Zig Zag Education

Computer Science

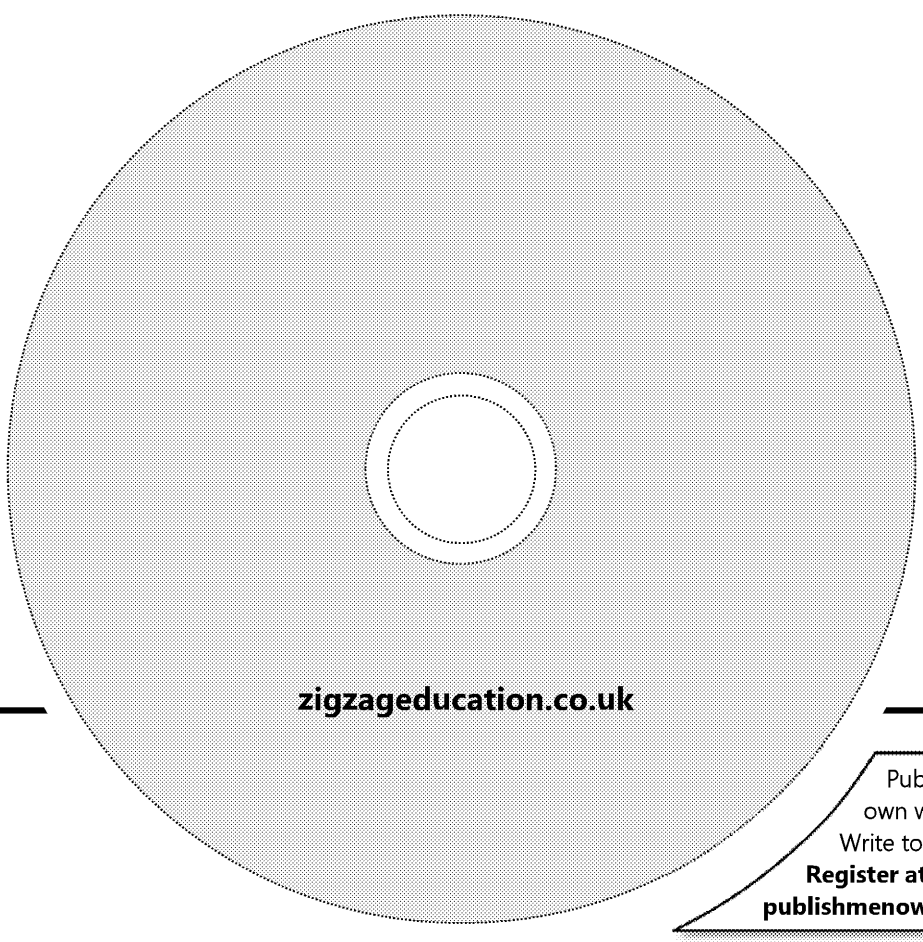
A Level | AQA | 7517

PAPER 1 EXAM RESOURCE PACK 2018

WORDS WITH AQA

JAVA

for A Level AQA Computer Science



CE2/
7892

POD
7892

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 (Java) – 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 Java 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 at random. Tiles are removed from the front of the queue and, once removed, are replaced by an identical tile at the back of the queue. The tile queue can be replenished any number of times, so the same tiles can appear more than once.

When the game begins, Player One and Player Two are each assigned 15 tiles taken from the queue, and 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 – for example, 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. The next step 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:

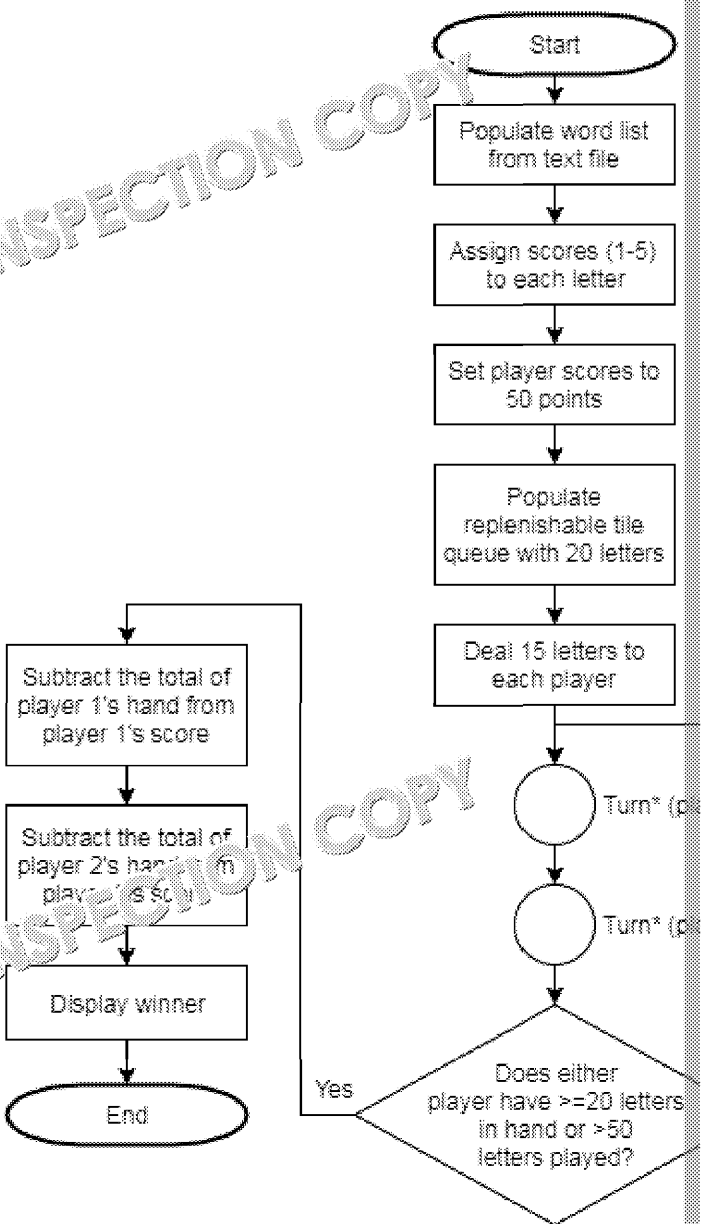
      - three tiles
      - a number of tiles equal to the length of the played word (so four tiles for a four-letter word)
      - a number of tiles equal to the length of the word plus three (so seven tiles for a four-letter word)
      - no tiles

   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 penalty points.

The game continues until either player has played a total of more than 50 tiles or has fewer than 3 (or more) in their hand. If either of these is true after Player One's turn, Player Two has one more turn and the game ends.

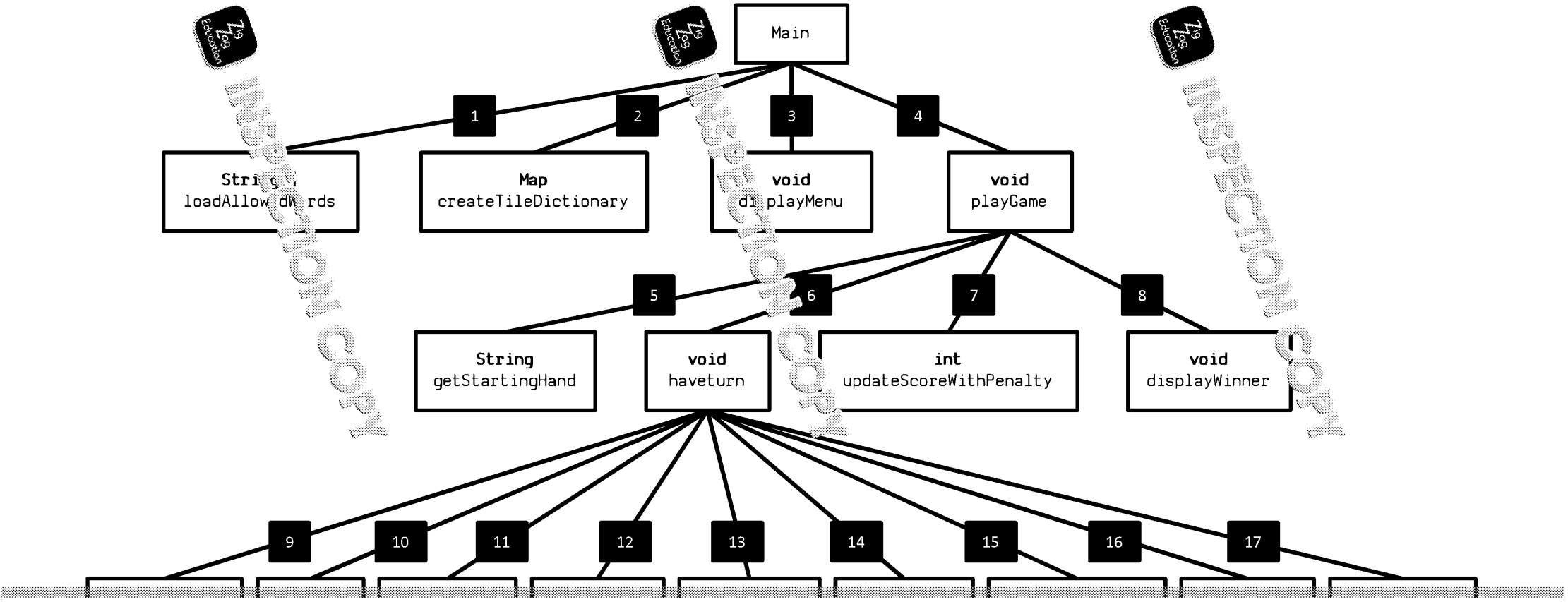
At the end of the game, the total value of each player's hand is subtracted from their score. The highest score is the winner.

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	-	String[]: allowedWords
2 Main calls createTileDictionary	-	Map: tileDictionary
3 Main calls displayMenu	-	-
4 Main calls playGame	String[]: allowedWords Map: tileDictionary Boolean: 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 Tiles: playerTiles TileCount: playerTilesPlayed Score: playerScore Map: tileDictionary QueueOfTiles: tileQueue String[]: allowedWords	-

Call	Parameters	Return
11 haveTurn calls displayTileValues	Map: tileDictionary String[]: allowedWords	-
12 haveTurn calls fillHandWithTiles	QueueOfTiles: tileQueue Tiles: playerTiles int: maxHandSize	-
13 haveTurn calls checkWordIsInTiles	String: word String: playerTiles	Boolean: inTiles
14 haveTurn calls checkWordIsValid	String: word String[]: allowedWords	Boolean: validWord
15 haveTurn calls updateAfterAllowedWord	String: word Tiles: playerTiles Score: playerScore TileCount: playerTilesPlayed Map: tileDictionary String[]: allowedWords	-
16 haveTurn calls getNewTileChoice	-	String: newTileChoice
17 haveTurn calls addEndOfTurnTiles	QueueOfTiles: tileQueue Tiles: playerTiles String: newTileChoice String: choice	-
18 updateAfterAllowedWord calls getScoreForWord	String: word Map: tileDictionary	int: score

Program Classes

The program contains five classes. Their purposes are described briefly in the table below.

Class	Description
`Main`	Most of the code resides in this c…nich handles all interaction with the user, validation, d…ning the flow of the game and identifying game end conditions.
`QueueOfTiles`	Class to store the structure which co…s tiles before they are passed to a player. The structure it…is an array, and an integer points to the rear of the queue. The front is a…s element zero in the array.
`Tiles`	Contains a string that consists of the ch…ers making up the player's hand. Each character function…one tile.
`TileCount`	Contains an integer that indicates the tota…mber of tiles played by a player.
`Score`	Contains an integer that indicates the score …player

Class Methods

Only `Main` and `QueueOfTiles` …ntain any methods. The functions Ⓕ and procedur…Ⓟ of these classes are described below.

Main – Methods	Description

`addEndOfTurnTiles` Ⓟ	Parameters: `QueueOfTiles: tileQueue`
	`Tiles: playerTiles`
	`String: newTileChoice`
	`String: choice`
	Returns: -
	CalledFrom: haveTurn
	1. Declare an integer variable without a value; this will ultimately contain the number of new tiles that will be taken
	2. If the user has entered option 1 (in _____), 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

Main – Methods	Description	
`checkWordIsInTiles` Ⓕ	Parameters: `String: word` `String: playerTiles` Returns: `Boolean` Called from: `haveTurn` Calls: -	1. Create a Boolean variable set to true 2. Create a copy of the player's tiles 3. Loop through the word being checked 4. For each character in the word 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` `String[]: allowedWords` Returns: `Boolean` Called from: `haveTurn` Calls: -	1. Create a Boolean variable set to false 2. Loop through each word in the array containing all of the words, stopping only when the end is reached or the attempted word is found 3. If a match is found, set the Boolean value to true 4. Return the Boolean to `haveTurn`
`createTileDictionary` Ⓕ	Parameters: - Returns: `Map` Called from: `Main` Calls: -	1. Create an empty map with the format 'character', 'integer'. 2. Set up a loop that runs 26 times 3. For each iteration, add the corresponding letter to the map (first add A, then add B, etc.) 4. For each letter, add the corresponding score (1, 2, 3 or 5), which depends on the letter 5. Return the map to `Main`
`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)

Main – Methods	Description	
`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: `QueueOfTiles: tileQueue`
	`Tiles: 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. Add a character to the player's hand by removing a tile from the front of the tile queue
		3. Add a tile to the back of the tile queue

`getChoice` Ⓕ	Parameters: - Returns: `String` Called from: `haveTurn` Calls: -	1. Present the player with the mid-game menu 2. Prompt the user to select a number or enter a word 3. Return this selection to `haveTurn`
`getNewTileChoice` Ⓕ	Parameters: - Returns: `String` Called from: `haveTurn` Calls: -	1. Declare an empty string 2. Prompt the user with a four-option menu 3. Return their response to this menu, which will be "1", "2", "3" or "4", to `haveTurn`
`getScoreForWord` Ⓕ	Parameters: `String: word` `Map: tileDictionary` Returns: `int` Called from: `updateAfterAllowedWord` Calls: -	1. Declare an integer variable and set it to zero 2. Loop through each character in the word, adding the score for each letter (taken as read from the map) 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

Main – Methods	Description
haveTurn Ⓟ	Parameters: String: playerName Tiles: playerTiles TileCount: playerTilesPlayed Score: playerScore Map: tileDictionary QueueOfTiles: tileQueue String[]: allowedWords int: maxHandSize int: noEndOfTurnTiles Returns: - Called from: playGame Calls: displayTilesInHand getChoice displayTileValues fillHandWithTiles checkWordIsInTiles checkWordIsValid updateAfterAllowedWord getNewTileChoice addEndOfTurnTiles

1. Display which player's turn it is
2. Display's the player's hand
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"
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
10. If the word is invalid based on a call to checkWordIsInTiles, validWord is set to false
11. If the word is valid, the move is processed by calling updateAfterAllowedWord and getNewTileChoice
12. If the word is invalid, a message is displayed saying 'Not a valid attempt, you lose your turn.'
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)
14. Console outputs the current game state

Main – Methods	Description	

`Main` Ⓟ	Parameters: -
	Returns: -
	Called from: -
	Calls: `loadAllowedW`
	`createTileDictionary`
	`displayMenu`
	`playGame`
		1. Game settings are initialised: `maxHandSize`, `maxTilesPlayed`, `noOfEndTurnTiles`, `startHandSize`
		2. Menu is displayed and use[illegible]mpted
		3. Loop continues until "9" is [illegible] to quit
		4. If "1" is entered, game is played with random tiles
		5. If "2" is entered, game is playe[illegible]h string literals defined in `playGame`

`playGame` Ⓟ	Parameters: `String[]: allowe[illegible]rds`
	`Map: tileDictiona`
	`Boolean: randomSta`
	`int: startHandSize`
	`int: maxHandSize`
	`int: maxTilesPlayed`
	`int: noOfEndOfTurnTi`
	Returns: -
	Called from: `Main`
	Calls: `getStartingHand`
	`haveTurn`
	`updateScoreWithPenalty`
	`displayWinner`
	`QueueOfTiles.QueueOfTiles`
		1. Set Player One score and Player [illegible]score to 50
		2. Set number of tiles (for both playe[illegible]o 0
		3. Create new tile queue containing 2[illegible]s
		4. If a random start has been requested (in `Main`), hands are populated randomly
		5. Otherwise, hands are populated with s[illegible] literals
		6. Loop to run until either player has reache[illegible]the maximum number of tiles played (50) or the maximum number of tiles in hand (20)
		7. Call `haveTurn` alternately for Player One an[illegible] Player Two until the loop terminates
		8. Update scores of both players
		9. Display the winner by calling `displayWinner`

`updateAfterAllowedWord` Ⓟ	Parameters: `String: word`
	`Tiles: playerTiles`
	`Score: playerScore`
	`TileCount: playerTilesPlayed`
	`Map: tileDictionary`
	`String[]: allowedWords`
		1. Add the length of the word just played to the total number of tiles played
		2. Loop through each character in the played word, removing a corresponding tile from the player's hand
		3. Update the player's score by calling `getScoreForWord`

QueueOfTiles – Methods	Description	
`QueueOfTiles` Ⓟ	Parameters: `int: maxSize` Returns: - Called from: `Main.playG` Calls: `add`	1. Constructor method – create a new `QueueOfTiles` object when called 2. Dimension an array of size `ze` (the parameter) 3. Set `rear` to -1. This variable `e` pointer to the back of the queue. Since the array is initially empty, there can be no meaningful rear pointer. 4. Call the `add` method repeatedly `` if `maxSize` is 20, call `add` 20 times.
`isEmpty` Ⓕ	Parameters: - Returns: `Boolean` Called from: remove Calls: -	1. If `rear` is -1 (meaning the pointer `ot` within the array, so the array can be considered empty) retu `ue` 2. For any other value of `rear`, return `f`

`remove` Ⓕ	Parameters: -
	Returns: `char`
	Called from: `Main.getStartingHand`
	`Main.addEndOfTurnTile`
	`Main.fillHandWithTile`
	Calls: `isEmpty`
		1. If the list is empty, return a line break c`cter`
		2. Otherwise:
		a. Store the character at element ze`o` the array
		b. Move all other characters in the ar`y` back one place, closer to element zero
		c. Add a line break character to the resul`g` empty element at the end of the array
		d. Subtract 1 from `rear`
		e. Return the character stored in (a)

`add` Ⓟ	Parameters: - Returns: - Called from: `Main.getStartingHand` `Main.addEndOfTurnTiles`	*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 from 0 to 25

Variables

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

Main – Variable	Type	Description	Created in
`allowedWords`	`String[]`	Contains all valid words read from a text file	`loadAllowedWords`
`choice`	`String`	A word played by a player. A reference passed as a parameter, its new name is 'word' instead of 'choice'.	`haveTurn`
`choice`	`String`	Contains user input at the main in-game menu, indicating whether they want to display player 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`
`newTileChoice`	`String`	Contains user input, indicating whether they want to replace the number of tiles they just played, take three tiles, take the number of tiles played plus three or take two tiles. User input is "1", "2", "3" or "4" respectively.	`getNewTileChoice`
`noOfEndOfTurnTiles`	`int`	The number of tiles drawn by default after a turn (this variable is only partially implemented – it is passed to subroutines and subsequently not used)	`Main`
`playerOneScore`	`int`	Player One's score	`playGame`
`playerTwoScore`	`int`	Player Two's score	`playGame`

QueueOfTiles – Variables	Type	Description	Created in
contents	char[]	Character 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 [illegible] used to add new tiles to the correct location in the array	N/A (instance variable)
maxSize	int	The largest size that the array can be	N/A (instance variable)

The following classes in the skeleton program each contain a single variable but no methods. They exist to allow their variables to be returned from a subroutine. As classes are used instead of global variables, multiple instances of each can be created, i.e. multiple scores, multiple sets of tiles, etc.

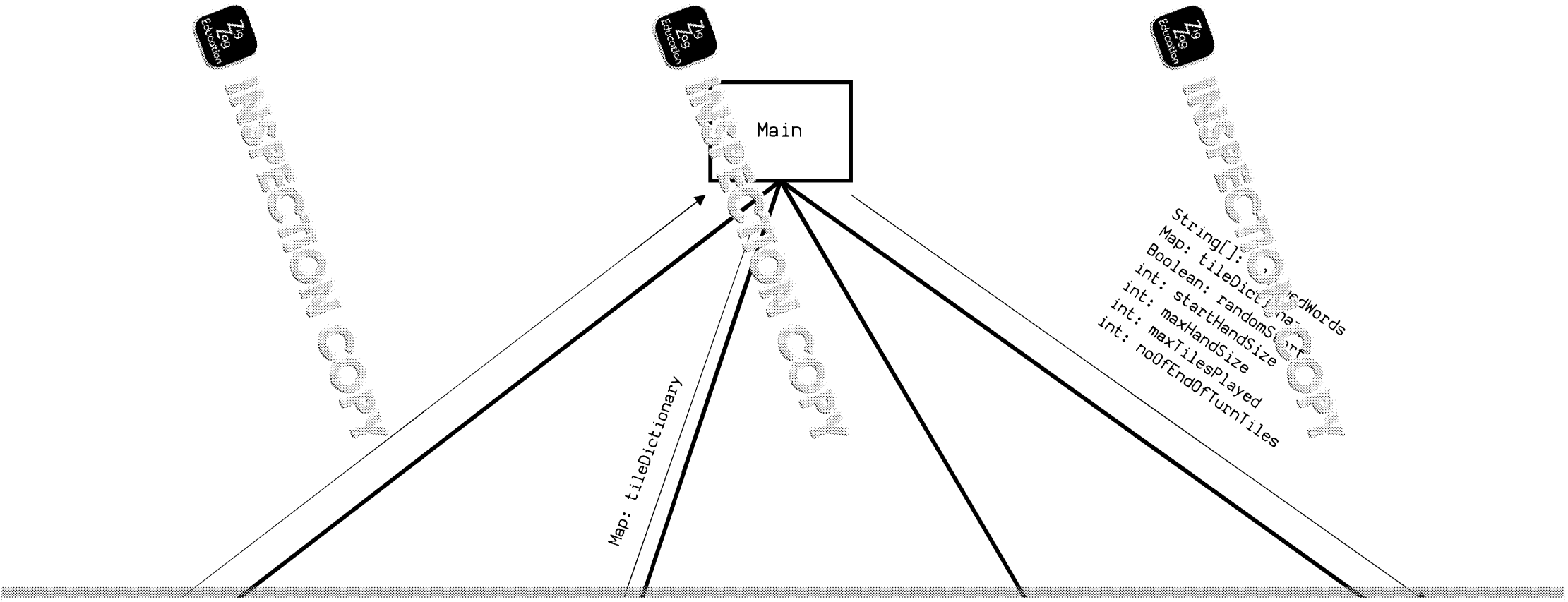
Tiles – Variables	Type	Description	Created in
playerTiles	String	A player's hand, with each tile represented by one character within the string	N/A (instance variable)

TileCount – Variables	Type	Description	Created in
numberOfTiles	int	The number of tiles played by a player	N/A (instance variable)

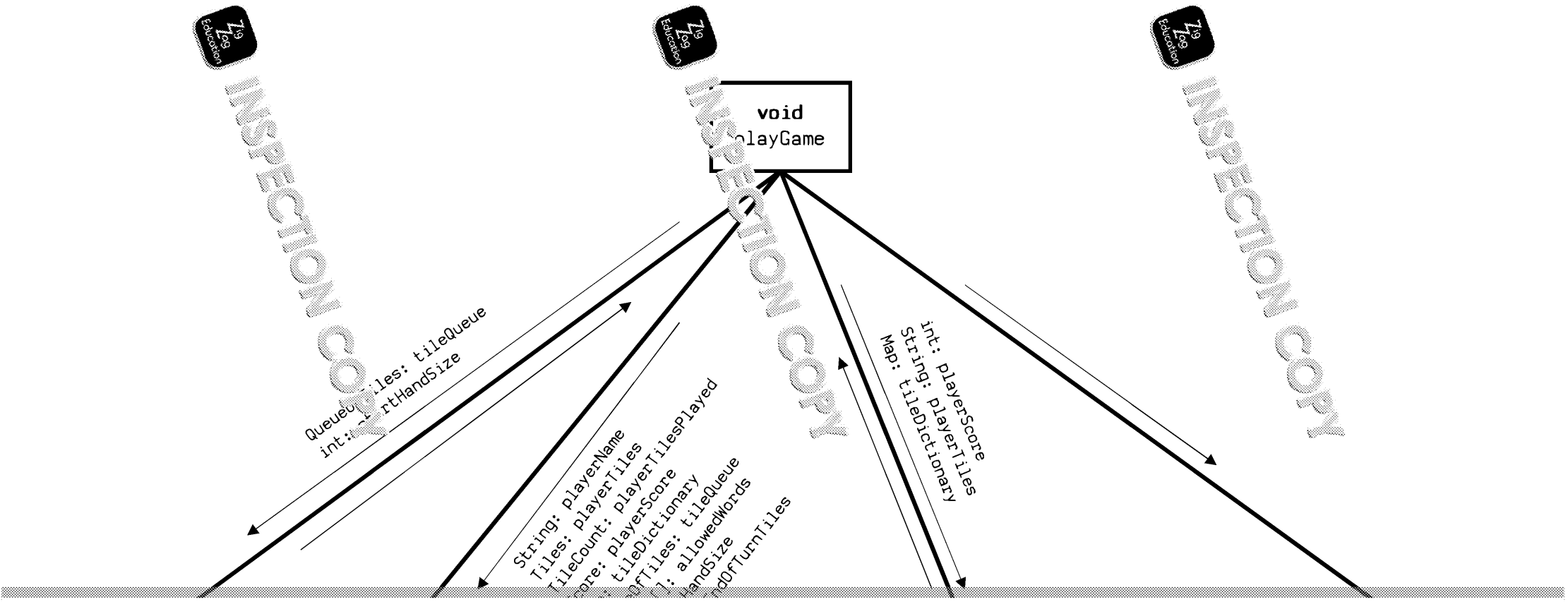
Score – Variables	Type	Description	Created in
score	int	The score of a player	N/A (instance variable)

Structure Diagram Activity 1

Main and subroutines called from **Main**

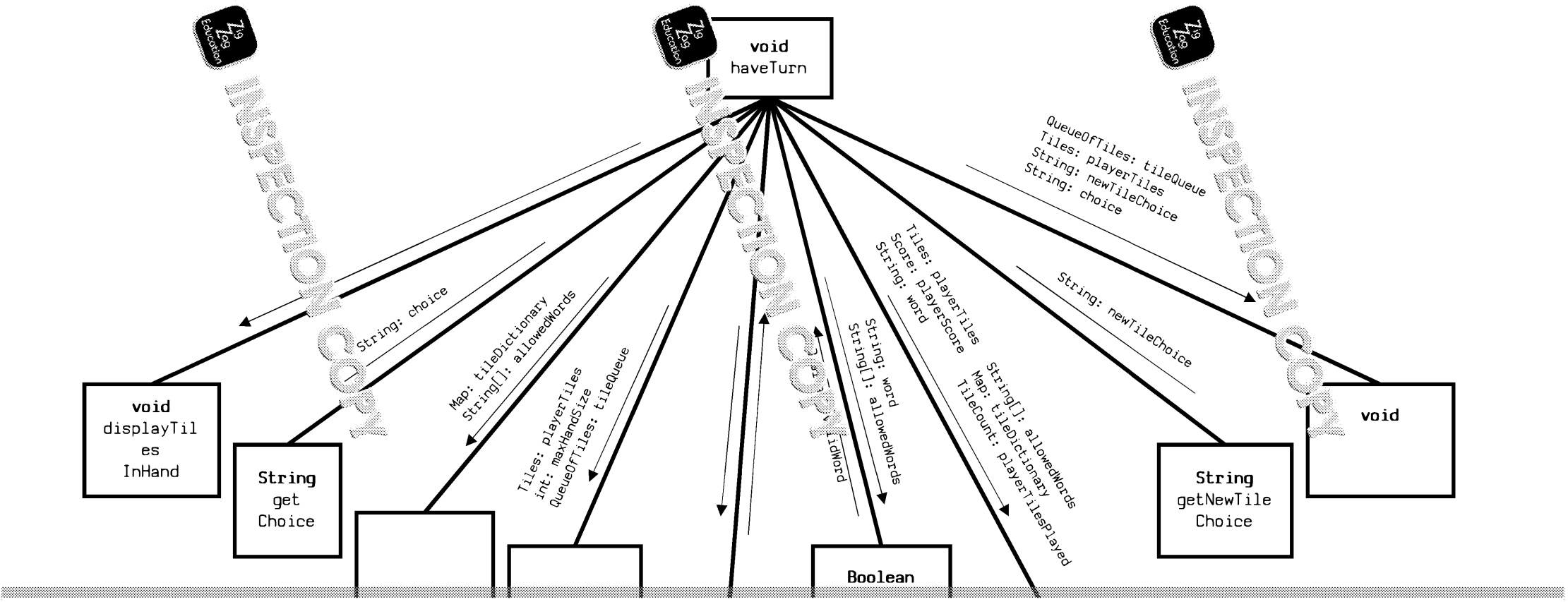


playGame and subroutines called from **playGame**



Structure Diagram Activity 3

haveTurn and subroutines called from **haveTurn**



Written Questions (Java)

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) A class [1]
   b) An array variable [1]
   c) A variable that is used to store a Boolean return value [1]
   d) A parameter whose data type is map [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. Write three lines of code from the skeleton program, each of which calls a di
3. Look at the subroutine `createTileDictionary`. Describe the purpose of

```
tileDictionary.put((char)(65 + count), 1);
```

4. State and describe the data structure returned by the `createTileDictio`
5. Look at the subroutine `checkWordIsInTil` Explain the role of the var
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.
8. Describe the actions performed in the following lines of the `loadAllowed`

```
List<String> stringList = Files.readAllLines(filePa
allowedWords = new String[stringList.size()];
```

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 the `QueueOfTiles` co
12. Describe in detail the purpose of the subroutine `updat coreWithPenal` and/or return values in your answer. [5]
13. Describe the operation of the follow within the subroutine `getSco`

```
int score = 0;
for (int  = 0; count < word.length(); count++)
{
    e += (int)tileDictionary.get(word.charAt(co
}
```

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 (Java)

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 variable [1]

   .....

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

   .....

   d) A parameter whose data type is map [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. Write three lines of code from the skeleton program, each of which calls a different

   .....

   .....

   .....

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

```
tileDictionary.put((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 var

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`

```
List<String> stringList = Files.readAllLines(filePa
allowedWords = new String[stringList.size()];
```

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

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... `updateScoreWithPenal`... and/or return values in your a...

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

```
int score = 0;
for (int count = 0; count < word.length(); count++)
{
    score += (int)tileDictionary.get(word.charAt(co
}
```

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

15. Explain why the vari... `TileChoice` is initialised to the string value "2"...

Programming Tasks (Java)

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
  - Press '4' to request no new tiles
- One screen capture showing
  sequence of events:
  - Beg
  - On *Player One*'s first turn, play the word 'BARD'
  - Press '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` 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

Task 3

This task 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 P... for `createTileDictionary`
- One screen capture sh... no... 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 t

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `playGame`
- Your amended SOURCE CODE PROGRAM for `displayWinner`
- One screen capture showing *Player One*'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 t... 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 store 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 `QueueOfTiles` object called `tileQueue` wi
for the constructor.

Create acc[illegible]ethods 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 th
are replaced by an equivalent number of letters from the tile queue. There shou
for changing letters.

Once the letters have been swapped, the new ha[illegible] be displayed, but pla
player.

Evidence you need to pr[illegible]:

- Yo[illegible]nd [illegible]OURCE CODE PROGRAM for `getChoice`
- You[illegible]ded 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) unti
the same as the previous letter. At that point, the letter sh[illegible]d be added to the

Evidence you need to provide:

- Your amended SOURCE C[illegible]OGRAM 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 only modify `loadAllowedWords` in your response to this task. You validation code and 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 (i.e. `Main()`, `public static void main()`) s returns an empty array and a 'File error' is displayed and the program end

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 ra[illegible]'

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 l
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 cla[illegible]`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 `displ` le to read '3. Settings'.
- Alter the code in `Ma` that if '3' is chosen from the menu, the user w following p turn:
  - te maximum hand size:
  - ter 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 v `maxTilesPlayed`, `noOfEndOfTurnTiles` and `startHandSize` re

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 t 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 selec 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 randomly ch sen vowel

ASCII values for the vowels are as follows: A=65, =6 =73, O=79, U=85

Evidence you need to provi

- Your CE CODE PROGRAM for the `QueueOfTiles` class
- On nshot showing *Player One*'s tiles after choosing to play with a

Task 15a

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 c play the role of blank
- For each dash entered, the user is given the prompt 'Enter value of b ide or 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 immediat `checkWordIsValid` 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.playerTiles = "BTAHA`
    TO: `playerOneTiles.pla.e Tiles = "--AHA`
    (i.e. change the first two characters to dashes)
  - Run the program and select the training hand option
  - Play the following: han--
- One screenshot showing the output that results from the following:
  - Before running the program, ensure that the line of code has bee
  - 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`

Task 15b

This task refers to `haveTurn` and `updateAfterAllowedWord`, and assumes t
this task is working correctly.

Currently, when a blank tile is played, it is not removed from the player's hand. If
HANDY tries to play the word 'handy' using the blanks, the letter tiles 'H', 'A', 'N',
player's hand, and the player keeps the blanks.

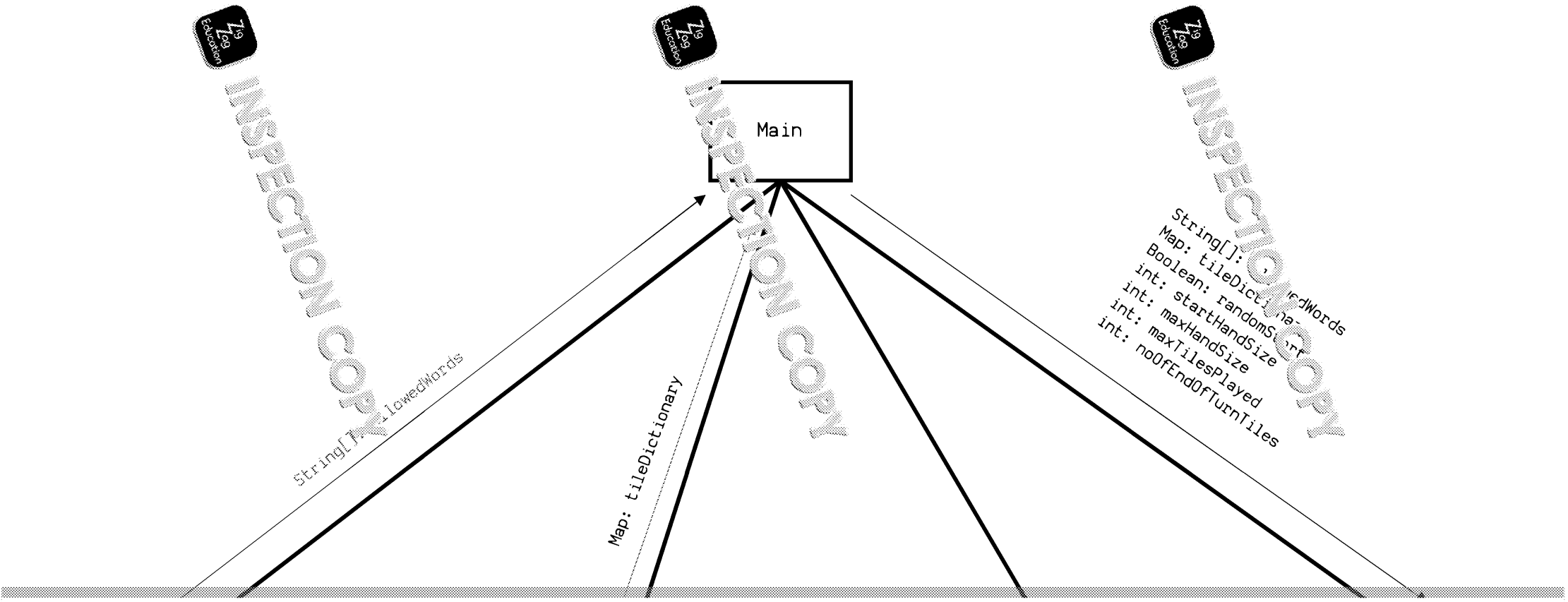
Modify the program so that the score for a tile is determined based on the letters that
any blank tiles that are used are now removed from the player's hand.

Before running the program, ensure that the value of `playerOneTiles.playe`
`--AHANDEM---RJ`.

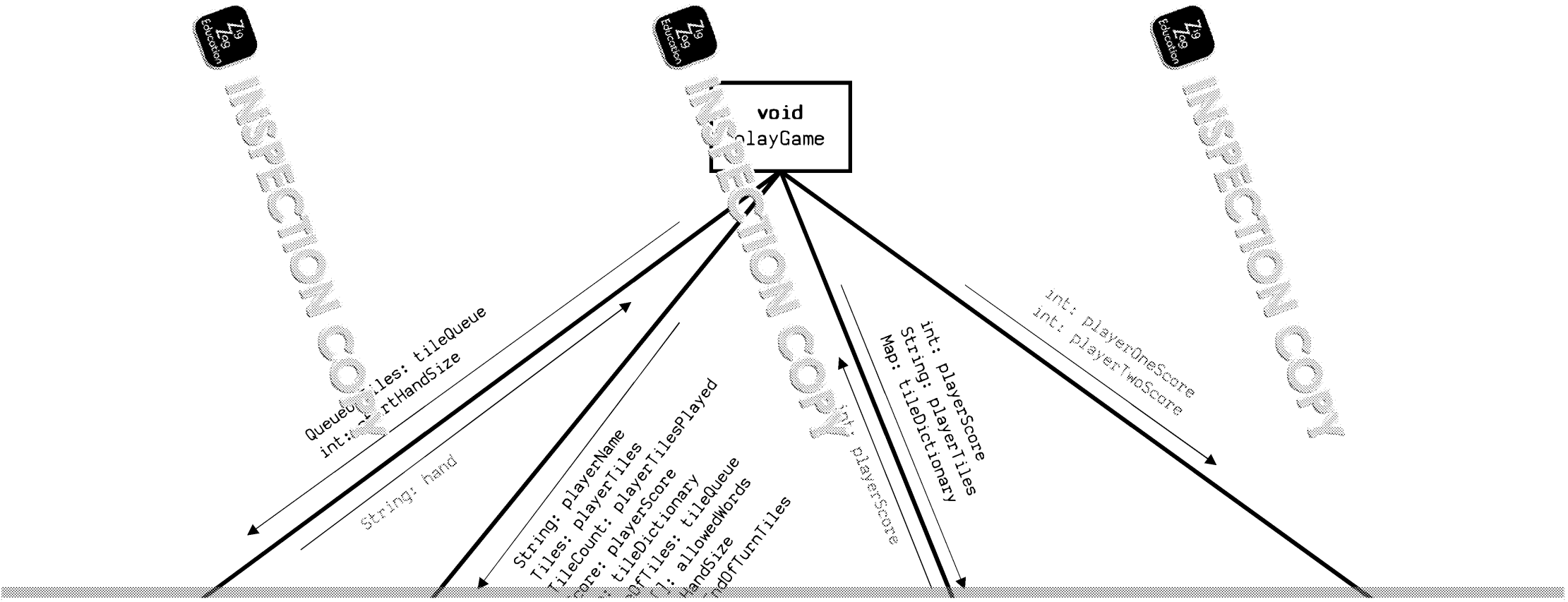
Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `haveTurn`
- Your amended SOURCE CODE PROGRAM for `updateAfterAllowedWo`
- One screenshot showing the output that results from the following:
  - 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`
  - Select option 4 to take no new tiles
  - Continue running the game, entering any inputs, until *Player One*
    turn is shown.

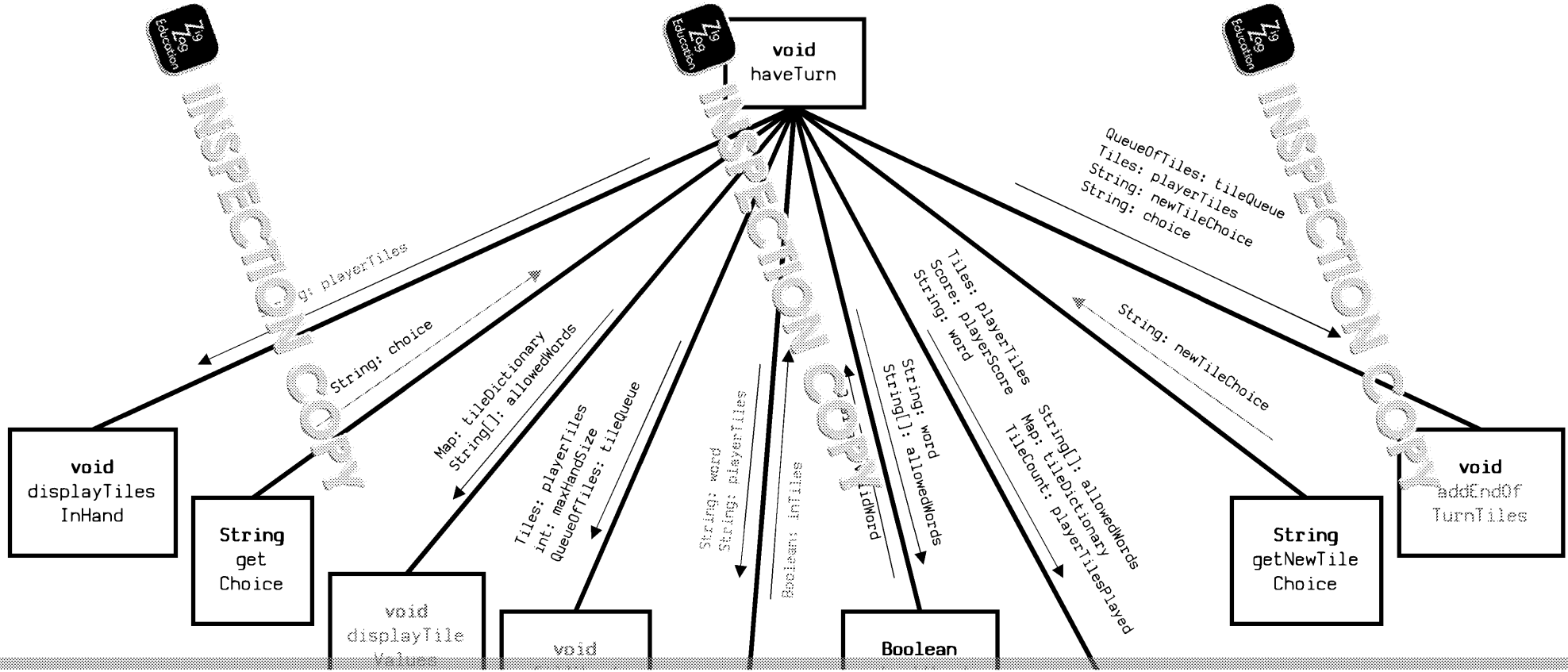
Main and subroutines called from **Main**

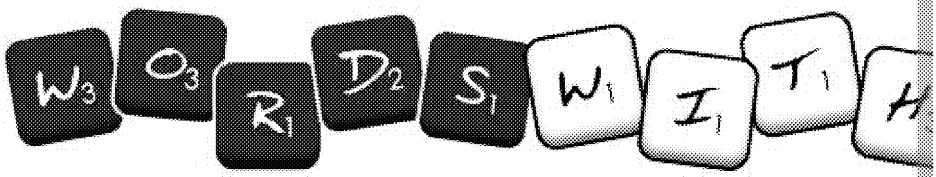


playGame and subroutines called from **playGame**



haveTurn and subroutines called from **haveTurn**



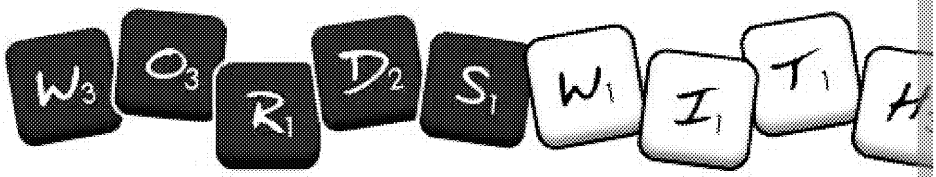


Written Questions: Suggested Answers and Mark Scheme

Q	Answer/Guidance
1a	`Main/QueueOfTiles/Tiles/TileCount/Score`
1b	`contents/allowedWords`
1c	`inTiles/validWord`
1d	`tileDictionary`
1e	`updateScoreWithPenalty`
1f	`displayMenu/Main/add/show`
1g	`newTileChoice/validChoice/validWord/choice`
1h	`rear/maxSize/contents`
2	1 mark each for up to three of the following (NB listed alphabetically). The whole line must be included for the mark: • `allowedWords = new String[stringList.size()];` • `allowedWords[count] = word.trim().toUpperCase();` • `Charset charset = Charset.defaultCharset();` • `newChoice = newChoice.toUpperCase();` • `copyOfTiles = copyOfTiles.replaceFirst(word.charAt(c` • `for(char letter : word.toCharArray())` • `if(copyOfTiles.contains(word.charAt(count) + ""))` • `int points = (int)tileDictionary.get(letter);` • `int randNo = rnd.nextInt(25);` • `List<String> stringList = Files.readAllLines(filePath` • `Path filePath = new File("aqawords.txt").toPath();` • `playerScore -= (int)tileDictionary.get(playerTiles.c` • `score += (int)tileDictionary.get(word.charAt(count));` • `tileDictionary.put((char)(65 + count), 1);` (or '2', '3' or '5' Also: • Any line containing `.equals` • Any line containing `.length` • Any line containing `Console.print / Console.println / Cons`
3	1 mark for each point: • Adds (an entry) to the dictionary • Entry is the character with ASCII value of 65 + 'count' • Entry includes integer value '1'

Q	Answer/Guidance
4	1 mark for stating data structure: • Map Up to 2 marks for description: • Contains unique keys • Keys are letters of the alphabet • Each key links to another value • Other value is an integer / t[illegible] for that letter
5	Up to 2 marks for explanation: • Creates a copy of the player's tiles • Matches are eliminated from the copy • Prevents tiles being eliminated from player's tiles (since word has not
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: • `add / QueueOfTiles.add` • `addEndOfTurnTiles` • `displayMenu` • `displayTilesInHand` • `displayTileValues` • `displayWinner` • `fillHandWithTiles` • `haveTurn` • `Main` • `playGame` • `QueueOfTi[illegible]` • `[illegible]ueOfTiles.show` • `[illegible]teAfterAllowedWord` 1 mark for identifying a function: • `checkWordIsInTiles` • `checkWordIsValid` • `createTileDictionary` • `getChoice` • `getNewTileChoice` • `getScoreForWord` • `getStartingHand` • `isEmpty / QueueOfTiles.isEmpty` • `loadAllowedWords` • `remove / QueueOfTiles.remove` • `updateScoreWithPenalty`
7	3 marks: • An exception would be thrown • Execution would jump to the 'catch' block • The 'catch' section contains no code/instructions • [illegible] array would be returned
8	4 marks: • String list is created • Populated with each line in turn from the file path • String array is created • Dimensioned / set to size of list

Q	Answer/Guidance
9	2 marks: • A constructor is a method called by the command `new` • Creates a new object based on the class in which it resides
10	2 marks: • Sets the instance variable (`maxSize`) • ... to the parameter (`maxSize`)
11	2 marks: • `Rear` points to the back of the queue • Value of -1 indicates an empty queue • Returns a dequeue value
12	1 mark for parameters: • Player's current score, tiles in the player's hand, `Map` that links tiles to 1 mark for return value • Player's updated score Up to 3 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 tile/character in the playe • Value of each tile is determined by looking up in map • Value is subtracted from player's score
13	4 marks: • Score/integer/variable is set to zero • Loop iterates through each character in the word • Value of character looked up in the map/`tileDictionary` • Value added to score
14	2 marks • Loop continues while 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 Sche

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 le

1 mark Score decremented correctly within the IF statement

```
if(word.length() > 7)
{
    score += 20;
}
else if(word.length() > 5)
{
    score += 5;
}
else if(word.length() == 2 || word.length() == 3)
{
    score--;
}
```

1 mark Screenshot shows 'HAD' was played, the new total is '55' and the total ...

```
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 this game.
```

Question 2

1 mark `displayTilesInHand` uses additional parameter of data type `Map` (A used to grant access to the map, but marks 6 and 7 for this question wo

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

1 mark Display the character (**R:** if the whole hand is displayed also displayed in its o

1 mark Display the value of the letter taken from the `Map`

1 mark Correct format to include brackets and a single space after each clo

```
displayTilesInHand(String PlayerTiles, Map tileD
{
    Console.println();
    for (int x = 0; x < PlayerTiles.length(); x++)
    {
        Console.print(PlayerTiles.charAt(x));
        Console.print("(");
        Console.print((int)tileDictionary.get(PlayerT
        Console.print(") ");
    }

}
```

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

```
Console.println(playerName + " it is your turn.");
displayTilesInHand(playerTiles.playerTiles, tileDict
String newTileChoice = "2";
```

1 mark Subsequent call for player's hand from `haveTurn` uses new argument corre

```
else if (choice.equals("7"))
{
  displayTilesInHand(playerTiles.playerTiles, tileDi
}
```

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 the case structure that assigns th

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

```
case 5:
case 7:
case 22:
    tileDictionary.put((char)(65 + count), 3);
    break;
case 10:
case 21:
case 24:
    tileDictionary.put((char)(65 + count), 4);
    break;
```

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

```
TILE VALUES

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

Question 4

1 mark Adding a score for *Player Three*

```
Score playerOneScore = new Score();
playerOneScore.score = 50;
Score playerTwoScore = new Score();
playerTwoScore.score = 50;
Score playerThreeScore = new Score();
playerThreeScore.score = 50;
```

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

```
TileCount playerOneTilesPlayed = new TileCount();
playerOneTilesPlayed.numberOfTiles = 0;
TileCount playerTwoTilesPlayed = new TileCount();
playerTwoTilesPlayed.numberOfTiles = 0;
TileCount playerThreeTilesPlayed = new TileCount();
playerThreeTilesPlayed.numberOfTiles = 0;
```

1 mark Creating an empty hand for *Player Three*

```
Tiles playerOneTiles = new Tiles();
Tiles playerTwoTiles = new Tiles();
Tiles playerThreeTiles = new Tiles();
```

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

```
if(randomStart)
{
  playerOneTiles.playerTiles = getStartingHand(tileC
  playerTwoTiles.playerTiles = getStartingHand(tileC
  playerThreeTiles.playerTiles = getStartingHand(tile
}
```

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

```
else
{
  playerOneTiles.playerTiles = "BTAHANDENONSARJ";
  playerTwoTiles.playerTiles = "CELZXIOTNESMUAA";
  playerThreeTiles.playerTiles = "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.numberOfTiles <= maxTiles
       playerTwoTilesPlayed.numberOfTiles <= maxTile
       playerThreeTilesPlayed.numberOfTiles <= maxT
       playerOneTiles.playerTiles.length() < maxHand
       playerTwoTiles.playerTiles.length() < maxHand
       playerThreeTiles.playerTiles.length() < maxHa
{
```

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

```
haveTurn("Player Two", playerTwoTiles, playerTwoTiles
        playerTwoScore, tileDictionary, tileQueue, a
        maxHandSize, noOfEndOfTurnTiles);
Console.println();
Console.println("Press Enter to continue");
Console.readLine();
Console.println();
haveTurn("Player Three", playerThreeTiles, playerThre
        playerThreeScore, tileDictionary, tileQueue,
        maxHandSize, noOfEndOfTurnTiles);
```

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

```
playerOneScore.score = updateScoreWithPenalty(player
        playerOneTiles.playerTiles, tileDictionary);
playerTwoScore.score = updateScoreWithPenalty(player
        playerTwoTiles.playerTiles, tileDictionary);
playerThreeScore.score = updateScoreWithPenalty(playe
        playerThreeTiles.playerTiles, tileDictionary
```

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

```
displayWinner(playerOneScore.score, playerTwoScore.
        playerThreeScore.score);
```

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

1 mark Score for *Player Three* is displayed

```
void displayWinner(int playerOneScore, int playerTwo
                   playerThreeScore)
{
  Console.println();
  Console.println("**** GAME OVER! ****");
  Console.println();
  Console.println("Player One your score is " + playe
  Console.println("Player Two your score is " + playe
  Console.println("Player Three your score is " + pla
```

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 a covers all other combinations; **R:** if other text is displayed)

```
if (playerOneScore > playerTwoScore && playerOneScore
{
  Console.println("Player One wins!");
}
else if (playerTwoScore > pla  on  core &&
        playerTwoScore  y  hreeScore)
{
  Console.p  l  Player Two wins!");
}
e  f (playerThreeScore > playerOneScore &&
        playerThreeScore > playerTwoScore)
{
    Console.println("Player Three wins!");
}
else
{
  Console.println("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  k  to play OR
     press 1 to di  th  tter values OR
     press 4  e tile queue OR
     es  iew your tiles again OR
     es 0 to fill hand and stop the game.
>


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
     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 y  tur

Your  arr  ABCDEFGHIJKLMNO

E
     nter 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.
>
```

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 the variable or place the concatenated string

1 mark Correct parameters – choice, tileDictionary

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 currentScore = getScoreForWord(choice, tileDic

  Console.println();
  Console.println("Valid word. " + choice + " scores
          currentScore + " points.");
  Console.println();
  updateAfterAllowedWord(choice, playerTiles, playerS
          playerTilesPlayed, tileDictionary, allowedW
  newTileChoice = getNewTileChoice();
}
```

1 mark Input of word 'ABANDONS' displays 'to... worth 30 points.
(**DPT:** spacing errors penalised in fourth mark)

```
Your current hand: BTAHANDENONSARJ

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.
> 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 names (**A:** alternative names if mea

1 mark All attributes use correct data types (**R:** if any additional attributes

1 mark Attributes declared `private`

```
private int score;
private int numberOfTiles;
private String tiles;
```

1 mark Constructor declared with single parameter `QueueOfTiles tileQue`

1 mark `score` and `numberOfTiles` (or their equivalents) initialised to `50` an

1 mark `tiles` (or its equivalent) initialised using a call to `getStartingHand`

1 mark `getStartingHand` contains correct parameters (**A:** integers other tha

```
Player(QueueOfTiles tileQueue)
{
    score = 50;
    numberOfTiles = 50;
    tiles = getStartingHand(tileQueue, 15);
```

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

1 mark Accessor methods all declared `public`

```
public int getScore()
{
    return score;
}

public int getNumberOfTiles()
{
    return numberOfTiles;
}

public String getTiles()
{
    return tiles;
```

Question 7

1 mark Addition of the extra option in `getChoice`

```
Console.println("Either:");
Console.println("      enter the word you would like
Console.println("      press 1 to display the letter
Console.println("      press 2 to swap your letters O
Console.println("      press 4 to view the tile queue
Console.println("      press 7 to view your tiles aga
Console.println("      press 0 to fill hand and stop
Console.print("> ");
```

1 mark Inclusion of 'else if' 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` 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.equals("2"))
{
    String newPlayerTiles = "";
    for (int x = 0; x < playerTiles.playerTiles.length
    {
        newPlayerTiles = newPlayerTiles + tileQueue.
        tileQueue.add();
    }
    playerTiles.playerTiles = newPlayerTiles;
    validChoice = true;
    newTileChoice = "4";
    Console.write("New tiles: " + playerTiles.player
}
```

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: BTAHANDENONSARJ

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 (**I:** access modi

```
protected char[] contents;
protected int rear;
protected int maxSize;
private char lastLetterAdded;
```

1 mark Loop inside existing 'if' to iterate until a duplicate has not bee
intent of loop i

1 mark G on 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 subrouti

```
void add()
{
    Random rnd = new Random();
    if (rear < maxSize - 1) {
        rear += 1;
        boolean duplicate = true;
        while (duplicate)
        {
            int randNo = rnd.nextInt(25);
            contents[rear] = (char) (65 + randNo);
            if (!(contents[rear] == lastLetterAdded))
            {
                duplicate = false;
            }
        }
        lastLetterAdded = contents[rear];
    }
}
```

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.println("(1) Default Dictionary");
Console.println("(2) Custom Dictionary");
Console.print("> ");
String choice = Console.readLine();
Console.println();
```

1 mark Declaration of variable to store file path (**A:** input concatenated directly the try block)

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 or concatenated within the try block

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

```
String path;
if (choice.equals("1"))
{
    path = "aqawords.txt";
}
else
{
    Console.println("Enter file name");
    Console.print("> ");
    path = Console.readLine() + ".txt";
}
```

1 mark File path specified by the user is accessed

```
try {
    Path filePath = new File(path).toPath();
```

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, i.e. == 0 or <1

1 mark Correct output message

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

```
String[] allowedWords = loadAllowedWords();

if (allowedWords.length == 0)
{
    Console.println("File error");
    System.exit(0);
}
```

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

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


File error
BUILD SUCCESSFUL (total time: 0 seconds)
```

Question 11

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

```
Console.println("You have " + playerTiles.playerTile
        " tiles remaining.");
newTileChoice = getNewTileChoice();
```

1 mark Message correctly placed (ever… sage is incorrect) immediately aft… addEndOfTurnTil… (**A:** f… is added outside the 'if' clause, but o… clause to prev… e message being redisplayed in the event that no n…

1 mark Si… ncatenation to display message in the specified format in both

```
if (!newTileChoice.equals("4"))
{
  addEndOfTurnTiles(tileQueue, playerTiles, newTileC
  Console.println("You have " + playerTi
          " tiles remaining.");
}
```

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
     get three extra tiles (2
     replace the tile… and get three extra tiles (3
     get no new …es (4)?
> 3
Y… …ve … tiles remaining.

Y… 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 mess… be 'Your word was: BRAT'

```
Valid word

You have 11 tiles remaining.
Do you want to:
     replace the tiles you used (1
     get three extra tiles (2
     replace the tile… and get three extra tiles (3
     get no ne… …es (4)?
> 4
Y… …rd 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 wi[illegible] names (**R:** if any other names are u

1 mark All [illegible] correct data types

1 mark At[illegible]s declared private

```
private char letter;
private int score;
private boolean isVowel;
```

1 mark Constructor written as `LetterTile`

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

1 mark Map parameter (**A:** if named other than '`tileDictionary`')

1 mark Letter attribute set using char parameter

1 mark Attribute always contains upper-case version of character

1 mark Score attribute set by extracting a value fr[illegible]e map (even if value ex

1 mark Score would always be [illegible] the parameter passed to 'get' should

```
[illegible](char letter, Map tileDictionary)
{
    this.letter = Character.toUpperCase(letter);
    this.score = (int)tileDictionary.get(this.letter
```

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 tha upper case or lower case

1 mark isVowel attribute set correctly with this selection

```
if (this.letter == 'A' || this.letter == 'E' ||
        this.letter == 'I' || this.letter == 'O' ||
        this.letter == 'U')
{
    isVowel = true;
}
else
{
    isVowel = false;
}
```

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

1 mark Accessor methods declared public

```
public char getLetter()
{
    return letter;
}

public int getScore()
{
    return score;
}

public boolean isVowel()
{
    return isVowel;
}
```

Question 13

1 mark New entry in `displayMenu`

```
void displayMenu()
{
  Console.println();
  Console.println("=========");
  Console.println("MAIN MENU");
  Console.println("=========");
  Console.println();
  Console.println("1. Play game with random start ha
  Console.println("2. Play game with training start
  Console.println("3. Settings");
  Console.println("9. Quit");
  Console.println();
}
```

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

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 → integer conversion for all inputs

1 mark Input prompts relate correctly to all four variables

```
else if (choice.equals("3"))
{
    Console.print("Enter maximum hand size: ");
    maxHandSize = Integer.parseInt(Console.readLine(
    Console.print("Enter maximum tiles played: ");
    maxTiles = Integer.parseInt(Console.readLi
    Console.print("Enter default new tiles: ");
    endOfTurnTiles = Integer.parseInt(Console.rea
    Console.print("Enter starting hand size: ");
    startHandSize = Integer.parseInt(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
2. Play game
3.
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

```
protected char[] contents;
protected int rear;
protected int maxSize;
int vowelCounter = 0;
```

1 mark `rear` is still incremented by 1 when 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 += 1;
    if (vowelCounter < 2)
    {
        int randNo = rnd.nextInt(25);
        contents[rear] = (char)(65 + randNo);
        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 changed to ensure that the next selection will be a random l

```
else
{
    int randNo = rnd.nextInt(4);
    switch (randNo)
    {
        case 0: contents[rear] = (char)(65);
            break;
        case 1: contents[rear] = (char)(69);
            break;
        case 2: contents[rear] = (char)(73);
            break;
        case 3: contents[rear] = (char)(79);
            break;
        default: contents[rear] = (char)(85);
    }
    vowelCounter = 0;
}
```

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

Question 15a

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)
{
  choice = resolveBlanks(choice);
  Console.println("Choice: " + choice);
  validWord = checkWordIsValid(choice, allowedWords);
```

1 mark M declaration for `resolveBlanks` with a string return type (**R:** a va 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 Selection statement to handle the presence of a dash

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

1 mark User input is stored in variable

1 mark Input is converted to upper case

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

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

1 mark Structure is such that all dashes would be replaced by characters enter

1 mark St rrectly returned

```
String resolveBlanks(String word)
{
    String newLetter;
    for(int x = 0; x < word.length(); x++)
    {
        if (word.charAt(x) == '-')
        {
            Console.print("Enter value of blank tile
            newLetter = Console.readLine().toUpperCa
            word = word.substring(0, x) + newLetter +
                    word.substring(x + 1);
        }
    }
    return word;
}
```

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

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

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
```

Question 15b

1 mark Additional parameter in `updateAfterAllowedWord` to contain word

1 mark This new parameter is iterated through to remove tiles from the player

```
void updateAfterAllowedWord(String wordWithBlanks, S
        Tiles playerTiles, Score playerScore, TileCo
        Map tileDictionary, String[] allowedWords)
{
    playerTiles.numberOfTiles += word.length();
    for (letter : wordWithBlanks.toCharArray())
    {
        playerTiles.playerTiles =
                playerTiles.playerTiles.replaceFirst
    }
    playerScore.score += getScoreForWord(word, tileD
}
```

1 mark Extra variable alongside `choice` in `haveTurn` - one will contain a wo the other will contain that word with the blank tiles resolved

```
while (!validChoice)
{
  String choice = getChoice();
  String wordWithBlanks = choice;
```

1 mark Call to `resolveBlanks` that c in one of the two variables conta contains the word with the blanks resolved

1 mark Ca `cerAllowedWord` with the new parameter included

```
if (validWord)
{
  choice = resolveBlanks(wordWithBlanks);
  Console.println("WORD WAS: " + choice);
  validWord = checkWordIsValid(choice, allowedWords);
  if (validWord)
  {
    Console.println();
    Console.println("Valid word");
    Console.println();
    updateAfterAllowedWord(wordWithBlanks, choice,
            playerTiles, playerScore, playerTilesPla
            tileDictionary, allowedWords);
    newTileChoice = getNewTileChoice();
  }
}
```

1 mark Player One's hand at start of second turn contains ADENONSARJ

```
Your word was:HANDY
Your new score is:60
You have played 5 tiles so far in this game.

Press Enter to continue



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 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>


Not a valid attempt, you lose your turn.


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

Player One it is your turn

Your current hand: ADENONSARJ
```

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 any additional pages

- Answer **all** questions
- The marks available 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	