Zig Zag Education

Computer Science A Level | AQA | 7517

2015 specification for the 2023 exam

PAPER 1 EXAM RESOURCE PACK 2023

Dastan

for A Level AQA Computer Science

JAVA EDITION

- DIGITAL RESOURCE -

This pack includes paper versions of the electronic files.

Go to **zzed.uk/ProductSupport** to download the electronic files.

POD 11729

zigzageducation.co.uk

Contents

** Note there are also electronic copies of the UML Diagrams ('Complete' & 'Activity' versions) which can be printed in A3, making them much more usable (especially when used as activities)*

Teacher's Introduction

This resource pack is designed to help you support your students taking the **A Level Computer Science Paper 1** exam. It is based on the ***Dastan*** preliminary material (Java) – for examination summer 2023.

DIGITAL RESOURCE

Once you have downloaded the files for this resource via (**zzed.uk/ProductSupport**) you will have access to the following:

`Dastan`	this folder contains all of the content (PDF/DOCX) accessible via a HTML interface
`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: Extract the files from the downloaded ZIP file and move the entire `Dastan` 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 accessible via the *Solutions* web page are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

- `j02a-UML-Diagam-Complete.pdf`
- `j06-TheoryQuestions-MS.pdf`
- `j07-CodingTasks-MS.pdf`

The resource pack consists of the following:

① **Code Breakdown**

This document gives a detailed technical overview of the skeleton program, describing in detail each class and method in turn – including their purpose/function, parameters and return values.

> **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.

② **Class Diagrams**

Two UML Class Diagrams help students explore the skeleton program; there is a completed version and a partially-complete version which contains a total of 15 missing class and method names, data types, associations and access types for students to fill in. The completed version is password-protected and accessible via the *Solutions* web page.

③ **Video**

A short video going over the *Dastan* game mechanics – intended as a visual aid to accompany the notes in the official AQA preliminary material.

④ **Written Questions**

Theory questions testing students' understanding of the skeleton program. These questions require access to the program, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on versions are available. Suggested answers are provided via the *Solutions* web page as a password-protected PDF.

⑤ **Coding Tasks**

Fifteen modification exercises put students' programming skills to the test. Example solutions with suggested mark schemes are provided via the *Solutions* web page as a password-protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

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

Dastan

Skeleton Code Breakdown

Class: Dastan

Identifier / Data		Description
<<constru		
Parameter	r : Int c : Int noOfPieces : Int	Initialises the following protected attri • noOfRows from parameter r • noOfColumns from paramet • moveOptionOfferPosition t
Return values	n/a	
		Instantiates two new Player objects parameter of 1 and Player 2 with the and appends them both to the protec Assigns the element at position 0 of t (Player 1) to the protected attribute c Invokes the following methods: • createMoveOptions() – to add to each playe • creat ptionOffer() – to o ior the move offer option reateBoard() – to create a sta • createPieces() – to add the sta to the board using the paramete
calculate a,urePoints (private)		
Parameter	finishSquare Reference : Int	Uses the getPieceInSquare method location from the finishSquareRefer
Return values	Integer	If there is a piece at that location, the that piece is returned. If there is no p returns 0.
checkIfGameOver (private)		
Parameters	n/a	Iterates through the board list check
Return values	Boolean	If the square contains a piece, the m contains a Kotla, and the piece in the to the opponent of the player that ow scenario, the play who owns the M opponent's f. If this isn't the cas confirm if the piece contains either a g the player1HasMirza and pla appropriately. A negated logical AND of these two players have lost their Mirza, the met returns false.

checkSquareInBounds (private)		
Parameters	squareReference : Int	Used as an error handling meth squareReference parameter is playing board.
Return values	Boolean	
		The method initialises two local DIV to s[illegible] if the row and MO the squareReference paramet confirm if Row is outside of the noOfRows and col is outside o noOfColumns and returns fals If both are in range, the method

checkSquareValid (private)		
Parameters	squareReference : Int startSquare : Boolean	Used to test if the squareRefe Square choice.
Return values	Boolean	The startSquare parameter is being used to check when the p of a piece to move from (a 'mov passed as false when the meth when the player is selecting the 'move to' check).
		The method firstly uses the che to confirm that the square refer the board and returns false if it The method then gets the piece the squareReference paramet location and this is a 'move fro false because the player has s startSquare parameter is true method instead returns true be a blank square. If there is a piece already at the the method checks to confirm i current player. If it does and th method returns true. If this is a returns false because the playe onto one of their own pieces. If the piece does not belong to 'move from' check, the method player is trying to select an opp a 'move to' check, the method attempting to take an opponen

createBoard (private)		
Parameters	n/a	Uses nested iteration using the noOfColumns attributes to po
Return values	n/a	
		Player 1's Kotla is placed to the Player 2's Kotla is placed in the there is an even number of col noOfRows attribute. The remaining locations are fill object.

createChowkidarMoveOption (private)		
Parameters	direction : Int	Instantiates a new MoveOptio method uses the direction par Move objects – one for each v option.
Return values	newMoveOption : MoveOption	
		The first new Move parameter fr st ng location to finishin Move parameter is the numbe starting location to finishing loc to the starting location.
		A direction of 1 moves down t direction of -1 moves up the b Move object is added to the ch object which is then returned.
		See pre-release document for valid move positions (shown fr

createCuirassierMoveOption (private)		
Parameters	direction : Int	Instantiates a new MoveOptio method uses the direction par Move objects – one for each v option.
Return values	newMoveOption : MoveOption	
		The first new Move parameter from starting location to finishin Move parameter is the numbe starting location to finishing loc to the starting location.
		A direction of 1 moves down t direction of -1 moves up the b Move object is added to the cu object which is then returned.
		See pre-release document for valid move positions (shown fr

createFaujdarMoveOption (private)		
Parameters	direction : Int	Instantiates a new MoveOptio method uses the direction par Move objects – one for each v option.
Return values	newMoveOption : MoveOption	
		The first new Move parameter from starting location to finishin Move parameter is the numbe starting location to finishing loc to the starting location.
		A direction of 1 moves down t direction of -1 moves up the b Move object is added to the fa which is then returned.
		See pre-release document for valid move positions (shown fr

createJazairMoveOption (private)		
Parameters	direction : Int	Instantiates a new MoveOptio method uses the direction par Move objects – one for each v option.
Return values	newMoveOption : MoveOption	
		The first new Move parameter fr st ng location to finishin Move parameter is the numbe starting location to finishing loc to the starting location.
		A direction of 1 moves down t direction of -1 moves up the b Move object is added to the ja which is then returned.
		See pre-release document for valid move positions (shown fr

createRyottMoveOption (private)		
Parameters	direction : Int	Instantiates a new MoveOptio method uses the direction par Move objects – one for each v option.
Return values	newMoveOption : MoveOption	
		The first new Move parameter from starting location to finishin Move parameter is the numbe starting location to finishing loc to the starting location.
		A direction of 1 moves down t direction of -1 moves up the b Move object is added to the ry which is then returned.
		See pre-release document for valid move positions (shown fr

createMoveOption (private)		
Parameters	name : String direction : Int	Uses selection on the name pa associated create*****MoveOp MoveOption from that method
Return values	MoveOption	

createMoveOptionOffer (private)		
Parameters	n/a	Adds th default moveOpt attribute.
Return values	n/a	

createMoveOptions (private)		
Parameters	n/a	Adds the five default moveOpt moveOptionQueue for each p This method calls the createM the move name and direction default m options, adding th MoveOptionQueue for Player l
Return values	n/a	

createPieces (private)		
Parameters	noOfPieces : Int	Places the default playing piec onto the board. The method uses the noOfPie many standard playing pieces Player 1 pieces on row 2 and P penultimate row. Pieces are gi which player they belong to, th their symbol on the board. Play symbol '!'. Player 2 pieces are using an escape character to c The method also places the Pl associated Kotlas by halving th work out the middle position in points value if captured of 5. P symbol of '1' and Player 2 Mirz
Return va	n/a	

displayBoard (private)		
Parameters	n/a	Iterates through the board list The method works by using the • Iterate through to the num column number and a spa • Iterate through to the num sequence of hyphens. • Use nested iteration to pri for each square on the bo a piece in the square the printed, otherwise a blank • Print a final '|' symbol at th • Iterate through to the num sequence of hyphens follo
Return values	n/a	

displayFinalResult (private)		
Parameters	n/a	The winner of the game is the when this method is called. Th of both players using the getS If Player 1 has a higher score t uses the getName() method to concatenated with 'is the winne Player 2 name concatenated w scores match, 'Draw!' is printed
Return values	n/a	

displayState (private)		
Parameters	n/a	Used as part of the main menu method to display information a
Return values	n/a	The method first calls the displ board to the screen followed by for a pla to choose if they w
		I the uses the getPlayerStat the score and move option que followed by the current player n

getIndexOf[illegible]		
Paramete	squareReference : Int	Used to convert a squareRefe the board list for the associate
Return values	Integer	The method initialises two loca DIV to split off the row and MO the squareReference parame
		1 is subtracted from both varia and then the row is multiplied b and added to the col attribute a

getPointsForOccupancyByPlayer (private)		
Parameters	currentPlayer : Player	Used to calculate the total poin the currentPlayer.
Return values	scoreAdjustment : Int	The method initialises an integ to 0 t e ates through the b hic re occupied by the curr
		The getPointsForOccupancy square in the board list which method is overridden by the K Kotla belongs to current playe player Mirza or a current playe Kotla occupied by a current pl the opponent player.
		Points are totaled up in the sc iteration progresses.
		This total is then returned.

getSquareReference (private)		
Parameters	description : String	Used to get a square reference
Return values	selectedSquare : Int	The method uses the descript an appropriate output to the us used for r a start or finish
		I t from the user is casted w stored in a local integer variabl returned.

playGame (public)		
Parameters	n/a	This method is the main game the loop using the local Boolea
Return values	n/a	The method firstly displays the includes the board and the cur the curr ayer to choose a m on queue or select 9 m e offer. If the user selects option 9, the useMoveOptionOffer() to disp and then displays the current g loops until the user selects a va The method then asks the use startSquareReference contai like to move. Using the getSqu checkSquareIsValid() method the user gives a valid location. The method then repeats this p finishSquareReference conta player wants to move the piece checkPlayerMove() method to startStartReference and finis for the selected move choice. If the move is legal, the metho • Calculates any points if th piece using the calculate and storing it in pointsFo • Updates the player score move option used from th changeScore() method. • Updates the player queue MoveOption choice to the updateQueueAfterMove • Calls the updateBoard() of pieces based on the st finishSquareReference. • Calls the updatePlayerS current player score with • Prints the updated score f screen. This method does not deal with not legal, it simply just ignores player t ithout informing th The method then checks which swaps to the opposing player. checkIfGameOver() to check their Mirza into the opponent K been captured which stops the After the main game playing lo displayState() method to print board and then calls the displa confirm which player has won.

useMoveOptionOffer (private)		
Parameters	n/a	Used to place the move i into the current player m
Return values	n/a	The method asks the pla current offer move from t wi using any error h l integer variable rep uses the updateMoveO method on the currentP selected position move w the moveOptionOffer lis the player score using th based on the position of t the player to replace. The method then update variable with a random n new move from the mov

updateBoard (private)		
Parameters	startSquareReference : Int finishSquareReference : Int	Performs the actual mov on the board to another.
Return values	n/a	The method uses the re board list index calculate startSquareReference p subse uently passed as nd to be placed at t om the finishSquareR

updatePlayerScore (private)		
Parameters	pieceCapture : Int	Calculates the change in which the player has just
Return va	n/a	The method calls the getPointsForOccupanc current player to create a Kotlas which are occupie added to the pointsForP contains the points for ar that move. The combined total is the player score using the c

Class: Piece

Identifier / Data		Description
<<constructor>>		
Parameters	t : String b : Player p : Int s : String	Initialises the following protected • type of piece from paramet • belongsTo from parameter • pointsIfCaptured from para • symbol from parameter s
Return values	n/a	
getBelongsTo (public)		
Parameters	n/a	Returns the value of the protecte
Return values	belongsTo : Player	
getPointsIfCaptured (public)		
Parameters	n/a	Returns the value of the protecte
Return values	pointsIfCaptured : Int	
getSymbol (public)		
Parameters	n/a	Returns the value of the protecte
Return values	symbol : String	
getTypeOfPiece (public)		
Parameters	n/a	Returns the value of the protecte
Return values	typeOfPiece : String	

Class: Square

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the following protected • pieceInSquare to null • belongsTo to null • symbol to ' '
Return values	n/a	
containsKotla (public)		
Parameters	n/a	If the Symbol attribute is a 'K' o to confirm that there is a Kotla p returns false.
Return values	Boolean	
getBelongsTo (public)		
Parameters	n/a	Returns the value of the protecte
Return values	belongsTo : Player	
getPieceInSquare (public)		
Parameters	n/a	Returns the value of the protecte
Return values	pieceInSquare : Piece	

getPointsForOccupancy (public)		
Parameters	currentPlayer : Player	Base class method for the get[illegible] method in the Kotla class to ov[illegible] If the method was not overridd[illegible]
Return values	Integer	
getSymbol (public)		
Parameters	n/a	R[illegible] the value of the protec[illegible]
Return values	symbol : String	
removePiece (public)		
Paramete[illegible]	[illegible]	Used for removing a piece fron[illegible] The method makes a temporar[illegible] attribute pieceInSquare in a lo[illegible] then sets the attribute to null to [illegible] It then returns the variable piec[illegible]
Return va[illegible]	pieceToReturn : Piece	
setPiece (public)		
Parameters	p : Piece	Assigns the p parameter to the pieceInSquare.
Return values	n/a	

Class: Kotla (inherits from Square)

Identifier / Data		Des[illegible]n
<<constructor>>		
Parameters	p : Player s : [illegible]	Initialises the following parent a[illegible] • belongsTo from paramete[illegible] • symbol from parameter s[illegible]
Return va[illegible]	n/a	
getPointsForOccupancy (public) <<overrides>>		
Parameters	currentPlayer : Player	Overrides the getPointsForOc[illegible] base class to return the score p[illegible] occupied. The method checks first to see [illegible] square. If there is not, the meth[illegible] If there is a piece in the Kotla s[illegible] to see if the Kotla square belon[illegible] currentPlayer passed in as a p[illegible] the piece in the Kotla is either a[illegible] also ow[illegible]y the currentPlay[illegible] p[illegible]s. [illegible]otla square belongs [illegible] is [illegible] Mirza or standard piece in[illegible] points. If the Kotla square belongs to t[illegible] piece in it is either a Mirza or a [illegible] currentPlayer, the method ret[illegible] returns zero points.
Return values	Integer	

Class: MoveOption

Identifier / Data		Description
<<constructor>>		
Parameters	n : String	Initialises the following p
Return values	n/a	ame from paramet possibleMoves to a
addToPossibleMoves (public)		
Parameters	m : Mo	Adds the m parameter to list.
Return va		
checkIfTh sAMoveToSquare (public)		
Parameters	startSquareReference : Int finishSquareReference : Int	Used to check if the star by the player are valid st MoveOption.
Return values	Boolean	
		The method initialises fo and startColumn togeth finishColumn. The meth startRow and MOD to s the startSquareReferen same techniques to split finishColumn from the parameter. Th ethod then iterates sibleMoves list chec startColumn and finish combination represent a possible positions a piec
getNam e)		
Parameter	n/a	Returns the value of the
Return values	name : String	

Class: Move

Identifier / Data		Description
<<constructor>>		
Parameters	r : Int c : Int	Initialises the following protected • rowCh ge from paramete • c nt hange from param
Return values	n/a	
getRowChange (public)		
Parameters	n/a	Returns the value of the protecte
Return va	ange : Int	
getColum ge (public)		
Parameters	n/a	Returns the value of the protecte
Return values	columnChange : Int	

Class: MoveOptionQueue

This class does not have a specific constructor and therefore uses the default constru

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the queue pr MoveOption list.
Return values	n/a	
add (public)		
Parameters	newMoveOption : MoveOption	Adds the newMoveOp queue list.
Return va	n/a	
getMoveOptionInPosition (public)		
Parameters	pos : Int	Returns the MoveOpti queue list.
Return values	MoveOption	
getQueueAsString (public)		
Parameters	n/a	Initialises a local empty queueAsString and a which it assigns 1. The method then iterat concatenating the cou name of each Move in getName() method), in reach loop. The method then return
Return values	queueAsString : String	
moveItemToBack (public)		
Parameter	: Int	Used for moving a Mov queue list. The method makes a te MoveOption at the ind The method then uses on the queue list to rem index position. It then appends the tem MoveOption back into effect of placing it at the
Return va	n/a	
replace (public)		
Parameters	position : Int newMoveOption : MoveOption	Assigns the newMove queue list at the index parameter.
Return values	n/a	

Class: Player

Identifier / Data		Description
<<constructor>>		
Parameters	n : String d : Int	Initialises the following p score to 100 • name from paramet • direction from para
Return values	n/a	
addToMoveOptionQueue (public)		
Parameters	newMoveOption : MoveOption	Adds the newMoveOpti queue attribute.
Return values	n/a	
changeScore (public)		
Parameters	amount : Int	Increments the protected parameter.
Return values	n/a	
checkPlayerMove (public)		
Parameters	pos : Int startSquareReference : Int finishSquareReference : Int	Used to check if a move using the checkIfTherel The method creates a te move selected from the p parameter. The method then passes finishSquareRefere checkIfThereIsAMoveT the references represent selected move option.
Return values	Boolean	
getDirection (public)		
Parameters	n/a	Returns the value of the
Return values	direction : Int	
getName (public)		
Parameters	n/a	Returns the value of the
Return values	name : String	
getPlayerStateAsString (public)		
Parameters	n/a	Used to expose the getQ MoveOptionQueue clas the player. The method returns a co attribute and the player q string using the getQueu
Return values	String	
getScore (public)		
Parameters	n/a	Returns the value of the
Return values	score : Int	

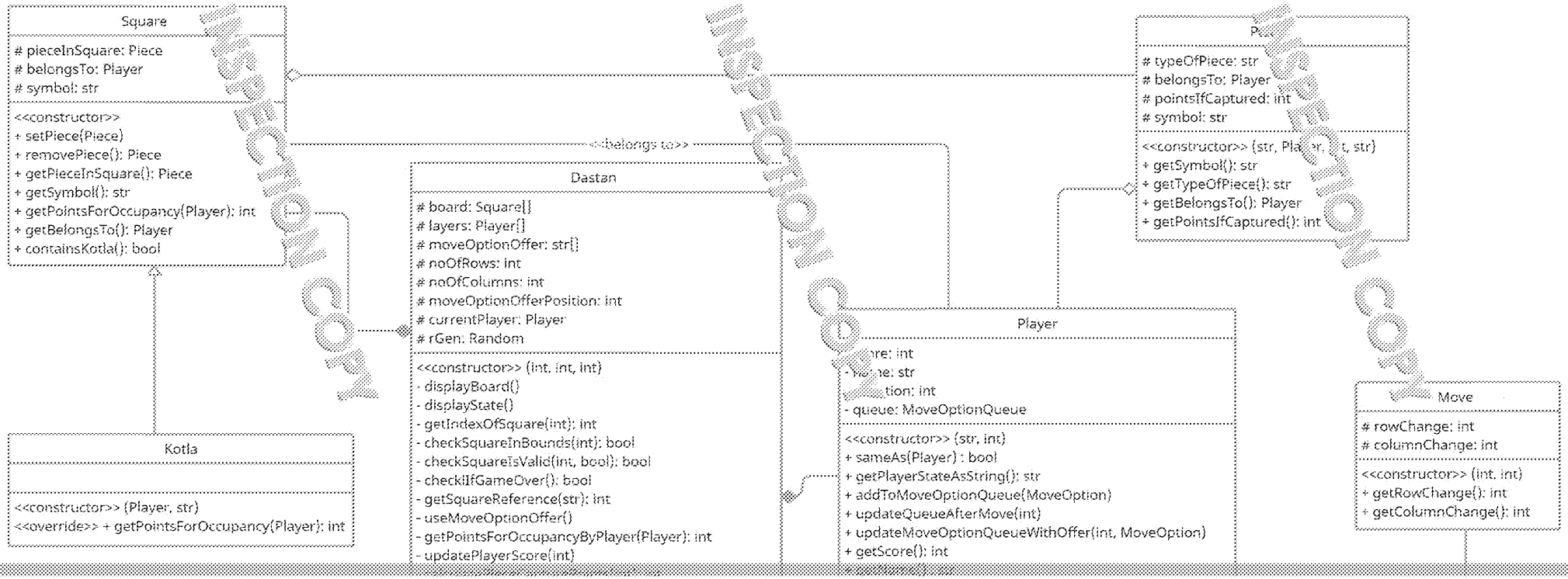
sameAs (public)		
Parameters	aPlayer : Player	Used to check if the aP as this player object.
Return values	Boolean	The method first checks object has been passed false if it is null.
		If not, the method comp parameter with the nam match, the method retu they are the same play

updateMoveOptionQueueWithOffer (public)		
Parameters	position : Int newMoveOption : MoveOption	Used to expose the rep MoveOptionQueue cla the player.
Return values	n/a	The method calls the re queue, passing the pos parameters. This will re index of position with t parameter.

updateQueueAfterMove (public)		
Parameters	position : Int	Used to expose the mo the moveOptionQueu through the player.
Return values	n/a	The method calls the m the player queue passi minus one to make it ze move option at that inde moved to the back of th

Dastan

UML Class Diagram



Dastan

Exam-style Questions

These questions refer to the **Preliminary Material** and the **Sk**
but **do not** require any additional programmin

TOTAL MARKS: 60

---

1 This question refers to the playGame method in the Dastan class.

The method contains a nested loop with multiple while loops inside th

(a) State the time complexity of this loop.

.....

(b) Explain the efficiency of this time complexity and how well it scale

.....

.....

.....

.....

.....

2 This question refers to the entire pre-release code.

Throughout the code there are many literals such as 'mirza', 'jazair', 'n
others.

(a) Describe one problem that could occur due to this.

.....

.....

.....

(b) Describe one possible solution to this problem.

.....

.....

.....

3 This question refers to the private method getPointsForOccupancyBy

Explain precisely how polymorphism is used when calculating the scor

.....

.....

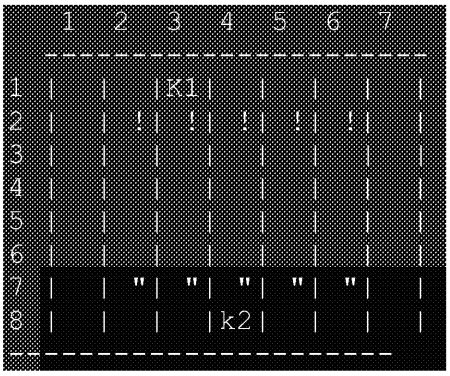
.....

.....

.....

4 This question refers to the main method that is executed at the start o

When thisGame is instantiated, currently the arguments 6, 6, 4 are pa



(a) Assume the arguments 8, 7 o passed to Dastan instead.

Explain why pl Kotla and Mirza would appear in colum opp o's as per the image above.

.....

.....

.....

(b) Describe how the code for the createBoard method of the Dastan so that where there are an odd number of columns, then the Kotla central column but when there are an even number it will remain a

.....

.....

.....

.....

.....

.....

5 This game refers to the private methods **createRyottMoveOption**, **createFaujdarMoveOption**, **createJazairMoveOption**, **createCuirassie**
createChowkidarMoveOption.

Currently the methods take a **direction** parameter which changes bet
to whose turn it is. Across the methods there is a lot of repeated use o
which always gets multiplied by any non-zero parameter to the constr

Without suggesting any specific code, de[illegible] alternative logic that c
the **direction** parameter by modifying the **addToMoveOptionQueue** a
updateMoveOptionQueueWithOther methods of the **Player** class.

.....

.....

.....

.....

.....

6 This question refers to the **MoveOptionQueue** class.

The game uses a queue data structure rather than a stack.

(a) Explain why a queue is a more suitable data structure than a stac

.....

.....

.....

(b) Currently this class uses a list to store the queue data structure; e
modified to use an array to implement a circular queue with five e

You should not write any actual code for this question but refer to
may be required and create any algorithms using structured/desc
Alternatively, you may produce an annotated diagram.

.....

.....

.....

.....

.....

.....

.....

.....

7 This question refers to the method getIndexOfSquare in the Dastan c

Explain how the private method getIndexOfSquare works.

.....

.....

.....

.....

.....

8 The board is currently represented as a one-dimensional array, but th alternative representations.

(a) Explain how the board could be represented as a two-dimensiona

.....

.....

.....

(b) State one reason why an array is more appropriate to store the bo

.....

.....

9 It woul possible to create a save game file for Dastan. At the start metadata.

Explain the purpose of metadata and give one example of metadata th Dastan.

.....

.....

.....

.....

10 This question refers to the `createMoveOptions`, `createMoveOption`, `createChowkidarMoveOption`, `createRyottMoveOption`, `createFaujdarMoveOption`, `createJazairMoveOption` and `createCuirassierMoveOption` methods in the `MoveOption` class.

(a) Currently the `MoveOption` class holds the details for whichever move is generated/populated by one of the `createChowkidarMoveOption`, `createRyottMoveOption`, `createFaujdarMoveOption`, `createJazairMoveOption`, `createCuirassierMoveOption` methods in the Dastan class.

Explain why this is NOT polymorphism.

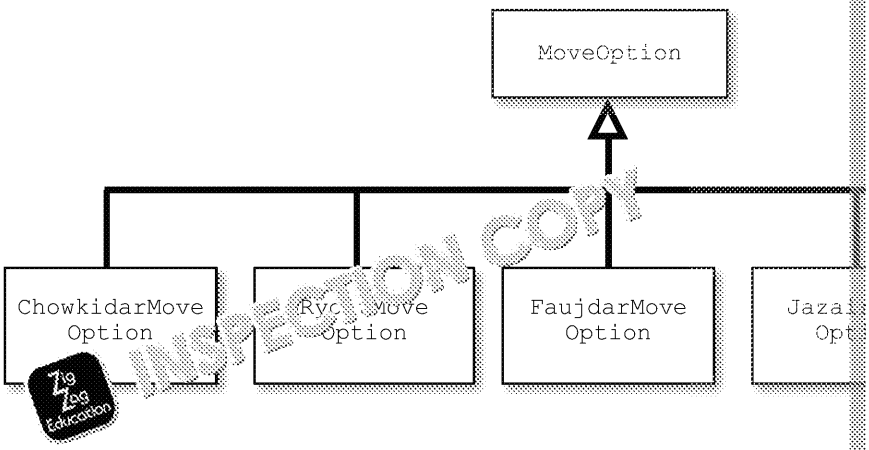
.....

.....

.....

.....

(b) An alternative would have been to create and use an inheritance structure as follows:



Explain how this inheritance structure could have been used effectively.

.....

.....

.....

11 This question refers to the `Kotla` class.

(a) The constructor includes a call using `super()`. Explain the purpose.

.....

.....

.....

(b) The method `getPointsForOccupancy` has a different implementation to the same name in the parent class. State the name for this OOP feature.

.....

11 **(c)** Explain what the OOP technique *overloading* is used for.

.....

.....

.....

.....

.....

12 The MoveOptionQueue class implements a normal queue, which is a data structure.

Explain the different between a normal queue and a priority queue.

.....

.....

.....

.....

.....

.....

13 This question [illegible] the constructor of the Piece class and the setP[illegible] Squa[illegible]s.

Both methods take a parameter *P* which is unclear. Explain why varia[illegible] meaningful names.

.....

.....

.....

14 This question is about access levels for attributes and methods and re[illegible]

(a) The Piece class has four protected a[illegible] ut[illegible] what does the word[illegible] context?

.....

.....

.....

14 **(b)** The Piece class has four public methods; what does the word 'pu

.....

.....

(c) There is one final level of access for attributes and methods which mean?

.....

.....

(d) W[illegible]t important to have access modifiers such as private, prot methods and attributes in OOP?

.....

.....

.....

.....

.....

15 This question refers to the checkSquare[illegible]s method of the Da

(a) This method uses intec[illegible]is[illegible], explain the difference between floating point div[illegible]

.....

.....

.....

(b) This method returns a Boolean value. Describe the meaning of B

.....

.....

END OF QUESTIONS

Dastan

Exam-style Questions

These questions refer to the **Preliminary Material** and the **Sk**
but **do not** require any additional programmin

TOTAL MARKS: 60

---

1 This question refers to the playGame method in the Dastan class.

The method contains a nested loop with multiple while loops inside th

(a) State the time complexity of this loop.

(b) Explain the efficiency of this time complexity and how well it scale

2 This question refers to the entire pre-release code.

Throughout the code there are many literals such as 'mirza', 'jazair', 'n
others.

(a) Describe one problem that could occur due to this.

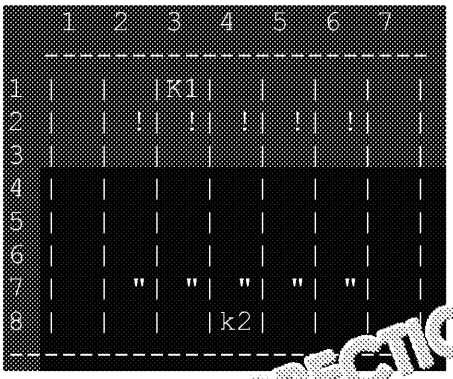
(b) Describe one possible solution to this problem.

3 This question refers to the private method getPointsForOccupancyBy

Explain precisely how polymorphism is used when calculating the score

4 This question refers to the main method that is executed at the start o

When thisGame is instantiated, currently the arguments 6, 6, 4 are pa



(a) As the arguments 8, 7, 5 were passed to Dastan instead.

Explain why player one's Kotla and Mirza would appear in column
opposite player two's as per the image above.

(b) Describe how the code for the createBoard method of the Dastan
so that where there are an odd number of columns, then the Kotla
central column but when there are an even number it will remain a

5 This game refers to the private methods createRyottMoveOption, cre
createJazairMoveOption, createCuirassierMoveOption and createCh

Currently the methods take a direction parameter which changes bet
to whose turn it is. Across the methods there is a lot of repeated use o
which always gets multiplied by any non-zero parameter to the constr

Without suggesting any specific code, describe alternative logic that c
the direction parameter by modifying the addMoveOptionQueue a
updateMoveOptionQueueWithOtherMoveOptions methods of the Player class.

6 This question refers to the MoveOptionQueue class.

The game uses a queue data structure rather than a stack.

(a) Explain why a queue is a more suitable data structure than a stac

(b) Currently this class uses a list to store the queue data structure; e
modified to use an array to implement a circular queue with five e

You should not write any actual code for this question but refer to
may be required and create any algorithms using structured/desc
Alternatively, you may produce an annotated diagram.

7 This question refers to the method getIndexOfSquare in the Dastan c

Explain how the private method getIndexOfSquare works.

8 The board is currently represented as a one-dimensional array, but th
alternative representations.

(a) Explain how the board could be represented as a two-dimensiona

(b) State the reason why an array is more appropriate to store the b

9 It would be possible to create a save game file for Dastan. At the start
metadata.

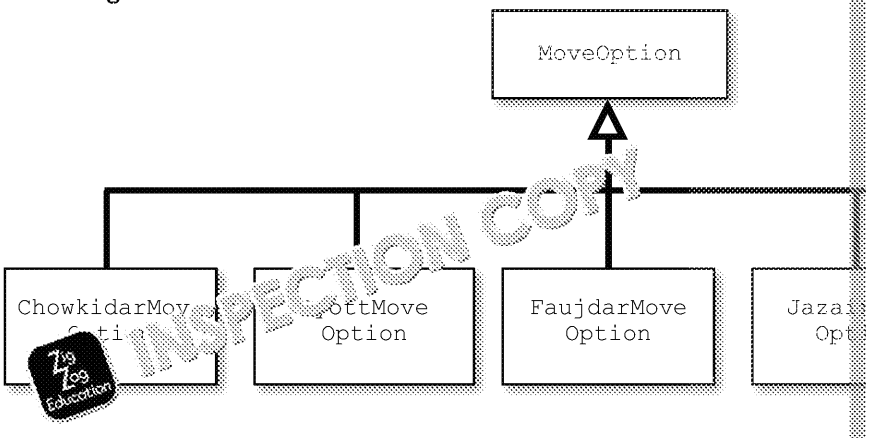
Explain the purpose of metadata and give one example of metadata t
Dastan.

10 This question refers to the createMoveOptions, createMoveOption,
createChowkidarMoveOption, createRyottMoveOption, createFaujda
createJazairMoveOption and createCuirassierMoveOption methods i
the MoveOption class.

(a) Currently the MoveOption class stores the details for whichever m
generated/populated by one of the createChowkidarMoveOptio
createRyottMoveOption, createFaujdarMoveOption, createJa
cr Cu MoveOption methods in the Dastan class.

Ex why this is NOT polymorphism.

10 **(b)** An alternative would have been to create and use an inheritance s
following:



Explain how this inheritance structure could have been used effec

11 This question refers to the Kotla class.

(a) The constructor includes a call using super() explain the purpose

(b) The method getPointsForOccupancy has a different implements
the same name in the parent class. State the name for this OOP

(c) Explain what the OOP technique *overloading* is used for.

12 The MoveOptionQueue class implements a normal queue, which is a
data structure.

Explain the different between a n queue and a priority queue.

13 This question refers to the constructor of the Piece class and the setP
Squa s.

Both methods take a parameter *p* which is unclear. Explain why varia
meaningful names.

14 This question is about access levels for attributes and methods and re

(a) The Piece class has four protected attributes; what does the word
context?

(b) The Piece class has four public methods; what does the word 'pu

(c) There is one final level of access for attributes and methods which
mean?

(d) Why is it important to have access modifiers such as private, prot
methods and attributes in OOP?

15 This question refers to the checkSquareInBounds method of the Da

(a) Th hod uses integer division; explain the difference between
floating point division.

(b) This method returns a Boolean value. Describe the meaning of E

END OF QUESTIONS

Dastan

Programming Tasks

These questions require you to load the **Skeleton Program** and to make

Note that alternative or additional code changes that you deemed appropriate ensuring that it is clear where in the Skeleton Program those chang

Task 1

This question refers to the Dastan class.

Introduce new functionality at the point at which both players are instantiat custom names set by the users. Ensure that players cannot both have the replace the two lines in the constructor that currently create the players wi method, createCustomPlayers.

What you need to do

Task 1

Create a new method createCustomPlayers in the Dastan class. Allow th for each player. Include checking in the code to ensure that two players ca name.

Allow the first player to enter any name they like, then repeatedly ask the name until they are both different.

Task 2

Test that the changes you have made work:

- run the skeleton program.
- enter 'Tom' as the first player name and then enter 'Tom' as the se prompted, enter 'Tom' again and then at the next prompt, enter 'Vi
- show the game using one of the custom names to address the play

Evidence that you need to provide

- PROGRAM SOURCE CODE showing creation of a new createCust Dastan class
- SCREEN CAPTURE(S) showing the required test

Task 2

This question refers to the createMoveOptionOffer, createMoveOption a methods and creation of a new method createFarisMoveOption in the Da

Develop a new move option called a 'Faris' (Knight). The Faris move optio chess – either two squares forward/backwards an[illegible] square left/right or left/right and one square forward/backwards. You should demonstrate the parameter.

What you need to

Task 1

i) Add new functionality into the createMoveOptionOffer & createMoveOption methods to perform a Faris move.

ii) Modify the createMoveOptions method to add the Faris after the Ryott for both players.

iii) Create a new method createFarisMoveOption which adds moves using the pattern shown, to the newMoveOption object.

Task 2

Test that the changes you have made work:

- run the program.
- play turns, showing both players making legal Faris moves.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the createM reateMoveOption and createMoveOptions methods
- PROGRAM SOURCE CODE showing a new method createFarisMo
- SCREEN CAPTURE(S) showing the required test

Task 3

Develop a new move option called a 'Sarukh' (Rocket). The Sarukh move c
rocket shape. You should demonstrate the use of the direction parameter

What you need to do

Task 1

i) Add new functionality into the `createMoveOptionOffer`, `cre veOption` and `createMoveOptions` methods to perform a Sarukh move.

ii) Modify the `createMoveOptions` method to add the Sarukh after the Ryott for both players.

iii) Create a new method `createSarukhMoveOption` which adds moves using the pattern below, to the new `MoveOption` object. The pattern is shown from the viewpoint of player two. For player one, the layout is inverted.

Task 2

Test that the changes you have made work

- run the skeleton program
- play turns showing both players making legal Sarukh moves.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the create
  `createMoveOption` and `createMoveOptions` methods
- PROGRAM SOURCE CODE showing a new method `createSarukh`
- SCREEN CAPTURE(S) showing the required test

Task 4

This question refers to the playGame method in the Dastan class and cre
awardWafr in the Dastan class, getWafrAwarded and setWafrAwarded
attribute wafrAwarded in the Player class.

Create a 'Wafr' (abundance) award which can be [illegible] to either player o
a 25% chance of being awarded to a pla[illegible] o[illegible] turn. On receipt of the
option of ANY move from their [illegible]ue rather than just being able to s
The 'Wafr' award rem[illegible]ove cost for the move the player selects f

Note: If th[illegible]r makes an invalid move then they 'lose' their Wafr and g
player should not be able to 'take the offer' if a Wafr is awarded.

What you need to do

Task 1

i) Create a new method in the Dastan class called awardWafr. This
chance of returning true.

ii) Add a new private attribute to the Player class called wafrAwarde
mutator (getter/setter) methods for this attribute.

Task 2

Update the playGame method in the Dastan [illegible] ca[illegible] the new awardW
hasn't already been awarded a Wafr, [illegible] u[illegible] message saying 'You hav
can select any move from you[illegible] for free this turn.' Adjust the input ra
in the queue to be [illegible]. Ensure that there is no score adjustment for p
value of th[illegible] to ensure that they cannot receive another Wafr.

Task 3

Test that the changes you have made work:

- run the skeleton program.
- play the game to show a player being awarded a Wafr.
- play a move option from position 4 or 5 in the move option queue.
- show the updated board and correctly modified score.

Evidence that you need to provide:

- PROGRAM SOURCE CODE sh[illegible] c[illegible]ges made to the playGa
class, creation of a new [illegible]ardWafr in the Dastan class
- PROG[illegible] CODE showing changes made to the Player c
meth[illegible]tWafrAwarded, setWafrAwarded together with one new
- SCREEN CAPTURE(S) showing the required test

Task 5

This question refers to the playGame method in the Dastan class and the getJustQueue in the Player class.

Introduce a new option 8 to the main game playing menu. On selecting this their opponent's queue to spy what move options an opponent might be an opponent's queue, however, carries a cost of 5 points from the player's opponent's queue, the player's turn would continue as normal.

What you need to do

Task 1

Create a new method in the Player class called getJustQueue which uses method from the Queue to return a string version of just the player's queue

Task 2

Modify the PlayGame method to introduce new functionality which adds a game playing menu. If the user selects this option, display the move option player.

(**Hint:** You can check the current player using the sameAs method and the Subtract 5 from the current player score and display the game state again their turn as normal.

Task 3

Test that the changes you have made work:

- run the skeleton program.
- show player one selecting option 8 from the main game menu.
- show the opponent queue being displayed clearly on the screen and reducing by 5 points.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGame class
- PROGRAM SOURCE CODE showing new method getJustQueue in
- SCREEN CAPTURE(S) showing the required test

Task 6

This question refers to the playGame method together with the modificatio
useMoveOptionOffer methods and creation of a new method getValidIn

Currently the game has a number of areas where it does not handle errons
error handling into the playGame, getSquareRe[illegible] and userMoveO
prevent unhandled exceptions from occurring if the user inputs data in an i
user to re-enter their input, until it is valid.

Note: There is no need to check that the square contains a player piece o
player sho[illegible] have a wasted turn if the move is invalid, the purpose of
from crashing.

What you need to do

Task 1

Create a new private method called getValidInt in the Dastan class which
valid integer. If the input is invalid, allow the user to keep trying again with

Task 2

Modify the getSquareReference method to use the new getValidInt meth
input. Add an error message if the user enters an invalid square.

Task 3

Modify the useMoveOptionOffer method to use the new getValidInt meth
input and test to confirm that the user input is within the correct range.

Task 4

Test that the changes you have made work:

- run the skeleton program.
- from the main game playing menu, enter 'help' as your choice and
message. Then choose move 1.
- For player one, enter a square of 19 and show the error message.
followed by 32 to make the move.
- For player two, select option 9 to take the offer move and choose p
message.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the getSqua
- PROGRAM SOURCE CODE showing changes made to the playGa
- PROGRAM SOURCE CODE showing changes made to the useMov
- PROGRAM SOURCE CODE showing the creation of new getValidI
- SCREEN CAPTURE(S) showing the required test

Task 7

This question refers to the playGame and useMoveOptionOffer methods
creation of a new attribute choiceOptionsLeft along with accessor and m
decreaseChoiceOptionsLeft and getChoiceOptionsLeft in the Player c

Currently a player can repeatedly select option 9 from the main game play
with new move options. Introduce a limit so that a player can only 'accept t
menu three times in a game. Every time a player accepts the offer, advise t
they have left and remove the menu for that player once they have used it t

What you [illegible] to do

Task 1

Modify the Player class to introduce a new private attribute called choiceO

i) Initialise choiceOptionsLeft to 3.

ii) Create a new accessor method called getChoiceOptionsLeft whic
attribute choiceOptionsLeft.

iii) Create a new mutator method called decreaseChoiceOptionsLeft
choiceOptionsLeft attribute and prints out how many options you

Task 2

Modify the playGame method to test the number of options the player has
three during the game.

i) Modify the playGame method so that if the player has used up all t
will no longer be available to the player.

ii) Modify the useMoveOptionOffer method so that when a move opt
the number of options available to them decreases by one.

Task 3

Test that the changes you have made work:

- run the skeleton program.
- select four sequential option moves from the move option list addin
the player one queue.
- show the removal of option 9 from the main game playing menu an
the player attempts to select option 9.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGam
- PROGRAM SOURCE CODE showing changes made to the useMove
Dastan class
- PROGRAM SOURCE CODE showing changes made to the Player c
- SCREEN CAPTURE(S) showing the required test

Task 8

This question refers to the playGame method in the Dastan class and crea
resetQueueBack in the moveOptionQueue class and resetQueueBackA

Introduce a new option that allows a player to undo their last move (after t
and before the next player makes their move), u y score gained o
returning the game to its previous state. Undoing a move costs a player 5
a player can then make an alte ove.

What you to

Task 1

Add the functionality to reset the queue if a move is undone.

i) Create a new method in the moveOptionQueue class called rese
should move the last element of the queue back to the original pos
method should take one parameter, Position, which is the place to
queue will be restored.

ii) Create a new method in the Player class called resetQueueBack
should call the newly created resetQueueBack method on the que
class. The method should take one parameter, position, which is t
from the menu.

Task 2

Modify the playGame introduce the new functionality.

i) If a is legal, store the player score prior to the move.
ii) Afte playing the board as a result of the move, give the player t
iii) If they choose to undo then: return the player score to the stored p
points and restore the board and the player's queue back to their p

Task 3

Test that the changes you have made work:

- run the skeleton program.
- show player one attempt a 'Chowkidar' move and then undo the m
- show the game board after the undo and the score set correctly an
a new move.

Evidence that you need to p

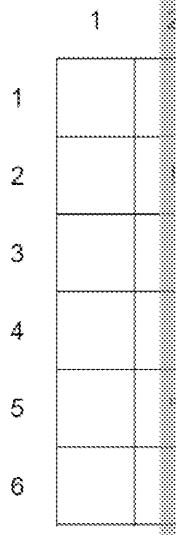
- PROGRAM SOU CODE showing changes made to the playGam
- PROG SOURCE CODE showing the creation of new methods re
moveO onQueue class
- PROGRAM SOURCE CODE showing the creation of the new method
in the Player class
- SCREEN CAPTURE(S) showing the required test

Task 9

This question refers to the PlayGame method together with the modificati and createMoveOption methods and creation of two new methods, creat calculateSahmMove, in the Dastan class – plus a new method, choicels

It also refers to a new attribute sahmUsed in the Player class along with c getSahmStatus and setSahmUsed, which operate as the accessor and m for the newly created sahmUsed attribute.

Implement a new 'Sahm' move option (arrow). The Sahm can only be fired and is fired instead of a piece moving. A Sahm can be fired by any piece. T line forwards from the player destroying any opponent piece(s) in its way except a Kotla, which is strong enough to withstand an attack and protect any piece inside it. The Sahm is only made available to a player through the moveOptionOffer method (they can choose to add it to their moves by using option 9 from the main menu at the start of the turn if a Sahm is offered to them). A Sahm will not show up normally in the MoveOptionQueue.



The image on the right shows the player 2 piece in square 54 firing the Sahm. The Sahm will fire forwards, destroying the player 1 pieces in squares 34 and 24.

What you need to do

Task 1

Add new functionality into the createMoveOptionOffer and createMoveO new private createSahmMoveOption method to perform a Sahm move.

i) Modify the createMoveOptionOffer method to offer the new 'Sahm

ii) Create the new private createSahmMoveOption method to allow t piece fires the Sahm and add only one possible new move Move(0, **Note:** The move should not actually move the piece anywhere, i.e.

iii) Modify the createMoveOption method to handle Sahm.

Task 2

Modify the Player class to allow the user to use the Sahm only once.

i) Add a new sahmUsed attribute in the Player class which is initialis

ii) Create two new methods, getSahmStatus and setSahmUsed, wh and to (getter/setter) methods for the newly created sahmUs

iii) Create a method choicelsSahm method which takes a parameter chosen is a Sahm move, whereupon it returns True.

(TASK CONTINUES ON THE NEXT PAGE)

Task 3

Modify the playGame method to test to see if the player has selected a Sa
moveOptionOffer menu and if it has already been used. If the selected fir
should destroy any opponent pieces in a straight line from the firing piece,
player should collect any points from multiple pieces destroyed by the Sah

i) Modify playGame to call the new meth d ch ceisSahm and only

ii) Create a new private meth n Dastan class called calculateS
calculate the poin Sahm move and destroy the pieces that a
Kotl

iii) Mod ayGame to so that is calls the new method calculateSah
the Sahm move and destroys the relevant pieces. It should also ca
for the current player.

Task 4

Test that the changes you have made work:

- run the skeleton program.
- select a Chowkidar move for player one (option 2) and choose squ
square 33 as the 'to' to diagonally move one piece in front of anoth
Kotla column.
- select 9 from menu for player two to accep fer. Choose 1 to
choose option 1 to select the Sah m ve. hoose the piece on sq
show the updated board player one pieces removed from
by player two, Mirza which is safely inside the Kotla.
- sh adjustment of player two's score.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGa
- PROGRAM SOURCE CODE showing changes made to the createM
createMoveOption methods
- PROGRAM SOURCE CODE showing the creation of new createSah
choiceIsSahm and calculateSahmMove methods
- PROGRAM SOURCE CODE showing changes made to the player c
- SCREEN CAPTURE(S) showing the required t

Task 10

This question refers to the playGame method in the Dastan class.

Introduce a new option 7 to the main game playing menu. On selecting this one of their own pieces to destroy and replace with a second Kotla. A new the square in which the piece was sacrificed. A p[illegible] can only replace one Replacing a piece with a Kotla should us[illegible] p[illegible] player turn and they should turn.

What you [illegible] to do

Task 1

Modify the playGame method in the Dastan class to introduce a new optio playing menu. Allow the player to select a piece which they would like to re validation to ensure that the user can only select one of their pieces and it confirmation, replace the piece with a second Kotla assigned to the correct

Task 2

Test that the changes you have made work:

- run the skeleton program.
- select option 7 for player one from the main [illegible] menu.
- show the user selecting 52 as an invalid square for the new Kotla.
- show the Kotla being placed for safety in square 22, a valid square,

Evidence you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGa
- SCREEN CAPTURE(S) showing the required test

Task 11

This question refers to the playGame method together with a new method in the Dastan class, additional new methods reverseQueue, swapFirstA moveItemToFront in the moveOptionQueue class together with new me swapQueue, getMoveOptionQueue, reversePlayer[illegible]eue, swapFirstA moveItemToFront in the Player class.

Introduce a new option 6 to the [illegible]e playing menu. On selecting this sub options for making [illegible] to their move queue using the following m

Options
a) Reverse the current player queue
b) Swap the current player queue with the opponent queue
c) Swap the first and last elements in the current player queue
d) Move one of the move options to the front of the current player queue
e) Nothing (make normal move)

Note: Options (a) – (d) cost 3 points, but the player can choose (e) for free
Note: This does not count as the player's turn and the player should still b

What you need to do

Task 1

Modify the [illegible] to introduce the new menu option.

i) Mo[illegible]e playGame method to add option 6 to the move options

ii) Create a new private method in the Dastan class called modifyQu player the above menu. Include validation to ensure that the user c choices from the menu.

iii) Adjust the score by 3 if options (a) – (d) are chosen but not if optio

Task 2

Modify the moveOptionQueue class to add the required methods.

i) Create new method reverseQueue to allow t[illegible]urrent player's qu

ii) Create new method swapFirstA[illegible] swap the first and last e queue.

iii) Create new [illegible]moveItemToFront to move the item from the the[illegible] the current player. There is no need to validate the i the [illegible]on from.

(TASK CONTINUES ON THE NEXT PAGE)

Task 3

Modify the Player class to create the required methods.

i) Create new methods reverseQueue, swapFirstAndLast, moveIte
to expose the new moveQueueOptions choices/methods to the D

ii) Create new method replaceQueue to allow t urrent player's qu
queue passed in as a parameter. Not th should return the cur

Task 4

Test that t nges you have made work:

- run the skeleton program.
- show player one selecting option 6 from the main game menu.
- show the player selecting each one of the queue options in turn an
screen as a result of the change.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGa
- PROGRAM SOURCE CODE for the new modifyQueueOptions me
- PROGRAM SOURCE CODE showing changes made to the moveO
- PROGRAM SOURCE CODE showing changes made to the Player c
- SCREEN CAPTURE(S) showing the required test

Task 12

This question refers to the creation of a new protected attribute noOfPiec playGame method and creation of two new methods checkReincarnatio the Dastan class.

Introduce a new feature whereby if a player manages t get one of their pi row, they are given a new piece to place on a upied space on the player cannot reincarnate pieces that a so they should not be a they started with.

What you n t

Task 1

Create a new private method in the Dastan class called countNormalPie of pieces that the current player has excluding the Mirza.

Task 2

i) Modify the constructor in the Dastan class to store the number of p protected attribute called noOfPieces.

ii) Modify the playGame method in the Dastan class to call a new pri checkReincarnation after the move is legal.

Task 3

Create a new private method checkReincarnation in the Dastan class. T which is the finishSquareReference for the current player's move. If the opponent's back row (e.g. row 6 for player one) and t layer has fewer p then allow them to reincarnate a piece on thei ck w in an empty squa the square is empty and allow them t it is not.

Task 4

Test that the have made work:

- add llowing four lines of code to the START of the private me **Dastan** class (*be certain to remove this afterwards!*):

```
noOfPieces = 2;
board.get(getIndexOfSquare(51)).setPiece(new Piece("piece",pla
board.get(getIndexOfSquare(21)).setPiece(new Piece("piece",pla
board.get(getIndexOfSquare(54)).setPiece(new Piece("piece",pla
```

- run the skeleton program.
- select a Ryott move for player one, enter a start square of 51 and a
- show player one attempting to reincarnate a piece in column 3 and saying that the square must be empty.
- show player one attempting to reincarnate a piece in column 4 and appropriately.
- select a Ryott move for player two, ent t square of 21 and a
- show player two not receiving tion message.
- Change back the method by removing the additiona

Evidenc y need to provide:

- PRO l SOURCE CODE showing the new countNormalPieces
- PROGRAM SOURCE CODE showing the new checkReincarnatio
- PROGRAM SOURCE CODE showing the other code changes to the
- SCREEN CAPTURE(S) showing the required test

Task 13

This question refers to the playGame method together with modification o the Dastan class. Additionally it involves the creation of a new Taziz class

Create a new type of game square, the Taziz (advantage castle, similar to the middle of the playing board (or slightly closer to player two if there are Either player can occupy the Taziz with any of their pieces. If a player can by both players (entering the taziz is considered a player's first turn), then have zero cost. This gives a player a zero cost move, but risks sitting in the to get it. If a player sits there for longer then they continue to get zero co

What you need to do

Task 1

Create a new Class taziz which should inherit from the Square class.

i) Add a new protected attribute campedTurns and initialise it to 0.

ii) Override the setPiece and removePiece methods from the Squar adjust the Taziz symbol to an upper case 'A' if player one owns the player two owns the Taziz (you may assume that the player with a the top – player one). When a player piece leaves the Taziz, owne set to null and the symbol set to a lower case 'x'.

iii) Create a new method getCampedTwoTurns. Each time the Taziz campedTurns should be reset back to zero. The getCampedTwo the number of turns using the campedTurns attribute and return tr of the player occupying the Taziz).

iv) Cre new method checkCamp that checks if the same player i increments the campedTurns attribute if they are.

Task 2

Modify the createBoard method in the Dastan class to place a Taziz on th of the board with a lower case 'x' symbol when the board is first created.

NOTE: *The Taziz should be correctly placed on the board even if the size should take account of the number of columns and rows.*

In the case where there are an even number of rows, the Taziz should be also if there are an even number of columns then it should be slightly close starting board this will place it on square 43. But it should work for any siz

The initial Taziz does not belong to either player.

Task 3

Modify the playGame method so that if a move is legal the game should t camped in for two full turns and, if so, give the selected move to the playe

Task 4

Test that the changes you have made work:

- run the skeleton program.
- use a Cuirassier move option 3 to move a player one piece into the
- play the game until both players have had two turns – leaving the p without attacking it using player two.
- after both players have had two turns, show a move option by playe (i.e. they should get 2 points for camping their Kotla, plus an addit piece that no longer charged anything for the move from the queue).

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGa
- PROGRAM SOURCE CODE showing changes made to the createB
- PROGRAM SOURCE CODE showing the new getCampedTwoTur Square class
- PROGRAM SOURCE CODE showing the new Taziz class
- SCREEN CAPTURE(S) showing the required test

Task 14

This question refers to the playGame method together with creation of a n
weatherEventOccurs method in the Dastan class. Additionally it involves
WeatherEvent with the methods countDownComplete, setWeatherLoc
getWeatherLocation.

The Weather Event has a 50% chance of appearing on any turn and can a
space on the board. On appearance on the board, both players are given a
Weather Event will destroy EVERY piece on the same column as the Wea
After two turns by each player, the Weather Event strikes and any piece fr
column is removed, including the Kotla.

***NOTE:** A Weather Event can only occur if a Weather Event is not already*

What you need to do

Task 1

Create a new class WeatherEvent which should include new methods co
setWeatherLocation and getWeatherLocation. On instantiation, the We
countdown to count the number of game turns before the event occurs. co
test to see if the countdown has expired. The setWeatherLocation and g
should set and get the location of the Weather Event on the board. Suitab
out each turn to indicate how long until the Weather Event will occur.

Task 2

Create a new method called weatherEventOccurs in the Dastan class w
creating a Weather Event and placing it on a random empty square on the boar
occurred, let the player know.

Task 3

Modify the playGame method in the Dastan class to test to see if a Weath
so if the Weather Event countdown has expired. If it has, use the Weather
piece (from either player) from the same column as the Weather Event, in
are awarded for this event.

Task 4

Test that the changes you have made work:

- run the skeleton program.
- when a weather event occurs, move player pieces to be on the sam
  event over the next two turns.
- show the board during the column of the Weather Event and af
  showing the pieces from both players removed from the board.

Evidence you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGa
- PROGRAM SOURCE CODE showing the new weatherEventOccu
- PROGRAM SOURCE CODE showing the new WeatherEvent class
- SCREEN CAPTURE(S) showing the required test

Task 15

This question refers to the playGame method together with modification o
createPieces methods and creation of three new private methods, check
and checkManhattanDistance in the Dastan class. Additionally it involve
method containsBarrier in the Square class and the [illegible]ation of a new B
Square.

Create a new game piece calle[illegible]. On creation of the board each p
would like to place th[illegible] on the board. The Barrier is 3 squares wid
the board [illegible] p[illegible]ion occupied by a normal piece or an opponent's Ba
be moved, [illegible]pied or jumped over by either player.

Some moves, however, do not move in a straight line, for example the Jaz
the direct move would be through the Barrier which is not allowed. A move
the Barrier, however, is possible which is, therefore allowed. Use the Man
there is a move route possible around the edge of the Barrier.

Manhattan distance is a heuristic function for calculating distance between
a grid. In the case of Dastan it is calculated by counting the sum of the nu
and then vertically (or vice versa) between a player starting location and th
in **Fig 2** below.

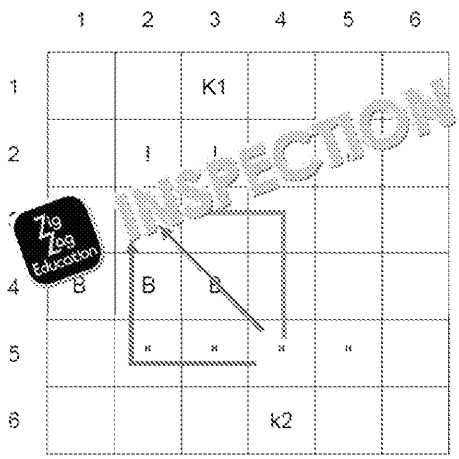
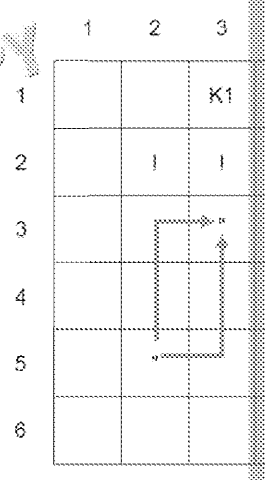


Fig 1



Fig

What you need to do

Task 1

i) Create a new class Barrier which sh[illegible]nt from the Square c
assigned an owner and gi[illegible] symbol of a capital 'B' if it belong
lowercase 'b' if it [illegible] player two.

ii) Cr[illegible] public method containsBarrier in the Square class
has [illegible] placed in that square.

(TASK CONTINUES ON THE NEXT PAGE)

Task 2

i) Modify the checkSquareIsValid method to check if the square bei that a piece cannot occupy it or attempt to move it.

ii) Create a new method checkBarrierIsValid in the Dastan class wh a Barrier being placed by a player fits within the bounds of the boa squares.

iii) Create a new method calle[illegible] a Barrier in the Dastan class whic Barrier onto the board. The Barrier will always be horizontal and th cent[illegible] being asked where to place the Barrier.

Task 3

i) Create a new method called checkManhattanDistance in the Das paths from a starting square reference to a finishing square referen starting row then down the finishing column and also down the star finishing row. This is used to check if a selected move can traverse over the top of it.

ii) Modify playGame to call checkManhattanDistance which should checkPlayerMove used to set the value of the variable moveLega

Note: This should be used for all moves even if they are too short to poten may be able to go round. For a single or double move [illegible]her horizontally o should be considered; only for diagonal move [illegible] you consider horizo vertical and then horizontal.

Task 4

Test that t[illegible]ngs you have made work:

- run the skeleton program.
- enter a position of 34 for the player one Barrier.
- enter a position of 42 for the player two Barrier.
- for player one: choose 9, then 1, then 1, then 24, then 46.
- for player two: choose 3, then 53, then 31.
- for player one: choose 2, then 25, then 45.
- for player two: choose 1, then 52, then 42, then 51.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the playGa
- PROGRAM SOURCE CODE [illegible] changes made to the checkS createPieces meth[illegible] the Dastan class
- PRO[illegible]M [illegible]CE CODE for the new private checkBarrierIsVal chec[illegible]hattanDistance methods in the Dastan class
- PROGRAM SOURCE CODE showing changes made to the Square
- PROGRAM SOURCE CODE showing the new Barrier class
- SCREEN CAPTURE(S) showing the required test

Dastan

Programming Tasks (Extensi

Extension 1

Introduce a health scoring system for pieces. Each piece (except the Kotla) health points. Each time a piece is landed on, it incurs damage, reducing it each time a piece's health is reduced. When a piece reaches 0 health poin board. Only one piece can attack another at one time. When a piece is be player symbol should be shown on the left of the piece and the target piec the right of the square.

Damage is determined using this formula:

> *Position of move choice in the queue + Manhattan distance from the piece (number of rows different + number of columns different).*

Manhattan distance is a heuristic function for calculating distance between two locations, for example in a grid. In the case of Dastan it is calculated by counting the sum of the number of squares horizontally and then vertically (or vice versa) between a player starting location and the destination location as shown in **Fig 1**.

An attack using move from position 1 in the move queue reduces health b An attack from position 3 in the move queue reduces health by 3 points. Th how far away the opponent is from the attacker. This is the sum of the row An attack from further away, therefore, incurs a greater level of damage.

Extension 2

Create a new game square called 'Qunbila Ghayr Muwajaha' (Unguided B has a 33% chance of appearing in any turn and is given to the current play a 10% chance of detonating. The player can either move away from the bo board. When the bomb is thrown the player can choose any board location location or a Kotla.

The 'Throw bomb' option should be available through the move offer menu

If the bomb is thrown to an opponent's square, the opponent takes owners back or moves away from it. Each turn carries a 10% chance of the bomb d thrown to a Kotla square, the bomb loses ownership from either player and The bomb remains at this location until a player moves to the square conta up again and be able to throw it. Each turn carries a 10% chance of the bo

If a player piece is captured while holding the bomb, the ownership of the b of the square.

Extension 3

Introduce the concept of a 'Makinat Taftish' (Inspection Machine). This is a computer-controlled piece which does not belong to either player. After each player turn, the Inspection Machine should measure the distance from itself to all the other pieces on the board using Manhattan distance. The machine should then move itself towards the closest piece on the board, regardless of whether two pieces are the same distance away, the machine should select one at random. The machine can move in any direction, but only one square at a time.

The machine should repeat this behaviour once a turn until it reaches a player piece and captures it. Neither player gains any points for a piece being captured.

Manhattan distance is a heuristic function for calculating distance between two locations, for example in a grid. In the case of Dastan it is calculated by counting the sum of the number of squares horizontally and then vertically (or vice versa) between a player starting location and the finishing location as shown in **Fig 1**.

The machine does not place any weighting on a 'target' to move towards and can capture a player piece or a Kotla.

A player loses the game if their Kotla is captured by the Inspection Machine.

Extension 4

Introduce the concept of a 'Multi-Move'. This allows a player to combine two moves at a significant points cost.

Introduce a new option 9 to the main game playing menu called 'Multi-Move'. The player can select two move options to execute sequentially. The player will then move option 2, choosing a 'move to…' square reference for each option. The reference for move option 2 must be a legal move based on the 'move to…' for option 1. Both moves must be legal. The program should use error handling when entering illegal references and allow them to re-enter.

Selected moves in a multi-move can be from any position in `MoveOptionQ` from the position of move in `MoveOptionQueue`.

On entering a legal multi-move, the game should move the selected player. The move should cost the player 2 points.

If the player lands on an opponent piece through either move 1 or move 2 the opponent piece should be captured as normal.

Extension 5

Introduce the concept of a 'Khalad' – a mole. Introduce a new submenu op
move option from the main game playing menu. The submenu should offe
activate a 'mole' mode for the selected move option.

On selecting 'mole' mode, the move operates as norm however, the pla
the board. A piece which is operating in 'mole' mode should be shown as a
for player 2, which is displayed on the left-hand side of each square instea
on the right-hand side of a square. This means that two pieces can occupy
'mole' mode and one on the board 'surface'.

A piece in 'mole' mode can move around underneath the board using norm
be captured by an opponent piece on the surface of the board. Once the p
be in 'mole' mode, the submenu should change to now give the player the
piece after moving it. If a piece in 'mole' mode resurfaces in a square refer
piece, the current player captures that opponent piece. Once a player resu
piece, the 'mole' mode submenu should no longer be offered to the player

A mole cannot move onto the Kotla square as the foundations are too dee

If an opponent also has a piece operating in 'mole' mode, one mole can ca
pieces on the board surface.

Extension 6

Introduce an option to 'preview a move' prior to making it. Add a new optio
menu. On selecting this option a player can select any move from position
valid player piece on the board. The player is then shown a 'preview' copy of
shows an X on all the squares which the selected move option can move t
current player piece.

The player should then be given the option to enter in a valid 'move to…' s
selected move option or go back to the main menu to choose a different m
valid 'move to…' square reference is selected, the game should make the

The player can 'preview a move' as many times as they like during the gam

The 'preview a move' option should not attempt to show the player 'move t
outside the bounds of the board.

Extension 7

Introduce a new option at the start of the game to allow the players to plac different formations prior to the game starting. Players can choose from an

All the positions are shown from the perspective of player 2.

Once the players have selected their chosen start positions, the game s

	1	2	3	4	5	6
1						
2						
3						
4						
5			"	"		
6		"		k2	"	

'Khandaq' Option (Trench)

	1	2
1		
2		
3		
4	"	
5		
6		

'Iltifa

	1	2	3	4	5	6
1						
2						
3						
4						"
5			"	"	"	
6				k2		

'Darba Rukniya' Option (Corner Kick)

	1	2
1		
2		
3		
4		
5		
6		

'Khanj

Extension 8

Introduce the concept of an 'Al Amlaq' (Giant), which is formed when a pla of their **own** player pieces. A Giant is shown as 'G' for player 1 and 'g' fo

Once a Giant has been created by combining a player Mirza with a norma and remains as a Giant for the rest of the game.

A Giant ca e around the board using the same move options as a no needs to land to within one square (in any direction) of any opposition piec opponent Kotla and Mirza.

A Giant can be captured by any opponent piece as normal and is worth 20

Extension 9

Introduce the concept of an 'Adra' (Chainmail). Add a new option 'C' to the selecting this option, the player should be asked which piece they would li player can only add chainmail to two pieces during the game. The chainm clothes and therefore a piece's symbol doesn't change when it has the cha forward-facing barrier which means that the piece cannot be attacked from opponent piece can be one square in front of current player piece, it can chainmail – it must approach the enemy piece from either side or behind

A player can add chainmail to any two pieces in the game including the

Extension 10

Introduce the load and save features to the game. Add new options 'L' and the player options to load a previously saved game or save the current gam

The load and save submenus should give the user the opportunity to ente program should have appropriate error handling to prevent it from attempti data or saving to an invalid location. The program should store appropriate separated values to store all of the program data required to rebuild a gam handling should be included when a game is being rebuilt to ensure that th all valid within the bounds of the board.

Extension 11

Introduce a new feature that allows the size of the playing board and pieces game, giving the player the option to choose the size of the board. The dime even; howe error handling should be included to prevent the board fro larger than 10 × 10. (Appropriate formatting needs introducing on a 10 × 1 line up correctly.)

For boards of 6 to 8 columns wide, ensure that both player Kotlas are plac and bottom rows of the board. A 7 column wide board should have 5 piece wide board should have 6 pieces per player.

For boards 9 and 10 columns wide, introduce a second Kotla for each player on the appropriate top and bottom rows of the board. The Kotlas should be evenly distributed across the board. The player should still only have 1 Mirza, whic should be placed in either of the Kotlas random. A 9 or 10 column wide board should have 6 pieces per player, 3 of which should be in front of one Kotla and 3 in front of the other, as per the example shown.

	1	2	3
1			K
2		!	!
3			
4			
5			
6			
7			
8		"	"
9			K2

Extension 12

Adjust the playing board to allow the sides to wrap around. On making a move, a player can move off the left- or right-hand side of the board and land on the correctly associated square on the opposite side of the board as if the board was wrapped around.

For example, a player can select a Cuirassier move for the piece in square 2,5 and move to square 3,1 which is one square forward followed by two squares to the left looking at the board from the point of view of the

Extension 13

Introduce the concept of an 'Muraqib' (Meerkat Lookout piece). At the start the opportunity to place their Muraqib on any empty square on the board. represented by an 'M' symbol and player 2 Muraqib is represented by an

The Muraqib is on constant lookout for the player it belongs to. For examp legal move and the board and player 1 queue are updated, the player 2 M player 1 piece left on the board and test each of moves 1, 2, 3 from the pla see if it could threaten to capture any player 2 piece. If such a threat is pos player 2 in case they have missed that possible threat.

A Muraqib cannot be captured. If either player lands on the square contain disappears down into its burrow underneath the board. While it is in its bur it belongs to of any threatening moves. If the player piece occupying t away from that square, the Muraqib should return to its lookout duties.

Extension 14

Introduce a new 'Aqrab' (Scorpion) option which can be added to any play Aqrab can only applied to one piece per player. Once applied, the piece sy for a player 1 piece or '£' for a player 2 piece. A piece chosen to be a Aqra the board; however, when it is one square away from an opponent piece (piece becomes paralysed and cannot move. This makes it vulnerable to b the Aqrab itself.

The Aqrab, however, can still be captured by any piece which can move fr squares away (in any direction). If the Aqrab moves away from a piece wit no longer paralysed and can move away as normal.

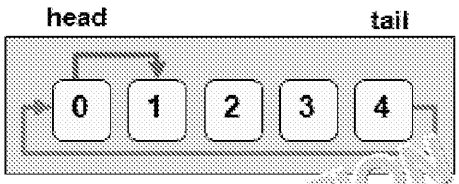
Extension 15

Introduce the concept of a 'Quantum Mirza'. Add a new option 'Q' to the m options to allow a player to swap their Mirza location with any of their other player selecting the Quantum menu, the player should be asked for the board loc swap. The piece must belong to the current player. The program should u to ensure that a valid piece is selected. The program should then swap the Mirza locations. The player turn should then continue as normal.

Dastan

Exam-style Questions (Mark Sc

Q		Suggested Solution	Marks
1	(a)	$O(n^2)$	1 mark
	(b)	1 mark for each point • … relatively efficient for smaller input sizes • … however, as the input size grows, the completion time increases • The rate of change is constantly changing using a quadratic function… • … which means that it does not scale up well	3 marks
2	(a)	1 mark for each point • You may mistype/misspell one of them • … which could mean that the code develops a logic error	2 marks
	(b)	1 mark for each point • One possible solution would be to define them as constants (1 mark) • … which would mean that you would get an error with an undefined identifier before running the program	2 marks
3	(a)	Every square in the board is treated as a square [1 mark] but some of them may be Kotlas (which inherit from Square) [1 mark] so will [illegible] the overridden method on the Kotla due to polymorphism because although treated as a Square will behave as a Kotla [1 mark]	3 marks
4	(a)	Because the position of player one's Kotla is determined by the number of columns DIV 2 which gives 3 [1 mark] and the position of player two's Kotla is determined by the number of columns DIV 2 and then add 1 which gives 4 [1 mark]	2 marks
	(b)	Change the calculation for player one [1 mark] to (noOfColumns+1) DIV 2 [1 mark] which will round up for odd numbers [1 mark] but round down for even numbers [1 mark]	4 marks
5	(a)	As the direction attribute is part of the Player class [1 mark] both of these methods could go modify the newMoveOption when it is received in the addToMoveOptionQueue and updateMoveOptionQueueWithOffer methods [1 mark] to modify each non-zero value for rowChange and columnChange by multiplying it by the direction for the current player [1 mark]	3 marks
6	(a)	A queue is more appropriate because move options are added to the back of the queue but could not be added to the back of the stack as it is a LIFO structure [1 mark] and removed from the front of the queue because it is a FIFO structure [1 mark]	2 marks

Q		Suggested Solution	Marks
6	(b)	A circular queue would need a head variable [1 mark] and a tail variable [1 mark]…  … so that when an item is added to the queue, the rear pointer could be incremented or wrapped back around to 0 if it was greater than 4 [1 mark] and when an item is removed from the queue the head pointer could be increased or wrapped back around to 0 if it was greater than 4 [1 mark]	4 marks
7	(a)	Each square is referred to by a two-digit number, the method extracts the first digit using MOD, subtracts one [1 mark] and then multiplies it by number of columns [1 mark], then extracts the second digit of the square reference using DIV, subtracts one and adds the two together. [1 mark]	3 marks
8	(a)	One dimension could be the row [1 mark] and the second dimension could be the column [1 mark]	2 marks
	(b)	An array is static so the amount of memory used will not change and the board size is fixed so this is appropriate	1 mark
9	(a)	Metadata describes the data in a file [1 mark] Possible examples (any sensible answer will do): • Board size (resolution) • Number of pieces for each player	2 marks
10	(a)	This is not polymorphism because each of the five methods creates a MoveOption object [1 mark] which is the same class but contains different data [1 mark]. In order to be polymorphism you need to have child classes being treated as their parent which is not the case here [1 mark].	2 marks
	(b)	This is polymorphism because each of the five different MoveOption methods (e.g. ChowkidarMoveOption) for each move inherits from MoveOption and so can be treated as a MoveOption [1 mark] but will actually behave as themselves [1 mark] meaning that you could still have a collection of MoveOptions, all of which would actually be children of MoveOption [1 mark]	2 marks
11	(a)	1 mark for each point • super() is used to refer to the base class object • And call a method within it	2 marks
	(b)	Overriding	1 mark
	(c)	1 mark for each point • Allows **multiple** implementations • of a method with the same name • by selecting which version to run based on the number and type of parameters passed • within the same class definition	3 marks

Q		Suggested Solution	Marks
12	a)	1 mark for each point • A priority queue has different points at which items can join the queue according to priority • They join at the back of the section according to their priority, almost like sub-queues • If there are no items queued in the corresponding priority section then they join the queue at the front of the next lower priority or at the back of the next higher • Items are still taken one at a time from the front of the entire queue but still join at the back of the appropriate sub-queue section	4 marks
13	(a)	1 mark for each point • The name describes the purpose of the variable • which makes the code easier to read/understand/follow	2 marks
14	(a)	1 mark for each point • It can be accessed by children/subclasses • and from within the class itself	2 marks
	(b)	It can be accessed from anywhere	1 mark
	(c)	It can only be accessed from within the class	1 mark
	(d)	They allow correct encapsulation [1 mark] of the class… which means that you can only interact with the class through the intended interface [1 mark]… but it still allows for direct access within the class where required [1 mark]. Also avoids exposing attributes and methods that are either dangerous to expose or unnecessary outside the class [1 mark].	3 marks
15	(a)	1 mark for each point • Integer division returns a whole number (and a remainder) • Floating point division returns a decimal value with a decimal point	2 marks
	(b)	It has two values, true or false	1 mark

Dastan

Programming Tasks (Mark Sch

Task 1

Coding:

- Create a new method [illegible]omPlayers which allows the user to enter in [illegible] continue u[illegible] [illegible]re different. [1 mark]

Example Solu[illegible]

Modify constructor in Dastan:

```
    public Dastan(int r, int c, int noOfPieces ){
        //CODE CHANGE
        createCustomPlayers();
        //END CHANGE
        createMoveOptions();
```

New private method:

```
    //CODE ADDED
    private void createCustomPlayers(){
        String p1;
        String p2;
        Console.write("Enter name for Player One: ");
        p1 = Console.readLine();
        players.add(new Player(p1, 1));
        Console.write("Enter name for P[illegible]");
        p2 = Console.readLine();
        while (p1.equals(p2)) {
            Console.w[illegible] = [illegible]e matches Player One, enter a different [illegible]
            [illegible]readLine();
        }
        play[illegible]dd(new Player(p2, 1));
    }
    //END ADDITION
```

Testing:

Display an appropriate error message if the user enters in two matching names. Co[illegible] custom name. [1 mark]

```
Enter name for Player One: Tom
Enter name for Player Two: Tom
Name matches Player One, enter a different name for Player Two: Tom
Name matches Player One, enter a different name for Player Two: Vict[illegible]

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option[illegible]: jazair

Tom
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj[illegible]

Turn: Tom
Choose move option to use from queue (1 to 3) or 9 to take the offer[illegible]
```

Task 2

Coding:

- Adding a new `moveOptionOffer` to the `createMoveOptionOffer` method, a [illegible] `createMoveOption` method, and adding the move option to both players in the [illegible] parameter set correctly. [1 mark]
- Adding a 'faris' to the `createMoveOption` method [illegible] option which calls th[illegible] method. [1 mark]
- Create a new method `createFari[illegible]` [illegible] on which correctly uses the `Dire[illegible]` the possible positions for the [illegible] move. [1 mark]

Example Sol[illegible]

Changes to c[illegible]`MoveOptionOffer`:

```
    private void createMoveOptionOffer() {
        moveOptionOffer.add("faris"); //LINE ADDED
        moveOptionOffer.add("jazair");
```

Changes to `createMoveOption`:

```
    private MoveOption createMoveOption(String name, int direction) {
        switch (name) {
            //CODE ADDED
            case "faris":
                return createFarisMoveOption(direction);
            //END ADDITION
            case "chowkidar":
```

Code for `createFarisMoveOption`:

```
    //CODE ADDED
    private MoveOption createFarisMoveOption(int direction) {
        MoveOption newMoveOption = new MoveOption("faris");
        Move newMove = new Move(1 * direction, 2 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(1 * direction, -2 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(-1 * direction, 2 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(-1 * direction, -2 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(2 * direction, 1 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(2 * direction, -1 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(-2 * direction, 1 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(-2 * direction, -1 * direction);
        newMoveOption.addToPossibleMoves(newMove);
        return newMoveOption;
    }
    //END ADDITION
```

Changes to `createMoveOptions`:

```
    private void createMoveOptions() {
        players.get(0).addToMoveOptionQueue(createMoveOption("ryott",
        players.get(0).addToMoveOptionQueue(createMoveOption("faris",
        pla[illegible]addToMoveOptionQueue(createMoveOption("chowkida
        pla[illegible]et(0).addToMoveOptionQueue(createMoveOption("cuirassi
        play[illegible]get(0).addToMoveOptionQueue(createMoveOption("faujdar"
        players.get(0).addToMoveOptionQueue(createMoveOption("jazair",
        players.get(1).addToMoveOptionQueue(createMoveOption("ryott",
        players.get(1).addToMoveOptionQueue(createMoveOption("faris",
```

Testing:

Displaying the Faris move option correctly in the player one queue and moving a pl
legal Faris move. [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !| !| !| !|  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: faris

Player One
Score: 100
Move option queue: 1. ryott   2. faris   3. chowkidar   4. cuirassie

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 101

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  |  |  |  |  |  |
4 |  |  | !|  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: faris

Player Two
Score: 100
Move option queue: 1. ryott   2. faris   3. chowkidar   4. jazair

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 102

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  |  |  |  |  |  |
4 |  |  | "|  |  |  |
5 |  | "| "| "|  |  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: faris

Player One
Score: 101
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Task 3

Coding:

- Adding a new `moveOptionOffer` to the `createMoveOptionOffer` method, a n
  `createMoveOption` method, and adding the move option to both players in the
  parameter set correctly. [1 mark]
- Adding a Sarukh to the `createMoveOption` meth a w option which calls th
  method. [1 mark]
- Create a new method `createSa` `Option` which correctly uses the Dir
  the possible positions for the sarukh move. [1 mark]

Example So

Changes to `cr eMoveOptionOffer`:

```
    private void createMoveOptionOffer() {
        moveOptionOffer.add("sarukh"); //LINE ADDED
```

Changes to `createMoveOption`:

```
    private MoveOption createMoveOption(String name, int direction) {
        switch (name) {
            //CODE ADDED
            case "sarukh":
                return createSarukhMoveOption(direction);
            //END ADDITION
```

Changes to `createMoveOptions`:

```
    private void createMoveOptions(){
        players.get(0).addToMoveOptio  e( ateMoveOption("ryott",
        players.get(0).addToMov  ue(createMoveOption("sarukh",
        players.get(0).ad  veOp  onQueue(createMoveOption("chowkida
        players.get(0  oveOptionQueue(createMoveOption("cuirassi
        pla  et  addToMoveOptionQueue(createMoveOption("faujdar"
        pla  et(0).addToMoveOptionQueue(createMoveOption("jazair",
        play  get(1).addToMoveOptionQueue(createMoveOption("ryott",
        players.get(1).addToMoveOptionQueue(createMoveOption("sarukh",
```

Code for `createSarukhMoveOption`:

```
    //CODE ADDED
    private MoveOption createSarukhMoveOption(int direction) {
        MoveOption newMoveOption  = new MoveOption("sarukh");
        Move newMove = new Move(0, -1);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(1 * direction,-1);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(2 * direction, 0);
        newMoveOption.addToPossibleMoves(newMove);
        newMove = new Move(0,1);
        newMoveOption.addToPossibleMoves(  w Mov
        newMove = new Move(1 * dire  1);
        newMoveOption.addToPo  s(newMove);
        return newMoveO
    }
    //END A  N
```

Testing:

Displaying the Sarukh move option correctly in the player one queue and moving a
a legal Sarukh move. [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !| !| !| !|  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: sarukh

Player One
Score: 100
Move option queue: 1. ryott   2. sarukh   3. chowkidar   4. cuirassi

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 101


   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  |  |  |  |  |  |
4 |  | !|  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: sar

Player Two
Score: 100
Move option queue: 1. ryott   2. sarukh   3. chowkidar   4. jazair

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 101


   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  |  |  |  |  |  |
4 |  | !|  |  |  |  |
5 | "|  | "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: sar

Player One
Score: 101
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Task 4

Coding:

- Change playGame to randomly award a Wafr to the current player and if one has that they can select any queue position without cost. [1 mark]
- Change playGame so that if a move is legal and a Wafr has been awarded to the points cost to the player. [1 mark]
- Create a new method awardWafr in the Dastar class which has a 25% chance
- Adding the wafrAwarded attribute to Player with get/set methods for wafrAwa

Example Solution

Changes to Player:

```
class Player {
    private boolean wafrAwarded = false; //LINE ADDED
```

```
    //CODE ADDED
    public boolean getWafrAwarded(){
        return wafrAwarded;
    }
    public void setWafrAwarded(){
        wafrAwarded=true;
    }
    //END ADDITION
```

Code for awardWafr:

```
    //CODE ADDED
    public boolean awardWafr(){
        return rGen.nextInt(1,5) == 1;
    }
    //END ADDITION
```

Changes to playGame:

```
    public void playGame(
        boolean gameOver = false;
        while (!gameOver) {
            displayState();
            boolean squareIsValid = false;
            int choice;
            //CODE CHANGE
            boolean wafr=false;
            if (awardWafr() && !currentPlayer.getWafrAwarded()){
                Console.writeLine("You have been awarded a Wafr, you ca
                queue for free this turn.");
                currentPlayer.setWafrAwarded();
                wafr=true;
            }
            if (wafr) {
                do {
                    Console.write("Choose move option to use from queu
                    choice = Integer.parseInt(Console.readLine());
                } while (choice < 1 || choice > 5)
            } else {
                do {
                    Console.write("Choose move option to use from queu
                    offer
                    choice = Integer.parseInt(Console.readLine());
                    if (choice == 9) {
                        useMoveOptionOffer();
                        displayState();
                    }
                } while (choice < 1 || choice > 3);
            }
            //END CHANGE
            int startSquareReference = 0;
            if (moveLegal) {
```

```
                int pointsForPieceCapture = calculatePieceCapturePoints
                //CODE CHANGE
                if (!wafr) {
                    currentPlayer.changeScore(-(choice + (2 * (choice
                }
                //END CHANGE
                currentPlayer.updateQueueAfterMove(choice);
```

Testing:

Show a player being awarded a Wafr and selecting a move from position 4 or 5 in the queue incurring a cost. [1 mark]

```
Move option queue: 1. jazair

Player Two
Score: 104
Move option queue: 1. chowkidar   2. jazair   3. faujdar   4. cuiras

Turn: Player Two

You have been awarded a Wafr, you can select any move from your queu
Choose move option to use from queue (1 to 5): 5
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 109


   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  | "|  |  |  |  |
4 |  |  |  |  |  |  |
5 |  |  | "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------
```

Task 5

Coding:

- Change playGame to give new menu option 8 and reduction of player score by 5
- Adding the opponent variable (or similar) to playGame and correctly assigning it [1 mark]
- Correctly printing out the opponent's queue. [1 mark]
- Creation of getJustQueue which calls the getQueueAsString method for the [1 mark]

Example Solution

Changes to playGame

```
int choice;
do {
    //CODE CHANGE
    Console.write("Choose move option to use from queue (1
    opponent's queue or 9 to take the offer: ");
    choice = Integer.parseInt(Console.readLine());
    if (choice == 9) {
        useMoveOptionOffer();
        displayState();
    } else if (choice == 8) {
        Player opponent = players.get(0);
        if (opponent.sameAs(currentPlayer)){
            opponent = players.get(1);
        }
         Console.writeLine(opponent.getJustQueue());
         currentPlayer.changeScore(-5);
         Console.writeLine("New score: " + currentPlayer.g
         System.lineSeparator();
    }
    //END CHANGE
} while (choice < 1 || choice > 3);
```

Changes to Player

```
    //CODE ADDED
    public String getJustQueue(){
        return queue.getQueueAsString();
    }
    //END ADDITION
```

Testing:

Display new menu option. Player one to selection option 8 to view Player two's queue

```
Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. faujdar

Turn: Player One

Choose move option to use from queue (1 to 3), 8 to see your opponents
offer: 8
1. ryott   2. chowkidar   3. cuirassier   4. faujdar   5. cuirassier
New score: 95

Choose move option to use from queue (1 to 3), 8 to see your opponents
offer:
```

Task 6

Coding:

- Create a new method `getValidInt` which returns true if the user enters in a va suitable message and force the user to retry until they have entered a valid integ
- Change `playGame` to use the `getValidInt` method on the main game menu to the move queue choice. [1 mark]
- Change `getSquareReference` to use the `getValidInt` method for choosing a suitable error message only allow valid input. [1 mark]
- Change `useMoveOptionOffer` to use the `getValidInt` method for choosing allow valid integer input. Include a range of 1 to 5 to prevent an invalid queue should inc valid error message and a prompt to re-enter until it is valid. [1

A: passing of prompt to `getValidInt()` instead, but do not award if the line sp

Example Solution

Code for `getValidInt`:

```
    //CODE ADDED
    private int getValidInt(){
        boolean valid = false;
        int number = 0;
        String userInput;
        while(!valid){
            try{
                number = Integer.parseInt(Console.readLine());
                valid = true;
            } catch (Exception e) {
                Console.write("Invalid input, you must enter an intege
            }
        }
        return number;
    }
    //END ADDITION
```

Changes to p me

```
            Console.write("Choose move option to use from queue (1 to
            choice = getValidInt(); //LINE CHANGED
            if (choice == 9) {
                useMoveOptionOffer();
                displayState();
            }
        } while (choice < 1 || choice > 3);
        int startSquareReference = 0;
        while (!squareIsValid) {
            startSquareReference = getSquareReference("containing t
            squareIsValid = checkSquareIsValid(startSquareReferenc
            //CODE ADDED
            if (!squareIsValid){
                Console.writeLine("You must enter a valid square."
            }
            //END ADDITION
        }
        int finishSquareRe r 0;
        squareIsVali
        while ( Isvalid) {
            in hsquareReference = getSquareReference("to move to
            quareIsValid = checkSquareIsValid(finishSquareReferen
            //CODE ADDED
            if (!squareIsValid){
                Console.writeLine("You must enter a valid square."
            }
            //END ADDITION
        }
```

Changes to getSquareReference:

```
    private int getSquareReference(String description) {
        int selectedSquare;
        Console.write("Enter the square " + description + " (row number
        selectedSquare = getValidInt(); //LINE CHANGED
        return selectedSquare;
    }
```

Changes to useMoveOptionOffer:

```
    private void useMoveOption[illegible]() {
        int replaceChoice;
        Consol[illegible] wr[illegible]ose the move option from your queue to repla
        //C[illegible]AA[illegible]
        do {
            replaceChoice = getValidInt();
            if (replaceChoice<1 || replaceChoice >5) {
                Console.writeLine("You must enter a number from 1-5.")
                Console.write("Please re-enter your selection: ");
            }
        } while (replaceChoice<1 || replaceChoice >5);
        //END CHANGE
        currentPlayer.updateMoveOptionQueueWithOffer(replaceChoice - 1
```

Testing:

Display an appropriate error message if the user enters in non-valid inputs for the n
to place moveOptionOffer in the queue. [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !| !| !| !|  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option [illegible] jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. faujdar   5.

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: help
Invalid input, you must enter an integer, please try again: 1
Enter the square containing the piece to move (row number followed by column
You must enter a valid square.
Enter the square containing the piece to move (row number followed by column
Enter the square to move to (row number followed by column number): 32
New score: 104

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  | !|  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jaza[illegible]

Player Two
Score: 100
Move option [illegible] 1. ryott   2. chowkidar   3. jazair   4. faujdar   5. cui

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer: 9
Choose the move option from your queue to replace (1 to 5): 8
You must enter a number from 1-5.
Please re-enter your selection:
```

Task 7

Coding:

- Adding the `choiceOptionsLeft` attribute to `Player` with getter method. Initialis
- Create a new method `decreaseChoiceOptionsLeft` in `Player` which decrem advise the player how many move options they have left. [1 mark]
- Change `playGame` to test if the player has used all their offer options and, if so,
- Change `userMoveOptionOffer` to call `decreaseChoiceOptionsLeft` for the a move option from the menu. [1 mark]

Example Solution:

Changes to Player:

```
class Player {
    private String name;
    private int direction, score, choiceOptionsLeft; //LINE CHANGED
    private MoveOptionQueue queue = new MoveOptionQueue();

    public Player(String n, int d) {
        score = 100;
        name = n;
        direction = d;
        choiceOptionsLeft = 3; //LINE ADDED
    }
    //CODE ADDED
    public int getChoiceOptionsLeft(){
        return choiceOptionsLeft;
    }
    public void decreaseChoiceOptionsLeft(){
        choiceOptionsLeft--;
    }
    //END ADDITION
```

Changes to playGame:

```
            int choice;
            do {
                //CODE CHANGED
                if (currentPlayer.getChoiceOptionsLeft()>0){
                    Console.write("Choose move option to use from queu
                    the offer: ");
                } else {
                    Console.write("Choose move option to use from queu
                }
                choice = Integer.parseInt(Console.readLine());
                if (choice == 9 && currentPlayer.getChoiceOptionsLeft(
                    //END CHANGE
                    useMoveOptionOffer();
```

Changes to `useMoveOptionOffer`:

```
        moveOptionOfferPosition = [illegible] Int(5);
        //CODE ADDED
        currentPlayer [illegible] ChoiceOptionsLeft();
        Cons[illegible] ("You have "+currentPlayer.getChoiceOptionsLe
        lef[illegible]
        //EN[illegible]ITION
    }
```

Testing:

Show player one selecting a move from the move option offer menu and decreasing

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Choose the move option from your queue to replace (1 to 5): 1
You have 2 move option choices left.

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: ryott

Player One
Score: 92
Move option queue: 1. jazair   2. chowkidar   3. cuirassier   4. fau

Turn: Player

Choose move option to use from queue (1 to 3) or 9 to take the offer
Choose the move option from your queue to replace (1 to 5): 2
You have 1 move option choices left.

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: faujdar

Player One
Score: 86
Move option queue: 1. jazair   2. ryott   3. cuirassier   4. faujdar
```

```
Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Choose the move option from your queue to replace (1 to 5): 3
You have 0 move option choices left.

   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ------------------

Move option queue: cuirassier

Player One
Score: 82
Move option queue: 1. jazair   2. ryott   3. faujdar   4. faujdar

Turn: Player One

Choose move option to use from queue (1 to 3): 9
Choose move option to use from queue (1 to 3):
```

Task 8

Coding:

- Writing the `resetQueueBackAfterUndo` method which calls the `resetQueueB` (the same one it was passed - `position` – but adjusted by -1 to make it an inde pops the item from the end of the queue and returns it to its original position. [1
- Asking the player if they would like to undo after the m y played their move an
- Correctly handling the undo to deduct 5 a reset the board and queue. [

Example Solution

Changes to P

```
    //CODE ADDED
    public void resetQueueBackAfterUndo(int choice){
        queue.resetQueueBack(choice-1);
    }
    //END ADDITION
```

Changes to `moveOptionQueue`:

```
    //CODE ADDED
    public void resetQueueBack(int position){
        MoveOption oldMove = queue.get(queue.size()-1);
        queue.remove(queue.size()-1);
        queue.add(position,oldMove);
    }
    //END ADDITION
```

Changes to `playGame`:

```
            boolean moveLegal = curr m yer.checkPlayerMove(choice, 
            finishSquareRefer ce ;
            //CODE CHANG
            St ing n "n";
            ov egal) {
                nt previousScore = currentPlayer.getScore();
                int pointsForPieceCapture = calculatePieceCapturePoints
                currentPlayer.changeScore(-(choice + (2 * (choice - 1)
                currentPlayer.updateQueueAfterMove(choice);
                updateboard(startSquareReference, finishSquareReferenc
                updatePlayerScore(pointsForPieceCapture);
                Console.writeLine("New score: " + currentPlayer.getScor
                displayState();
                Console.write("Would you like to undo your move(y/n)? 
                undo = Console.readLine().toLowerCase().strip().substr
                if (undo.equals("y")){
                    updateboard(finishSquareReference, startSquareRefe
                    currentPlayer.resetQueueBackAfterUndo(choice);
                    currentPlayer.changeScore(previousScore-currentPla
                } else {
                    undo = "n";
                }
            }
            if (undo.equals("n")) {
                if (cur .sameAs(players.get(0))) {
                    tPlayer = players.get(1);
                e e {
                    currentPlayer = players.get(0);
                }
            }
            //END CHANGE
            gameOver = checkIfGameOver();
```

Testing:

- Showing that a move can be undone and that 5 points are deducted. [1 mark]
- Showing that the same player can still play their turn and that the game can con
  [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 101


   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !! !! !!  |
3 | !!  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer:

Player One
Score: 101
Move option queue: 1. ryott   2. cuirassier   3. faujdar   4. jazair

Turn: Player One

Would you like to undo your move (y/n)? y

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 95
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 99
```

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  | !|  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 99
Move option queue: 1. chowkidar   2. cuirassier   3. faujdar   4. ja

Turn: Player

Would you like to undo your move (y/n)? n

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  | !|  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3.   r   4. faujdar

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Task 9

Coding:

- `createMoveOptionOffer` method has been modified to the append "sahm" as dealing with the `Name` parameter of "sahm" in the `createMoveOption` method.
- Making the Sahm the move option for both players on their first turn. [1 mark]
- Correctly creating the `sahmUsed` attribute with getter/setter methods. [1 mark]
- Only allowing a player to fire a single Sahm in a game. [1 mark]
- Correctly removing all the pieces in the Sahm's line of fire from the board (except `calculateSahmMove` method). [1 mark]
- Correctly awarding points for all removed/destroyed pieces (even if a piece was `calculateSahmMove` method. [1 mark]

Example Solution

Changes to `createMoveOptionOffer`:

```
    private void createMoveOptionOffer() {
        moveOptionOffer.add("sahm"); //LINE ADDED
        moveOptionOffer.add("jazair");
```

Code for `createSahmMoveOption`:

```
    //CODE ADDED
    private MoveOption createSahmMoveOption(int direction) {
        MoveOption newMoveOption  = new MoveOption("sahm");
        Move newMove = new Move(0, 0);
        newMoveOption.addToPossibleMoves(newMove);
        return newMoveOption;
    }
    //END ADDITION
```

Changes to `createMoveOption`:

```
    private MoveOption cr[illegible]Option(String name, int direction) {
        switch (nam[illegible]
            [illegible]
            [illegible]"sahm":
                return createSahmMoveOption(direction);
            //END ADDITION
            case "chowkidar":
```

Changes to `Player`:

```
class Player {
    private String name;
    private int direction, score;
    private MoveOptionQueue queue = new MoveOptionQueue();
    private boolean sahmUsed; //LINE ADDED

    public Player(String n, int d) {
        score = 100;
        name = n;
        direction = d;
        sahmUsed = false; //LINE ADDED
    }
    //CODE ADDED
    public boolean gets[illegible]{
        retur[illegible]ah[illegible]
    }
    public v[illegible]etSahmUsed(){
        sahmUsed=true;
    }
    public boolean choiceIsSahm(int choice) {
        return queue.getMoveOptionInPosition(choice-1).getName().equal
    }
    //END ADDITION
```

Changes to playGame:

```
            while (!squareIsValid) {
                startSquareReference = getSquareReference("containing
                squareIsValid = checkSquareIsValid(startSquareReferenc
            }
            //CODE CHANGE
            if (currentPlayer.choiceIsSahm(choice)){
                if (currentPlayer.getSahmUsed()) {
                    Console.writeLine("You have already used your Sahm
                } else {
                    currentPlayer.setSahmUsed();
                    int pointsForPieceCapture = calculateSahmMove(star
                    currentPlayer.changeScore(choice);
                    currentPlayer.changeScore(-(choice + (2 * (choice
                    currentPlayer.updateQueueAfterMove(choice);
                    updatePlayerScore(pointsForPieceCapture);
                    Console.writeLine("New score: " + currentPlayer.ge
                    System.lineSeparator());
                }
            } else {
                int finishSquareReference = 0;
                squareIsValid = false;
                while (!squareIsValid) {
                    finishSquareReference = getSquareReference("to mov
                    squareIsValid = checkSquareIsValid(finishSquareRef
                }
                boolean moveLegal = currentPlayer.checkPlayerMove(choi
                finishSquareReference);
                if (moveLegal) {
                    int pointsForPieceCapture = calculatePieceCapturePo
                    currentPlayer.changeScore(-(choice + (2 * (choice
                    currentPlayer.updateQueueAfterMove(choice);
                    updateboard(startSquareReference, finishSquareRefe
                    updatePlayerScore(pointsForPieceCapture);
                    Console.writeLine("New score: " + currentPlayer.ge
                    System.lineSeparator());
                }
            }
            //CODE CHANGE
            if (currentPlayer.sameAs(players.get(0))) {
```

Code for calculateSahmMove:

```
    //CODE ADDED
    public int calculateSahmMove(int squareReference){
        int row = squareReference / 10;
        int col = squareReference % 10;
        int score = 0;
        int endRow = 0;
        int direction = currentPlayer.getDirection();

        if (direction == 1){
            endRow = 6;
        } else {
            endRow = 1;
        }

        while (row!=endRow){
            row+=direction;
            Square boardSquare = board.get(getIndexOfSquare(10*row+col
            if (boardSquare.getPieceInSquare() != null && !boardSquare
                score += boardSquare.getPieceInSquare().getPointsIfCap
                boardSquare.removePiece();
            }
        }
        return score;
    }
    //END ADDITION
```

Testing:

- Showing the board correctly after the Sahm has been fired (allow follow-through
  The pieces on 23 and 33 must have been destroyed to award the mark. [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: sahm

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 101

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !! !! !!  |
3 |  |  | !!  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: sahm

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Choose the move option from your queue to replace (1 to 5): 1

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !! !! !!  |
3 |  |  | !!  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 92
Move option queue: 1. [illegible] chowkidar   3. jazair   4. faujdar
```

```
Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
New score: 98

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  |  | !! !!|  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 101
Move option queue: 1. ryott   2. cuirassier   3. faujdar   4. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: raaket

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 101

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !! !! !!|  |
3 |  |  | !!|  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: raaket

Player Two
Score: 100
Move option queue: 1. ryott   2. [illegible]   3. jazair   4. faujdar
```

```
Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Choose the move option from your queue to replace (1 to 5): 1

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !! !! !! |
3 |  |  | !! |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option queue: 1. out

Player Two
Score: 92
Move option queue: 1. raaket   2. chowkidar   3. jazair   4. faujdar

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
New score: 98
```

Task 10

Coding:

- Add option 7 to the menu to create a Kotla. [1 mark]
- Checking that the square in which the player wishes to create the Kotla is empty
- Creating a Kotla of the correct type in the square and removing the piece. [1 ma

Example Solution

Changes to playGame:

```
        ch
        //CODE CHANGE
            Console.write("Choose move option to use from queue (1
            or 9 to take the offer: ");
            choice = Integer.parseInt(Console.readLine());
            if (choice == 9) {
                useMoveOptionOffer();
                displayState();
            }
        } while ((choice < 1 || choice > 3) &&(choice!=7));
        int startSquareReference = 0;
        if (choice == 7) {
            while (!squareIsValid){
                startSquareReference = getSquareReference("contain
                for a new Kotla");
                squareIsValid = checkSquareIsValid(startSquareRefe
            }
            if (currentPlayer.sameAs(players g  ))) {
                board.set(getIndexOfSqua  (st  SqdareReference),n
            } else {
                board.set(ge  ex  qdare(startSquareReference),n
            }
        } else {
            wh  squareIsValid) {
                startSquareReference = getSquareReference("contain
                squareIsValid = checkSquareIsValid(startSquareRefe
            }
            int finishSquareReference = 0;
            squareIsValid = false;
            while (!squareIsValid) {
                finishSquareReference = getSquareReference("to mov
                squareIsValid = checkSquareIsValid(finishSquareRef
            }
            boolean moveLegal = currentPlayer.checkPlayerMove(choi
            finishSquareReference);
            if (moveLegal) {
                int pointsForPieceCapture = calculatePieceCapturePo
                currentPlayer.changeScore(-(choice + (2 * (choice
                currentPlayer.updateQueueAfterMove(choice);
                updateboard(startSquareReferen  finishSquareRefe
                updatePlayerScore(points  ie  Capture);
                Console.writeLine("  s  e: " + currentPlayer.ge
                System.lines  t  ));
            }
        }
        urrentPlayer.sameAs(players.get(0))) {
```

Testing:

Showing the creation of the new Kotla (even if the letter is wrong) and removal of th

```
   1  2  3  4  5  6
  -------------------
1 |   |  |K1|  |  |  |
2 |   | !| !| !| !|  |
3 |   |  |  |  |  |  |
4 |   |  |  |  |  |  |
5 |   | "| "| "| "|  |
6 |   |  |  |k2|  |  |
  -------------------

Move option offer:

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3), 7 to create a Kotla o
Enter the square containing the piece to sacrifice for a new Kotla (
number): 22

   1  2  3  4  5  6
  -------------------
1 |   |  |K1|  |  |  |
2 |   |K | !| !| !|  |
3 |   |  |  |  |  |  |
4 |   |  |  |  |  |  |
5 |   | "| "| "| "|  |
6 |   |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar

Turn: Playe

Choose move option to use from queue (1 to 3), 7 to create a Kotla o
```

Task 11

Coding:

- Adding option 6 to the menu which brings up a list of options to modify the queu
- Option a correctly reverses the player's queue in a method inside MoveOptionQ
- Option b correctly swaps queue with the opponent without breaking encapsulati that shouldn't be exposed. [1 mark]
- Option c correctly swaps the first and last elements of your queue. [1 mark]
- Option d correctly moves an element to the front of the queue and shuffles up th
- Option e quits and doesn't cost points but the other options all cost 3 points.

Example So

Changes to playGame:

```
            int choice;
            do {
                //CODE CHANGE
                Console.write("Choose move option to use from queue (1
                or 9 to take the offer: ");
                choice = Integer.parseInt(Console.readLine());
                if (choice == 6) {
                    modifyQueueOptions();
                } else if (choice == 9) {
                    useMoveOptionOffer();
                    displayState();
                }
                //END CHANGE
            } while (choice < 1 || choice > 3);
```

Code for modifyQueueOptions:

```
    //CODE ADDED
    private void modifyQueueOp
        String choice;
        Consol
        Con
        Con
        Cons
        Console.writeLine("c) Swap the first and last move options in y
        Console.writeLine("d) Move an option of your choice to positio
        Console.writeLine("e) Don't modify the queue, let me play my m
        Console.write("Enter your choice (a-e): ");
        choice = Console.readLine().toLowerCase();
        while (!(choice.equals("a") || choice.equals("b") || choice.eq
        choice.equals("d") || choice.equals("e"))) {
            Console.write("You must choose a letter from a to e: ");
            choice = Console.readLine().toLowerCase();
        }
        if (choice.equals("e")) {
            return;
        } else if (choice.equals("a")) {
            currentPlayer.reverseQueue();
        } else if (choice.equals("b")) {
            MoveOptionQueue p1Queue = players.g
            MoveOptionQueue p2Queue = playe
            players.get(0).replaceQu
        } else if (choice.equa
            currentPlayer
        } else if (
            te
            le.write("Enter position of item to move to the front
            = Integer.parseInt(Console.readLine());
            currentPlayer.moveItemToFront(item);
        }
        currentPlayer.changeScore(-3);
        displayState();
    }
    //END ADDITION
```

Changes to moveOptionQueue:

```
    //CODE ADDED
    public void reverseQueue(){
        List<MoveOption> reversedQueue = new ArrayList<>();
        for (int i=queue.size()-1;i>=0;i--){
            reversedQueue.add(queue.get(i));
        }
        queue = reversedQueue;
    }

    public void moveItemToFront(int itemNumber){
        MoveOption queueItem = queue.get(position);
        queue.remove(position);
        queue.add(0,queueItem);
    }

    public void swapFirstAndLast(){
        MoveOption last = queue.get(queue.size()-1);
        MoveOption first = queue.get(0);
        queue.remove(queue.size()-1);
        queue.remove(0);
        queue.add(first);
        queue.add(0,last);
    }
    //END ADDITION
```

Changes to Player:

```
    //CODE ADDED
    public void reverseQueue(){
        queue.reverseQueue();
    }

    public MoveOptionQueue replaceQueue(MoveOptionQueue newQueue){
        MoveOptionQueue oldQueue = queue;
        queue=newQueue;
        return oldQueue;
    }

    public void swapFirstAndLast(){
        queue.swapFirstAndLast();
    }

    public void moveItemToFront(int itemNumber){
        queue.moveItemToFront(itemNumber-1);
    }
    //END ADDITION
```

Testing:

- Showing at least one of options a–d working. [1 mark]
- Showing the remaining three options working. [1 mark]
- Showing option e and the scoring working correctly. [1 mark]

```
   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ------------------

Move option offer: jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3), 6 to modify the queue
You have the following options to modify your queue:
a) Reverse your queue
b) Swap queues with your opponent
c) Swap the first and last move options in your queue
d) Move an option of your choice to position 1 in the queue
e) Don't modify the queue, let me play my move!
Enter your choice (a-e): a

   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |  |  |  |
  ------------------

Move option offer: jazair

Player One
Score: 97
Move option queue: 1. jazair   2. faujdar   3. cuirassier   4. chowk

Turn: Player One

Choose move option to use from queue (1 to 3), 6 to modify the queue
You have the following options to modify your queue:
a) Reverse your queue
b) Swap queues with your opponent
c) Swap the first and last move options in your queue
d) Move an option of your choice to position 1 in the queue
e) Don't modify the queue, let me play my move!
Enter your choice (a-e): b

   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ------------------

Move option offer: jazair
```

```
Player One
Score: 94
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar

Turn: Player One

Choose move option to use from queue (1 to 3), 6 to modify the queue
You have the following options to modify your queue:
a) Reverse your queue
b) Swap queues with your opponent
c) Swap the first and last move options in your queue
d) Move an option of your choice to position 1 in the queue
e) Don't modify the queue, let me play my move!
Enter your choice (a-e): e

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 91
Move option queue: 1. cuirassier   2. chowkidar   3. jazair   4. fau

Turn: Player One

Choose move option to use from queue (1 to 3), 6 to modify the queue
You have the following options to modify your queue:
a) Reverse your queue
b) Swap queues with your opponent
c) Swap the first and last move options in your queue
d) Move an option of your choice to position 1 in the queue
e) Don't modify the queue, let me play my move!
Enter your choice (a-e): d
Enter position of item to move to the front(2-5): 4

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 88
Move option queue: 1. faujdar   2. cuirassier   3. chowkidar   4. jaz

Turn: Player One

Choose move option to use from queue (1 to 3), 6 to modify the queue
You have the following options to modify your queue:
a) Reverse your queue
b) Swap queues with your opponent
c) Swap the first and last move options in your queue
d) Move an option of your choice to position 1 in the queue
e) Don't modify the queue, let me play my move!
Enter your choice (a-e): e
Choose move option to use from queue (1 to 3), 6 to modify the queue
```

Task 12

Coding:

- Creating and storing the number of pieces correctly in the new protected attribut
- Adding a call to `checkReincarnation` in the correct place. [1 mark]
- Creating `countNormalPieces` to correctly return the numbe of pieces excludin [1 mark]
- Correctly detecting when a piece reaches the op ion s back row. [1 mark]
- Having a condition to only allow reinca a the player has fewer pieces than
- Correctly handling the reinca ion n the player's own back row and checking

Example Sol

Additional im tement to enable the use of a ListIterator (at the top with the ot

```
import java.util.ListIterator; //LINE ADDED
```

Changes to Dastan:

```
    protected Random rGen = new Random();
    protected int noOfPieces; //LINE ADDED

    public Dastan(int r, int c, int noOfPieces ){
        this.noOfPieces = noOfPieces; //LINE ADDED
```

Changes to playGame:

```
            if (moveLegal) {
                checkReincarnation(); //LINE ADDED
                int pointsForPieceCapture = calculatePieceCapturePoint
```

New methods countNormalPieces and checkReinca ion:

```
    //CODE ADDED
    private int countNormalPie s() {
        int pieces = 0;
        ListIterat e> boardSquares = board.listIterator();
        whi ar qares.hasNext()){
            pieceInSquare = boardSquares.next().getPieceInSquare
            (pieceInSquare != null && pieceInSquare.getBelongsTo().
            pieceInSquare.getTypeOfPiece().equals("piece")){
                pieces++;
            }
        }
        return pieces;
    }

    private void checkReincarnation(int squareReference) {
        int row = squareReference / 10;
        int col = squareReference % 10;
        if (currentPlayer.sameAs(players.get(0))) {
            if (row == noOfRows && countNormalPieces() < noOfPieces) {
                Console.writeLine("Congratulations, you have earned a
                Console.write("Which column wou you like you piece t
                int reincarnationCol = Inte er. sInt(Console.readLi
                while (board.get(get fSquare(10+reincarnationCol)
                    Console.wr / The square must be empty.");
                    Cons + ("Which column would you like you pie
                    nationCol = Integer.parseInt(Console.readLi

                board.get(getIndexOfSquare(10+reincarnationCol)).setPi
                (new Piece("piece",players.get(0),1,"!"));
            } else {
                Console.writeLine("Congratulations, you have earned a
                Console.write("Which column would you like you piece t
                int reincarnationCol = Integer.parseInt(Console.readLi
```

```
                while (board.get(getIndexOfSquare(noOfRows*10+reincarn
                 != null){
                    Console.writeLine("The square must be empty.");
                    Console.write("Which column would you like you pie
                    reincarnationCol = Integer.parseInt(Console.readLi
                }
                board.get(getIndexOfSquare(noOfRows*10+reincarnationCo
                ("piece",players.get(1),1,"\""));
            }
        }
    }
    //END ADDITION
```

Testing:

Correctly shows the moves as requested in the tests, specifically including the ch
reincarnate on and then the correct one. [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 | "| !! !!|  |  |  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 | !! "| "| "|  |  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. [illegible]ssier   4. fauj

Turn: Player One

Choose move option to use [illegible] (1 to 3) or 9 to take the offer
Enter the square con[illegible] the piece to move (row number followed b
Enter the s[illegible] t[illegible]o (row number followed by column number):
Congratula[illegible] you have earned a reincarnation!
Which colum[illegible]d you like your piece to be reincarnated on? 3
The square must be empty.
Which column would you like your piece to be reincarnated on? 4
New score: 104

   1  2  3  4  5  6
  -------------------
1 |  |  |K1| !!|  |  |
2 | "| !! !!|  |  |  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "|  |  |
6 | !!|  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar
```

```
Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 104

   1  2  3  4  5  6
  -------------------
1 | "|  |K1| !!|  |  |
2 |  | !! !!|  |  |  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "|  |  |
6 | !!|  |  |k2|  |  |
  -------------------

Move option queue: jazair

Player One
Score: 104
Move option queue: 1. chowkidar   2. cuirassier   3. faujdar   4. ja

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Task 13

Coding:

- Putting the Taziz in the correct place regardless of board size. [1 mark]
- Having a mechanism that correctly counts the number of turns that the Taziz has
- Resetting the `campedTurns` attribute if the square becomes empty or changes o
- Allowing the player to make a move that costs zero p[illegible] when they have held t
- Showing the correct A and a symbols when the Taziz is occupied by overriding s
- Correctly resetting the symbol for the Taziz to 'x' when a player leaves by overrid

Example Solution

Changes to [illegible]Board:

```
            } else if (row == noOfRows && column == noOfColumns / 2
                s = new Kotla(players.get(1), "k");
            //CODE ADDED
            } else if (row == noOfRows / 2 + 1 && column == noOfCo
                s = new Taziz();
            //END ADDITION
            } else {
                s = new Square();
```

Changes to playGame:

```
    public void playGame() {
        boolean gameOver = false;
        Taziz taziz = (Taziz)board.get(getIndexOfSquare((noOfRows / 2 +
        noOfColumns % 2))); //LINE ADDED
        while (!gameOver) {
```

```
            if (moveLegal) {
                int pointsForPieceCapture = calculatePieceCapturePoints
                //CODE CHANGE
                currentPlayer = updateQueueAfterMove(choice);
                updateBoard(startSquareReference, finishSquareReferenc
                if (taziz.getCampedTwoTurns(currentPlayer)) {
                    currentPlayer.changeScore(-(choice + (2 * (choice
                } else {
                    Console.writeLine(currentPlayer.getName() + " got a
                }
                taziz.checkCamp(currentPlayer);
                //END CHANGE
                updatePlayerScore(pointsForPieceCapture);
```

Code for Taziz:

```
//CODE ADDED
class Taziz extends Square {

    private int campedTurns;

    public Taziz() {
        super();
        campedTurns = 0;
        symbol = "x";
    }

    public [illegible]heckCamp(Player p){
        if [illegible]InSquare != null){
            [illegible](pieceInSquare.getBelongsTo().getDirection() == p.getDi
            campedTurns += 1;
            Console.writeLine(p.getName() + " has camped the Taziz fo
        }
    }
}
```

```
    public boolean getCampedTwoTurns(Player p){
        return campedTurns >= 2 && pieceInSquare.getBelongsTo().getDir
    }

    @Override
    public void setPiece(Piece p) {
        super.setPiece(p);
        if (p.getBelongsTo().getDirection() == 1){
            symbol = "A";
        } else {
            symbol = "a";
        }
    }

    @Overri
    public P removePiece() {
        campedTurns = 0;
        symbol = "x";
        return super.removePiece();
    }
}
//END ADDITION
```

Testing:

- Show the Taziz being occupied and changing from x to A. [1 mark]
- Show player one getting a free move. [1 mark]

```
   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |x |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option queue: jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
Player One has camped the Taziz for 1 turns.
New score: 98


   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !!  | !! !!|  |
3 |  |  |  |  |  |  |
4 |  |  |A!|  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ----------  ------

Move option queue: jazair

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar
```

```
Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 104

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !!  | !! !!  |
3 |  |  |  |  |  |  |
4 |  | "|A!|  |  |  |
5 |  |  | "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option queue: jazair

Player One
Score: 98
Move option queue: 1. ryott   2. chowkidar   3. faujdar   4. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
Player One has camped the Taziz for 2 turns.
New score: 99

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !!  | !!  |  |
3 |  |  |  |  |  | !!
4 |  | "|A!|  |  |  |
5 |  |  | "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option queue: jazair

Player Two
Score: 104
Move option queue: 1. chowkidar   2. jazair   3. faujdar   4. cuiras

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 108

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !!  | !!  |  |
3 |  |  |  |  |  | !!
4 |  | "|A!|  |  | "|
5 |  |  | "| "|  |  |
6 |  |  |  |k2|  |  |
  -------------------

Move option queue:

Player One
Score: 99
Move option queue: 1. ryott   2. faujdar   3. jazair   4. cuirassier
```

```
Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
Player One got a free move.
Player One has camped the Taziz for 3 turns.
New score: 105


   1  2  3  4  5  6
  -------------------
1 |  |  |Kl|  |  |  |
2 |  | !|  | !|  |  |
3 |  |  |  |  |  |  |
4 |  | "|A |  |  |  |
5 |  |  | "|  |  |  |
6 |  |  |  |  |  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 108
Move option queue: 1. jazair   2. faujdar   3. cuirassier   4. ryott

Turn: Player Two
Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Task 14

Coding:

- Using a method to track the Weather Event (this is the weatherEvent variable
- Having the countdown timer allow precisely two complete turns from when it is a
- Announcing to the players when the Weather Event started with a 2 turns warni
- Destroying all pieces in the same column as the Weather Event when the timer
- Destroying a Kotla in the Weather Event column when the timer expires. [1 mar
- Correctly selecting a random empty square. [1 mark]
- Creating a weatherEvent class with getWeatherLocation and setWeather
- Implementing countDownTimerComplete so that it returns an appropriate value inclu expires. [

Example Solution

Changes to playGame:

```
    public void playGame() {
        boolean gameOver = false;
        WeatherEvent weatherEvent = null; //LINE ADDED
        while (!gameOver) {
```

```
            squareIsValid = false;
            while (!squareIsValid) {
                finishSquareReference = getSquareReference("to move to
                squareIsValid = checkSquareIsValid(finishSquareReferen
            }
            //CODE ADDED
            if (weatherEvent == null){
                weatherEvent = weatherEventOccurs();
            } else {
                if (weatherEvent.countDownTimerComplete()){
                    int colToDestroy = weatherEvent.getWeatherLocation
                    for (int row = 1;row<=noOfRows;row++){
                        if (board.get(getIndexOfSquare(row*10+colToDestro
                            board.get(getIndexOfSquare(row*10+colToDes
                        }
                        if (board.get(getIndexOfSquare(row*10+colToDes
                            board.set(getIndexOfSquare(row*10+colToDes
                        }
                    }
                }
            }
            //END ADDITION
            boolean moveLegal = currentPlayer.checkPlayerMove(choice,
            finishSquareReference);
```

Code for weatherEventOccurs:

```
    //CODE ADDED
    public WeatherEvent weatherEventOccurs(){
        WeatherEvent weatherEvent = null;
        int randomSquare=0;

        if (rGen.nextInt(2) == 1){
            boolean squareFound = false;
            while (!squareFound){
                randomSquare = rGen.nextInt(1,noOfRows+1)*10 + rGen.ne
                if (board.get(getIndexOfSquare(randomSquare)).getPiece
                !board.get(getIndexOfSquare(randomSquare)).containsKot
                    squareFound = true;
                }
            }
            weatherEvent = new WeatherEvent(randomSquare);
        }
        return weatherEvent;
    }
    //END ADDITION
```

Code for WeatherEvent:

```
//CODE ADDED
class WeatherEvent {
    private int squareReference, countDownTimer;

    public WeatherEvent(int squareReference){
        this.squareReference = squareReference;
        countDownTimer = 3;
        Console.writeLine("A weather event has occurred at location " +
        Console.writeLine("After two more turns, all pieces on the sa
    }

    public void setWeatherLocation(int squareReference) {
        this.squareReference = squareReference;
    }

    public int getWeatherLocation(){
        return squareReference;
    }

    public boolean countDownTimerComplete(){
        if (countDownTimer == 0){
            Console.writeLine("The weather event destroys all the piec
            +squareReference % 10);
            return true;
        } else {
            countDownTimer--;
            if (countDownTimer > 1){
                Console.writeLine("The weather event at location " + s
                " will occur after one more turn.");
            } else {
                Console.writeLine("The weather event at location " + s
                " will occur next turn.");
            }
            return false;
        }
    }
}
//END ADDITI
```

Testing:

- Having at least one piece owned by each player in the column where the weathe players will lose at least one piece each. [1 mark]
- Showing all pieces in the column destroyed. [1 mark]

```
   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !! !! !! !!  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ------------------

Move option offer: jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
```

```
A weather event has occured at location 61
After two complete turns, all pieces on the same column will be dest
New score: 104

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  | !|  |  |  |  |
4 |  |  |  |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer:

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
The weather event at location 61 will occur after one more turn
New score: 104

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  | !| !| !|  |
3 |  | !|  |  |  |  |
4 |  |  |  |  |  |  |
5 | "|  | "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player One
Score: 104
Move option queue: 1. chowkidar   2. cuirassier   3. faujdar   4. ja

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
The weather event at location 61 will occur next turn
New score: 108

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 | !|  | !| !| !|  |
3 |  |  |  |  |  |  |
4 |  |  |  |  |  |  |
5 | "|  | "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 104
Move option queue: 1. chowkidar   2. jazair   3. faujdar   4. cuiras
```

```
Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
The weather event at location 61 will occur next turn
New score: 108

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 | !!|  | !!| !!| !!|  |
3 |  |  |  |  |  |  |
4 |  | "|  |  |  |  |
5 | "|  |  | "| "|  |
6 |  |  |  |  |  |  |
  -------------------

Move option offer: jazair

Player One
Score: 108
Move option queue: 1. cuirassier   2. faujdar   3. jazair   4. ryott

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
The weather event destroys all the pieces in column 1
New score: 112

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  |  |  | !!| !!|  |
3 |  |  |  |  |  |  |
4 |  | "| !!|  |  |  |
5 |  |  |  | "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: jazair

Player Two
Score: 108
Move option queue: 1. jazair   2. faujdar   3. cuirassier   4. ryott

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Task 15

Coding:

- Creating a `Barrier` class that accepts the parameters `player` and `symbol` and [1 mark]
- Creating `containsBarrier` that returns true for B or b and false otherwise. [1 m
- Modifying `checkSquareIsValid` to return false if the [illegible] square contains a ba
- Creating `checkBarrierIsValid` which checks that all the squares for the barr
- Creating `checkBarrierIsVali[illegible]` checks that all the squares for the barr
- Creating `placeBarrier` [illegible] input messages that calls `checkBarrier` creates a b[illegible] 3 [illegible] wide on the board. [1 mark]
- Modifying [illegible]ePieces to make two calls to `placeBarrier`, one for each pla
- Creating `checkManhattanDistance` and modifying `playGame` to call that inste for the line starting `moveLegal=`. [1 mark]
- Inside `checkManhattanDistance`, checking that the start and end squares are `checkPlayerMove`). [1 mark]
- Inside `checkManhattanDistance`, iterating along the row and column and vice and has only been attempted. [1 mark]
- Inside `checkManhattanDistance`, correctly iterating along the row and colum all combinations of up, down, left and right (with and without vertical/horizontal m possible move orientations (in the code below this was achieved by using the ho `verticalDirection`). [1 mark]

Example Solution

Code for `Barrier`:

```
//CODE ADDED
class Barrier extends Square {
    public Barrier(Player p, St[illegible]
        super();
        belongsTo [illegible]
        sym[illegible]s,
    }
}
//END ADDITION
```

Changes to `Square`:

```
    //CODE ADDED
        public boolean containsBarrier() {
        if (symbol.equals("B") || symbol.equals("b")) {
            return true;
        } else {
            return false;
        }
    }
    //END ADDITION
```

Changes to `checkSquareIsValid` (Dastan class):

```
    private boolean checkSquar[illegible] int squareReference, boolean st
        if (!checkSquareI[illegible]squareReference)) {
            return [illegible]
        }
        //C[illegible]DED
        if (board.get(getIndexOfSquare(squareReference)).containsBarri
            return false;
        }
        //END ADDITION
        Piece pieceInSquare = board.get(getIndexOfSquare(squareReferen
```

Code for checkBarrierIsValid (Dastan class):

```
    //CODE ADDED
    private boolean checkBarrierIsValid(int barrierCentre){
        boolean barrierValid = true;
        int barrierRow = barrierCentre / 10;
        int barrierCol = barrierCentre % 10;

        for (int col=barrierCol-1;col<=barrierC[illegible]col++){
            if (!checkSquareInBounds(barrier[illegible]row[illegible]+col)){
                barrierValid = false;
            } else {
                if (boa[illegible]indexOfSquare(barrierRow*10+col)).con[illegible]
                bo[illegible]getIndexOfSquare(barrierRow*10+col)).contain[illegible]
                oa[illegible].get(getIndexOfSquare(barrierRow*10+col)).getPiec[illegible]
                    barrierValid = false;
                }
            }
        }
        return barrierValid;
    }
    //END ADDITION
```

Code for placeBarrier (Dastan class):

```
    //CODE ADDED
    public void placeBarrier(Player p, String s){
        boolean barrierValid = false;
        int barrierSquare=0;

        while (!barrierValid) {
            Console.write("Which square would you [illegible] to be the centr[illegible]
            3-square-wide barrier? ");
            barrierSquare = Integer.par[illegible]nt(C[illegible]nsole.readLine());
            if (checkBarrierIsVal[illegible](ba[illegible]eSquare)){
                barrierVali[illegible] = tr[illegible]
            } else {
                Co[illegible]riteLine("You must choose the centre of 3 emp[illegible]
                orizontally.");

        }

        int barrierRow = barrierSquare / 10;
        int barrierCol = barrierSquare % 10;

        for (int col=barrierCol-1;col<=barrierCol+1;col++){
            board.set(getIndexOfSquare(barrierRow*10+col),new Barrier(p[illegible]
        }
    }
    //END ADDITION
```

Changes to createPieces (Dastan class):

```
    private void createPieces(int noOfPieces) {
        Piece currentPiece;
        for (int count = 1; count <= [illegible]; count++) {
            currentPiece = new [illegible]iece", players.get(0), 1, "!");
            board.get(get[illegible]Of[illegible]are(2 * 10 + count + 1)).setPiece(c[illegible]
        }
        cur[illegible]ec[illegible] = new Piece("mirza", players.get(0), 5, "1");
        boa[illegible](getIndexOfSquare(10 + noOfColumns / 2)).setPiece(cur[illegible]
        //CO[illegible]ADDED
        Console.writeLine(players.get(0).getName()+", it's time to pla[illegible]
        placeBarrier(players.get(0), "B");
        //END ADDITION
```

```
        for (int count = 1; count <= noOfPieces; count++) {
            currentPiece = new Piece("piece", players.get(1), 1, "\"");
            board.get(getIndexOfSquare((noOfRows - 1) * 10 + count + 1
        }
        currentPiece = new Piece("mirza", players.get(1), 5, "2");
        board.get(getIndexOfSquare(noOfRows * 10 + (noOfColumns / 2 +
        //CODE ADDED
        Console.writeLine(players.get(1).getName()+" It's time to pla
        placeBarrier(players.get(1), "b");
        //END ADDITION
    }
```

Changes to playGame (D[illegible]ass):

```
            (squareIsValid) {
            finishSquareReference = getSquareReference("to move to"
                squareIsValid = checkSquareIsValid(finishSquareReferen
            }
            boolean moveLegal = checkManhattanDistance(choice, startSq
finishSquareReference); //LINE CHANGED
            if (moveLegal) {
```

Code for checkManhattanDistance (Dastan class):

```
    //CODE ADDED
    public boolean checkManhattanDistance(int choice, int startSquare,
        if (currentPlayer.checkPlayerMove(choice,startSquare,endSquare
            int horizontalDirection, verticalDirection;
            boolean route1Valid = true;
            boolean route2Valid = true;

            if (endSquare % 10 < startSquare % 10){
                horizontalDirection = -1;
            } else {
                horizontalDirection = 1;
            }
            if (endSquare / 10 < startSquare / 10){
                verticalDirection = -1;
            } else {
                verticalDirection = 1;
            }

            //Route 1
            int row=startSquare / 10;
            do {
                row += verticalDirection;
                if (board.get(getIndexOfSquare(row*10+(startSquare % 10
                    route1Valid=false;
                }
            } while (row != endSquare / 10 + verticalDirection);

            int col = startSquare % 10 - horizontalDirection;
            do {
                col += horizontalDirection;
                if (board.get(getIndexOfSquare((endSquare / 10)*10+col
                    route1Valid=false;
                }
            } while (col != endSquare % 10);

            //Route 2
            int col = startSquare % 10;
            do {
                col += horizontalDirection;
                if (board.get(getIndexOfSquare((startSquare / 10)*10+co
                    route2Valid=false;
                }
            } while (col != endSquare % 10 + horizontalDirection);
```

```
            row=startSquare / 10 - verticalDirection;
            do {
                row += verticalDirection;
                if (board.get(getIndexOfSquare(row*10+(endSquare % 10)
                    route2Valid=false;
                }
            } while (row != endSquare / 10 + verticalDirection);
            return route1Valid || route2Valid;
        } else {
            return false;
        }
    }
    //END ADDITION
```

Testing:

- Moving the piece correctly when only one route is valid. [1 mark]
- Not moving the piece for a cuirassier move when there is a barrier in the way. [1
- Not moving the piece when the end square is a barrier. [1 mark]
- Not moving the piece when there is a barrier in the way on both routes and the t (right to left and bottom to top). [1 mark]

```
Player One, it's time to place your barrier
Which square would you like to be the centre of your horizontal, 3-s
Player Two, it's time to place your barrier
Which square would you like to be the centre of your horizontal, 3-s

   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !| !| !| !|  |
3 |  |  |B |B |B |  |
4 |b |b |b |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ------------------

Move optio : jazair

Player One
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. cuirassier   4. fauj

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
Choose the move option from your queue to replace (1 to 5): 1

   1  2  3  4  5  6
  ------------------
1 |  |  |K1|  |  |  |
2 |  | !| !| !| !|  |
3 |  |  |B |B |B |  |
4 |b |b |b |  |  |  |
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  ------------------

Move option offer: ryott

Player One
Score: 92
Move option queue: 1. jazair   2. chowkidar   3. cuirassier   4. fau
```

```
Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer:
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
New score: 96


   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !| !|  | !|  |
3 |  |  |B |B |B |  |
4 |b |b |b |  |  | !|
5 |  | "| "| "| "|  |
6 |  |  |  |  |  | !|
  -------------------

Move option offer: ryott

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer:
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !| !|  | !|  |
3 |  |  |B |B |B |  |
4 |b |b |b |  |  | !|
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer:

Player One
Score: 96
Move option queue: 1. chowkidar   2. cuirassier   3. faujdar   4. ja

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer:
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):

   1  2  3  4  5  6
  -------------------
1 |  |  |K1|  |  |  |
2 |  | !| !|  | !|  |
3 |  |  |B |B |B |  |
4 |b |b |b |  |  | !|
5 |  | "| "| "| "|  |
6 |  |  |  |k2|  |  |
  -------------------

Move option offer: ryott

Player Two
Score: 100
Move option queue: 1. ryott   2. chowkidar   3. jazair   4. faujdar
```

```
Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer
Enter the square containing the piece to move (row number followed b
Enter the square to move to (row number followed by column number):
Enter the square to move to (row number followed by column number):
New score: 104


   1  2  3  4  5  6
  -------------------
1 |   |   |K1|   |   |   |
2 |   | !! | !! |   | !! |   |
3 |   |   |B |B |B |   |   |
4 |b |b |b |
5 | "|   | '                "| |
6 |   |   |                |   |
  -------------------

Move option offer: ryott

Player One
Score: 96
Move option queue: 1. chowkidar   2. cuirassier   3. faujdar   4. ja

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer
```

Name	

ZigZag Education supporting

A Level AQA Computer Science Paper 1

Summer 2023

Dastan

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:

Exam-style Questions

Answer all questions. Remember to save this document

Q		Answer
1	(a)	
	(b)	
2	(a)	
	(b)	
3		
4	(a)	
	(b)	
5		
6	(a)	
	(b)	
7		
8	(a)	
	(b)	
9		
10	(a)	
	(b)	
11	(a)	
	(b)	
	(c)	
12		
13		
14	(a)	
	(b)	
	(c)	
	(d)	
15	(a)	
	(b)	

Exam-style Programming Task

Answer all questions. Remember to save this document

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