each tile can only be played once; if the word 'HAMMER' were to be attempted, the player would need two 'M' tiles).

   Each letter tile has an integer value, which determines the score, so the word 'HAMMER' would be worth 11 points.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $A_1$ | $B_2$ | $C_2$ | $D_2$ | $E_1$ | $F_3$ | $G_2$ | $H_3$ | $I_1$ |
| $N_1$ | $O_1$ | $P_3$ | $Q_{10}$ | $R_1$ | $S_1$ | $T_1$ | $U_2$ | $V_3$ |

2. After a word is played, the program checks that it exists in the array of legal words. This depends on whether the word is allowed.

   a. If the word is allowed, that word's score is calculated using the tile values. If the word is seven characters long, 5 bonus points are awarded. If the word is eight characters long, 10 bonus points are awarded. They may then choose how many new tiles they will take from the queue, from the following options:

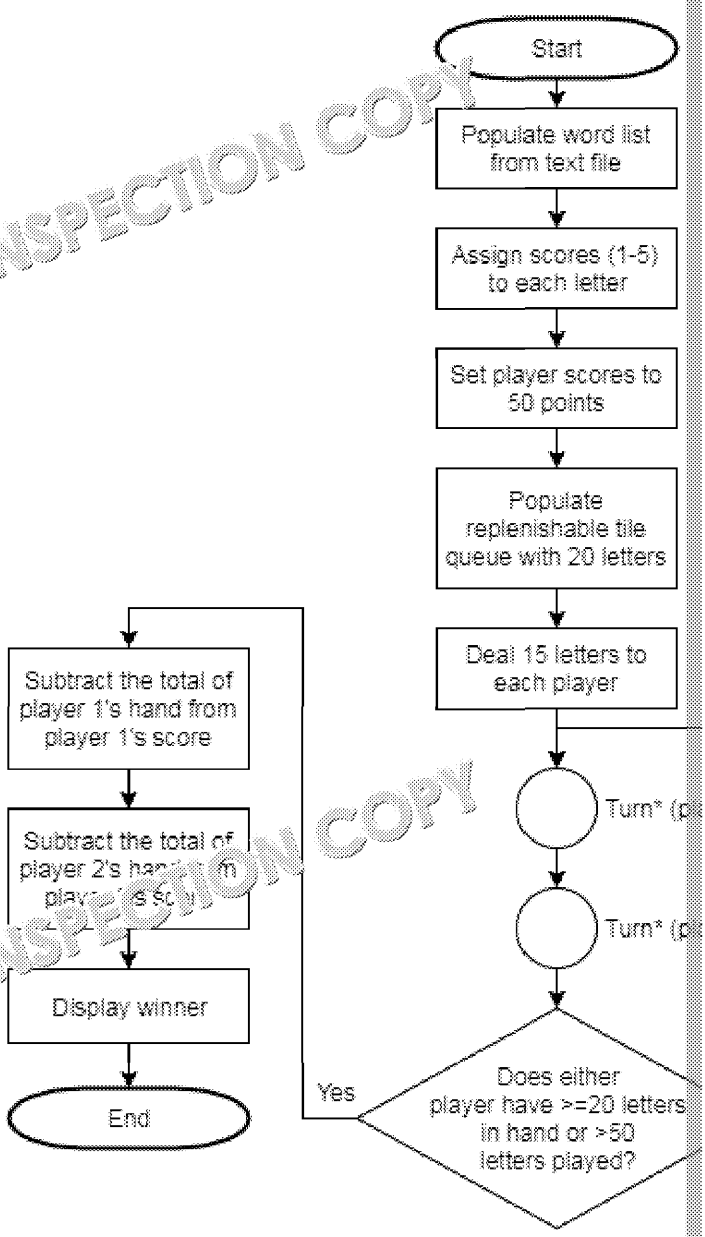
      - three tiles
      - a number of tiles equal to the length of the played word (so four tiles for a four-letter word)
      - a number of tiles equal to the length of the word plus three (so seven tiles for a four-letter word)
      - no tiles

   b. If the word is not allowed (is not in the array), the player's turn is over, they are not permitted to attempt a second word. They are then given three tiles.

The game continues until either a player has played a total of more than 50 tiles (or more) in their hand, or one of these is true after Player One's turn, Player Two's turn the game ends.

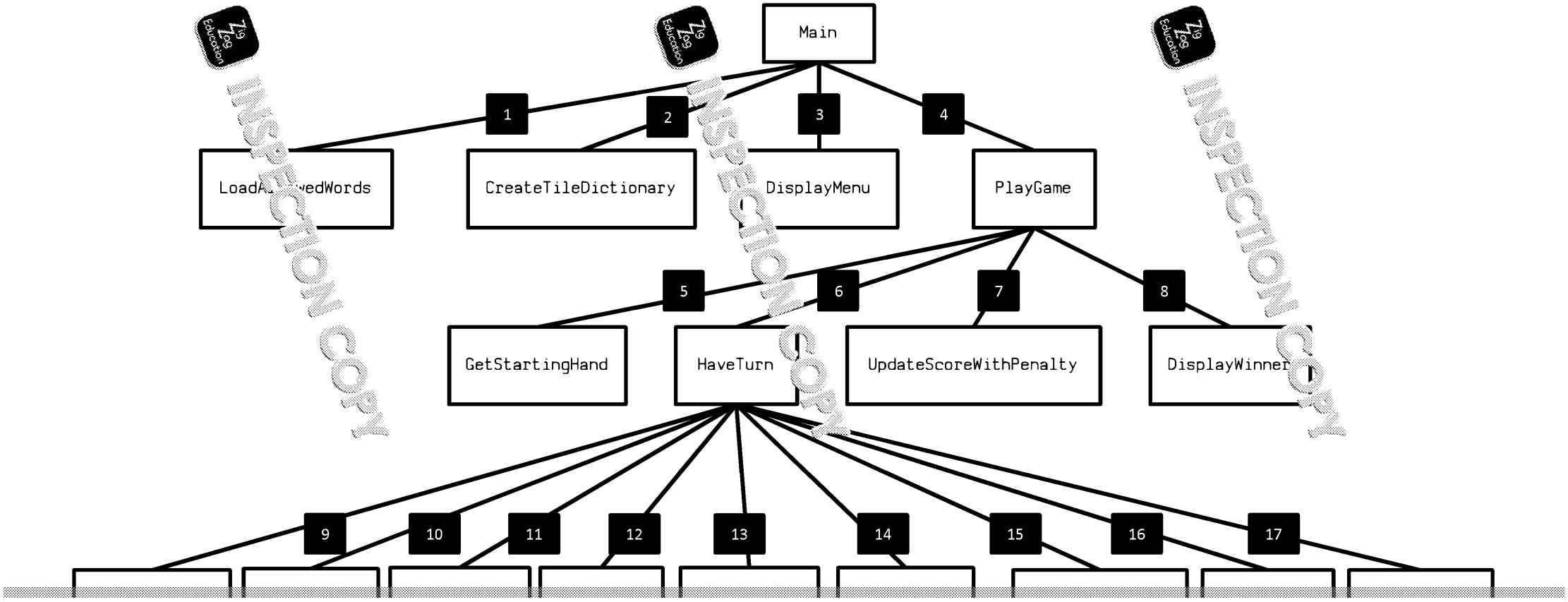
At the end of the game, the total value of each player's hand is subtracted from their score. The highest score is the winner.

# Program Flowchart




* For the details of the 'turn' subroutines, see steps 1 and 2 on th

# Structure Diagram: Overview



# Subroutine Calls, Parameters and Return Values

The numbers to the left do **not** indicate the order in which subroutines are called, as there are multiple possible orders. Instead, these numbers relate to the numbers in the structure diagram on page 3.

| Call | Parameters | Return |
|---|---|---|
| 1 Main calls LoadAllowedWords | - | AllowedWords |
| 2 Main calls CreateTileDictionary | - | TileDictionary |
| 3 Main calls DisplayMenu | - | - |
| 4 Main calls PlayGame | AllowedWords<br>TileDictionary<br>RandomStart<br>StartHandSize<br>MaxHandSize<br>MaxTilesPlayed<br>NoOfEndOfTurnTiles | - |
| 5 PlayGame calls GetStartingHand | TileQueue<br>StartHandSize | Hand |
| 6 PlayGame calls HaveTurn | PlayerName<br>PlayerTiles<br>PlayerTilesPlayed<br>PlayerScore<br>TileDictionary<br>TileQueue<br>AllowedWords | PlayerTiles<br>PlayerTilesPlayed<br>PlayerScore<br>TileQueue |

| Call | Parameters | Return |
|---|---|---|
| 11 HaveTurn calls DisplayTileValues | TileDictionary<br>AllowedWords | - |
| 12 HaveTurn calls FillHandWithTiles | TileQueue<br>PlayerTiles<br>MaxHandSize | TileQueue<br>PlayerTiles |
| 13 HaveTurn calls CheckWordIsInTiles | Word<br>PlayerTiles | InTiles |
| 14 HaveTurn calls CheckWordIsValid | Word<br>AllowedWords | ValidWord |
| 15 HaveTurn calls UpdateAfterAllowedWord | Word<br>PlayerTiles<br>PlayerScore<br>PlayerTilesPlayed<br>TileDictionary<br>AllowedWords | PlayerTiles<br>PlayerScore<br>PlayerTilesPlayed |
| 16 HaveTurn calls GetNewTileChoice | - | NewTileChoice |
| 17 HaveTurn calls AddEndOfTurnTiles | TileQueue<br>PlayerTiles<br>NewTileChoice<br>Choice | TileQueue<br>PlayerTiles |
| 18 UpdateAfterAllowedWord calls GetScoreForWord | Word<br>TileDictionary | Score |

# Program Subroutines

The functions Ⓕ and procedures Ⓟ are described below.

| Main Program Subroutines | Description | |
|---|---|---|
| `AddEndOfTurnTiles`Ⓕ | Parameters: `TileQueue`<br>`PlayerTiles`<br>`NewTileChoice`<br>`Choice`<br>Returns: `TileQueue, PlayerTiles`<br>CalledFrom: `HaveTurn`<br>Calls: `QueueOfTiles.Add`<br>`QueueOfTiles.Remove` | 1. If the user has entered option 1 ( in GetNewTileChoice() ) set `NoOfEndOfTurnTiles` to contain the number of tiles played in the last move<br>3. If the user has entered option 2, set `NoOfEndOfTurnTiles` to contain the number 3<br>4. If the user has entered neither 1 nor 2 set `NoOfEndOfTurnTiles` to contain the number of tiles played plus 3<br>5. Set up a loop that runs once per tile to be drawn<br>6. Add a character to the string containing the player's hand by removing a tile from the front of the tile queue<br>7. Add a tile to the back of the tile queue |
| `CheckWordIsInTiles` Ⓕ | Parameters: `Word`<br>`PlayerTiles`<br>Returns: `InTiles`<br>Called from: `HaveTurn`<br>Calls: - | 1. Create a Boolean variable (`InTiles`) set to true<br>2. Create a copy of the player's tiles<br>3. Loop through the word being checked<br>4. For each character in the word, check that it exists in the copy of player's tiles<br>5. If it is present, it is removed from the copy of the player's tiles<br>6. If it is not present, `InTiles` is set to false<br>7. Return `InTiles` to `HaveTurn` |
| `CheckWordIsValid` Ⓕ | Parameters: `Word` | 1. Create a Boolean variable (`ValidWord`) set to false |

| Main Program – Subroutines | Description | |
|---|---|---|
| `CreateTileDictionary` Ⓕ | Parameters: -<br>Returns: `TileDictionary`<br>Called from: `Main`<br>Calls: - | 1. Create an empty dictionary.<br>2. Set up a loop that runs 26 times<br>3. For each iteration, add the corresponding letter to the map (first add A, then add B, etc.)<br>4. For each letter, add the corresponding score (1, 2, 3 or 5), which depends on the letter<br>5. Return the dictionary to `Main` |
| `DisplayMenu` Ⓟ | Parameters: -<br>Returns: -<br>Called from: `Main`<br>Calls: - | 1. Present options to play the game, play a random start, training start (string literals) or quit (this subroutine does not accept input or return a value) |
| `DisplayTilesInHand` Ⓟ | Parameters: `PlayerTiles`<br>Returns: -<br>Called from: `HaveTurn`<br>Calls: - | 1. Output a blank line<br>2. Output the player's hand |
| `DisplayTileValues` Ⓟ | Parameters: `TileDictionary`<br>`AllowedWords`<br>Returns: -<br>Called from: `HaveTurn`<br>Calls: - | 3. Loop through the `TileDictionary` dictionary<br>4. Display each entry in the dictionary in the form 'Points for A: 1', with each entry on a different line |
| `DisplayWinner` Ⓟ | Parameters: `PlayerOneScore`<br>`PlayerTwoScore`<br>Returns: -<br>Called from: `PlayGame`<br>Calls: | 1. Display 'GAME OVER!' message<br>2. Display "Player One your score is" concatenated with PlayerOneScore<br>3. Display "Player Two your score is" concatenated with |

| Main Program – Subroutines | Description | |
|---|---|---|
| GetChoice Ⓕ | Parameters: -<br>Returns: Choice<br>Called from: HaveTurn<br>Calls: - | 1. Present the player with the mid-game menu<br>2. Prompt the user to select a number or enter a word<br>3. Convert their entry to uppe<br>3. Return this selection to Ha |
| GetNewTileChoi Ⓕ | Parameters: -<br>Returns: NewTileChoice<br>Called from: HaveTurn<br>Calls: - | 1. Declare an empty string<br>2. Prompt the user with a four-op menu<br>3. Return their response to this me HaveTurn; their response is validated to ensure that it is only er "1", "2", "3" or "4", |
| GetScoreForWord Ⓕ | Parameters: Word<br>TileDictionary<br>Returns: Score<br>Called from: UpdateAfterAllowedW<br>Calls: - | 1. Asssign the variable Score, the va zero<br>2. Loop through each character in the , adding the score for each letter (taken as read from the map) to total<br>3. If the length of the word is greater tha add 20 to the score<br>4. If the length of the word is 6 or 7, add 5 the score<br>5. Return the score to UpdateAfterAll edWord |
| GetStartingHand Ⓕ | Parameters: TileQueue<br>StartHandSize<br>Returns: Hand<br>Called from: PlayGame<br>Calls: QueueOfTiles.Add<br>QueueOfTiles.Remove | 1. Create an empty string<br>2. Loop once per tile in startHandSize (in al / 20 times).<br>3. Add a character to the string by removing a t rom the front of the tile queue<br>4. Add a tile to the back of the tile queue<br>5. Return the string (Hand) to PlayGame |

| Main Program – Subroutines | Description | |
|---|---|---|
| `HaveTurn`Ⓕ | Parameters: `PlayerName`<br>`PlayerTiles`<br>`PlayerTilesPlayed`<br>`PlayerScore`<br>`TileDictionary`<br>`TileQueue`<br>`AllowedWords`<br>`MaxHandSize`<br>`NoEndOfTurnTiles`<br>Returns: `PlayerTiles`<br>`PlayerTilesPlayed`<br>`PlayerScore`<br>`TileQueue`<br>Called from: `PlayGame`<br>Calls: `DisplayTilesInHand`<br>`GetChoice`<br>`DisplayTileValues`<br>`FillHandWithTiles`<br>`CheckWordIsInTiles`<br>`CheckWordIsValid`<br>`UpdateAfterAllowedWord`<br>`GetNewTileChoice`<br>`AddEndOfTurnTiles`<br>`QueueOfTiles.Show` | 1. Display which player's turn it is<br>2. Display's the player's hand<br>3. A loop runs prompting the player to enter option "1", "4", "7", "0" or enter a word, via a call to `GetChoice`, until they enter a word or option "0"<br>4. If they enter "1", the value of all tiles are displayed by calling `DisplayTileValues`<br>5. If they enter "4", the tile queue is displayed by calling `QueueOfTiles.Show`<br>6. If they enter "7", redisplay the player's hand by calling `DisplayTilesInHand`<br>7. If they enter "0", fill the player's hand by calling `FillHandWithTiles`<br>8. If they enter none of those options, the assumption is that they have entered a word, so its length is checked<br>9. If the length is 0, a variable called `ValidWord` is set to false<br>10. If the word is invalid based on a call to `CheckWordIsInTiles`, `ValidWord` is set to false<br>11. If the word is valid, the move is processed by calling `UpdateAfterAllowedWord` and `GetNewTileChoice`<br>12. If the word is invalid, a message is displayed saying 'Not a valid attempt. You lose your turn.'<br>13. New tiles are drawn by calling `AddEndOfTurnTiles` unless the player requested no tiles in the call to `GetNewTileChoice` (if the move was invalid, the player has no choice and three new tiles will be drawn)<br>14. Output the current game state |

| Main Program – Subroutines | Description | |
|---|---|---|
| `Main` Ⓟ | Parameters: -<br>Returns: -<br>Called from: -<br>Calls: `LoadAllowedWords`<br>`CreateTileDictionary`<br>`DisplayMenu`<br>`PlayGame` | 1. Game settings are initialised: `MaxHandSize`, `MaxTilesPlayed`, `NoOfEndOfTurnTiles`, `StartHandSize`<br>2. Menu is displayed and user prompted<br>3. Loop continues until "9" is entered to quit<br>4. If "1" is entered, game is played with random tiles<br>5. If "2" is entered, game is played with string literals defined in `PlayGame` |
| `PlayGame` Ⓟ | Parameters: `AllowedWords`<br>`TileDictionary`<br>`RandomStart`<br>`StartHandSize`<br>`MaxHandSize`<br>`MaxTilesPlayed`<br>`NoOfEndOfTurnTiles`<br>Returns: -<br>Called from: `Main`<br>Calls: `GetStartingHand`<br>`HaveTurn`<br>`UpdateScoreWithPenalty`<br>`DisplayWinner`<br>`QueueOfTiles` (constructor) | 1. Set Player One score and Player Two score to 50<br>2. Set number of tiles (for both players) to 0<br>3. Create new tile queue containing 20 tiles<br>4. If a random start has been requested (in `Main`), hands are populated randomly<br>5. Otherwise, hands are populated with string literals<br>6. Loop to run until either player has reached the maximum number of tiles played (50) or the maximum number of tiles in hand (20)<br>7. Call `HaveTurn` alternately for Player One and Player Two until the loop terminates<br>8. Update scores of both players by calling `UpdateScoreWithPenalty` for each player<br>9. Display the winner by calling `DisplayWinner` |
| `UpdateAfterAllowedWord` Ⓕ | Parameters: `Word`<br>`PlayerTiles`<br>`PlayerScore`<br>`PlayerTilesPlayed`<br>`TileDictionary`<br>`AllowedWords` | 1. Add the length of the word just played to the total number of tiles played<br>2. Loop through each character in the played word, removing a corresponding tile from the player's hand<br>3. Update the player's score by calling `GetScoreForWord` |

## Program Classes

The program contains one class. Its purpose is described briefly in the table below.

| Class | Description |
|---|---|
| `QueueOfTiles` | Class to store the structure that contains tiles before they are passed to a player. The structure is a list, and an integer points to the rear of the queue. The front is always element zero in the list. |

## Class Methods

The only class is `QueueOfTiles`. The functions Ⓕ and procedures Ⓟ are described below.

| QueueOfTiles – Methods | Description | |
|---|---|---|
| `__init__` Ⓕ | Parameters: `MaxSize`<br>Returns: `QueueOfTiles`<br>Called from: `PlayGame`<br>Calls: `QueueOfTiles.Add` | 1. Constructor method – create a new `QueueOfTiles` object when called (and return this object)<br>2. Create an empty list to store the queue<br>3. Set the attribute MaxSize to the parameter MaxSize<br>4. Set `_Rear` to -1. This variable is the pointer to the back of the queue. Since the list is initially empty, there cannot be no meaningful rear pointer.<br>5. Call the `Add` method repeatedly, e.g. if `MaxSize` is 20, call `Add` 20 times. |
| `IsEmpty` Ⓕ | Parameters: -<br>Returns: `True or False (Boolean)`<br>Called from: `QueueOfTiles.Remove` | 1. If `rear` is -1 (meaning the pointer is not within the list, so the list can be considered empty) return true<br>2. For any other value of `_Rear`, return false |

| QueueOfTiles – Methods | Description | |
|---|---|---|
| `Add` Ⓟ | Parameters: -<br>Returns: -<br>Called from: `GetStartingHand`<br>`AddEndOfTurnTiles`<br>`FillHandWithTiles`<br>Calls: - | *NB. This subroutine will do nothing if `_Rear` already points to the end of the list, since there would be no room to add a character.*<br>1. Generate a random integer from 0 to 25<br>2. Increment `_Rear` by 1, indicating the queue is now one larger<br>3. At the rear of the queue, add the character whose ASCII code is equal to the randomly generated integer plus 65, e.g. if it was 0, add 'A', if it was 1, add 'B', if it was '2' add 'C', etc. |
| `Show` Ⓟ | Parameters: -<br>Returns: -<br>Called from: `HaveTurn`<br>Calls: - | *NB. This subroutine will do nothing if the queue is empty.*<br>1. Print a blank line<br>2. Print "The contents of the queue are:" followed by<br>3. looping through each item in the queue and printing it out<br>1. Print a blank line |

## Variables

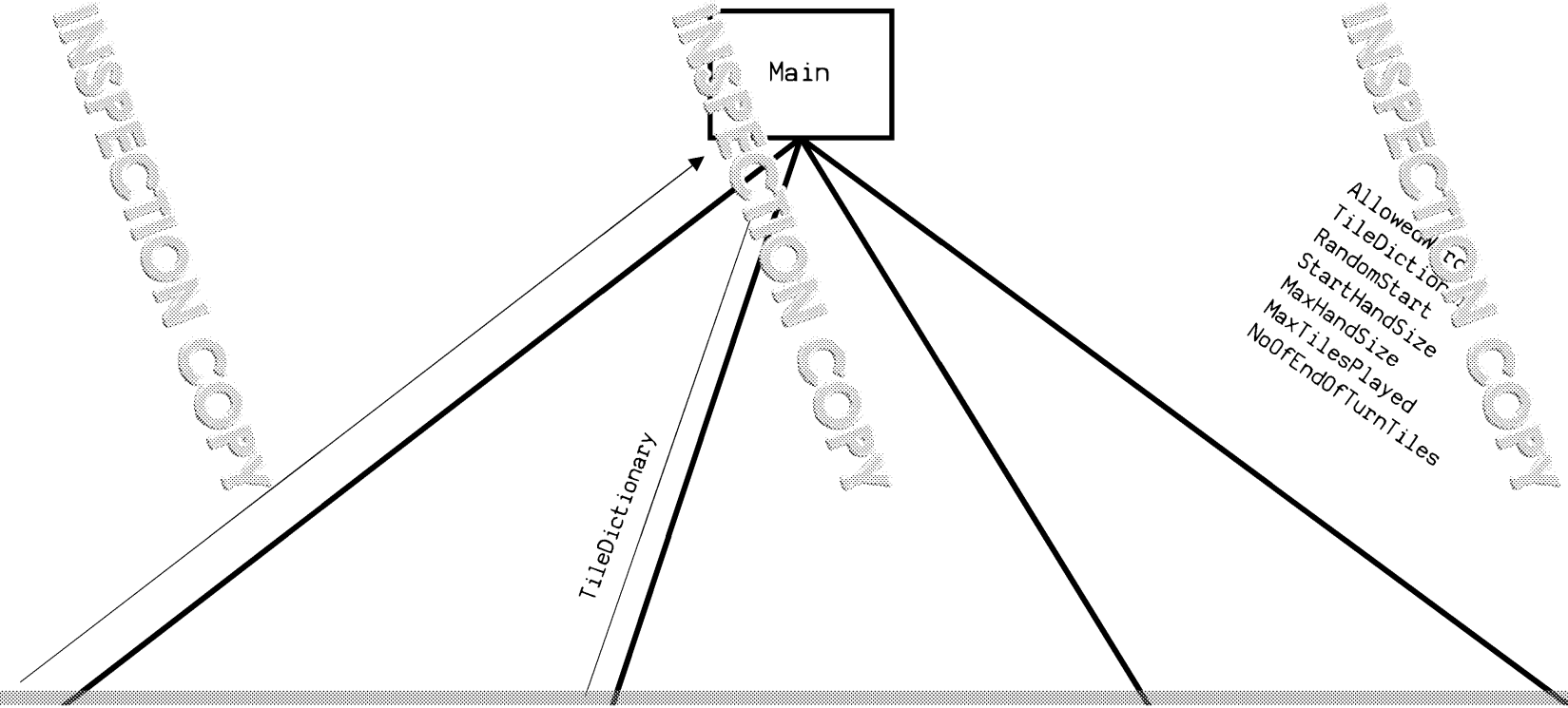
The following table contains variables that are declared locally and passed to at least one other method.

| Main Program – Variables | Type | Description | Created in |
|---|---|---|---|
| `AllowedWords` | `list of strings` | Contains all valid words read from a text file | `Main` via a call to `LoadAllowedWords` subroutine |
| `Choice` | `String` | Contains user input at the main in-game menu, indicating whether they want random hands or the training hands (or to quit). | `Main` |
| `MaxHandSize` | `integer` | The largest number of tiles that a player can hold – | `Main` |

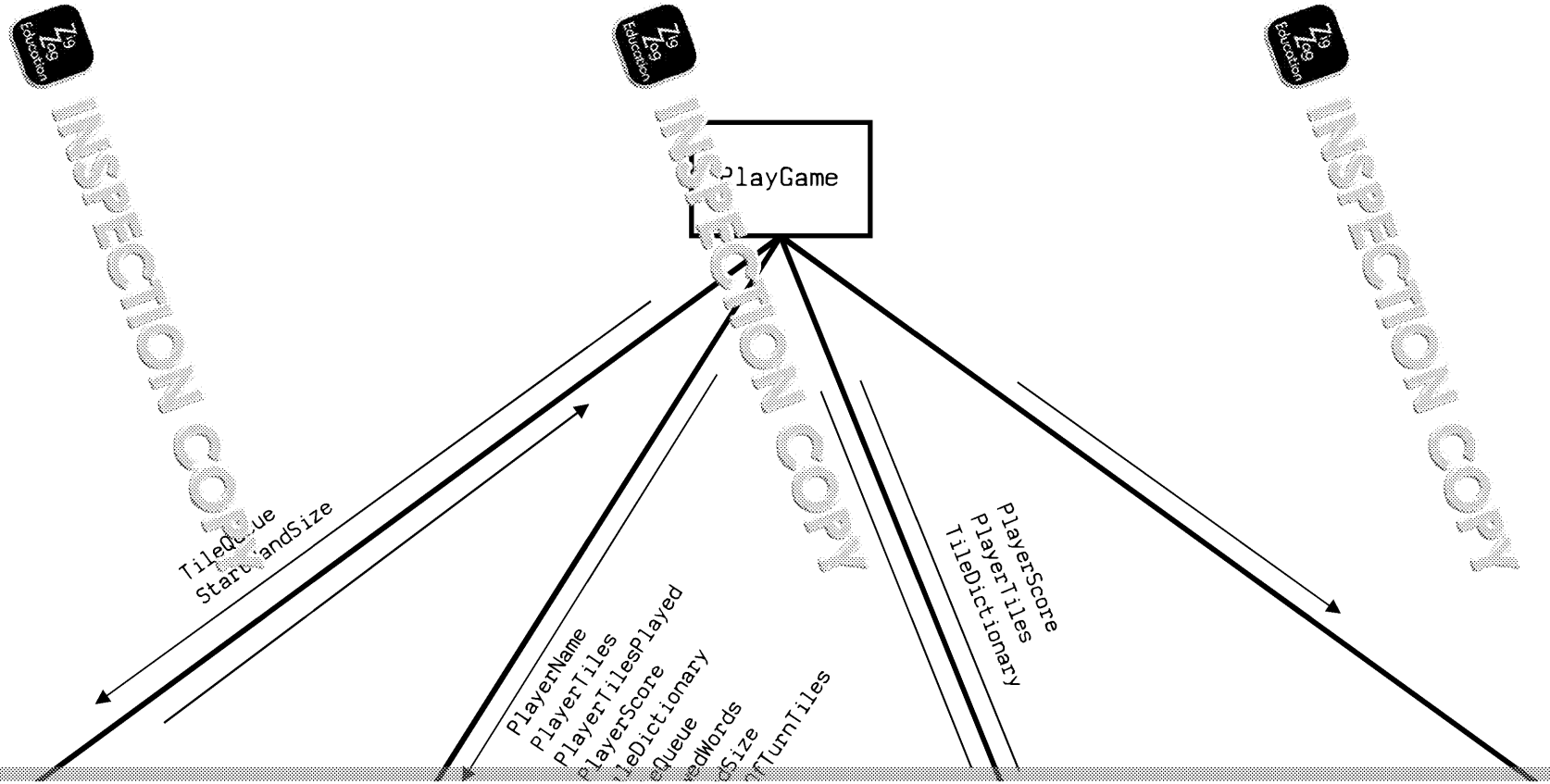
The following table contains the attributes of the `QueueOfTiles` class.

| QueueOfTiles – Attributes | Type | Description | Created in |
|---|---|---|---|
| `_Contents` | `list` | List to contain all letters in the queue before they are passed to a player's hand | `__init__` (the constructor) |
| `_Rear` | `integer` | The index of the back of the queue, used to add new tiles to the correct location in the list | `__init__` (the constructor) |
| `_MaxSize` | `integer` | The largest size that the queue can be | `__init__` (the constructor) |

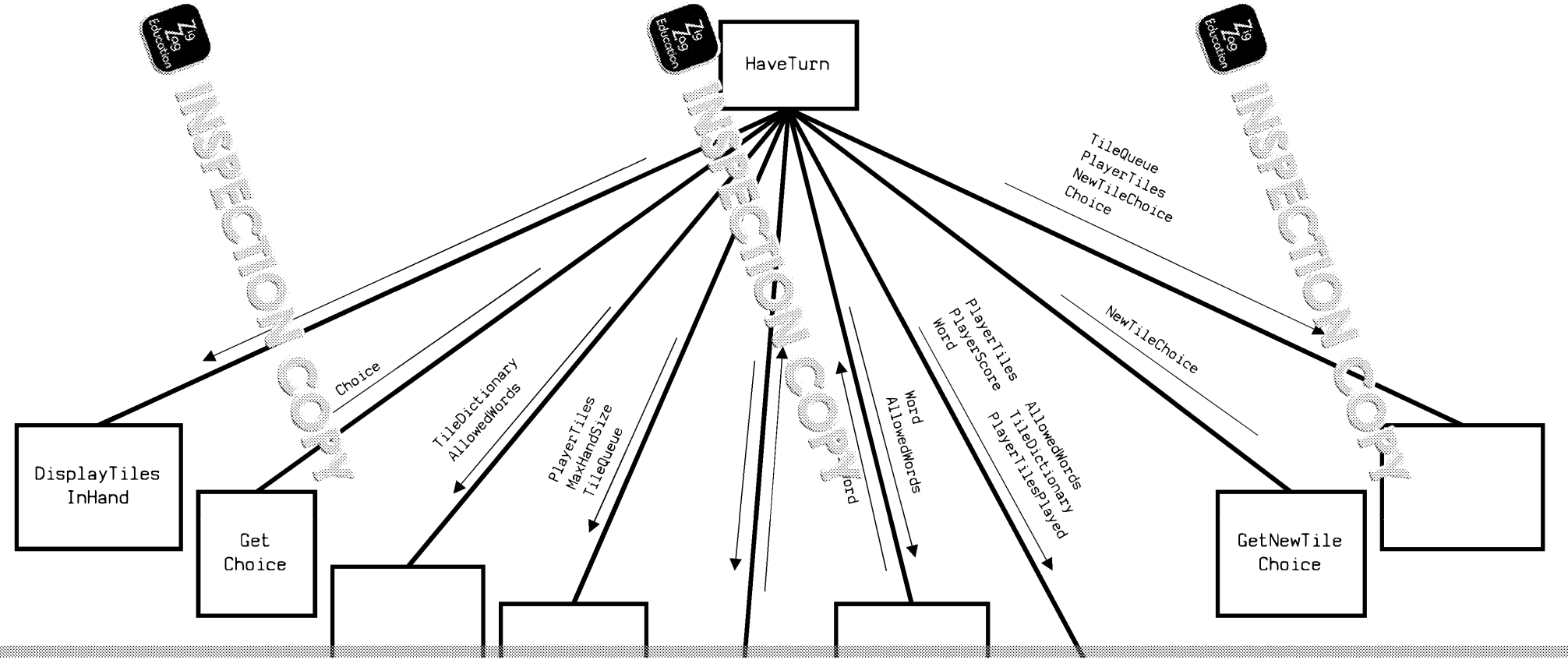
# **Main** and subroutines called from **Main**



# **PlayGame** and subroutines called from **PlayGame**



# **HaveTurn** and subroutines called from **HaveTurn**



# Written Questions (Python)

**These questions refer to the preliminary material and require you to load the skel any additional programming.**

1. State the name of an object for:
   a) [illegible] ss [1]
   b) [illegible] variable [1]
   c) A variable that is used to store a Boolean return value [1]
   d) A parameter whose data type is dictionary [1]
   e) A function with three parameters [1]
   f) A procedure with no parameters [1]
   g) A local variable (excluding parameters) within the `HaveTurn` subro
   h) An attribute of `QueueOfTiles` [1]

2. Write one line of code from the skeleton program which calls a library subrou

3. Look at the subroutine `CreateTileDictionary`. Describe the purpose of

```
TileDictionary[chr(65 + Count)] = 1
```

4. State and describe the data structure returned by the `CreateTileDiction`

5. Look at the subroutine `CheckWordIsInTil` [illegible] ain the role of the vari

6. Describe the difference between a procedure and a function. State one exam code. Do not include any methods from the class `QueueOfTiles` in your an

7. Describe what would happen if, during a call to `LoadAllowedWords`, the fil found. [

8. Describe the actions performed in the following lines of the `LoadAllowed`

```
for Word in WordsFile:
    AllowedWords.append(Word.strip().upper()
```

9. The `QueueOfTiles` class contains a constructor. Describe what is meant by

10. Describe the effect of the following instruction within the `QueueOfTiles` c

```
self._MaxSize = MaxSize
```

11. Explain why the variable `_Rear` is initialised to -1 in the `QueueOfTiles` co

12. Describe in detail the purpose of the subroutine `UpdateScoreWithPenalt` and/or return values in your answer. [5]

13. Describe the operation of the following code within the subroutine `GetScor`

```
Score = 0
for Count in range (len(Word)):
    Score = TileDictionary[Word[Count]]
```

14. Explain the role of the iterative structure within the subroutine `GetNewTile`

15. Explain why the variable `NewTileChoice` is initialised to the string value "2"

# Written Questions (Python)

**These questions refer to the preliminary [illegible] and require you to load the skel[illegible] any additional programming.**

1. State th[illegible]e [illegible]entifier for:

   a) [illegible]ass [1]

   .....

   b) A list variable [1]

   .....

   c) A variable that is used to store a Boolean return value [1]

   .....

   d) A parameter whose data type is dictionary [1]

   .....

   e) A function with three parameters [1]

   .....

   f) A procedure with [illegible] parameters [1]

   .....

   g) [illegible]cal variable (excluding parameters) within the `HaveTurn` subro[illegible]

   .....

   h) An attribute of `QueueOfTiles` [1]

   .....

2. Write one line of code from the skeleton program which calls a library subrou[illegible]

   .....

3. Look at the subroutine `CreateTileDictio[illegible]y`. [illegible]escribe the purpose of

```
TileDictionary[chr(ord(c)+count)] = 1
```

   .....

   .....

   .....

   .....

4. State and describe the data structure returned by the `CreateTileDictio`

.....

.....

.....

.....

.....

.....

5. Look at the subroutine `CheckWordIsInTiles`. Explain the role of the var

.....

.....

.....

.....

6. Describe the difference between a procedure and a function. State one exam code. Do not include any methods from the class `QueueOfTiles` in your a

.....

.....

.....

.....

.....

.....

7. Describe what would happen if, during a call to `LoadAllowedWords`, the f found. [4]

.....

.....

.....

.....

.....

.....

8. Describe the actions performed in the following lines of the `LoadAllowedW`

```
for Word in WordsFile:
    AllowedWords.append(Word.strip().upper()
```

.....

.....

.....

.....

.....

9. The `QueueOfTiles` class contains a constructor. Describe what is meant b

.....

.....

.....

10. Describe the effect of the following instruction within the `QueueOfTiles c`

```
self._MaxSize = MaxSize
```

.....

.....

.....

.....

11. Explain why the variable `_Rear` is initialised to -1 in the `QueueOfTiles` co

.....

.....

.....

12. Describe in detail the purpose of the subroutine `UpdateScoreWithPenal` and/or return values in your answer. [5]

.....

.....

.....

.....

.....

.....

.....

13. Describe the operation of the following code within the subroutine GetSco

```
Score = 0
for Count in range (len(Word)):
    Score += TileDictionary[Word[Count]]
```

.....

.....

.....

.....

.....

14. Explain the role of the iterative structure within the subroutine GetNewTil

.....

.....

.....

15. Explain why the variable NewTileChoice is initialised to the string value "2"

.....

.....

.....

.....

# Programming Tasks (Python)

**The following questions require you to open the skeleton program and make modi**

## Question 1

This question refers to `GetScoreForWord`.

Currently, the source code assigns bonus points for words that contain more than
`GetScoreForWord` so that words of two or three letters incur a one-point pena
'BAR' would normally be worth four points (B=2, A=1, R=1). Following the new ru
be applied, meaning 'BAR' would only be worth three points.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `GetScoreForWord`
- One screen capture showing the word played, the new score and the tota
  sequence of events:
  - Begin the game with the training hand
  - On *Player One*'s first turn, play the wo[illegible]
  - Press '4' to request no new tiles
- One screen capture showing the word played, the new score and the tota
  sequence of events:
  - Begin the game again with the training hand
  - On *Player One*'s first turn, play the word 'BARS'
  - Press '4' to request no new tiles

## Question 2

This question refers to `DisplayTilesInHand` and `HaveTurn`.

The program currently displays the letters in the player's hand as a single string
`DisplayTilesInHand` so that an **additional** line is printed out in which each l
its points value in brackets, followed by a space; for example:

`A(1) F(3) M(2) E(1)` etc.

Modify `HaveTurn` to allow `DisplayTilesInHand` to have access to the point

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `DisplayTilesInHand`
- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing *Player One*'s hand at the beginning of the ga
  hand

## Question 3

This question refers to `CreateTileDictionary`.

Currently, there are no letters worth four points. Modify the code in `CreateTi` letters 'K', 'V' and 'Y' are each worth four points.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `CreateTileDictionary`
- One screen capture showing the letter values after any player's turn

## Question

This question relates to `PlayGame` and `DisplayWinner`.

'Words With AQA' is currently a two-player game. Add code to `PlayGame` to incl should be assigned the same values as *Player One* and *Player Two*. The call to `Di` an additional parameter, and, if *Player Three* has the highest score, they should be

The training hand for *Player Three* should be 'ABCDEFGHIJKLMNO'.

When displaying the winner, the output should be 'Player One wins!', 'Player Two 'No clear winner'. This last message should be displayed if any two players are ti

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `PlayGame`
- Your amended SOURCE CODE PROGRAM for `DisplayWinner`
- One screen capture showing *Player Three*'s turn and the prompt that marks turn (the action taken by *Player Two* is unimportant). Begin with the train

## Question

This question refers to `HaveTurn`.

Presently, if a valid word is played, the program displays the text 'valid word'. M followed, on the same line, with the word and its score. If the player has played points, the output should be as follows:

Valid word. FARM scores 7 points.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing *Player One*'s first turn with the training hand ABANDON

## Question 6

This question refers to a new class, `Player`.

The program does not currently allow efficient creation of additional players. Th
creation of a class called `Player`.

Create a `Player` class that contains private attributes to store that player's tiles,
they have played. There should be a constructor to initialise these attributes to t
`PlayGame`. In order to do this, a `QueueOfTiles` object called `TileQueue` wil
for the constructor.

Create accessor methods within the class to grant public visibility to each of the a

**Evidence you need to provide:**

- The SOURCE CODE for a new class, `Player`

## Question 7

This question refers to `GetChoice` and `HaveTurn`.

Currently, there is no option for the player to swap their letters. Extend the menu
option 'Press 2 to swap all your letters OR'. Modify `HaveTurn` so that all tiles in
and are replaced by an equivalent number of letters from the tile queue. There s
tiles) for changing letters.

Once the letters have been swapped, the new hand should be displayed, but play
player.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `GetChoice`
- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing *Player One*'s hand both before and after sele
  'before' hand should be the training hand.

## Question 8

This question refers to `Add` within the `QueueOfTiles` class.

Modify this subroutine to prevent two indentical consecutive letters being added
be used to cause new letters to be generated (and ignored) until a letter is gener
previous letter. At that point, the letter should be added to the queue.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `Add`

## Question 9

This question refers to `LoadAllowedWords`.

Currently, all words are taken from the `aqawords.txt` file. Modify the `LoadA` that the following takes place:

1. The user is asked to press option '1' for the default dictionary or '2' for a c be validation for this input.
2. If the user presses '1', the `aqawords.txt` file is used as normal.
3. If the user presses '2' they are prompted for the name of a file. The prog the specified file name, and this will be used instead of `aqawords.txt`.
4. If the user enters a file name that cannot be found then the program will inste

You should **only** modify `LoadAllowedWords` in your response to this question.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `LoadAllowedWords`
- One screen capture showing the menu (press '1' for default, press '2' for c program's response to '2' being entered.

## Question 10

This question refers to `Main`.

The program currently attempts to load words from a text file called `aqawords` or not found, the game is allowed to continue. Under these circumstances, howe the game effectively cannot be played.

Modify the `Main` subroutine so that if `LoadAllowedWords` returns an empty lis time error with message "Dictionary file not found", before terminating.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `Main`
- One screen capture showing the entire starting output of the program. P should change `aqawords.txt` in the `LoadAllowedWords` subroutine

## Question 11

This question refers to `HaveTurn`.

The program currently displays a player's hand, but does not display how many le
makes the decision of how many tiles to draw more difficult than it needs to be.

Modify `HaveTurn` so that both before and after new tiles are drawn, the number
displayed. If no new tiles are drawn, the message should only be displayed once
decision not to draw any new tiles. The output should consist of the following fo

```
You have XX tiles remaining.
```

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing the output having done the following:
  - Selected '2' to play with the training hand
  - Played the word BAT for *Player One*
  - Pressed '3' to indicate that you want to replace the tiles played p
- One screen capture showing the output having done the following:
  - Selected '2' to play with the training hand
  - Played the word HAND for *Player One*
  - Pressed '4' to indicate that you want no replacement tiles

## Question 12

This question refers to a new class, `LetterTile`.

The program currently uses characters to represent tiles, with the value of each le
structure. An alternative would be to create a new class, from which objects coul
representing a tile.

Create a new class called `LetterTile`. It should be assembled according to the

- There should be three attributes – `_Letter` (string), `_Score` (integer) a
  last attribute would be set to '`True`' for a vowel (A, E, I, O, U) and '`False`
- The constructor should have two parameters – the letter and the dictiona
  letter. The constructor should set all three attributes correctly. For exam
  attributes would be set as follows:
  - `_Letter`: A (the letter should always be stored in the attribute i
  - `_Score`: 1 (since 'A' is worth a single point)
  - `_IsVowel`: `True`' (since 'A' is a vowel)
- There should be a public accessor method to grant access to each attribu

**Evidence you need to provide:**

- The SOURCE CODE for a new class `LetterTile`.

## Question 13

This question refers to `Main` and `DisplayMenu`.

Presently, the values of `MaxHandSize`, `MaxTilesPlayed`, `NoOfEndOfTurnT` written into the code and cannot be changed by the user.

- Add a menu option in `DisplayMenu` to re[illegible] 'Settings'.
- Alter the code in `Main` so that if '3' is chosen from the menu, the user wi[illegible] following prompts in turn:
  - Enter maximum hand size:
  - Enter maximum tiles played:
  - Enter default new tiles:
  - Enter starting hand size:
- The user input (which will not need to be validated) should go into the va[illegible] `MaxTilesPlayed`, `NoOfEndOfTurnTiles` and `StartHandSize` resp[illegible]

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- Your amended SOURCE CODE PROGRAM for `Main`
- One screen capture showing the following:
  - Option 3 is chosen from the first menu
  - Set maximum hand size to 25
  - Set maximum tiles played to 40
  - Set default new tiles to 2
  - Set starting hand size to 10
  - Begin the game with a random start hand

## Question 14

This questi[illegible]s to the `QueueOfTiles` class.

Presently, letters are chosen purely at random, which can result in a queue and a[illegible] sufficient number of vowels.

Modify the add subroutine and any other necessary code so that letters are select[illegible] order:

- The first letter is chosen purely at random, as is currently the case
- The second letter is also chosen at random
- The third letter is chosen at random from the vowels only (A, E, I, O, U)
- Every letter that is chosen other than the first two and every third letter is [illegible]
- Every third letter should be a randomly chosen vowel (A, E, I , O, U)

It may help you to know that the ASCII values for the vowels are as follows: A=65[illegible]

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for the `QueueOfTiles` class
- One screen capture showing *Player One*'s tiles after choosing to play with a r[illegible]

## Question 15

This question refers to `HaveTurn` and a new subroutine called `ResolveBlank`

The game is currently played without blank tiles. In other games, such as Scrab
part of a word. That blank tile can count for any letter as long as it results in the

Create a new subroutine called `ResolveBlanks`. This subroutine should behav

- It accepts a single variable `Word` which is a string that may or may not c
  play the role of blank tiles
- For each dash encountered, the user is given the prompt 'Enter value of b
  ide for all instances of dash in the word)
- The user inputs a character, which replaces a dash (in the event of multip
  should be replaced in the order in which they appear). No validation is re
- The word, now without any dashes, is returned to `HaveTurn`

You should also modify `HaveTurn` so that `ResolveBlanks` is called immediate
`CheckWordIsValid` is called.

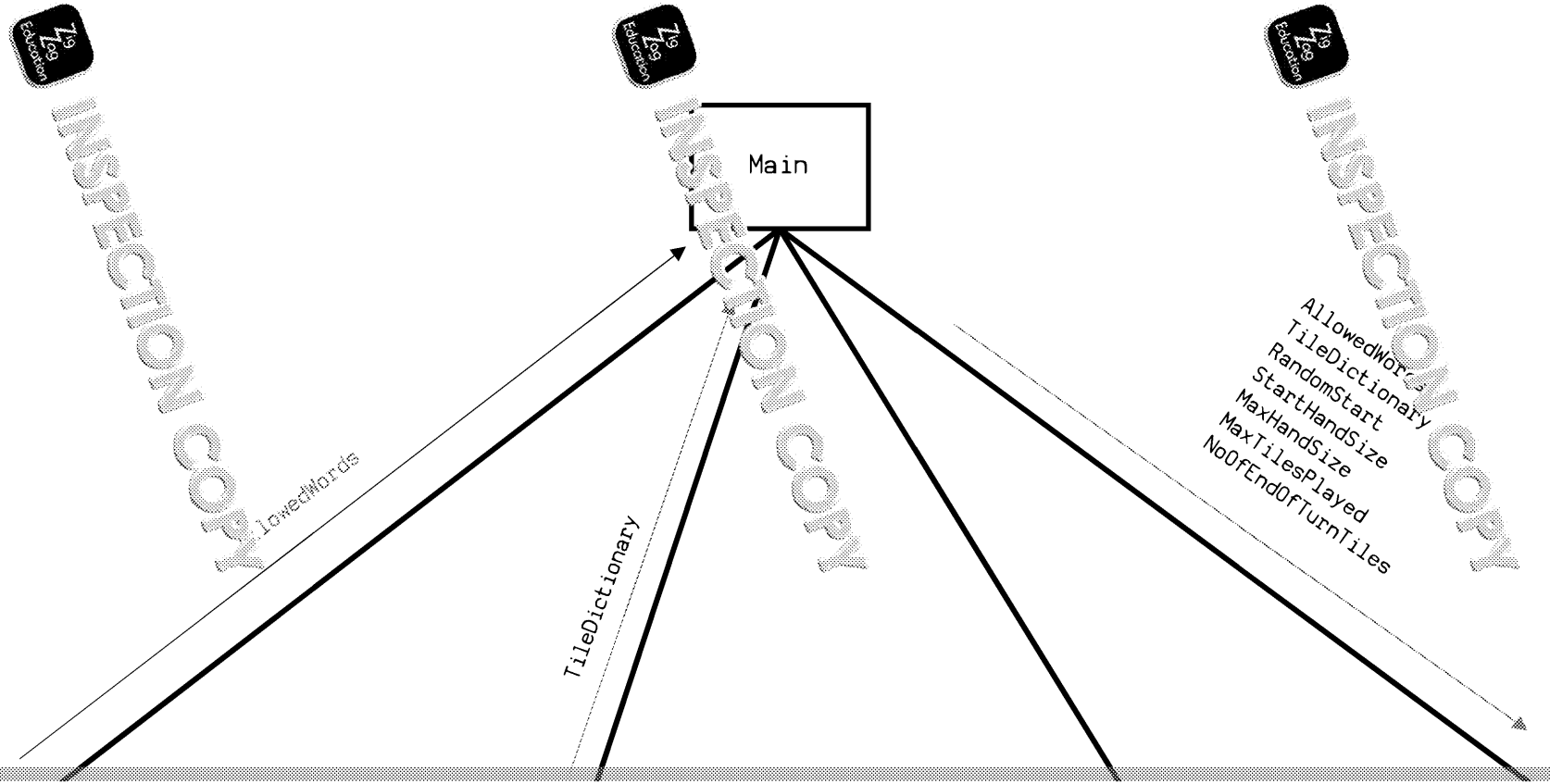
You should also add the "-" tile to `TileDictionary` and assign it a value of 0 p

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- Your SOURCE CODE PROGRAM for the new `ResolveBlanks` subroutine
- One screenshot showing the output that results from the following:
  - Before running the program, change the following line in the `Pl`
    - FROM: `PlayerOneTiles = "TAHANDENONSARJ"`
    - TO: `PlayerOneTiles = "--AHANDENONSARJ"`
      (i.e. change the two characters to dashes)
  - Run the program and select the training hand option
  - Play the following: han--
  - At the first prompt, enter `D`
  - At the second prompt, enter `Y`
- One screenshot showing the output that results from the following:
  - Run the program and select the training hand option
  - Play the following: ha---

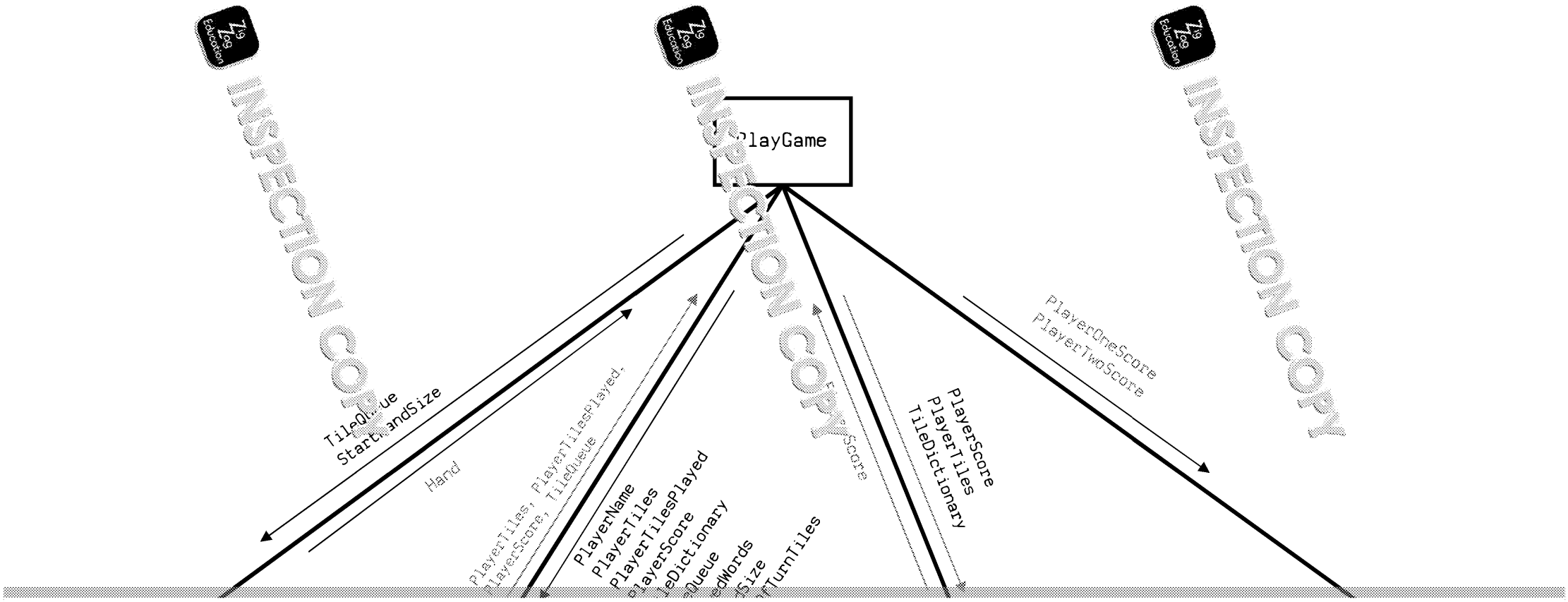
## Structure Diagram 1

# **Main** and subroutines called from **Main**



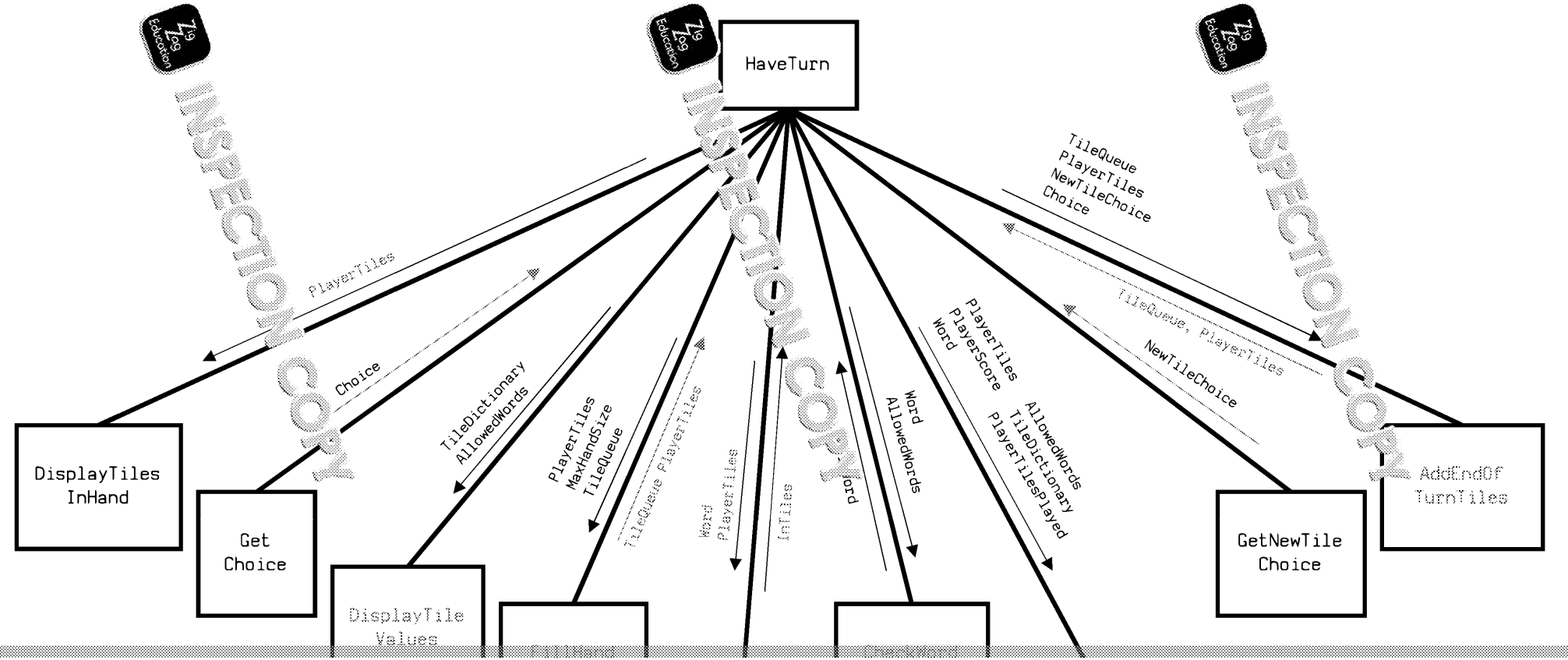
## Structure Diagram 2

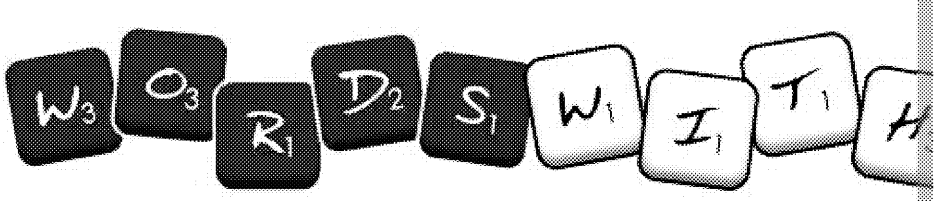
# **PlayGame** and subroutines called from **PlayGame**



## Structure Diagram 3

# **HaveTurn** and subroutines called from **HaveTurn**



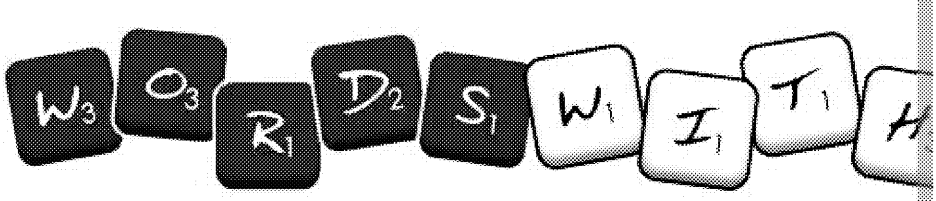


# Written Questions: Suggested Answers and Mark Scheme

| Q | Answer/Guidance |
|---|---|
| 1a | `QueueOfTiles` |
| 1b | `_Contents / AllowedWords` |
| 1c | `InTiles / ValidWord` |
| 1d | `TileDictionary` |
| 1e | `FillHandWithTiles / UpdateScoreWithPenalty` |
| 1f | `DisplayMenu / Main` |
| 1g | `NewTileChoice / ValidChoice / ValidWord / Choice` |
| 1h | `_Contents / _Rear / _MaxSize` |
| 2 | The whole line must be included for the mark:<br>• `RandNo = random.randint(0, 2)` |
| 3 | 1 mark for each point:<br>• Adds (an entry) to dictionary<br>• Entry is the character with ASCII value of 65 + 'count'<br>• Entry includes integer value '1' |
| 4 | 1 mark for stating data structure:<br>• Dictionary<br><br>Up to 3 marks for description:<br>• Contains unique keys<br>• Keys are letters of the alphabet<br>• Each key links to another value<br>• Other value is an integer / the score for that letter<br><br>Also accept key-value pairs for 2 marks (3rd mark for describing key as a letter o |
| 5 | Up to 2 marks for explanation:<br>• Creates a copy of the player's tiles<br>• Characters are eliminated from the copy<br>• Prevents tiles being eliminated from player's tiles (since word has not b |

| Q | Answer/Guidance |
|---|---|
| 6 | 2 marks for difference between procedure and function:<br>• A *procedure* performs a sequence of events but *does not* return a value<br>• A *function* also performs a sequence of events but *does* return a value<br><br>Sufficient for 2 marks: a function returns a value – a procedure does not<br><br>1 mark for identifying a procedure:<br>• `DisplayMenu`<br>• `DisplayTilesInHand`<br>• `DisplayTileBag`<br>• `DisplayBoard`<br>• `[illegible]`<br>• `PlayGame`<br><br>1 mark for identifying a function:<br>• `CheckWordIsInTiles`<br>• `CheckWordIsValid`<br>• `CreateTileDictionary`<br>• `FillHandWithTiles`<br>• `GetChoice`<br>• `GetNewTileChoice`<br>• `GetScoreForWord`<br>• `HaveTurn`<br>• `GetStartingHand`<br>• `LoadAllowedWords`<br>• `UpdateAfterAllowedWord`<br>• `UpdateScoreWithPenalty` |
| 7 | 1 mark each up to 4 marks:<br>• An exception would be thrown<br>• Execution would go to the 'except' block<br>• The code to populate the AllowedWords list would not be executed<br>• The 'except' block contains no code/instructions<br>• An empty list would be returned |
| 8 | 1 mark each up to 4 marks:<br>• Goes through the file one line at a time<br>• Assigning the contents of each line to the variable Word<br>• The Word/line is stripped of any trailing spaces and end of line characters<br>• The Word/variable is converted to upper case<br>• And appended to the list AllowedWords (or list of allowed words) |
| 9 | 1 mark for each up to 2 marks:<br>• A constructor is a method called by creating a variable using the class name Class()<br>• Creates a new object based on the class in which it resides<br>• Constructor method is called __init__<br>• Ensures that a new object in the class is initialised correctly |
| 10 | 2 marks:<br>• Sets the instance variable (`_MaxSize`)<br>• [illegible] to the parameter (`MaxSize`) |
| 11 | 2 marks:<br>• `rear` points to the back of the queue<br>• Value of -1 indicates an empty queue<br>• -1 is a rogue/special value |

| Q | Answer/Guidance |
|---|---|
| 12 | 1 mark for parameters:<br>• Player's current score, tiles in the player's hand, `TileDictionary` th<br><br>1 mark for return value<br>• Player's updated score<br><br>Up to 3 marks from the following:<br>• Purpose of function is to subtra[illegible] al[illegible] player's tiles/hand from thei<br>• Loop is established to i[illegible] through each tile/character in the playe<br>• Value of each [illegible] determined by looking up in a dictionary<br>• Value i[illegible] ted from player's score |
| 13 | 4 mar<br>• Score/integer/variable is set to zero<br>• Loop iterates through each character in the word<br>• Value of character looked up in the (Tile)Dictionary<br>• Value added to score |
| 14 | 2 marks<br>• Loop continues until user has selected '1', '2', '3' or '4'<br>• Validates user input |
| 15 | 3 marks<br>• Contents of this variable are passed to `AddEndOfTurnTiles`<br>• Selection of '2' indicates that three new tiles will be drawn<br>• The only way to replace this selection is to have played a valid word |
| **TOTAL MARKS** | |



# Programming Tasks: Suggested Solutions and Mark Scheme

*That the following are recommended solutions, and not an exhaustive list of all possible so guidance should be used as a guide only. Discretion should be used in awarding credit wh*

## Question 1

**1 mark** An IF statement that evaluates to TRUE for a word length of either two

**1 mark** The IF statement evaluates to TRUE for a word length of two or three le

**1 mark** Score decremented correctly within the IF statement

```
elif len(Word) in [2,3]:
  Score -= 1
```

**1 mark** Screenshot shows 'HAD' was played, the new total is '55' and the total t

```
Your word was: HAD
Your new score is: 55
You have played 3 tiles so far in this g
```

**1 mark** Screenshot shows 'BARS' was played, the new total is '55' and the total

```
Your word was: BARS
Your new score is: 55
You have played 4 tiles so far in this g
```

## Question 2

**1 mark** `DisdisplayTilesInHand` uses additional parameter containing the *Tile*
means has been used to grant access to the dictionary, but marks 6 and
not be available)

**1 mark** Loop to iterate through each character in the p's hand

**1 mark** Display the character

**1 mark** Display the value of the letter taken from the `TileDictionary`

**1 mark** C for string, to include brackets and a single space after each clo

```
def DisplayTilesInHand(PlayerTiles, TileDicti
  print
  sys.stdout.write("\nYour current hand: " +
  sys.stdout.write("With values: ")
  for letter in PlayerTiles:
    sys.stdout.write(letter+" "+str(TileDicti
  print
```

**1 mark** Initial call for player's hand from `HaveTurn` uses new argument corre

```
DisplayTilesInHand(PlayerTiles, TileDict
```

**1 mark** Second call for player's hand from `HaveTurn` uses new argument corr

```
elif Choice == "7":
  DisplayTilesInHand(PlayerTiles, TileDi
```

**1 mark** Screenshot showing correct output format, which should also include t

```
Your current hand: BTAHANDENONSARJ
With values: B (2) T (1) A (1) H (3) A (1) N (1) D (2) E (1) N (1) O (
```

## Question 3

**1 mark** Removal of values 10, 21 and 24 from the selection structure that assig

**1 mark** Inclusion of values 10, 21 and 24 in a new selection structure

**1 mark** These values, and only these values, assigned a score of 4

```
elif Count in [5, 7, 2[illegible]:
  TileDictionary[chr(65 + Count)] = 3
elif Count in [10, 21, 24]:
  TileDictionary[chr(65 + Count)] = 4
```

**1 mark** Screenshot shows that K, V and Y are worth four points each

```
TILE VALUES

Points for A: 1
Points for B: 2
Points for C: 2
Points for D: 2
Points for E: 1
Points for F: 3
Points for G: 2
Points for H: [illegible]
Points for I: 1
Points for J: 5
Points for K: 4
Points for L: 2
Points for M: 2
Points for N: 1
Points for O: 1
Points for P: 2
Points for Q: 5
Points for R: 1
Points for S: 1
Points for T: 1
Points for U: 2
Points for V: 4
Points for W: 3
Points for X: 5
Points for Y: 4
Points for Z: 5
```

## Question 4

**1 mark** Adding a score for *Player Three*

```
PlayerThreeScore = 50
```

**1 mark** Adding a tile count for *Player Three* and setting it to zero

```
PlayerThreeTilesPlayed = 0
```

**1 mark** *Player Three*'s hand filled with random tiles

```
if RandomStart:
    PlayerOneTiles = GetStartingHand(TileQueu
    PlayerTwoTiles = GetStartingHand(TileQueu
    PlayerThreeTiles = GetStartingHand(TileQu
```

**1 mark** *Player Three*'s hand filled with tiles of letters A–O in the 'else' block

```
else:
    PlayerOneTiles = "BTAHANDENONSARJ"
    PlayerTwoTiles = "CELZXIOTNESMUAA"
    PlayerThreeTiles = "ABCDEFGHIJKLMNO"
```

**1 mark** Logic expression in 'while' loop include tiles played for *Player Three*

**1 mark** Logic expression includes check for size of hand and all logic is sound

```
while PlayerOneTilesPlayed <= MaxTilesPlayed and PlayerTwoTilesPlayed <= MaxTilesPlayed and Play
    and len(PlayerOneTiles) < MaxHandSize and len(PlayerTwoTiles) < MaxHandSize and len(Player
```

**1 mark** Call to `HaveTurn` with variables for tiles, tiles played and score for *Pla*

```
HaveTurn("Player Three", PlayerThreeTiles, PlayerThreeTilesPlayed, PlayerThreeScore, TileDictionary, TileQueue, A
```

**1 mark** Call to `UpdateScoreWithPenalty` for *Player Three*

```
PlayerThreeScore = UpdateScoreWithPenalty(PlayerThreeScore, Player
```

**1 mark** Call to `DisplayWinner` passes scores for all three players

```
DisplayWinner(PlayerOneScore, PlayerTwoScore
```

**1 mark** Subroutine `DisplayWinner` has three parameters instead of two

**1 mark** Score for *Player Three* is displayed

```
def DisplayWinner(PlayerOneScore, PlayerTwoScore, PlayerT
    print "**** GAME OVER! ****"
    sys.stdout.write("Player One your score is " + str(Play
    sys.stdout.write("Player Two your score is " + str(Play
    sys.stdout.write("Player Three your score is " +  str(P
```

**1 mark** Correct logic for displaying 'Player One wins!'

**1 mark** Correct logic for displaying 'Player Two wins!'

**1 mark** Correct logic for displaying 'Player Three wins!'

**1 mark** 'No clear winner' displayed in 'else' block (**A:** if this has been written as
covers all other combinations; **R:** if other text is displayed)

```
if PlayerOneScore > max([PlayerTwoScore, PlayerT
  print "Player One wins!"
elif PlayerTwoScore > max([PlayerOneScore, Playe
  print "Player Two wins!"
elif PlayerThreeScore > max([PlayerOneScore, Pla
  print "Player Three wins!"
else:
  print "No clear winner!"
print
```

**1 mark** Prompt for *Player Three* to move after *Player Two* has moved with *Player*
(ABCDEFGHIJKLMNO)

```
Player Two it is your turn.

Your current hand: CELZXIOTNESMUAA

Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 2 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>plague


Not a valid attempt, you lose your turn.


Your word was: PLAGUE
Your new score is: 50
You have played 0 tiles so far in this game.

Press Enter to continue


Player Three it is your turn.

Your current hand: ABCDEFGHIJKLMNO
```

## Question 5

**1 mark** Use of a variable to store the score for the word (**A:** if no variable is use
`GetScoreForWord` forms part of the string concatenation to display '
14 points.')

**1 mark** Call to `GetScoreForWord` to either initialise the variable or place the
concatenated string

**1 mark** Correct parameters – `ChecksAgainstDictionary`

**1 mark** Concatenation incorporates all components stated in the question, incl
(**A:** full stop is not included, difference in case). Concatenation mus
the variable used to store the score (if a variable was used).

```
if ValidWord:
  print
  print "Valid word. " + Choice + " scores " + str(GetScoreFor
  print
```

**1 mark** Input of word 'abandon' displays score worth 14 points (**DPT:** spacing e

```
Valid word. ABANDON scores 14

Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get thre
     get a new tiles (4)?
```

## Question 6

**1 mark** Class declaration (**R:** if name or case incorrect)

```
class Player:
```

**1 mark** Attributes declared with appropriate names (or alternative names if mea

**1 mark** All three attributes initialised to the correct type of data (**R:** if any additi attributes only declared (first used) inside the constructor)

```
_Score = 50
_NumberOfTiles = 0
_Tiles = ""
```

**1 mark** Constructor declared with single parameter (in addition to `self`) `Tile`

**1 mark** `_Score` and `_NumberOfTiles` (or their equivalents) initialised to `50` initialising `_Score` and `_NumberOfTiles` if there were initialised wi

**1 mark** `_Tiles` (or its equivalent) initialised using a call to `GetStartingHan`

**1 mark** `GetStartingHand` contains correct parameters (**A:** **positive** integers

```
def __init__(self, TileQueue):
  self._Score = 50
  self._NumberOfTiles = 0
  self._Tiles = GetStartingHand(TileQueue, 15)
```

**1 mark** Accessor methods (get and set) for _Score

```
def GetScore(self):
  return self._Score

def SetScore(self, score):
  self._Score = score
```

**1 mark** Accessor methods (get and set) for _NumberOfTiles

```
def GetNumberOfTiles(self):
  return self._NumberOfTiles

def SetNumberOfTiles(self, NumberOfTiles):
  self._NumberOfTiles = NumberOfTiles
```

**1 mark** Accessor methods (get and set) for _Tiles

```
def GetTiles(self):
  return self._Tiles

def SetTiles(self, Tiles):
  self._Tiles = tiles
```

## Question 7

**1 mark** Addition of the extra option in `GetChoice`

```
def GetChoice():
  print
  print "Either:"
  print "     enter the word you would like to play OR"
  print "     press 1 to display the letter values OR"
  print "     press 2 to swap all your letters OR"
  print "     press 4 to view the tile queue OR"
  print "     press 7 to view your tiles again OR"
  print "     press 0 to fill hand and stop the game."
```

**1 mark** Inclusion of 'elif' clause to deal with 'choice' being '2'

**1 mark** Variable to temporarily store the new letters (**A:** valid repurposing of `P`

**1 mark** Loop that runs once per letter in the original hand

**1 mark** Calling `Remove` and appending the character to the player's hand

**1 mark** Calling `Add` to keep the tile queue full

**1 mark** `PlayerTiles` contains the new letters

**1 mark** `ValidChoice` set to 'true' to prevent the main loop in `HaveTurn` rep

**1 mark** `NewTileChoice` set to '4' to ensure that no new tiles are taken on to

**1 mark** Output of new hand

```
elif Choice == "2":
  NewPlayerTiles = ""
  for ReplaceTile in PlayerTiles:
    NewPlayerTiles = TileQueue.Remove()
    TileQueue.Add()
  PlayerTiles = NewPlayerTiles
  ValidChoice=True
  NewTileChoice="4"
  print "Your new hand is: " + PlayerTiles
```

**1 mark** Output to show the original hand, selection of '2' from the menu and th comprise random letters)

```
==============
MAIN MENU
==============

1. Play game with random start hand
2. Play game with training start hand
9. Quit

Enter your choice: 2

Player One it is your turn.

Your current hand: BTAHANDTAKA

Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 2 to swap all your letters OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>2

Your new hand: WKQRXKBNUQKPFZA

Press Enter to continue
```

## Question 8

**1 mark** Suitable iterative structure used

**1 mark** Generation of a new letter exists inside the loop

**1 mark** Termination of loop depends on correctly choosing a letter that is not in that was added to the queue

**1 mark** Incrementation of '_Rear' inside the loop

**1 mark** New letter is correctly added to the rear of the queue

```
def Add(self):
  if self._Rear < self._MaxSize - 1:
    RandNo = random.randint(0, 25)
    while self._Contents[self._Rear] ==
      RandNo = random.randint(0, 25)
    self._Rear += 1
    self._Contents[self._Rear] = chr(65
```

## Question 9

**1 mark** Options displayed to user

**1 mark** Declaration of variable to store input from this menu

**1 mark** Input assigned to variable

**1 mark** Declaration of variable to store file name

**1 mark** A choice of '1' on the previous menu will cause `aqawords.txt` to be l

**1 mark** A [illegible] results in the user being prompted for a file name

**1 mark** Response of other than "1" or "2" results in them being asked to choose

**1 mark** String '.txt' appended to the user input file

**1 mark** Failure to open the file that they chose results in the `aqawords.txt` f

```
def LoadAllowedWords():
  AQAfile = "aqawords.txt"
  AllowedWords = []
  print
  print "1: Default Dictionary"
  print "2: Custom Dictionary"
  Answer = raw_input("> ")
  while Answer not in ["1","2"]:
    Answer = raw_input("Please choose either 1 or 2: ")
  if Answer == "1":
    FileName = AQAfile
  else:
    FileName = raw_input("Please enter the file name (without
  try:
    WordsFile = open(FileName, "r")
  except:
    print "File not found: " + FileName +" - using default i
    WordsFile = open(AQAfile, "r")
  try:
    for Word in WordsFile:
      AllowedWords.append(Word.strip().upper())
    WordsFile.close()
  except:
    pass
  return AllowedWords
```

**1 mark** Output shows the new menu, selection of '2' and prompting for the file

```
1: Default Dictionary
2: Custom Dictionary
> 2
Please enter the file name (without the .txt): testwords
```

## Question 10

**1 mark** 'if' clause after call to `LoadAllowedWords`

**1 mark** 'if' clause compares list length correctly, i.e. == 0 or <1

**1 mark** Suitable output message

**1 mark** Exception correctly raised (and of the correct type)

```
if len(AllowedWords) == 0:
    raise RuntimeError("Dictionary file not found")
```

**1 mark** Output displays welcome message followed by 'Dictionary file not found' and has ended

```
++++++++++++++++++++++++++++++++++++++
+ Welcome to the WORDS WITH AQA game +
++++++++++++++++++++++++++++++++++++++


Traceback (most recent call last):
  File "/Users/appler/Documents/ZigZag/AQA prelim 2017-18/Python
/Python3 Q10.py", line 288, in <module>
    Main()
  File "/Users/appler/Documents/ZigZag/AQA prelim 2017-18/Python
/Python3 Q10.py", line 272, in Main
    raise RuntimeError("Dictionary file not found")
RuntimeError: Dictionary file not found
>>>
```

## Question 11

**1 mark** Message correctly placed (even if message is incorrect) before call to G

```
PlayerTiles, PlayerScore, PlayerTilesPlayed = UpdateA
print "You have " + str(len(PlayerTiles)) + " tiles
NewTileChoice = GetNewTileChoice()
```

**1 mark** Message correctly placed (ev[illegible]sage is incorrect) immediately aft[illegible] AddEndOfTurnTi[illegible] (**A**: [illegible]ode is added outside the 'if' clause, but [illegible] clause to pr[illegible]e [illegible]essage being redisplayed in the event that no n[illegible]

**1 mark** St[illegible]ncatenation to display message in the specified format in both

```
if NewTileChoice != "4":
  TileQueue, PlayerTiles = AddEndOfTurnTiles(TileQueu
  print "You have " + str(len(PlayerTiles)) + " tiles
```

**1 mark** Output showing 12 tiles before the draw and 18 afterwards

```
Valid word

You have 12 tiles remaining.
Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get [illegible] extra tiles (3)
     get no new tiles (4)?
>3
You have 18 tiles rem[illegible]

Your word [illegible]T
Y[illegible]w [illegible]ore is: 54
Yo[illegible] played 3 tiles so far in this game.

Press Enter to continue
```

**1 mark** Output showing 11 tiles before the draw, then not displaying the messa[illegible] be 'Your word was: HAND'

```
Valid word

You have 11 tiles remaining.
Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and [illegible]re extra tiles (3)
     get no new tiles (4)?
>4

Your word [illegible]N
Y[illegible]w [illegible]e is: 57
Yo[illegible] played 4 tiles so far in this game.

Press Enter to continue
```

## Question 12

**1 mark** Class declaration (**R:** if name or case incorrect)

**1 mark** Attributes declared with appropriate names (**R:** if any other names are u

**1 mark** Constructor has a parameter for the letter (**A:** if named other than 'Lett

**1 mark** Constructor has a parameter for the dictionary (**A:** if named other than

**1 mark** _Letter attribute set using letter parameter

**1 mark** Attribute always contains upper-case version of character

**1 mark** _Score attribute set by extracting a value from the dictionary (even if incorrect)

**1 mark** Selection to isolate either vowels or consonants (even if it would not w

**1 mark** Selection would always isolate vowels/consonants, bearing in mind that upper case or lower case

**1 mark** _isVowel attribute set correctly

```
class LetterTile:
  _Letter = "" # a single character string containing the
  _Score = 0 # the score value for the letter
  _IsVowel = False # True if the letter is A, E, I O or U

  def __init__(self, Letter, TileDictionary):
    self._Letter = Letter.upper()
    self._Score = TileDictionary[Letter]
    self._IsVowel = self._Letter in ["A","E","I","O","U"]
```

**1 mark** Access methods with appropriate names (**A:** any reasonable/sensible

```
def GetLetter(self):
  return _Letter

def GetScore(self):
  return _Score

def IsVowel(self):
  return _IsVowel
```

## Question 13

**1 mark** New entry in `DisplayMenu`

```
print "1. Play game with random start hand"
print "2. Play game with training start hand"
print "3. Settings"
print "9. Quit"
```

**1 mark** 'elif' added to main tr[illegible] entry of '3'

**1 mark** Prom[illegible] new inputs (**R:** alternative wording) (A: doing this e[illegible] s[illegible] function)

```
elif Choice == "3":
  MaxHandSize, MaxTilesPlayed, NoOfEndOfTurnTiles, StartHandSi
```

**1 mark** Attempt (even if unsuccessful) to convert inputs to integers

**1 mark** Syntactically valid string → integer conversion for all inputs

**1 mark** Input prompts relate correctly to all four variables

```
def GetSettings():
  MaxHandSize = int(input("Enter maximum hand size: "))
  MaxTilesPlayed = int(input("Enter maximum tiles played
  NoOfEndOfTurnTiles = int(input("Enter default new tile
  StartHandSize = int(input("Enter starting hand size: "))
  return MaxHandSize, MaxTilesPl[illegible]d, [illegible]oOfEndOfTurnTiles,
```

**1 mark** Output showing values [illegible] and 10 entered, with 10 indicating the hand should [illegible] characters long instead of 15

```
=[illegible]
M[illegible]U
==========

1. Play game with random start hand
2. Play game with training start hand
3. Settings
9. Quit

Enter your choice: 3
Enter maximum hand size: 25
Enter maximum tiles played: 40
Enter default new tiles: 2
Enter starting hand size: 10

==========
MAIN MENU
==========

1. Play game with [illegible] hand
2. Play game [illegible] start hand
3 [illegible]
9 [illegible]
Enter your choice: 1

Player One it is your turn.

Your current hand: TMWPVBXCOV
```

## Question 14

**1 mark** Variable to track the iterations so that a vowel is guaranteed every third

```
_NumTilesAdded = 0
```

**1 mark** `_Rear` is still incremented before any letter is added

**1 mark** Selection structure to determine whether to add a vowel or a random le every third time

**1 mark** New variable changed to ensure the switch between selecting a vowe current `_NumTilesAdded` in this example, but any equivalent app

**1 mark** Random letter still correctly added to the array

**1 mark** Random number generator selects from vowels, giving each equal prob

**1 mark** Vowel is correctly added to the array

**1 mark** Variable is changed to ensure that the next selection will be a random l

```
def Add(self):
  if self._Rear < self._MaxSize - 1:
    RandNo = random.randint(0, 25)
    if self._NumTilesAdded >=3:
      if self._NumTilesAdded % 3 == 0: #choose a vowel
        while RandNo not in [0, 4, 8, 14, 20]:
          RandNo = random.randint(0, 25)
      else: # choose a consonant
        while RandNo in [0, 4, 8, 14, 20]:
          RandNo = random.randint(0, 25)
    self._NumTilesAdded += 1
    self._Rear += 1
    self._Contents[self._Rear] = chr(65 + RandNo)
```

**1 mark** Selecting the random starting hand should display every third letter as a letters could be anything)

```
Enter your choice: 1

Player One it is your turn.

Your current hand: KEIMQEHXUDBUCLU
```

## Question 15

**1 mark** Call to `ResolveBlanks`, with Choice as an argument, before the call t

**1 mark** Value returned from call to `ResolveBlanks` stored in a new variable

**1 mark** New variable passed as the first argument to `CheckWordIsValid`

```
else:
  ValidWord = CheckWordIsValidGivenTiles(Choice, PlayerTil
if ValidWord:
  WordToPlay = ResolveBlanks(Choice)
  ValidWord = CheckWordIsValid(WordToPlay, Allowed
```

**1 mark** "-" letter added to `TileDictionary` with a value of 0 (by modifying C

```
  else:
    TileDictionary[chr(65 + Count)] = 5
TileDictionary["-"] = 0
return TileDictionary
```

**1 mark** Method declaration for `ResolveBlanks` with one parameter (**R:** any o variation in case) (**A**: Any sensible name for the parameter)

**1 mark** Variable to store user input for the value of a blank tile

**1 mark** Loop to ensure all dashes are found (**R:** if pr ould fail in the abs

**1 mark** Selection statement to handle m sence of a dash

**1 mark** User is prompt er value of blank tile:' (**R:** alternative wording

**1 mark** U t is stored in variable

**1 mark** Input is converted to upper case

**1 mark** Dash would be replaced by the user input in all cases

**1 mark** Structure ensures that all dashes would be replaced by characters enter

**1 mark** Word correctly returned (with letters instead of blanks)

```
def ResolveBlanks(PlayerWord):
  Word = ""
  for Letter in PlayerWord:
    if Letter == "-":
      Letter = raw_input("Enter value of blank tile
    Word += Letter
  return Word
```

**1 mark** Output for han--, followed by entering ‘d’ then ‘y’, results in ‘Valid word’

```
>han--

Enter value of blank tile:d
Enter value of blank tile:y

Valid word

Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get three extra tiles (3)
     or no new tiles (4)?
>4

Your word was: HAN--
Your new score is: 55
You have played 5 tiles so far in this game.

Press Enter to continue
```

**1 mark** Output for `ha---` results in ‘Not a valid attempt, you lose your turn.’

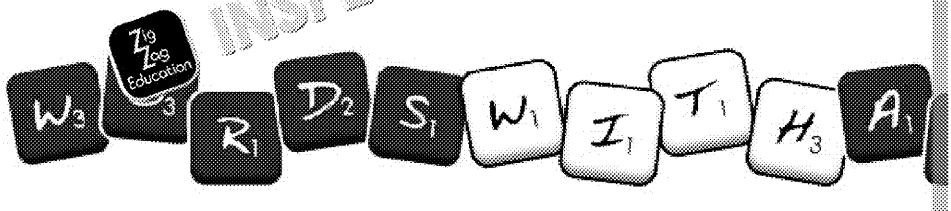
```
Your current hand: --AHANDENONSARJ

Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to give up hand and stop the game.
>

Not a valid attempt, you lose your turn.


Your word was: HA---
Your new score is: 50
You have played 0 tiles so far in this game.

Press Enter to continue
```

| Name | |
|---|---|

ZigZag Education supporting

# A Level AQA Computer Science Pap

# Summer 2018



## Electronic Answer Document (EAD)

**Instructions**

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and ar additional pages

- Answer **all** questi
- The ma ailable for each question are shown in brackets

- You will need:
  - access to a computer
  - access to a printer
  - access to appropriate software
  - electronic copies of the required skeleton code
  - EAD (Electronic Answer Document)

**Total marks:**

# Written Questions

Answer all questions.
Remember to save this document regularly.

| Q | | Answer |
|---|---|---|
| 1 | (a) | |
| | (b) | |
| | (c) | |
| | (d) | |
| | (e) | |
| | (f) | |
| | (g) | |
| | (h) | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |
| 11 | | |
| 12 | | |
| 13 | | |
| 14 | | |
| 15 | | |

# Programming Tasks

Answer all questions.
Remember to save this document regularly.

| Q | Answer |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | |