Zig Zag Education

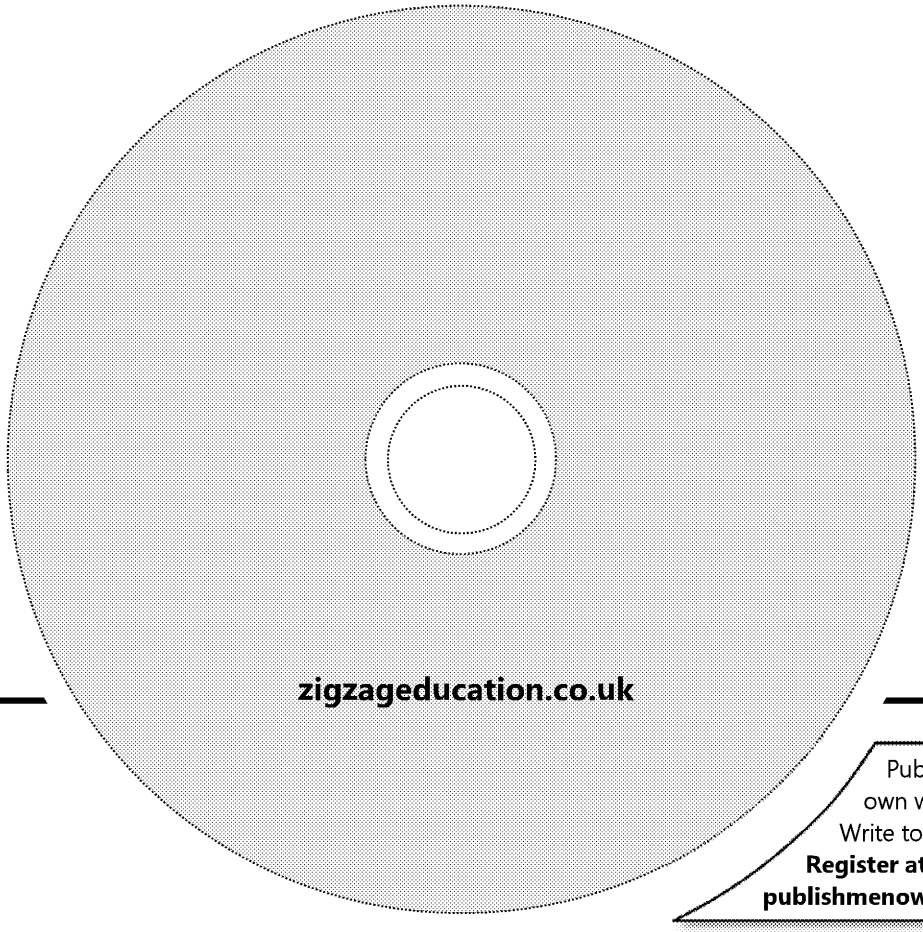
Computer Science A Level | AQA | 7517

PAPER 1 EXAM RESOURCE PACK 2018

WORDS WITH AQA

C#

for A Level AQA Computer Science



CE1/
7891

POD
7891

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 (C#) – 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 C# 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.

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 appear multiple times.

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 all 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, so 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. What happens next 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 wish to take from the queue, from the following options:

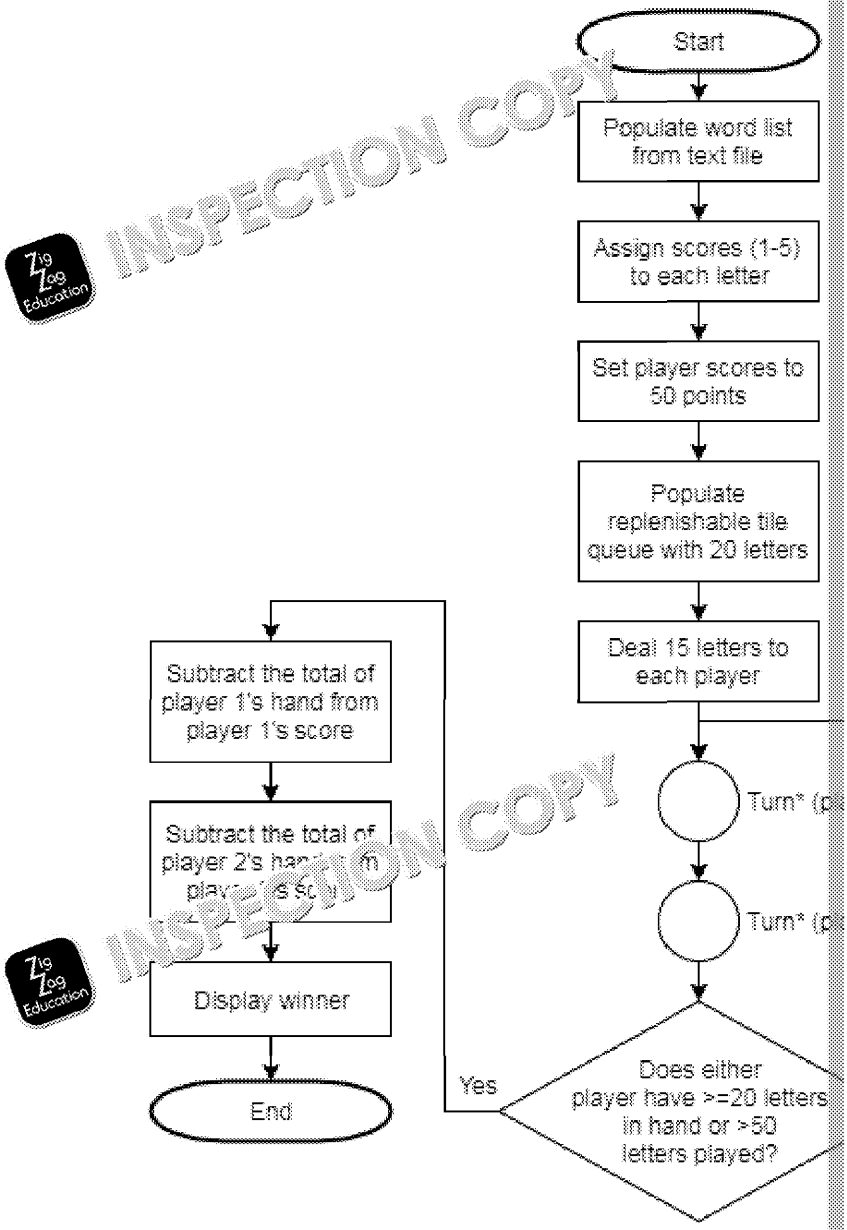
      - three tiles
      - a number of tiles equal to the length of the played word (so four tiles for a four-letter word)
      - a number of tiles equal to the length of the word plus three (so seven tiles for a four-letter word)
      - no tiles

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

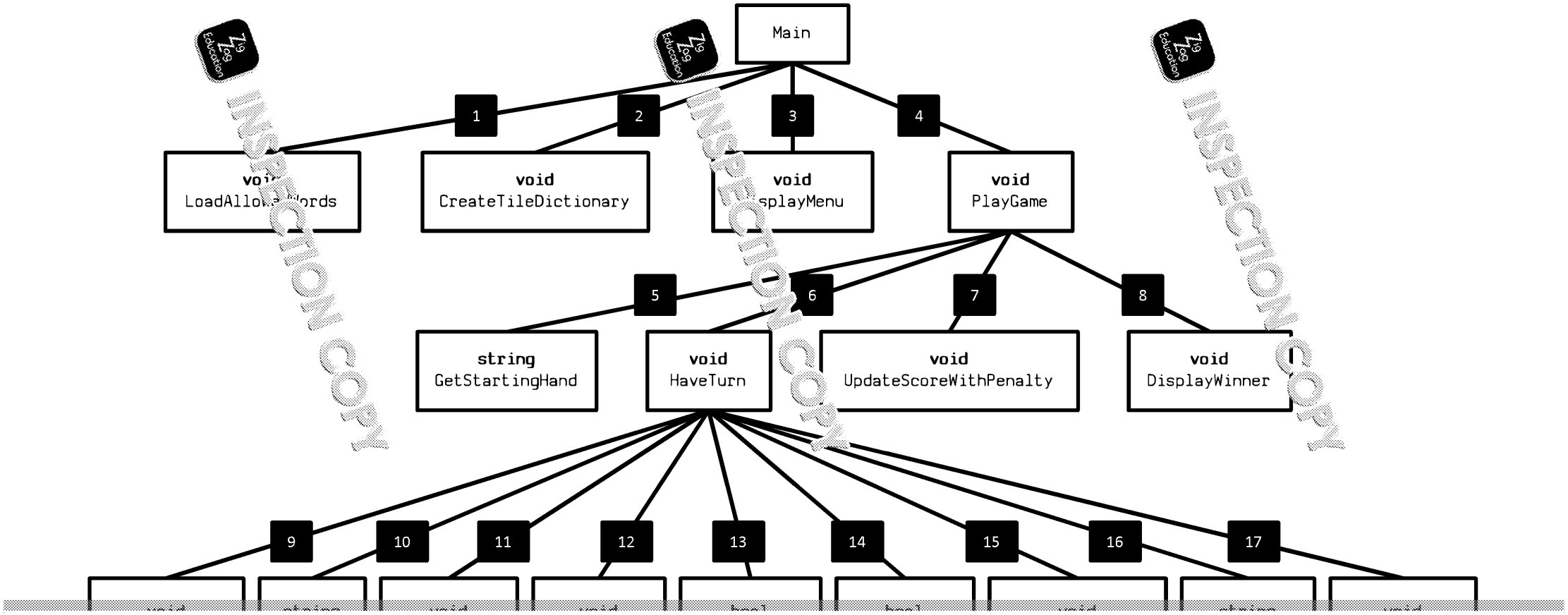
At the end of the game, the total value of each player's hand is subtracted from their score. The highest score is the winner.

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	ref List<string>: AllowedWords	-
2 Main calls CreateTileDictionary	ref Dictionary<char, int> TileDictionary	-
3 Main calls DisplayMenu	-	-
4 Main calls PlayGame	List<string>: AllowedWords Dictionary<char, int>: TileDictionary bool: RandomStart int: StartHandSize int: MaxHandSize int: MaxTilesPlayed int: NoOfEndOfTurnTiles	-
5 PlayGame calls GetStartingHand	QueueOfTiles: TileQueue int: StartHandSize	string: Hand
6 PlayGame calls HaveTurn	string: PlayerName ref string: PlayerTiles ref int: PlayerTilesPlayed ref int: PlayerScore Dictionary<char, int>: TileDictionary ref QueueOfTiles: TileQueue List<string>: AllowedWords int: MaxHandSize	-

Call	Parameters	Return
11 HaveTurn calls DisplayTileValues	Dictionary<char, int>: TileDictionary List<string>: AllowedWords	-
12 HaveTurn calls FillHandWithTiles	ref QueueOfTiles: TileQueue ref PlayerTiles int MaxHandSize	
13 HaveTurn calls CheckWordIsInTiles	string: Word String: PlayerTiles	bool: InTiles
14 HaveTurn calls CheckWordIsValid	string: Word List<string>: AllowedWords	bool: ValidWord
15 HaveTurn calls UpdateAfterAllowedWord	string: Word ref string: PlayerTiles ref int: PlayerScore ref int: PlayerTilesPlayed Dictionary<char, int>: TileDictionary List<string>: AllowedWords	-
16 HaveTurn calls GetNewTileChoice	-	string: NewTileChoice
17 HaveTurn calls AddEndOfTurnTiles	ref QueueOfTiles: TileQueue ref string: PlayerTiles string: NewTileChoice string: Choice	-
18 UpdateAfterAllowedWord calls GetScoreForWord	string: Word Dictionary<char, int>: TileDictionary	int: Score

Program Classes

Class Methods

Only `Paper1Alevel2018CS` and `QueueOfTiles` contain any methods. The functions Ⓕ and procedures Ⓟ of these classes are described below.

Paper1Alevel2018CS – Methods	Description	

`AddEndOfTurnTiles` Ⓟ	Parameters: `ref QueueOfTiles: TileQueue`
	`ref strng: PlayerTiles`
	`string: NewTileChoice`
	`string: Choice`
	Returns: -
	CalledFrom: `HaveTurn`
	Calls: `QueueOfTiles.Add`
	`QueueOfTiles.Remove`
		1. Declare an integer variable and initialise it to 0; this will ultimately contain the number of new tiles that will be taken
		2. If the user has entered option 1 (NewTileChoice), set the integer variable to contain the number of tiles played in the last move
		3. If the user has entered option 2, set the variable to contain the number 3
		4. If the user has entered neither 1 nor 2, set the variable to contain the number of tiles played plus 3
		5. Set up a loop that runs once per tile to be drawn
		6. Add a character to the string by removing a tile from the front of the tile queue
		7. Add a tile to the back of the tile queue

`CheckWordIsInTiles` Ⓕ	Parameters: `string: Word` `string: PlayerTiles` Returns: `bool` Called from: `HaveTurn` Calls: -	1. Declare a boolean variable set to true 2. Create a copy of the player's tiles 3. Loop through each character of the word being checked 4. For each character, check that it exists in the copy of player's tiles 5. If it is present, it is removed from the copy of the player's tiles 6. If it is not present, the boolean is set to false 7. Return the boolean to `HaveTurn`
`CheckWordIsValid` Ⓕ	Parameters: `string: Word` `List<string>: AllowedWords` Returns: `bool`	1. Declare a boolean variable set to false 2. Loop through each word in the array containing all of the allowed words, stopping only when the end is reached or the attempted

Paper1Alevel2018CS – Methods	Description	
`DisplayMenu` Ⓟ	Parameters: - Returns: - Called from: `Main` Calls: -	1. Present options to play the game with a random start, training start (string literals) or quit (this subroutine does not accept input or return a value)
`DisplayTilesInHand` Ⓟ	Parameters: `string: PlayerTiles` Returns: - Called from: `HaveTurn` Calls: -	1. Output a blank line 2. Output the player's hand
`DisplayTileValues`	Parameters: `Dictionary<char, int>: TileDictionary` `List<string>: AllowedWords` Returns: - Called from: `HaveTurn` Calls: -	3. Loop through each key-value entry in `TileDictionary` 4. Display each entry in the format 'Points for A: 1', with each entry on a different line
`DisplayWinner` Ⓟ	Parameters: `int: PlayerOneScore` `int: PlayerTwoScore` Returns: - Called from: `PlayGame` Calls: -	1. Display 'GAME OVER!' message 2. Display 'Player One wins!', 'Player Two wins!' or 'It is a draw!' accordingly

`FillHandWithTiles` Ⓟ	Parameters: `ref QueueOfTiles: TileQueue`
	`ref string: PlayerTiles`
	`int: MaxHandSize`
	Returns: -
	Called from: `HaveTurn`
	Calls: `QueueOfTiles.Add`
	`QueueOfTiles.Remove`
		1. Set up a loop that runs until the size of the player's hand is up to the maximum
		2. Remove a tile from the front of the tile queue and add it to the player's hand
		3. Add a tile to the back of the tile queue

Paper1Alevel2018CS – Methods	Description	
`GetScoreForWord` Ⓕ	Parameters: `string: Word` `Dictionary<char, int>: TileDictionary` Returns: `int` Called from: `UpdateAfterAllowedWord` Calls: -	1. Declare an integer variable and set it to 0 2. Loop through each character in the word, adding the score for each letter to the total 3. If the length of the word is greater than 7, add 20 to the score 4. If the length of the word is 6 or 7, add 5 to the score 5. Return the score to `UpdateAfterAllowedWord`

`GetStartingHand` Ⓕ	Parameters: `QueueOfTiles: TileQueue`
	`int: StartHandSize`
	Returns: `string`
	Called from: `PlayGame`
	Calls: `QueueOfTiles.Add`
	`QueueOfTiles.Remove`
		1. Create an empty string
		2. Loop once per tile in `StartHandSize` (initially 20 times).
		3. Add a character to the string by removing a tile from the front of the tile queue
		4. Add a tile to the back of the tile queue
		5. Return the string to `PlayGame`

`HaveTurn` Ⓟ	Parameters: `string: PlayerName`
	`ref string: PlayerTiles`
	`ref int: PlayerTilesPlayed`
	`ref int: PlayerScore`
	`Dictionary<char, int>: TileDictionary`
	`ref QueueOfTiles: TileQueue`
	`List<string>: AllowedWords`
	`int: MaxHandSize`
	`int: NoEndOfTurnTiles`
	Returns: -
	Called from: `PlayGame`
	Calls: `DisplayTilesInHand`
	`GetChoice`
	`DisplayTileValues`
		1. Display which player's turn it is
		2. Display's the player's hand
		3. A loop runs prompting the user to enter option "1", "4", "7", "0" or enter a word, via a call to `GetChoice`, until they enter a word or option "0"
		4. If they enter "1", the value of all tiles are displayed by calling `DisplayTileValues`
		5. If they enter "4", the tile queue is displayed by calling `QueueOfTiles.Show`
		6. If they enter "7", redisplay the player's hand by calling `DisplayTilesInHand`
		7. If they enter "0", fill the player's hand by calling `FillHandWithTiles`
		8. If they enter none of those options, the assumption is that they have entered a word, so its length is checked
		9. If the length is 0, a variable called `ValidWord` is set to false

Paper1Alevel2018CS – Methods	Description	
`LoadAllowedWords` Ⓟ	Parameters: `ref List<string>: AllowedWords` Returns: - Called from: `Main` Calls: -	1. Open the text file using a StringReader 2. Populate the string list such that each string is a line from the text file, with leading and trailing white space removed, and converted to uppercase 3. Return the array to `Main`

`Main` Ⓟ	Parameters: `string[]: args`
	Returns: -
	Called from: -
	Calls: `LoadAllowedWords`
	`CreateTileDictionary`
	`DisplayMenu`
	`PlayGame`
		1. Game settings are initialised: `StartHandSize`, `MaxTilesPlayed`, `NoOfEndTurnTiles`, `StartHandSize`
		2. Menu is displayed and user is prompted
		3. Loop continues until "9" is entered to quit
		4. If "1" is entered, game is played with random tiles
		5. If "2" is entered, game is played with string literals defined in `PlayGame`

`PlayGame` Ⓟ	Parameters: `List<string>: AllowedWords`
	`Dictionary<char, int>`
	`TileDictionary`
	`bool: RandomStart`
	`int: StartHandSize`
	`int: MaxHandSize`
	`int: MaxTilesPlayed`
	`int: NoOfEndOfTurnTiles`
	Returns: -
	Called from: `Main`
	Calls: `GetStartingHand`
	`HaveTurn`
	`UpdateScoreWithPenalty`
	`DisplayWinner`
	`QueueOfTiles.QueueOfTiles`
		1. Set player 1 score and player 2 score to 0
		2. Set number of tiles (for both players) to 0
		3. Create new tile queue containing 20 tiles
		4. If a random start has been requested (in `Main`), hands are populated randomly
		5. Otherwise, hands are populated with string literals
		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)
		7. Call `HaveTurn` alternately for player 1 and player 2 until the loop terminates
		8. Update scores of both players
		9. Display the winner by calling `DisplayWinner`

Paper1Alevel2018CS – Methods	Description	

`UpdateScoreWithPenalty` Ⓟ	Parameters: `ref int: PlayerScore`
	`string: PlayerTiles`
	`Dictionary<char, int>: tileDiction`
	Returns: -
	Called from: `PlayGame`
	Calls: -
		1. Loop through each tile that a player has in their hand
		2. Subtract the point value for each tile from the player's total score
		3. There is no need to return `PlayerScore` as it was passed by reference

QueueOfTiles – Metho	Description	
`QueueOfTiles` Ⓟ	Parameters: `int: MaxSize` Returns: - Called from: `Main.PlayGame` Calls: -	1. Constructor method – create a new `QueueOfTiles` object when called 2. Set the property `MaxSize` to the parameter `MaxSize` 3. Set `Rear` to -1. This variable is the pointer to the back of the queue. Since the array is initially empty, there can be no meaningful rear pointer, as 0 would mean the first element 4. Call the `Add` method repeatedly, e.g. if `MaxSize` is 20, call `Add` 20 times.
`IsEmpty` Ⓕ	Parameters: - Returns: `bool` Called from: Remove Calls: -	1. If `Rear` is -1 (meaning the pointer is not within the array, so the array can be considered empty) return true 2. For any other value of `Rear`, return false
`Remove` Ⓕ	Parameters: - Returns: `string` Called from: `Main.GetStartingHand`	1. If the list is empty, return an empty string 2. Otherwise: a. Store the character at element zero of the array

QueueOfTiles – Methods	Description	

`Add` Ⓟ	Parameters: -
	Returns: -
	Called from: `Main.GetStartingHand`
	`Main.AddEndOfTiles`
	`Main.FillHandTiles`
	Calls: -
		NB This subroutine will do nothing if `Rear` already points to the end of the array, since there would be no room to add a character.
		
		1. Generate a random integer between 0 and 25 (inclusive)
		2. Increment `Rear` by 1, indicating the queue is now one larger
		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.

Variables

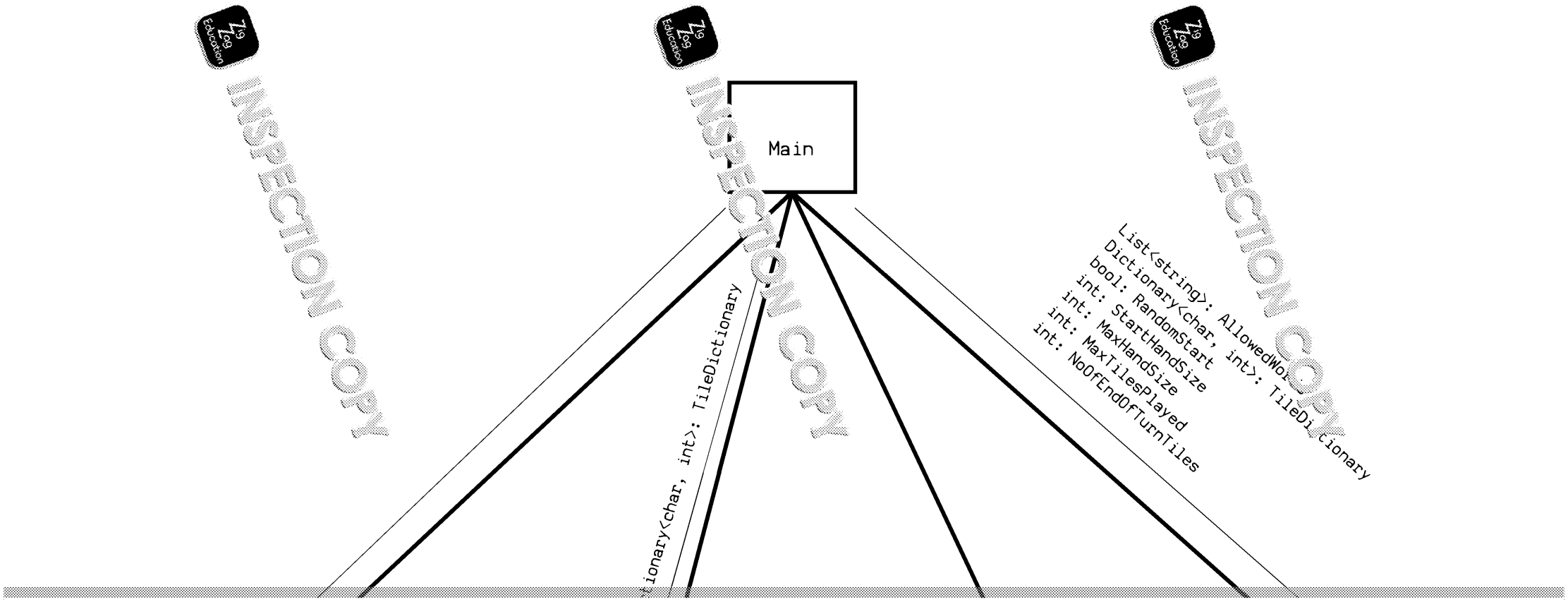
There are no instance variables in the `Paper1Alevel2018CS` class. The following table contains variables that are declared locally and passed to at least one other method.

Paper1Alevel2018CS – Variable	Type	Description	Created in
`AllowedWords`	`List<string>`	Contains all valid words read from a text file.	`LoadAllowedWords`
`Choice`	`string`	A word played by a player.	`HaveTurn`
`Choice`	`string`	Contains user input at the main in-game menu, indicating whether they want to display letter values, view the tile queue, view the player hand or end the game. User input is "1", "4", "7" or "0" respectively.	`GetChoice`
`MaxHandSize`	`int`	The largest number of tiles that a player can hold – going above this number ends the game	`Main`

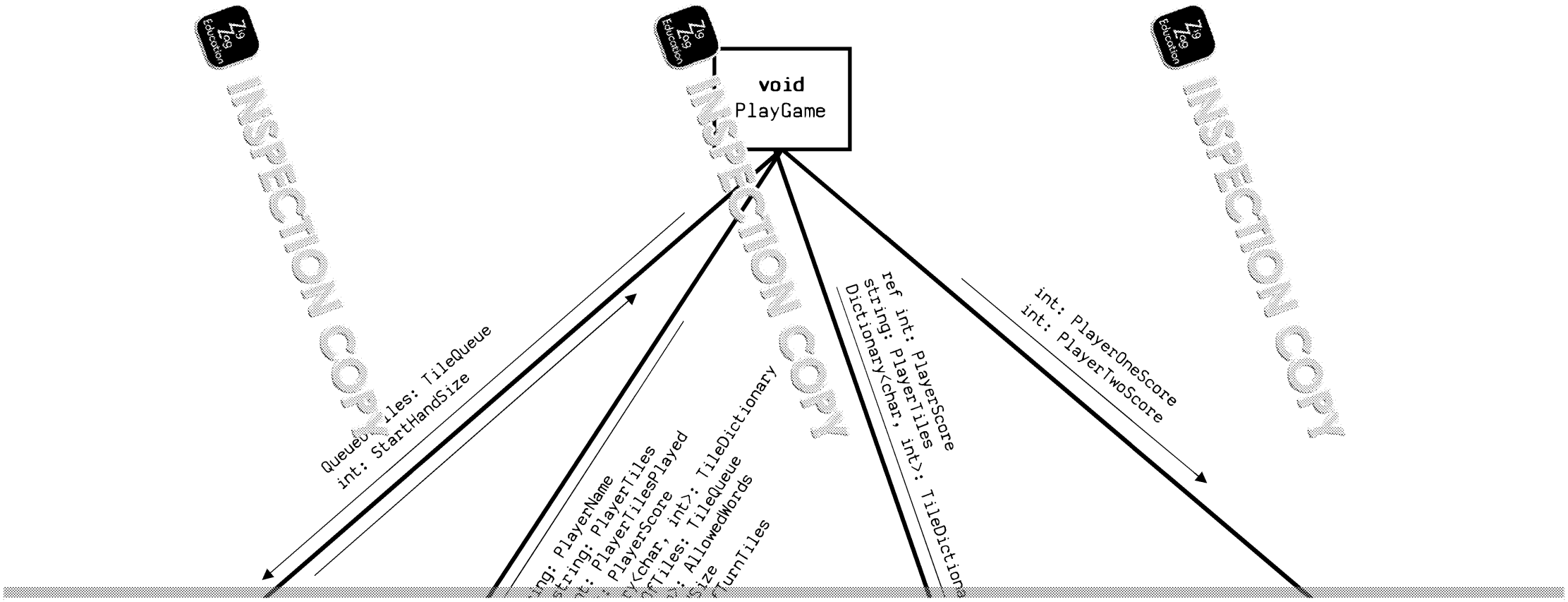
Paper1Alevel2018CS – Variables	Type	Description	Created in
`StartHandSize`	`int`	The number of tiles in a player's starting hand	`Main`
`TileDictionary`	`Dictionary<char, int>`	Contains all letters, A-Z, along with a corresponding integer value which depends on the letter. The skeleton program uses values between 1 and 5 for each letter.	`CreateTileDictionary`
`TileQueue`	`QueueOfTiles`	Contains all letters currently in the queue, from where they are passed to the player's hand	`PlayGame`

QueueOfTiles – Variables	Type	Description	Created in
`Contents`	`List<string>`	Array to contain all letters in the queue before they are passed to a player's hand	N/A (instance variable)
`Rear`	`int`	The index of the back of the queue, used to add new tiles to the correct location in the `Contents` array	N/A (instance variable)
`MaxSize`	`int`	The largest size that the array can be	N/A (instance variable)

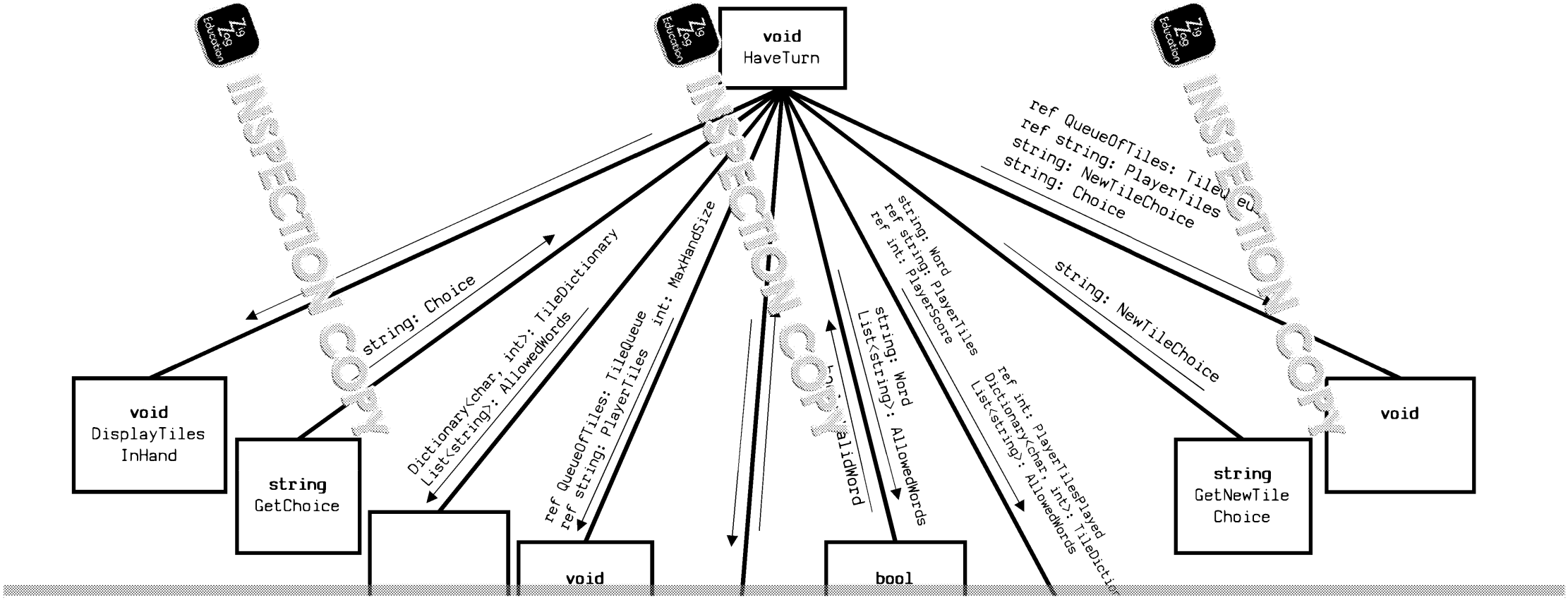
Main and subroutines called from **Main**



PlayGame and subroutines called from **PlayGame**



HaveTurn and subroutines called from **HaveTurn**



Written Questions (C#)

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

1. State the name of an identifier for:
   a) [illegible]
   b) [illegible]ray/collection 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 within the `HaveTurn` subroutine [1]
   h) An attribute within `QueueOfTiles` [1]
2. State which three subroutines remove a tile from an instance of `QueueOfTi`
3. Look at the subroutine `CreateTileDictionary`. Describe the purpose o

```
TileDictionary.Add((char)(65 + Count), 1);
```

4. State and describe the data structure returned by the `CreateTileDictio`
5. Look at the subroutine `CheckWordIsInTiles`. Ex[illegible] the role of the var
6. Describe the difference between a proced[illegible]re and[illegible]unction. State one exam code. [4]
7. Describe what would h[illegible]pen during a call to `LoadAllowedWords`, the fi found. [3]
8. Describ[illegible]ctions performed in the following lines of the `loadAllowed`

```
StreamReader FileReader = new StreamReader("aqawords
while (!FileReader.EndOfStream)
{
    AllowedWords.Add(FileReader.ReadLine().Trim().To
}
FileReader.Close();
```

9. The `QueueOfTiles` class contains a constructor. Describe what is meant b
10. Describe the effect of the following instruction within the `QueueOfTiles`

```
this.MaxSize = MaxSize;
```

11. Explain why the variable `Rear` is initialised to -1 in t[illegible]ueueOfTiles co
12. Describe in detail the purpose of the subr[illegible]ut[illegible]e [illegible]ateScoreWithPenal and/or return values in your answ[illegible]
13. Describe the operation[illegible]following code within the subroutine `GetSco`

```
int[illegible]
fo[illegible] Count = 0; Count < Word.Length; Count++)
{
    Score = 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"

Written Questions (C#)

These questions refer to the preliminary material and require you to load the skeleton program and any additional programs.

1. State the name of an identifier for:

   a) A class [1]

   .....

   b) An array/collection 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 within the `HaveTurn` subroutine [1]

   .....

   h) An attribute within `QueueOfTiles` [1]

   .....

2. State which three subroutines remove a tile from an instance of `QueueOfTiles`

   1 .....

   2 .....

   3 .....

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

```
TileDictionary.Add((char)(65 + Count), 1);
```

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

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

6. Describe the difference between a procedure and a function. State one exam code. [4]

7. Describe what would happen if, during a call to `LoadAllowedWords`, the fi
found. [3]

.....

.....

.....

.....

.....

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

```
StreamReader FileReader = new StreamReader("aqawords
while (!FileReader.EndOfStream)
{
    AllowedWords.Add(FileReader.ReadLine().Trim().To
}
FileReader.Close();
```

.....

.....

.....

.....

.....

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

.....

.....

.....

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

```
this.MaxSize = MaxSize;
```

.....

.....

.....

.....

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

.....

.....

.....

12. Describe in detail the purpose of the su[illegible]t `UpdateScoreWithPenal`
and/or return values in your a[illegible]

.....

.....

.....

.....

.....

.....

.....

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

```
int Score = 0;
for (int Count = 0; Count < Word [illegible]h; Count++)
{
    Score = Score + Tile[illegible]ory[Word[Count]];
}
```

.....

.....

.....

.....

.....

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

.....

.....

.....

15. Explain why the va[illegible] `TileChoice` is initialised to the string value "2"

.....

.....

.....

.....

Programming Tasks (C#)

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

Task 1

This task 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 r 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]AD
  - Press '4' to request no new tiles
- One screen capture showing [illegible] played, the new score and the tota sequence of events:
  - Begi[illegible] game again with the training hand
  - [illegible] *Player One*'s first turn, play the word 'BARD'
  - [illegible]ess '4' to request no new tiles

Task 2

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

The program currently displays the letters in the player's hand as a single string `DisplayTilesInHand` so that each letter is displayed, followed by its points v space; for example:

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

Modify `HaveTurn` to allow `DisplayTilesInH[illegible]` to [illegible]ive access to the point

Evidence you need to provide:

- Your amended [illegible]URCE CODE PROGRAM for `DisplayTilesInHand`
- Yo[illegible] SOURCE CODE PROGRAM for `HaveTurn`
- One [illegible] capture showing *Player One*'s hand at the beginning of the g hand

Task 3

This task refers to `CreateTileDictionary`.

Currently, there are no letters worth four points.

Modify the code in `CreateTileDictionary` so that the letters 'K', 'V' and 'Y' a

Evidence you need to provide:

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

Task 4

This task 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 b

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 Two*'s turn and the prompt that marks
turn (the action taken by *Player Two* is unimportant). Begin with the train

Task 5

This task 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
ABANDONS

Task 6

This task 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 attrib[illegible]to[illegible] that player's tiles
they have played. There should be a constr[illegible]or to initialise these attributes to
`PlayGame`. In order to do this, a [illegible]`QueueOfTiles` object called `TileQueue` wi
for the constructor.

Create acc[illegible]et[illegible]ds within the class to grant public visibility to each of the

Evidence you need to provide:

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

Task 7

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

Currently, there is no option for the player to swap their letters. Extend the men
option 'Press 2 to swap your letters OR'.

Modify `HaveTurn` so that all tiles in the player's hand are removed and are repla
letters from the tile queue. There should be no penalty (e.g. extra tiles) for chan

Once the letters have been swapped, the ne[illegible]ha[illegible]hould be displayed, but play
player.

Evidence you need to provide:

- You[illegible]ded 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.

Task 8

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

Modify this subroutine to prevent two consecutive letters being added to the que

Iteration should be used to cause new letters to be generated (and ignored) until
the same as the previous letter. At that point, the let[illegible]ou[illegible]d be added to the

Evidence you need to provide:

- Your amended SO[illegible]E PROGRAM for `Add`

Task 9

This task 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
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`.

You should modify `LoadAllowedWords` in your response to this task. You validation could you can assume the user will enter either '1' or '2'.

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.

Task 10

This task refers to `Main`.

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

Modify the `Main` method so that if `LoadAllowedWords` returns an empty array displayed and the progra

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

Task 11

This task 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 BRAT for *Player One*
  - Pressed '4' to indicate that you want no replacement tiles

Task 12

This task 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 private attributes – `Letter` (char), `Score` (int) an
  attribute would be set to 'true' for a vowel (A, E, I, O, U) and 'false' for a c
- The constructor should have two parameters the letter and the map, cont
  The constructor should set all three attributes correctly. For example, if t
  would be set as follows:
  - `Letter`: A (the letter should always be stored in the attribute in
  - `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`.

Task 13

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

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

Modify the program as follows:

- Add a menu option in `Disp` de to read '3. Settings'.
- Alter the code in `Ma` tha `3` is chosen from the menu, the user w following pa turn:
  - ter 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 `MaxTilesPlayed`, `NoOfEndOfTurnTiles` and `StartHandSize` res

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

Task 14

This task refers to the `QueueOfTiles` class.

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

Modify the add subroutine and any other necessary code so that letters are select 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)
- After this, every third letter should be a random chosen vowel

ASCII values for the vowels are as follow , E=69, I=73, O=79, U=85

Evidence you need to p :

- Yo nd SOURCE CODE PROGRAM for the `QueueOfTiles` class
- One shot showing *Player One*'s tiles after choosing to play with a r

Task 15

This task refers to `HaveTurn` and a new subroutine called `ResolveBlanks`.

The game is currently played without blank tiles. In other games, such as *Scrabb*
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 which is a string that may or may not
play the role of blank tiles
- For each dash entered, the user is given the prompt 'Enter value of
ide for all instances of dash in the word)
- The 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 r
- The word, now without any dashes, is returned to `HaveTurn`

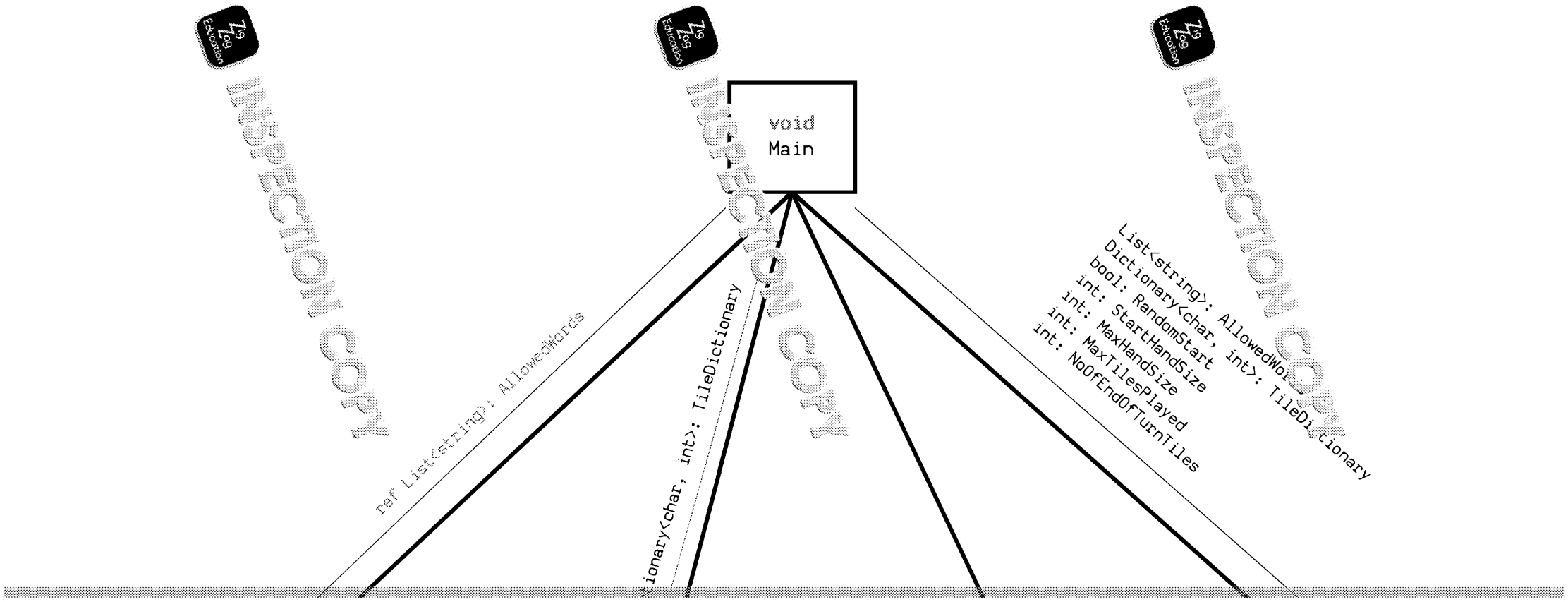
You should also modify `HaveTurn` so that `ResolveBlanks` is called immediate
is called.

Evidence you need to provide:

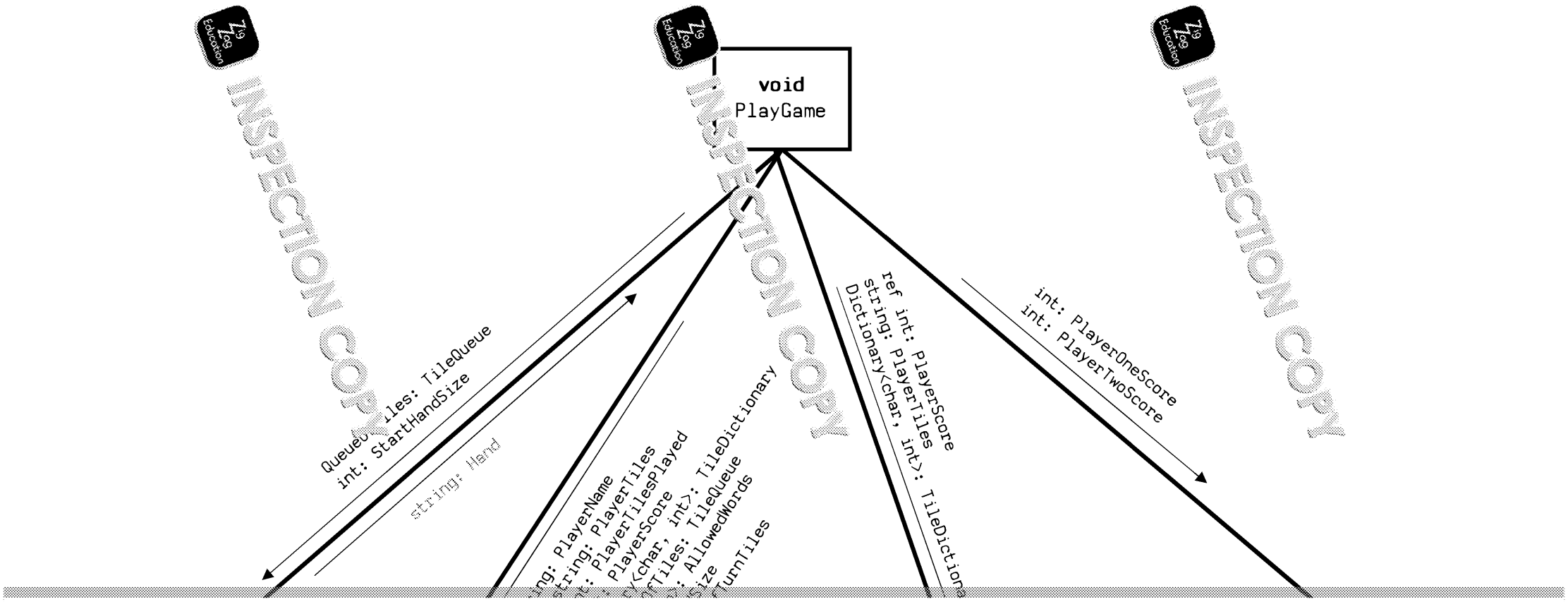
- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- Your SOURCE CODE PROGRAM for the new `ResolveBlanks` subroutine
- One screenshot showing the output that results from the following:
  - Before running the program, change the following line in the Pla
    FROM: `PlayerOneTiles = "BTAHANDENONSARJ"`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

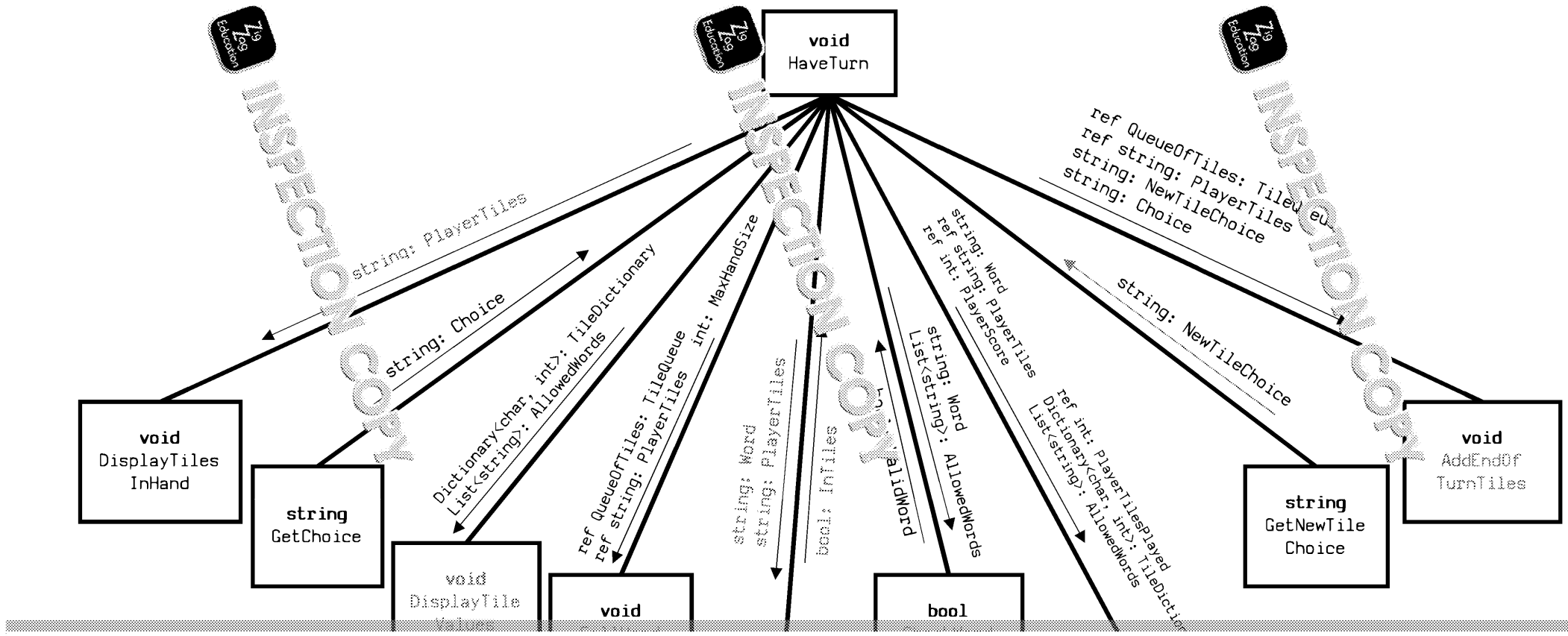


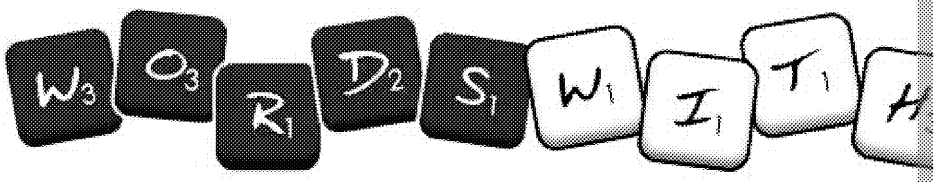
PlayGame and subroutines called from **PlayGame**



Structure Diagram 3

HaveTurn and subroutines called from **HaveTurn**



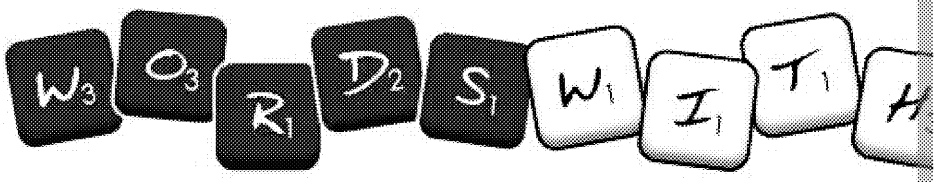


Written Questions:
Suggested Answers and Mark Sche

Q	Answer/Guidance
1a	`Paper1Alevel2018CS/QueueOfTiles`
1b	`Contents/AllowedWords/args`
1c	`InTiles/ValidWord`
1d	`TileDictionary`
1e	`FillHandWithTiles/UpdateScoreWithPenalty`
1f	`QueueOfTiles.Add/QueueOfTiles.Show/DisplayMenu`
1g	`NewTileChoice/ValidChoice/ValidWord/Choice`
1h	`Contents/Rear/MaxSize/Rnd`
2	3 marks: • `GetStartingHand` • `AddEndOfTurnTiles` • `FillHandWithTiles`
3	1 mark for each: • (a) entry/key-value pair) to the dictionary. • y includes character with ASCII value of 65 + 'count', which is an u • Entry includes integer value 1.
4	1 mark for stating data structure: • Dictionary Up to 2 marks for description: • Contains unique keys • Keys are letters of the alphabet • Each key links to a value • Values are integers which represent the score for that letter
5	Up to 2 marks for explanation: • Creates/contains a copy of the player's tiles • Characters are eliminated from the copy • Prevents tiles being eliminated from layer tiles (since word has not

Q	Answer/Guidance
6	2 marks for difference between procedure and function: • A *procedure* performs a sequence of events but *does not* return a value • A *function* also performs a sequence of events but *does* return a value Sufficient for 2 marks: a function returns a value – a procedure does not 1 mark for identifying a procedure: • `QueueOfTiles.Add` • `QueueOfTiles.Sho` • `AddEndOfTu` • `Creat` • `ictionary` • `lyMenu` • `DisplayTilesInHand` • `DisplayTileValues` • `DisplayWinner` • `FillHandWithTiles` • `HaveTurn` • `LoadAllowedWords` • `Main` • `PlayGame` • `UpdateAfterAllowedWord` • `UpdateScoreWithPenalty` 1 mark for identifying a function: • `CheckWordIsInTiles` • `CheckWordIsValid` • `GetChoice` • `GetNewTileChoice` • `GetScoreForWord` • `GetStartin` • `Queu` • `.IsEmpty` • `ueOfTiles.Remove`
7	3 marks: • An exception would be thrown • Execution would pass to the 'catch' block • The AllowedWords array would be made empty
8	4 marks: • The text file ("aqawords.txt") is opened • Each line in the text file is added to the list of allowed words • Each word has leading and trailing whitespace removed • Each word is converted to uppercase • The text file is closed
9	2 marks: • A constructor is a method called by the ke • Creates a new object based on the cl ew ich it resides
10	2 marks: • Sets the pro ce variable (`MaxSize`) • ter (`MaxSize`)
11	2 mar • `Rear` points to the back of the queue • Value of -1 indicates an empty queue • -1 is a special value

Q	Answer/Guidance
12	1 mark for parameters: • Player's current score, tiles in the player's hand, `Map` that links tiles to Up to 4 marks from the following: • Purpose of function is to subtract value of player's tiles/hand from the • Loop is established to iterate through each ti[illegible]aracter in the playe • Value of each tile is determined by lo[illegible]n map • Value is subtracted from pla[illegible]ore • Player's score is upd[illegible]erence ('ref' keyword) OR void/no retur
13	4 marks: • [illegible]e/integer/variable is set to zero • [illegible] iterates through each character in the word • Value of character looked up in the map/dictionary/`TileDictionary` • Value added to score variable, effectively summing the scores
14	2 marks • Loop continues until user has selected '1', '2', '3' or '4' • Validates user input
15	3 marks • Contents of this variable are passed to `AddEndOfTurnTiles` • Selection of '2' indicates that three new tiles will be drawn • The only way to replace this selection is to have played a valid word
TOTAL MARKS	



Programming Tasks:
Suggested Solutions and Mark Sch

That the ... recommended solutions, and not an exhaustive list of all possible so... guidance ... 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 l

1 mark Score decremented correctly within the IF statement

```
if (Word.Length > 7)
{
    Score = Score + 20;
}
else if (Word.Length > 5)
{
    Score = Score + 5;
}
el...F(W...Length == 2 || Word.Length == 3)
{
    Score--;
}
```

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 game.
```

1 mark Screenshot shows 'BARD' was played, the new total is '56' and the total ...

```
Your word was:BARD
Your new score is:56
You have played 4 tiles so far in th...ame
```

Question 2

1 mark `DisplayTilesInHand` uses additional parameter of data type `Dict` some other means has been used to grant access to the map, but mark then not be available)

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

1 mark Display the character (**R:** if the wh's ha's still *also* displayed in its o

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

```
private static void DisplayTilesInHand(string PlayerTiles,
    Dictionary<char, int> TileDictionary)
{
    Console.WriteLine();
    for (int i = 0; i < PlayerTiles.Length; i++) {
        char Tile = PlayerTiles[i];
        Console.Write(Tile + "(" + TileDictionary[Tile] + ")
    }
    Console.WriteLine();
}
```

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

```
Console.WriteLine(PlayerName + " it is your turn.");
DisplayTilesInHand(PlayerTiles, TileDictionary);
string NewTileChoice = "2";
```

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

```
else if NewTileChoice == "7")
{
    DisplayTilesInHand(PlayerTiles, TileDictionary);
}
```

1 mark Screenshot showing correct output format, which should no longer incl

```
Player One it is your turn.

B(2) T(1) A(1) H(3) A(1) N(1) D(2) E(1) N(1) O(1) N(1) S(1)
```

Question 3

1 mark Removal of values 10, 21 and 24 from `int[] Value3`, the structure th

1 mark Inclusion of values 10, 21 and 24 in a `int[] Value4`.

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

```
int[] Value1 = { 0, 4, 8, 13, 14, 17, 18, 19 };
int[] Value2 = { 1, 2, 3, 6, 9, 12, 15, 20 };
int[] Value3 = { 7, 22 };
int[] Value4 = { 10, 21, 24 };
```

```
e (Array.IndexOf(Value4, Count) > -1)
{
    TileDictionary.Add((char)(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: 3
Points for I: 1
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*

```
int PlayerOneScore = 50;
int PlayerTwoScore = 50;
int PlayerThreeScore = 50;
```

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

```
int PlayerOneTilesPlayed = 0;
int PlayerTwoTilesPlayed = 0;
int PlayerThreeTilesPlayed = 0;
```

1 mark Creating an empty hand for *Player Three*

```
string PlayerOneTiles = "";
string PlayerTwoTiles = "";
string PlayerThreeTiles = "";
```

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

```
if (RandomStart)
{
    PlayerOneTiles = GetStartingHand(TileQueue, StartHandSize
    PlayerTwoTiles = GetStartingHand(TileQueue, StartHandSize
    PlayerThreeTiles = GetStartingHand(TileQueue, StartHandSi
}
```

1 mark *Player Three*'s hand filled with tiles called ‘ABCDEFGHIJKLMNO’ in the ‘else’ block

```
else
{
    PlayerOneTiles = "BTAHANDENONSARJ";
    PlayerTwoTiles = "CELZXIOTNESMUAA";
    PlayerThreeTiles = "ABCDEFGHIJKLMNO";
}
```

1 mark Logic expression in ‘while’ loop includes tiles played for *Player Three*

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

```
while (
        PlayerOneTilesPlayed <= MaxTilesPlayed
        && PlayerTwoTilesPlayed <= MaxTilesPlayed
        && PlayerThreeTilesPlayed <= MaxTilesPlayed
        && PlayerOneTiles.Length < MaxHandSize
        && PlayerTwoTiles.Length < MaxHandSize
        && PlayerThreeTiles.Length < MaxHandSize
) {
```

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

```
HaveTurn("Player Two", ref PlayerTwoTiles, ref PlayerTwoTile
ref PlayerTwoScore, TileDictionary, ref TileQueue, AllowedWo
MaxHandSize, NoOfEndOfTurnTiles);
Console.WriteLine();
Console.WriteLine("Press Enter to continue");
Console.ReadLine();
Console.WriteLine();
HaveTurn("Player Three", ref PlayerThreeTiles, ref
PlayerThreeTilesPlayed, ref PlayerThreeScore, TileDictionary,
TileQueue, AllowedWords, MaxHandSize, NoOfEndOfTurnTiles);
```

1 mark Ca `pdateScoreWithPenalty` for *Player Three*

```
UpdateScoreWithPenalty(ref PlayerOneScore, PlayerOneTiles, Ti
UpdateScoreWithPenalty(ref PlayerTwoScore, PlayerTwoTiles, Ti
UpdateScoreWithPenalty(ref PlayerThreeScore, PlayerThreeTiles,
```

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

```
DisplayWinner(PlayerOneScore, PlayerTwoScore, PlayerThreeScor
```

1 mark Subroutine `DisplayWinner` requires three parameters instead of two

1 mark Score for *Player Three* is displayed

```
private static void DisplayWinner(int PlayerOneScore, int Pl
    int PlayerThreeScore)
{
    Console.WriteLine();
    Console.WriteLine("**** GAME OVER! ****");
    Console.WriteLine();
    Console.WriteLine("Player One your score is " + PlayerOne
    Console.WriteLine("Player Two your score is " + PlayerTw
    Console.WriteLine("Player Three your score is " + Player
```

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 > PlayerTwoScore && PlayerOneScore > Playe
{
    Console.WriteLine("Player One wins!");
}
else if (PlayerTwoScore > PlayerOneSc  & layerTwoScore >
{
    Console.WriteLine("P  e  wins!");
}
else if (Pl  nreScore > PlayerOneScore && PlayerThreeSco
{
    sole.WriteLine("Player Three wins!");
}
else
{
    Console.WriteLine("No clear winner");
}
```

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 O
     press 4 to view the tile queue OR
     press 7 to view your tile  a  OR
     press 0 to fill ha  t  the game.
> cleats

  rd

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
     get no new tiles (4)?
> 1

Your word was:CLEATS
Your new score is:63
You have played 6 tiles so far in this game.

Press Enter to continue



Player Three it is your turn.

Your current hand: ABCDEFGHIJKLMNO

Either:
     enter the  o  ld like to play OR
     pres  1 to play the letter values OR
     ss 4 to view the tile queue OR
     ss 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>
```

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 '
30 points.')

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

1 mark Correct parameters – `leDictionary`

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)
{
    int Score = GetScoreForWord(Choice, TileDictionary);
    Console.WriteLine();
    Console.WriteLine("Valid word. " + Choice + " scores " +
    Console.WriteLine();
    UpdateAfterAllowedWord(Choice, ref PlayerTiles, ref Playe
        ref PlayerTilesPlayed, TileDictionary, AllowedWords);
    NewTileChoice = GetNewTileChoice();
}
```

1 mark Input of word 'ABANDONS' displays score worth 30 points.
(**DPT:** spacing errors penalised on fourth ma

```
Your current hand: BTAR
Either:
    nt
    ess 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 fill hand and stop the game.
> abandons

Valid word. ABANDONS scores 30 points.
```

Question 6

1 mark Class declaration with opening and closing braces (**R:** if name or case in

```
class Player
{...
}
```

1 mark Attributes declared with appropriate na[illegible] (A: alternative names if me

1 mark All three attributes u[illegible] correct data types (**R:** if any additional attributes

1 mark At[illegible]es [illegible] declared `private`

```
pr[illegible] int Score;
private string Tiles;
private int NumTilesPlayed;
```

(**A**: C# auto-implemented properties)

```
public int Score { get; set; }
public string Tiles { get; set; }
public int NumTilesPlayed { get; set; }
```

1 mark Constructor declared with parameter `QueueOfTiles TileQueue` (**A**: if they are used)

1 mark `Score` and `NumTilesPlayed` (or their equivalents) initialised to `50` a

1 mark `Tiles` (or its equivalent) initialised using a call to `GetStartingHand`

1 mark `GetStartingHand` called with correct arguments (**A:** integers other t is passed to construct

```
public P[illegible]QueueOfTiles TileQueue, int StartHandSize) {
    [illegible]s.Score = 50;
    [illegible]is.Tiles = GetStartingHand(TileQueue, StartHandSize);
    this.NumTilesPlayed = 0;
}
```

1 mark Accessor methods with appropriate data types to return all attributes (**D**

1 mark Accessor methods all declared `public` (**A**: C# auto-implemented prope

```
public int GetScore() {
    return Score;
}

public string GetTiles() {
    return Tiles;
}

public int GetNum[illegible]esPlayed() {
    return [illegible]sPlayed;
}
```

Question 7

1 mark Addition of the extra option in GetChoice

```
Console.WriteLine("Either:");
Console.WriteLine("     enter the word you would like to play
Console.WriteLine("     press 1 to display the letter values
Console.WriteLine("     press 2 to swap your letters OR");
Console.WriteLine("     press 4 to view the tile queue OR");
Console.WriteLine("     press 7 to view your tiles again OR")
Console.WriteLine("     press 0 to fill hand and stop the gam
Console.W...
```

1 mark Inclusion of 'else if' clause in HaveTurn to deal with 'choice' being '2'

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

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 repe

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

1 mark Output of new hand

```
else if (Choice == "2")
{
    string NewPlayerTiles = "";
    while (NewPlayerTiles.Length < PlayerTiles.Length)
    {
        NewPlayerTiles += TileQueue.Remove();
        TileQueue.Add();
    }
    PlayerTiles = NewPlayerTiles;
    ValidChoice = true;
    NewTileChoice = "4";
    Console.WriteLine("New tiles: " + PlayerTiles);
}
```

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

```
Player One it is your turn.

Your current hand: BIAHANDENONSARJ

Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 2 to swap 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

New tiles: AVMMJKQWLPIPMEN
Press Enter to continue
```

Question 8

1 mark Character attribute in `QueueOfTiles` of suitable name (**A**: loop reads
but must check array is not empty first; **I:** access modifier, initialisation)

```
private List<string> Contents = new List<string>();
private int Rear;
private int MaxSize;
Random Rnd = new Random();
private string LastLet
```

1 mark Loop inside e structure to iterate until a duplicate has not bee
ir lo clear)

1 mark Generation of a new letter exists inside the loop

1 mark Comparison of new letter and previous letter is made inside the loop

1 mark Incrementation of '`Rear`' outside the loop

1 mark Termination of loop depends on correct comparison of new letter and t

1 mark Attribute is set to the most recent new letter by the end of the subrout

```
public void Add()
{
    if (Rear < MaxSize - 1)
    {
        while(true)
        {
            int RandNo = Rnd.Next(0,26);
            string Tile = Convert.ToChar(65 + RandNo).ToStri
            if (Tile != LastLetterAdded) {
                Rear++;
                Contents[Rear] = Tile;
                LastLetterAdded = Tile;
                break;
            }
        }
    }
}
```

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

```
Console.WriteLine("(1) Default Di
Console.WriteLine("(2) Cu
Console.Write("> ")
Choice = Co
Co
```

1 mark De ation of variable to store file path

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

1 mark A choice of '2' results in the user being prompted for a path

1 mark User response is placed into the appropriate variable

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

```
string Path;
if (Choice == "1")
{
    Path = "aqawords.txt";
}
else
{
    Console.WriteLine("
    Console.Write
    Path =
}
```

1 mark File path specified by the user is accessed

```
try
{
    StreamReader FileReader = new StreamReader(Path);
```

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

```
(1) Default Dictionary
(2) Custom Dictionary
> 2

Enter file name
>
```

Question 10

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

1 mark 'if' clause compares array length correctly, e.g. `== 0` or `< 1`

1 mark Correct output message

1 mark Instruction to end program (**I:** exit code)

```
LoadAllowedWords(    AllowedWords);

if (    Words.Count == 0) {
    Console.WriteLine("File error");
    Environment.Exit(1);
}
```

1 mark Output displays welcome message followed by 'file error' and evidence

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


File error
```

Question 11

1 mark Message correctly placed (even if message is incorrect) before call to Ge
message is printed by calling a new subroutine, e.g. DisplayTimesRe

```
DisplayTilesRemaining(PlayerTiles);
NewTileChoice = GetNewTileChoice();
```

```
private static void DisplayTilesR
    Console.WriteLine("You
}
```

1 mark Me e c placed (even if message is incorrect) immediately af
Ad fTurnTiles (**A:** if code is added outside the 'if' clause, but o
clause to prevent the message being redisplayed in the event that no n

1 mark String concatenation to display message in the specified format in both
called)

```
if (NewTileChoice != "4")
{
    AddEndOfTurnTiles(ref TileQueue, ref PlayerTiles, NewTile
    DisplayTilesRemaining(PlayerTiles);
}
```

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 til
     get three
     rep
     et
You have 18 tiles remaining.

Your word was:BAT
Your new score is:54
You have 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
be 'Your word was: BRAT'

```
Valid word

You have 11 tiles remaining.
Do you want to:
     replace the til
     get three
     rep
     et
Your word was:BRAT
Your new score is:55
You have played 4 tiles so far in this game.

Press Enter to continue
```

Question 12

1 mark Class declaration with opening and closing braces (**R:** if name or case i

```
class LetterTile
{…
}
```

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

1 mark All attributes use correct …

1 mark Attributes dec… p…ate

```
p… char Letter;
pr… int Score;
private bool IsVowel;
```

(**A**: C# auto-implemented properties)

```
public char Letter { get; }
public int Score { get; }
public bool IsVowel { get; }
```

1 mark Constructor written as `LetterTile`

1 mark `char` parameter (**A:** if named other than '`Letter`')

1 mark `Dictionary<char, int>` parameter (**A:** if named other than '`TileD`

1 mark Letter attribute set using char parameter

1 mark Attribute always contains upper-case version o… acter

1 mark Score attribute set by extracting a value fr… the map (**A**: value extracte

1 mark Score would always be … the parameter passed to 'get' should

```
public L…(char Letter, Dictionary<char, int> TileDic
{
    …s.Letter = Char.ToUpper(Letter);
    this.Score = TileDictionary[this.Letter];
```

1 mark Selection to isolate either vowels or consonants (**A**: it would not work

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

1 mark `IsVowel` attribute set correctly with this selection

```
this.IsVowel = (new char[] { 'A', 'E', 'I', 'O', 'U' }).Cont
```

1 mark Accessor methods with appropriate data types (**A:** any sensible names; … properties)

1 mark Accessor methods declared public

```
public char GetLetter() {
    return Letter;
}

p… in… GetScore() {
    …urn Score;
}

public bool Vowel() {
    return IsVowel;
}
```

Question 13

1 mark New entry in `DisplayMenu`

```
private static void DisplayMenu()
{
    Console.WriteLine();
    Console.WriteLine("=========");
    Console.WriteLine("MAIN MENU");
    Console.WriteLine("=========");
    Console.WriteLine();
    Console.WriteLine("1. Play game with random start hand");
    Console.WriteLine("2. Play game with training start hand");
    Console.WriteLine("3. Settings");
    Console.WriteLine("9. Quit");
    Console.WriteLine();
}
```

1 mark 'else if' added to main to detect entry of '3' in `Main`

1 mark Prompts for all four new inputs (**R:** alternative wording)

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

1 mark Syntactically valid string-to-integer conversion for all inputs

1 mark Input prompts relate correctly to all four variables

```
else if (Choice == "3")
{
    Console.Write("Enter maximum hand size: ");
    MaxHandSize = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter maximum tiles played: ");
    MaxTilesPlayed = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter default new tiles: ");
    DefaultNewTiles = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter starting hand size: ");
    StartHandSize = Convert.ToInt32(Console.readLine());
}
```

1 mark Output showing values 25, 40, 2 and 10 entered, with 10 indicating the hand should now be 10 characters long instead of 15

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

Enter your choice:
1

Player One it is your turn.

Your current hand: SFQIYLPYRQ
```

Question 14

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

```
private List<string> Contents = new List<string>();
private int Rear;
private int MaxSize;
private int VowelCounter = 0;
Random Rnd = new Random();
```

1 mark `Rear` is still incremented before any letter is added

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

1 mark New variable is changed to ensure the switch between selecting a vowe
(`VowelCounter++` in this example, but any equivalent approach can b

1 mark Random letter still correctly added to the array

```
if (Rear < MaxSize - 1)
{
    Rear++;
    if (VowelCounter < 2)
    {
        int RandNo = Rnd.Next(0, 26);
        Contents[Rear] = Convert.ToChar(65 + RandNo).ToStrin
        VowelCounter++;
    }
```

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 reset to ensure that the next selection will be a random l

```
    {
        int RandNo = Rnd.Next(0, 5);
        char[] Vowels = { 'A', 'E', 'I', 'O', 'U' };
        Contents[Rear] = Vowels[RandNo];
        VowelCounter = 0;
    }
}
```

1 mark Selecting the random starting hand should display every third letter as

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

1. Play game with random starting hand
2. Play game with trained starting hand
9. Quit

Enter your choice:
1

Player One it is your turn.

Your current hand: MHOBNEBMOOSAGGE
```

Question 15

1 mark Call to `ResolveBlanks`, with choice as parameter, before call to `Che`

1 mark Value returned from call to `ResolveBlanks` stored in choice

```
if (ValidWord)
{
    string ChoiceWithBlanks = Choice;
    Choice = ResolveBlanks(Choice);
    Console.WriteLine("Word was: " + Choice);
    IsValid = CheckWordIsValid(Choice, AllowedWords);
```

1 mark Me declaration for `ResolveBlanks` with a string return type (**R:** a variation in case)

1 mark String parameter (**R:** if any other parameters)

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 program would fail in the abs

1 mark User is prompted with 'Enter value of blank tile:' (**R:** alternative wording

1 mark User input is converted to upper case and stored in variable

1 mark Attempt to incorporate the new input to replace the dash (even if unsuc

1 mark Loop ensures that all dashes would be repla characters entered i

1 mark String correctly returned (**A:** pa passed by reference)

```
private sta g ResolveBlanks(string Word) {
    for i = 0; i < Word.Length; i++)
    {
        if (Word[i] == '-')
        {
            Console.Write("Enter value of blank tile: ")
            string NewLetter = Console.ReadLine().ToUpp
            Word = Word.Substring(0, i) + NewLetter[0] +
        }
    }
    return Word;
}
```

1 mark Modified the signature of `UpdateAfterAllowedWord` to accept both blanks left as dashes (`Word`/`Choice`), and with b ks resolved to lette (`WordWithBlanks`/`ChoiceWithBla`

```
UpdateAfterAllowedW hoice, ChoiceWithBlanks, ref Pl
PlayerScore  erTilesPlayed, TileDictionary, All
```

```
p  tatic void UpdateAfterAllowedWord(string Word,
r ing PlayerTiles, ref int PlayerScore, ref int Pla
Dictionary<char, int> TileDictionary, List<string> Allow
```

1 mark Modified `UpdateAfterAllowedWord` to only remove letters from `P` played as blanks

```
foreach (var Letter in WordWithBlanks)
{
    if (Letter != '-') {
        PlayerTiles = PlayerTiles.Remo PlayerTiles.In
    }
}
```

1 mark Word with any [illegible] is passed to `GetScoreForWord`.

```
P core = PlayerScore + GetScoreForWord(Word, TileD
```

1 mark Output for `han--`, followed by entering 'd' then 'y', results in 'Valid wo

```
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 fill hand and stop the game.
> han--

Enter value of blank tile: d
Enter value of blank tile: y
WORD WAS: HANDY

Valid word
```

1 mark Output for `ha---` results in 'Not [illegible]d attempt, you lose your turn.'

```
Your cur[illegible]d --AHANDENONSARJ

E
     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 fill hand and stop the game.
> ha---


Not a valid attempt, you lose your turn.
```

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[illegible] additional pages
- Answer **all** questi[illegible]
- The ma[illegible]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	