

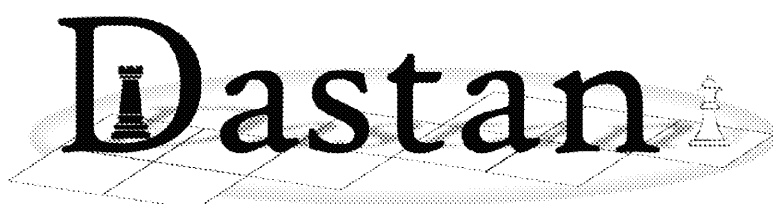
```

20 self._NoOfColumns = C
21 self._MoveOptionOfferPosition = 0
22 self.__CreateMoveOptionOffer()
23 self.__CreateBoard()
24 self.__CreatePieces(NoOfPieces)
25 self._CurrentPlayer = self._Players[0]
26
27 def __DisplayBoard(self):
28     print("\n" + " ", end="")
29     for Column in range(1, self._NoOfColumns + 1):
30         print(str(Column) + " ", end="")
31     print("\n" + " ", end="")
32     for Count in range(1, self._NoOfColumns + 1):
33         print("---", end="")
34     print("-")
35     for Row in range(1, self._NoOfRows + 1):
36         print(str(Row) + " ", end="")
37         for Column in range(1, self._NoOfColumns + 1):
38             Index = self.__GetIndexOffSquare(Row * 10 + Column)
39             print "[" + self._Board[Index].GetSymbol() + " ", end=""]

```

2015 specification
for the 2023 exam

PAPER 1 EXAM RESOURCE PACK 2023



for A Level AQA Computer Science

C# EDITION

- DIGITAL RESOURCE -

This pack includes paper versions of the electronic files.

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



POD
11728

zigzageducation.co.uk

Publish your own work... Write to a brief...
Register at publishmenow.co.uk

Follow us on Twitter @ZigZagComputing

Contents

Product Support from ZigZag Education	ii
Terms and Conditions of Use	iii
Teacher's Introduction	iv

Printouts of electronic resources (for reference)

- Code Breakdown (14 pages)
- UML Class Diagram – Complete (1 page)*
- Theory Questions: Write-on version (7 pages)
- Theory Questions: Non-write-on version (3 pages)
- Coding Tasks (19 pages)
- Additional Tasks (Extension) (6 pages)
- Theory Questions: Mark Scheme (3 pages)
- Programming Tasks: Mark Scheme (48 pages)
- Electronic Answer Document (3 pages)

** 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 (C#) – 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:

- c02a-UML-Diagram-Complete.pdf
- c06-TheoryQuestions-MS.pdf
- c07-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/method names and access levels, associations and data 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.

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

Dastan

Skeleton Code Breakdown

Class: Dastan

Identifier	Description
<<constructor	
Parameters	R : Int C : Int NoOfPieces : Int
Return values	n/a
	<p>Initialises the following protected attributes:</p> <ul style="list-style-type: none"> NoOfRows from parameter C NoOfColumns from parameter R MoveOptionOfferPos from parameter NoOfPieces <p>Instantiates two new Player objects with Direction parameter of 1 and parameter of -1 – and append to the list attribute Players.</p> <p>Assigns the element at position 0 of the list attribute Players (Player 1) to the protected attribute Player1.</p> <p>Invokes the following methods:</p> <ul style="list-style-type: none"> CalculatePieceCapturePoints() – to calculate capture points for each player. CreateMoveOptionOffer() – to create move options to the move list attribute. CreateBoard() – to create the board. CreatePieces() – to add pieces to the board using the board.
CalculatePieceCapturePoints (private)	
Parameters	FinishSquareReference : Int
Return values	Integer
	<p>Uses the GetPieceInSquare() method to get the piece at the Board location from the FinishSquareReference parameter.</p> <p>If there is a piece at that location, the method returns the location attribute for that piece is returned. If there is no piece at that location the method returns 0.</p>
CheckIfGameOver (private)	
Parameters	n/a
Return values	Boolean
	<p>Iterates through the Board list attribute to check if there is a piece at the location of the piece.</p> <p>If the Board contains a piece at the location of the piece, and the piece is a Kotla, and the piece belongs to the opposite player of the player that square. Under this scenario, if the player has just captured their opponent's piece, the method checks if the player has lost their Mirza. If the player has lost their Mirza, the method returns true. Otherwise, the method checks if the player has lost their Mirza. If the player has lost their Mirza, the method returns true. Otherwise, the method returns false.</p> <p>A negated logical AND of the two conditions. If both players have lost their Mirza, the method returns true. Otherwise, it returns false.</p>




INSPECTION COPY

COPYRIGHT
PROTECTED



CheckSquareInBounds (private)		
Parameters	SquareReference : Int	Used as an error handling method. The SquareReference parameter is playing board. The method initialises two local variables, Row and Col, split off the row and column from the SquareReference parameter. It then checks to confirm if Row is outside the range of the attribute NoOfRows and Col is outside the range of the attribute NoOfColumns and if so, returns false. If both are in range, the method returns true.
Return values	Boolean	
CheckSquareValid (private)		
Parameters	SquareReference : Int StartSquare : Boolean	Used to test if the SquareReference parameter is a valid Square choice.
Return values	Boolean	The StartSquare parameter is used to check when the location of a piece to move from is being used to check when the location of a piece to move to, otherwise it is passed as false. The method checks to confirm if the location is valid to check when the player is selecting a piece to (a 'move to' check). The method firstly uses the CheckSquareInBounds method to confirm that the square is within the bounds of the board and returns true if it is. If not, it returns false. The method then gets the piece at the location specified by the SquareReference parameter. If the piece is not at the location, it returns false because the player has selected an invalid location. If the StartSquare parameter is true, the method instead returns true because it is a 'move from' check and the location is a blank square. If there is a piece already at the location, the method checks to confirm if it belongs to the current player. If it does and this is a 'move from' check, the method returns true. If this is a 'move to' check, it returns false because the player is attempting to place a piece onto one of their own pieces. If the piece does not belong to the current player, the method returns false. If the player is trying to select an opponent's piece for a 'move to' check, the method returns false because the player is attempting to take an opponent's piece.
CreateBoard (private)		
Parameters	n/a	Uses nested iteration using the NoOfColumns attributes to populate the board with pieces. Player 1's Kotla is placed in the top left corner. Player 2's Kotla is placed in the middle left corner. The remaining locations are filled with empty objects.
Return values	n/a	



CreateChowkidarMoveOption (private)		
Parameters	Direction : Int	Instantiates a new MoveOption method uses the Direction parameter Move objects – one for each value option.
Return values	NewMoveOption : MoveOption	
		<p>The first new Move parameter from starting location to finishing location Move parameter is the number starting location to finishing location to the starting location.</p> <p>A Direction of 1 moves down the board Direction of -1 moves up the board Move object is added to the chowkidar object which is then returned.</p> <p>See pre-release document for valid move positions (shown from page 10)</p>
CreateCuirassierMoveOption (private)		
Parameters	Direction : Int	Instantiates a new MoveOption method uses the Direction parameter Move objects – one for each value option.
Return values	NewMoveOption : MoveOption	
		<p>The first new Move parameter from starting location to finishing location Move parameter is the number starting location to finishing location to the starting location.</p> <p>A Direction of 1 moves down the board Direction of -1 moves up the board Move object is added to the cuirassier object which is then returned.</p> <p>See pre-release document for valid move positions (shown from page 10)</p>
CreateFaujdarMoveOption (private)		
Parameters	Direction : Int	Instantiates a new MoveOption method uses the Direction parameter Move objects – one for each value option.
Return values	NewMoveOption : MoveOption	
		<p>The first new Move parameter from starting location to finishing location Move parameter is the number starting location to finishing location to the starting location.</p> <p>A Direction of 1 moves down the board Direction of -1 moves up the board Move object is added to the faujdar object which is then returned.</p> <p>See pre-release document for valid move positions (shown from page 10)</p>

CreateJazairMoveOption (private)		
Parameters	Direction : Int	Instantiates a new MoveOption method uses the Direction parameter. Move objects – one for each valid option.
Return values	NewMoveOption : MoveOption	
		<p>The first new Move parameter is from starting location to finishing location. Move parameter is the number of starting location to finishing location to the starting location.</p> <p>A Direction of 1 moves down the board. Direction of -1 moves up the board. Move object is added to the jazair which is then returned.</p> <p>See pre-release document for a valid move positions (shown from</p>
CreateRyottMoveOption (private)		
Parameters	Direction : Int	Instantiates a new MoveOption method uses the Direction parameter. Move objects – one for each valid option.
Return values	NewMoveOption : MoveOption	
		<p>The first new Move parameter is from starting location to finishing location. Move parameter is the number of starting location to finishing location to the starting location.</p> <p>A Direction of 1 moves down the board. Direction of -1 moves up the board. Move object is added to the ryott which is then returned.</p> <p>See pre-release document for a valid move positions (shown from</p>
CreateMoveOption (private)		
Parameters	Name : String Direction : Int	Uses selection on the Name parameter associated Create****MoveOption MoveOption from that method.
Return values	MoveOption	
CreateMoveOptionOffer (private)		
Parameters	n/a	Adds the default MoveOption to the default attribute.
Return values	n/a	

COPYRIGHT
PROTECTED

CreateMoveOptions (private)		
Parameters	n/a	Adds the five default MoveOptions to the MoveOptionQueue for each player.
Return values	n/a	This method calls the CreateMoveOptions() method for each player, passing the move Name and Direction. It then adds the five default move options, adding them to the MoveOptionQueue for Player 1 and Player 2's list.
CreatePieces (private)		
Parameters	Pieces : Int	Places the default playing pieces onto the board.
Return values	n/a	The method uses the NoOfPieces to determine how many standard playing pieces to place on the board. Player 1 pieces are on row 2 and Player 2 pieces are on the penultimate row. Pieces are given a color and a symbol which player they belong to, then their symbol on the board. Player 1 pieces are given the symbol '1'. Player 2 pieces are given the symbol '2' using an escape character to avoid conflict with the board symbol of '1' and Player 2 Mirza.
DisplayBoard (private)		
Parameters	n/a	Iterates through the Board list to display the board.
Return values	n/a	The method works by using the following steps:
		<ul style="list-style-type: none"> • Iterate through to the number of rows and a space character. • Iterate through to the number of columns and a space character. • Use nested iteration to print the board. For each square on the board, if there is a piece in the square the piece symbol is printed, otherwise a blank space is printed. • Print a final '\n' symbol at the end of each row. • Iterate through to the number of rows and a space character.
DisplayFinalResult (private)		
Parameters	n/a	The winner of the game is the player who has the highest score when this method is called. The scores are calculated using the GetScore() method.
Return values	n/a	If Player 1 has a higher score than Player 2, the Player 1 name concatenated with 'is the winner' is printed. If Player 2 has a higher score than Player 1, the Player 2 name concatenated with 'is the winner' is printed. If the scores match, 'Draw!' is printed.


**COPYRIGHT
PROTECTED**



DisplayState (private)		
Parameters	n/a	Used as part of the main menu method to display information a
Return values	n/a	
		<p>The method first calls the DisplayBoard to the screen followed by for a player to choose if they want to move.</p> <p>It then uses the GetPlayerState to get the score and move option queue followed by the current player name.</p>
GetIndexOnBoard (private)		
Parameters	SquareReference : Int	Used to convert a SquareReference to the Board list for the associated
Return values	Integer	
		<p>The method initialises two local variables using DIV to split off the row and column from the SquareReference parameter.</p> <p>1 is subtracted from both variables and then the Row is multiplied by the Board width and added to the Col attribute of the Board list.</p>
GetPointsForOccupancyByPlayer (private)		
Parameters	CurrentPlayer : Player	Used to calculate the total points for the CurrentPlayer.
Return values	ScoreAdjustment : Int	
		<p>The method initialises an integer to 0 and iterates through the Board list to find squares occupied by the current player.</p> <p>The GetPointsForOccupancyByPlayer method is overridden by the Kotla class. Kotla belongs to current player Mirza or a current player Mirza occupied by a current player Mirza or the opponent player.</p> <p>Points are totaled up in the ScoreAdjustment variable as iteration progresses.</p> <p>This total is then returned.</p>
GetSquareReference (private)		
Parameters	Description : String	Used to get a square reference from the Board list.
Return values	SelectedSquare : Int	
		<p>The method uses the Description to find an appropriate output to the user. If the user enters a start or finish square, the input from the user is casted with Integer.parseInt and stored in a local integer variable before being returned.</p>

**COPYRIGHT
PROTECTED**



PlayGame (public)		
Parameters	n/a	This method is the main game loop using the local Boolean variable <code>isGameRunning</code> .
Return values	n/a	The method firstly displays the board and the current player. It then allows the current player to choose a move from the move option queue or select 9 to move offer.
		If the user selects option 9, the method calls <code>UseMoveOptionOffer()</code> to display the move offer and then displays the current game board. It loops until the user selects a valid move option.
		The method then asks the user to select a square reference. It checks if the <code>StartSquareReference</code> contains a valid square reference like to move. Using the <code>GetSquareReference()</code> method, it checks if the <code>CheckSquaresValid()</code> method returns true. If the user gives a valid location, the method proceeds.
		The method then repeats this process until the <code>FinishSquareReference</code> contains a valid square reference. The player wants to move the piece. It calls the <code>CheckPlayerMove()</code> method to check if the move is legal. If the move is legal, the method proceeds.
		<ul style="list-style-type: none"> • Calculates any points if the move is legal using the <code>CalculatePoints()</code> method and storing it in <code>PointsForMove</code>. • Updates the player score based on the move option used from the <code>ChangeScore()</code> method. • Updates the player queue based on the move option choice to the <code>UpdateQueueAfterMove()</code> method. • Calls the <code>UpdateBoard()</code> method to update the board of pieces based on the <code>StartSquareReference</code> and <code>FinishSquareReference</code>. • Calls the <code>UpdatePlayerScore()</code> method to update the current player score with <code>PointsForMove</code>. • Prints the updated score for the current player on the screen.
		This method does not deal with illegal moves. If the move is not legal, it simply just ignores it and continues with the player turn without informing the user.
		The method then checks which player's turn it is and swaps to the opposing player. It calls the <code>CheckIfGameOver()</code> method to check if the game is over. If the game is over, it prints their Mirza into the opponent's Ke. If the game has not been captured which stops the game.
		After the main game playing loop, the method calls the <code>DisplayState()</code> method to print the current state of the board and then calls the <code>DisplayWinner()</code> method to confirm which player has won.

UseMoveOptionOffer (private)		
Parameters	n/a	Used to place the move from the current offer move into the current player move.
Return values	n/a	
		<p>The method asks the player to select a move from the current offer move from the list using any error handling. It uses an integer variable <code>Random</code> to select a move from the <code>MoveOptionOffer</code> list. It then uses the <code>UpdateMoveOptionOffer</code> method on the <code>CurrentPlayer</code> to select a position move from the <code>MoveOptionOffer</code> list. It then updates the player score using the <code>UpdatePlayerScore</code> method based on the position of the move. It then updates the player score using the <code>UpdatePlayerScore</code> method based on the position of the move.</p> <p>The method then updates the player score using the <code>UpdatePlayerScore</code> method based on the position of the move.</p>
UpdateBoard (private)		
Parameters	StartSquareReference : Int FinishSquareReference : Int	Performs the actual move on the board to another.
Return values	n/a	The method uses the <code>Random</code> list index calculate the <code>StartSquareReference</code> position and subsequently passed as a parameter to be placed at the <code>FinishSquareReference</code> position.
UpdatePlayerScore (private)		
Parameters	StartSquareReference : Int	Calculates the change in which the player has just made.
Return values	n/a	
		<p>The method calls the <code>GetPointsForOccupancy</code> method on the current player to create a <code>Kotlas</code> which are occupied. It then adds the <code>PointsForOccupancy</code> to the <code>PointsForOccupancy</code> list. It then contains the points for that move.</p> <p>The combined total is then used to update the player score using the <code>UpdatePlayerScore</code> method.</p>

COPYRIGHT
PROTECTED

Class: Piece

Identifier / Data		Description
<<constructor>>		
Parameters	T : String B : Player P : Int S : String	Initialises the following protected <ul style="list-style-type: none"> • TypeOfPiece from parameter T • BelongsTo from parameter B • PointsIfCaptured from parameter P • Symbol from parameter S
Return values	n/a	
GetBelongsTo (public)		
Parameters	n/a	Returns the value of the protected
Return values	BelongsTo : Player	
GetPointsIfCaptured (public)		
Parameters	n/a	Returns the value of the protected
Return values	PointsIfCaptured : Int	
GetSymbol (public)		
Parameters	n/a	Returns the value of the protected
Return values	Symbol : String	
GetTypeOfPiece (public)		
Parameters	n/a	Returns the value of the protected
Return values	TypeOfPiece : String	

Class: Square

Identifier		Description
<<constructor>>		
Parameters	n/a	Initialises the following protected
Return values	n/a	<ul style="list-style-type: none"> • PieceInSquare to null • BelongsTo to null • Symbol to ' '
ContainsKotla (public) <<virtual>>		
Parameters	n/a	If the Symbol attribute is a 'K' or
Return values	Boolean	to confirm that there is a Kotla piece returns false.
GetBelongsTo (public) <<virtual>>		
Parameters	n/a	Returns the value of the protected
Return values	BelongsTo : Player	
GetPieceInSquare (public) <<virtual>>		
Parameters	n/a	Returns the value of the protected
Return values	PieceInSquare : Piece	

INSPECTION COPY

COPYRIGHT
PROTECTED



GetPointsForOccupancy (public) <<virtual>>		
Parameters	CurrentPlayer : Player	Base class method for the GetPointsForOccupancy method in the Kotla class to override. If the method was not overridden, it returns zero points.
Return values	Integer	
GetSymbol (public) <<virtual>>		
Parameters	n/a	Return the value of the protected attribute Symbol.
Return values	Symbol : String	
RemovePiece (public)		
Parameters		Used for removing a piece from the board.
Return values	PieceToReturn : Piece	The method makes a temporary attribute PieceInSquare in a local variable, sets the attribute to null, then sets the attribute to null to return the variable PieceToReturn.
SetPiece (public) <<virtual>>		
Parameters	P : Piece	Assigns the P parameter to the PieceInSquare attribute.
Return values	n/a	

Class: Kotla (inherits from Square)

Identifier / Data		Description
<<constructor>>		
Parameters	P : Player	Initialises the following parent attributes: <ul style="list-style-type: none">• BelongsTo from parameter P• Symbol from parameter S
Return values	n/a	
GetPointsForOccupancy (public) <<override>>		
Parameters	CurrentPlayer : Player	Overrides the GetPointsForOccupancy method in the base class to return the score for the piece occupied. The method checks first to see if the piece is in the square. If there is not, the method returns zero points. If there is a piece in the Kotla square, the method checks to see if the Kotla square belongs to the CurrentPlayer passed in as a parameter. If the piece in the Kotla is either a Mirza or a standard piece, the piece is also owned by the CurrentPlayer, then the method returns 5. If the Kotla square belongs to the CurrentPlayer, there is no Mirza or standard piece, the method returns zero points. If the Kotla square belongs to the CurrentPlayer, the piece in it is either a Mirza or a standard piece, the method returns zero points.
Return values	Integer	

**COPYRIGHT
PROTECTED**



Class: MoveOption

Identifier / Data		Description
<<constructor>>		
Parameters	N : String	Initialises the following protected members: • PossibleMoves to a new list.
Return values	n/a	
AddToPossibleMoves (public)		
Parameters	M : MoveOption	Adds the M parameter to the PossibleMoves list.
Return values	n/a	
CheckIfTheMoveIsValid (public)		
Parameters	StartSquareReference : Int FinishSquareReference : Int	Used to check if the start and finish squares by the player are valid squares on the board. The method initialises four variables: StartRow, StartColumn, FinishRow and FinishColumn. The method then uses StartRow and MOD to split the StartSquareReference into StartRow and StartColumn. The same techniques to split the FinishSquareReference into FinishRow and FinishColumn from the FinishSquareReference parameter. The method then iterates through the PossibleMoves list checking if the StartColumn and FinishColumn combination represent a valid move between possible positions a piece can move to.
Return values	Boolean	
GetName (public)		
Parameters	n/a	Returns the value of the Name property.
Return values	Name : String	

Class: Move

Identifier / Data		Description
<<constructor>>		
Parameters	R : Int C : Int	Initialises the following protected members: <ul style="list-style-type: none">• RowChange from parameter R• ColumnChange from parameter C
Return values	n/a	
GetColumnChange (public)		
Parameters	n/a	Returns the value of the protected ColumnChange property.
Return values	ColumnChange : Int	
GetRowChange (public)		
Parameters	n/a	Returns the value of the protected RowChange property.
Return values	RowChange : Int	

INSPECTION COPY

COPYRIGHT
PROTECTED



Class: MoveOptionQueue

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

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the Queue property to an empty MoveOption list.
Return values	n/a	
Add (public)		
Parameters	NewMoveOption : MoveOption	Adds the NewMoveOption to the Queue list.
Return values	n/a	
GetMoveOptionInPosition (public)		
Parameters	Pos : Int	Returns the MoveOption at the specified position in the Queue list.
Return values	MoveOption	
GetQueueAsString (public)		
Parameters	n/a	Initialises a local empty QueueAsString and assigns it the value of 1.
Return values	QueueAsString : String	The method then iterates through the Queue list concatenating the Count property of each MoveOption (using the GetCount() method), incrementing the index in each loop.
		The method then returns the QueueAsString variable.
MoveItemToBack (public)		
Parameters	Position : Int	Used for moving a MoveOption from the Queue list.
Return values	n/a	The method makes a temporary MoveOption at the index Position.
		The method then uses RemoveAt() on the Queue list to remove the item at index Position.
		It then appends the temporary MoveOption back into the Queue list with the effect of placing it at the end of the list.
Replace (public)		
Parameters	Position : Int NewMoveOption : MoveOption	Replaces the MoveOption at the specified index in the Queue list with the NewMoveOption parameter.
Return values	n/a	

INSPECTION COPY

COPYRIGHT
PROTECTED



Class: Player

Identifier / Data		Description
<<constructor>>		
Parameters	N : String D : Int	Initialises the following parameters: • Score to 100 • Name from parameter N • Direction from parameter D
Return values	n/a	
AddToMoveOptionQueue (public)		
Parameters	MoveOption : MoveOption	Adds the NewMoveOption Queue attribute.
Return values	n/a	
ChangeScore (public)		
Parameters	Amount : Int	Increments the protected Amount parameter.
Return values	n/a	
CheckPlayerMove (public)		
Parameters	Pos : Int StartSquareReference : Int FinishSquareReference : Int	Used to check if a move is valid using the CheckIfThereIsAMoveTo parameter. The method creates a temporary move selected from the parameter. The method then passes the temporary move and FinishSquareReference to CheckIfThereIsAMoveTo. The references represent the selected move option.
Return values	Boolean	
GetDirection (public)		
Parameters	n/a	Returns the value of the Direction parameter.
Return values	Direction : Int	
GetName (public)		
Parameters	n/a	Returns the value of the Name parameter.
Return values	Name : String	
GetPlayerStateAsString (public)		
Parameters	n/a	Used to expose the GetQueue method of the MoveOptionQueue class through the player.
Return values	String	The method returns a comma-separated string of the MoveOptionQueue attribute and the player's score using the GetScore method.
GetScore (public)		
Parameters	n/a	Returns the value of the Score parameter.
Return values	Score : Int	

INSPECTION COPY

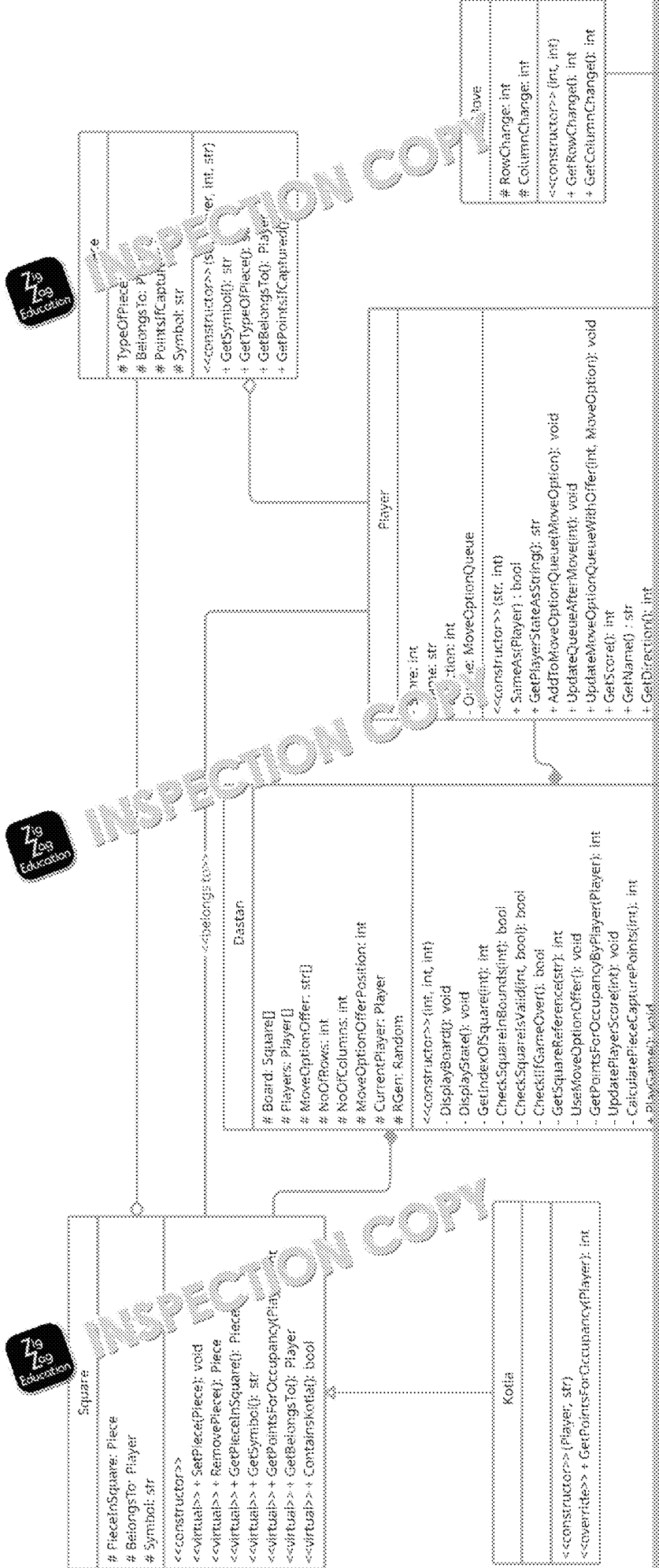
COPYRIGHT
PROTECTED



SameAs (public)		
Parameters	APlayer : Player	Used to check if the APlayer is the same as this player object.
Return values	Boolean	
		The method first checks if the APlayer object has been passed as null. If it is null, it returns false. If not, the method compares the APlayer parameter with the name of the player. If they match, the method returns true, otherwise it returns false.
UpdateMoveOptionQueueWithOffer (public)		
Parameters	Position : Int NewMoveOption : MoveOption	Used to expose the RegisterMoveOptionQueue class to the player.
Return values	n/a	
		The method calls the RegisterMoveOptionQueue class, passing the Position and NewMoveOption parameters. This will register the index of Position with the NewMoveOption parameter.
UpdateQueueAfterMove (public)		
Parameters	Position : Int	Used to expose the MoveOptionQueue class to the player.
Return values	n/a	
		The method calls the MoveOptionQueue class, passing the Position parameter minus one to make it zero-based. It then moves the move option at that index to the back of the queue.

**COPYRIGHT
PROTECTED**





COPYRIGHT
PROTECTED



INSPECTION COPY

Dastan

Exam-style Questions

These questions refer to the **Preliminary Material** and the **Sketch** but **do not** require any additional programming.



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 the loop.
- (a) State the time complexity of this loop.

.....

- (b) Explain the efficiency of this time complexity and how well it scales.

.....

.....

.....



- 2 This question refers to the entire pre-release code.
- Throughout the code there are many string literals such as 'mirza', 'jazz', 'some others'.

- (a) Describe one problem that could occur due to the use of string literals.

.....

.....

.....

- (b) Describe one possible solution to this problem.

.....

.....

.....

INSPECTION COPY

**COPYRIGHT
PROTECTED**



- 3 This question refers to the private method `GetPointsForOccupancyBy`. Explain how polymorphism is used, and why it is useful, when calculating in this method.

.....

.....

.....

.....

.....



- 4 This question refers to the `Main` static method that is executed at the start of the program. When `ThisGame` is instantiated, currently the arguments 6, 6, 4 are passed to the `CreateBoard` method.

	1	2	3	4	5	6	7
1			K1				
2			! !	! !	! !	! !	
3							
4							
5							
6							
7			"	"	"	"	"
8				k2			

- (a) The figure above shows how the board appears if these arguments are passed. Explain why player 1's Kotla and Mirza appear in column 3 rather than column 4, and player 2's as per the image above.



.....

.....

.....

- (b) Describe how the code for the `CreateBoard` method of the `Dastan` class works so that where there are an odd number of columns, then the Kotla is in the central column but when there are an even number it will remain at the left.

.....

.....

.....

.....

.....



**COPYRIGHT
PROTECTED**



- 5 This game refers to the private methods `CreateRyottMoveOption`, `CreateFaujdarMoveOption`, `CreateJazairMoveOption`, `CreateCuirassierMoveOption` and `CreateChowkidarMoveOption`.

Currently the methods take a `Direction` parameter which changes between 0 and 360 degrees depending on whose turn it is. Across the methods there is a lot of repeated use of the `Direction` parameter which always gets multiplied by any non-zero parameter to get the final direction.

Without suggesting any specific code, describe an alternative logic that could be used to calculate the final direction by modifying the `Direction` parameter by modifying the `addToMoveOptionQueue` and `updateMoveOptionQueue` 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 stack.



- (b) Currently this method uses a list to store the queue data structure. Modify the method to use an array to implement a circular queue with five elements.

You should not write any actual code for this question but refer to the method signature and create any algorithms using structured/descriptive notation. Alternatively, you may produce an annotated diagram.



**COPYRIGHT
PROTECTED**



- 7 This question refers to the method `GetIndexOfSquare` in the `Dastan` class. Explain how the private method `GetIndexOfSquare` works.

.....

.....

.....

.....



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

(a) Explain how the board could be represented as a two-dimensional array.

.....

.....

.....

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

.....

.....



- 9 It would be possible to create a save game file for `Dastan`. At the start of the game, the file would contain metadata.

Explain the purpose of metadata and give one example of metadata that would be stored in the file for `Dastan`.

.....

.....

.....

.....



**COPYRIGHT
PROTECTED**



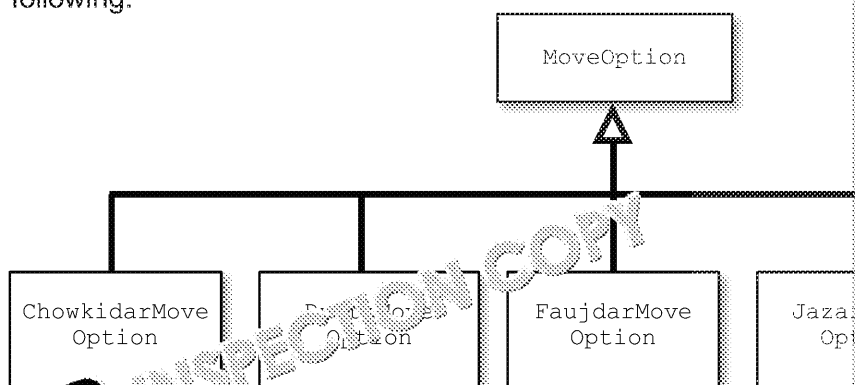
10 This question refers to the `CreateMoveOptions`, `CreateMoveOption`, `CreateChowkidarMoveOption`, `CreateRyottMoveOption`, `CreateFaujd`, `CreateJazairMoveOption` and `CreateCuirassierMoveOption` methods in the `MoveOption` class.

- (a) Currently the `MoveOption` class holds the details for whichever method generated/populated by one of the `CreateChowkidarMoveOption`, `CreateRyottMoveOption`, `CreateFaujd`, `CreateJazairMoveOption` and `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 `base()`. Explain the purpose of this call.



- (b) The method `GetPointsForOccupancy` in the `Kotla` class has a different signature than the method with the same name in the parent class. State the name of the method in the parent class.

COPYRIGHT
PROTECTED



- 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 is about the constructor of the `Piece` class and the `SetPiece` method of the `Square` class.

Both methods take a parameter *P* which is unclear. Explain why variables need meaningful names.

.....

.....

.....

- 14 This question is about access levels for attributes and methods and related concepts.

- (a) The `Piece` class has four protected attributes. What does the word 'protected' mean in this context?

.....

.....

.....

**COPYRIGHT
PROTECTED**



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

.....

.....

(c) There is one final level of access for attributes and methods which means 'private'. What does this mean?

.....

.....

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

.....

.....

.....

.....

.....

15 This question refers to the CheckSquare method of the Data class.

(a) This method uses integer division. Explain the difference between integer and floating point division.

.....

.....

.....

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

.....

.....

**COPYRIGHT
PROTECTED**

END OF QUESTIONS



Dastan

Exam-style Questions

These questions refer to the **Preliminary Material** and the **Source Code** but **do not** require any additional programming.



TOTAL MARKS: 60

- This question refers to the `PlayGame` method in the `Dastan` class.
The method contains a nested loop with multiple while loops inside the loop.
 - State the time complexity of this loop.
 - Explain the efficiency of this time complexity and how well it scales.
- This question refers to the entire pre-release code.
Throughout the code there are many string literals such as 'mirza', 'jaz', and some others.
 - Describe one problem that could occur due to the use of string literals.
 - Describe one possible solution to this problem.
- This question refers to the private method `GetPointsForOccupancyByPlayer`.
Explain how polymorphism is used, and why it is useful, when calculating points in this method.
- This question refers to the `Main` static method that is executed at the start of the program.
When `ThisGame` is instantiated, currently the arguments 6, 6, 4 are passed to the `CreateBoard` method.

	1	2	3	4	5	6	7
1			K1				
2			!	!	!	!	
3							
4							
5							
6							
7			"	"	"	"	
8				k2			

- The image above shows how the board appears if these arguments are passed to the `CreateBoard` method. Explain why player one's Kotla and Mirza appear in column 3 rather than column 4, opposite player two's as per the image above.
- Describe how the code for the `CreateBoard` method of the `Dastan` class should be modified so that where there are an odd number of columns, then the Kotla and Mirza appear in the central column but when there are an even number it will remain in the central column.

INSPECTION COPY

**COPYRIGHT
PROTECTED**



- 5 This game refers to the private methods `CreateRyottMoveOption`, `CreateFaujdarMoveOption`, `CreateJazairMoveOption`, `CreateCuirassierMoveOption` and `CreateChowkidarMoveOption`.

Currently the methods take a `Direction` parameter which changes between 0 to 4 depending on whose turn it is. Across the methods there is a lot of repeated use of the `Direction` parameter which always gets multiplied by any non-zero parameter to get the correct direction.

Without suggesting any specific code, describe an alternative logic that could be used to modify the `Direction` parameter by modifying the `addToMoveOptionQueue`, `updateMoveOptionQueue` and `removeMoveOption` methods of the `Player` class.

- 6 This question refers to the `MoveOptionQueue` class.

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

- Explain why a queue is a more suitable data structure than a stack.
- Currently this method uses a list to store the queue data structure. Modify the code to use an array to implement a circular queue with five elements.

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

- 7 This question refers to the method `GetIndexOfSquare` in the `Dastan` class. Explain how the private method `GetIndexOfSquare` works.

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

- Explain how the board could be represented as a two-dimensional array.
- State one reason why an array is more appropriate to store the board data.

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

Explain the purpose of metadata and give one example of metadata that could be stored in the Dastan class.

- 10 This question refers to the `CreateMoveOptions`, `CreateMoveOption`, `CreateChowkidarMoveOption`, `CreateRyottMoveOption`, `CreateFaujdarMoveOption`, `CreateJazairMoveOption` and `CreateCuirassierMoveOption` methods in the `MoveOption` class.

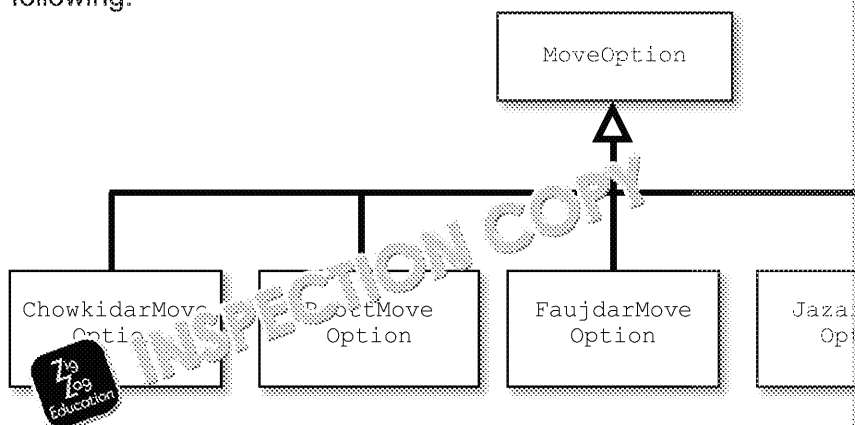
- Currently the `MoveOption` class holds the details for whichever move is generated/processed by one of the `CreateChowkidarMoveOption`, `CreateRyottMoveOption`, `CreateFaujdarMoveOption`, `CreateJazairMoveOption` and `CreateCuirassierMoveOption` methods in the `Dastan` class.

Explain why this is NOT polymorphism.

**COPYRIGHT
PROTECTED**



- (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.
- The constructor includes a call using **base()**. Explain the purpose.
 - The method **GetPointsForOccupancy** in the Kotla class has a difference from the method with the same name in the parent class. State the name of the difference.
 - Explain what the OOP technique *overloading* is used for.

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

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

- 13 This question refers to the constructor of the **Piece** class and the **SetPiece** method.

Both methods take a parameter *P* which is unclear. Explain why variable names are not meaningful names.

- 14 This question is about access levels for attributes and methods and references.

- The **Piece** class has four protected attributes; what does the word 'protected' mean in this context?
- The **Piece** class has four public methods; what does the word 'public' mean in this context?
- There is one final level of access for attributes and methods which is not mentioned. What is it?
- Why is it important to have access modifiers such as private, protected, and public for methods and attributes in OOP?

- 15 This question refers to the **CheckSquareInBounds** method of the **DaySquare** class.


- This method uses integer division; explain the difference between integer and floating point division.
- This method returns a Boolean value. Describe the meaning of Boolean values.

END OF QUESTIONS

Dastan

Programming Tasks

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

Note that a  may make any change or additional code changes that you deemed appropriate, ensuring that it is clear where in the Skeleton Program those changes

Task 1

This question refers to the Dastan class.

Introduce new functionality at the point at which both players are instantiated with custom names set by the users. Ensure that players cannot both have the same name. Replace the two lines that currently create the players with a single call to a new method `CreateCustomPlayers`.

What you need to do

Task 1

Create a new method `CreateCustomPlayers` in the Dastan class. Allow the user to enter custom names for each player. Add checks in your code to ensure that two players cannot have the same custom name.

Allow the first player to enter any name they like, then repeatedly ask the user for a 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 second player name. When prompted, enter 'Tom' again and then at the next prompt, enter 'Vic'.
- show the game using one of the custom names to address the players.

Evidence that you need to provide

- PROGRAM SOURCE CODE showing creation of a new `CreateCustomPlayers` method in the Dastan class.
- SCREEN CAPTURE(S) showing the required test.

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 2

This question refers to the `CreateMoveOptionOffer`, `CreateMoveOption` methods and creation of a new method `CreateFarisMoveOption` in the `Dastan` class.

Develop a new move option called a 'Faris' (Knight). The Faris move option in chess – either two squares forward/backwards and one square left/right or one square left/right and one square forward/backwards. You should demonstrate the use of the `Faris` parameter.

What you need to do:

Task 1

- Add new functionality into the `CreateMoveOptionOffer` & `CreateMoveOption` methods to perform a Faris move.
- Modify the `CreateMoveOptions` method to add the Faris after the Ryott for both players.
- 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 the game, showing both players making legal Faris moves.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `CreateMoveOptionOffer`, `CreateMoveOption` and `CreateMoveOptions` methods
- PROGRAM SOURCE CODE showing a new method `CreateFarisMoveOption`
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



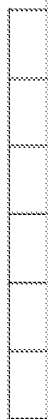
Task 3

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

What you need to do

Task 1

- Add new functionality into the `CreateMoveOptionOffer`, `CreateMoveOption` and `CreateMoveOptions` methods to perform a Sarukh move.
- Modify the `CreateMoveOptions` method to add the Sarukh after the Ryott for both players.
- 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 two turns showing both players making legal Sarukh moves.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `CreateMoveOptionOffer`, `CreateMoveOption` and `CreateMoveOptions` methods
- PROGRAM SOURCE CODE showing a new method `CreateSarukhMoveOption`
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 4

This question refers to the `PlayGame` method in the `Dastan` class and create `AwardWafr` in the `Dastan` class, and `GetWafrAwarded` and `SetWafrAwarded` attribute `WafrAwarded` in the `Player` class.

Create a 'Wafr' award (Abundance) which can be awarded to either player on a 25% chance of being awarded to a player on their turn. On receipt of the option of ANY move from their queue rather than just being able to select a move. The Wafr award removes the move cost for the move the player selects for the turn.

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

What you need to do

Task 1

- Create a new method in the `Dastan` class called `AwardWafr`. This method should return a 25% chance of returning true.
- Add a new private attribute to the `Player` class called `WafrAwarded`. Add the appropriate getter/setter methods for this attribute.

Task 2

Update the `PlayGame` method in the `Dastan` class to call the new `AwardWafr` method. If the player hasn't already been awarded a Wafr, print out a message saying 'You have been awarded a Wafr. You can select any move from your queue for free this turn.' Adjust the input range in the queue to be 0 to 9. Ensure that there is no score adjustment for the player who receives the Wafr. 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 showing changes made to the `PlayGame` method in the `Dastan` class, creation of a new method `AwardWafr` in the `Dastan` class
- PROGRAM SOURCE CODE showing changes made to the `Player` class to add the attribute `WafrAwarded`, `GetWafrAwarded`, `SetWafrAwarded` together with one new method
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 5

This question refers to the `PlayGame` method of the `Dastan` class and the `GetJustQueueAsString` 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 the 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 could continue as normal.

What you need to do



Task 1

Create a new method in the `Player` class called `GetJustQueueAsString` in the `GetQueueAsString` method to return a string version of just the player's queue.

Task 2

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

(Hint: You can check the current player using the `SameAs` method and the `Subtract` method. Subtract 5 from the current player score and display the game state again. Then it's the opponent's 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 the current player's score reducing by 5 points.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method in the `Dastan` class
- PROGRAM SOURCE CODE showing new method `GetJustQueueAsString` in the `Player` class
- SCREEN CAPTURE(S) showing the result of the test



INSPECTION COPY

COPYRIGHT
PROTECTED



Task 6

This question refers to the `PlayGame` method together with the modification of the `UseMoveOptionOffer` methods and creation of a new method `GetValidInt`.

Currently the game has a number of areas where it does not handle errors or error handling into the `PlayGame`, `GetSquareReference`, and `UserMoveOptions` prevent unhandled exceptions from occurring if the user inputs data in an invalid way. The user should be prompted to re-enter their input, until it is valid.

Note: There is no need to check that the square contains a player piece or that a player should not have a wasted turn if the move is invalid, the purpose of this task is to prevent the program from crashing.

What you need to do

Task 1

Create a new private method called `GetValidInt` in the `Dastan` class which returns a valid integer. If the input is invalid, allow the user to keep trying again without crashing.

Task 2

Modify the `GetSquareReference` method to use the new `GetValidInt` method to get a valid input. Add an error message if the user enters an invalid square.

Task 3

Modify the `UseMoveOptions` method to use the new `GetValidInt` method to get a valid input and test to ensure 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 see the help message. Then choose move 1.
- For player one, enter a square of 19 and show the error message.
- For player two, select option 9 to take the offer move and choose player one to move.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `GetSquareReference` method.
- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method.
- PROGRAM SOURCE CODE showing changes made to the `UseMoveOptions` method.
- PROGRAM SOURCE CODE showing the creation of new `GetValidInt` method.
- SCREEN CAPTURE(S) showing the required test results.

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 7

This question refers to the `PlayGame` and `UseMoveOptionOffer` methods, the creation of a new attribute `ChoiceOptionsLeft` along with accessor and mutator methods `GetChoiceOptionsLeft` and `DecreaseChoiceOptionsLeft` in the `Player` class.

Currently a player can repeatedly select option 9 from the main game playing menu with new move options. Introduce a limit so that a player can only 'accept' the offer menu three times in a game. Ensure that a player accepts the offer, advise them they have left and remove the offer menu for that player once they have used it.

What you need to do

Task 1

Modify the `Player` class to introduce a new private attribute called `ChoiceOptionsLeft`.

- Initialise `ChoiceOptionsLeft` to 3.
- Create a new accessor method called `GetChoiceOptionsLeft` which returns the `ChoiceOptionsLeft` attribute.
- Create a new mutator method called `DecreaseChoiceOptionsLeft` which decreases the `ChoiceOptionsLeft` attribute.

Task 2

Modify the `PlayGame` method to test the number of options the player has left during the game.

- Modify the `PlayGame` method so that if the player has used up all their options, they will be advised that no more options are available to the player.
- Modify the `UseMoveOptionOffer` method so that when a move option is selected, the number of options available to them decreases by one. Advise the player of the choices they have left.

Task 3

Test that the changes you have made work:

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

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `UseMoveOptionOffer` method in the `Dastan` class
- PROGRAM SOURCE CODE showing changes made to the `Player` class
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Task 8

This question refers to the `PlayGame` method of the `Dastan` class and create `ResetQueueBack` in the `MoveOptionQueue` class and `ResetQueueBack`

Introduce a new option that allows a player to undo their last move (after the move and before the next player makes their move), undoing any score gained or returning the game to its previous state. If undoing a move costs a player 5 points, a player can then make an alternative move.

What you need to do



Task 1

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

- Create a new method in the `MoveOptionQueue` class called `ResetQueueBack`. This method should move the last element of the queue back to the original position. The method should take one parameter, `Position`, which is the place to which the queue will be restored.
- Create a new method in the `Player` class called `ResetQueueBackAfterUndo`. This method should call the newly created `ResetQueueBack` method on the `Queue` class. The method should take one parameter, `Position`, which is the position made from the menu.

Task 2

Modify the `PlayGame` method to introduce the new functionality.

- If a move is illegal, store the player score prior to the move.
- After playing the board as a result of the move, give the player the option to undo.
- If they choose to undo then: return the player score to the stored previous points and restore the board and the player's queue back to their previous state.

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 move.
- show the game board after the undo and the score set correctly and then make a new move.

Evidence that you need to provide

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing the creation of new methods `ResetQueueBack` in the `MoveOptionQueue` class
- PROGRAM SOURCE CODE showing the creation of the new method `ResetQueueBackAfterUndo` in the `Player` class
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 9

This question refers to the `PlayGame` method together with the modification of `CreateMoveOption` methods and creation of two new methods, `CreateSahmMoveOption` and `CalculateSahmMove`, in the `Dastan` class – plus a new method, `ChoiceIsSahm`.

It also refers to a new attribute `SahmUsed` in the `Player` class along with `GetSahmStatus` and `SetSahmUsed`, which operate as the accessor and mutator methods for the newly created `SahmUsed` attribute.

Implement a new `Sahm` move option (Arrow). The `Sahm` can only be fired once per turn and is fired instead of a piece moving. A `Sahm` can be fired by any piece at any time. The `Sahm` moves in a straight line forwards from the player, all the way across the board until it reaches an 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` option menu. (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.

	1
1	
2	
3	
4	
5	
6	

What you need to do

Task 1

Add new functionality into the `CreateMoveOptionOffer` and `CreateMoveOption` methods and create a new private `CreateSahmMoveOption` method to perform a `Sahm` move.

- Modify the `CreateMoveOptionOffer` method to offer the new 'Sahm' move option.
- Create the new private `CreateSahmMoveOption` method to allow a piece to fire the `Sahm` and add only one possible new move `Move(0,0)` to the `MoveOptionQueue`.
Note: The move should not actually move the piece anywhere, i.e. it should only be added to the queue.
- Modify the `CreateMoveOption` method to handle `Sahm`.

Task 2

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

- Add a new `SahmUsed` attribute to the `Player` class which is initialised to `false`.
- Create two new methods, `GetSahmStatus` and `SetSahmUsed`, which operate as the accessor and mutator (getter/setter) methods for the newly created `SahmUsed` attribute.
- Create a method `ChoiceIsSahm` method which takes a parameter `choice`. If the chosen is a `Sahm` move, whereupon it returns `True`.

(TASK CONTINUES ON THE NEXT PAGE)

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 3

Modify the `PlayGame` method to test to see if the player has selected a `Sahm` `MoveOptionOffer` menu and if it has already been used. If the selected firing piece should destroy any opponent pieces in a straight line from the firing piece, the player should collect any points from multiple pieces destroyed by the `Sahm` move.

- Modify `PlayGame` to call the new method `ChoicesSahm` and only.
- Create a new private method in the `Dastan` class called `CalculateSahm` calculate the points for a `Sahm` move and destroy the pieces that are in the line from the firing piece.
- Modify `PlayGame` to so that it calls the new method `CalculateSahm` the `Sahm` move and destroys the relevant pieces. It should also call `UpdateScore` 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 square 33 as the 'to' to diagonally move one piece in front of another `Kotla` column.
- select 9 from menu for player two to accept the offer. Choose 1 to choose option 1 to select the `Sahm` move. Choose the piece on square 33 to fire. Show the updated board with the player one pieces removed from the board by player two, but the `Marza` which is safely inside the `Kotla`.
- show the correct adjustment of player two's score.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `CreateMoveOption` methods
- PROGRAM SOURCE CODE showing the creation of new `CreateSahm` `ChoicesSahm` and `CalculateSahmMove` methods
- PROGRAM SOURCE CODE showing changes made to the `Player` class
- SCREEN CAPTURE(S) showing the required test

**COPYRIGHT
PROTECTED**



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 player can only replace one Replacing a piece with a Kotla should use up a player turn and they should turn.

What you need to do



Task 1

Modify the `PlayGame` method in the `Dastan` class to introduce a new option playing menu. Allow the player to select a piece which they would like to replace validation to ensure that the user can only select one of their pieces and if 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 game menu.
- show the user selecting 52 as an invalid value for the new Kotla.
- show the Kotla being placed correctly in square 22, a valid square.

Evidence you need to provide:

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 11

This question refers to the `PlayGame` method together with a new method in the `Dastan` class, additional new methods `ReverseQueue`, `SwapFirstAndLast`, `MoveItemToFront` in the `MoveOptionQueue` class together with new methods `GetMoveOptionQueue`, `ReversePlayerQueue`, `SwapFirstAndLast` and `Player` class.

Introduce a new option 6 to the menu while playing menu. On selecting this sub options for making a move to their move queue using the following menu.

Options
1. Reverse the current player queue
2. Swap the current player queue with the opponent queue
3. Swap the first and last elements in the current player queue
4. Move one of the move options to the front of the current player queue
5. Nothing (make normal move)

Note: Options 1–4 cost 3 points, but the player can choose option 5 for free.

Note: This does not count as the player's turn and the player should still be able to make a move.

What you need to do

Task 1

Modify the `PlayGame` method to introduce the new menu option.

- Modify the `PlayGame` method to add option 6 to the move options menu.
- Create a new private method in the `Dastan` class called `ModifyQueue` to show the player the above menu. Include validation to ensure that the user can only choose from the menu.
- Adjust the score by 3 if options 1–4 are chosen but not if option 5 is chosen.

Task 2

Modify the `MoveOptionQueue` class to add the required methods.

- Create new method `ReverseQueue` to allow the current player's queue to be reversed.
- Create new method `SwapFirstAndLast` to swap the first and last elements in the current player's queue.
- Create new method `MoveItemToFront` to move the item from the end of the current player's queue to the front of the current player's queue. There is no need to validate the item is in the current player's queue.

(TASK CONTINUES ON THE NEXT PAGE)

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 3

Modify the `Player` class to create the required methods.

- Create new methods `ReversePlayerQueue`, `SwapFirstAndLast`, and `ModifyQueueOptions` in the `Player` class to expose the new `MoveQueueOptions` choices/methods.
- Create a new method `ReplaceQueue` to allow the current player's queue to be replaced with the queue passed in as a parameter.
- Create a new method `GetPlayerQueue` which returns the player's current queue. This method should return the newly created `PlayerQueue` object.

Task 4

Test that the changes 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 and the resulting queue on the screen as a result of the change.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method.
- PROGRAM SOURCE CODE for the new `ModifyQueueOptions` method.
- PROGRAM SOURCE CODE showing changes made to the `MoveOptions` method.
- PROGRAM SOURCE CODE showing changes made to the `Player` class.
- SCREENSHOTS showing the required test results.

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 12

This question refers to the creation of a new protected attribute `NoOfPieces`, the `PlayGame` method and creation of two new methods `CheckReincarnation` in the `Dastan` class.

Introduce a new feature whereby if a player manages to get one of their pieces to the back row, the player is given a new piece to place on any unoccupied space on the board. If the player cannot reincarnate pieces that are already dead so they should not be able to reincarnate more than they started with.

What you need to do:

Task 1

Create a new private method in the `Dastan` class called `CountNormalPieces` which returns the number of pieces that the current player has excluding the `Mirza`.

Task 2

- Modify the constructor of the `Dastan` class to store the number of pieces in a protected attribute called `NoOfPieces`.
- Modify the `PlayGame` method of the `Dastan` class to call a new private method `CheckReincarnation` following a legal move.

Task 3

Create a new private method `CheckReincarnation` in the `Dastan` class. This method should take the `FinishSquareReference` for the current player's move. If the player's move is to the opponent's back row (e.g. row 6 for player one) and the player has fewer pieces than they started with then allow the player to reincarnate a piece on the opponent's back row in an empty square. If the square is empty and allow the player to reincarnate if it is not.

Task 4

Test that the changes you have made work:

- add the following four lines of code to the START of the private method `CountNormalPieces` in the `Dastan` class: (be certain to remove this after testing to ensure that the code works)

```
NoOfPieces = 2;  
Board[GetIndexOfSquare(51)].SetPiece(new Piece("piece", "white", 1));  
Board[GetIndexOfSquare(21)].SetPiece(new Piece("piece", "white", 1));  
Board[GetIndexOfSquare(54)].SetPiece(new Piece("piece", "white", 1));
```

- run the skeleton program.
- select a Ryott move for player one, enter a start square of 51 and an end square of 21.
- 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 saying that the square must be empty.
- select a Ryott move for player two, enter a start square of 21 and an end square of 51.
- show player two not receiving a reincarnation message.
- remove any modifications to the `CreatePieces` method after testing the changes. The `CountNormalPieces` method should return to normal.

Evidence you need to provide:

- PROGRAM SOURCE CODE showing the new `CountNormalPieces` method.
- PROGRAM SOURCE CODE showing the new `CheckReincarnation` method.
- PROGRAM SOURCE CODE showing the other code changes to the `Dastan` class.
- SCREEN CAPTURE(S) showing the required test results.

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 13

This question refers to the `PlayGame` method together with modification of the `Dastan` class. Additionally it involves the creation of a new method `GetSquare` class and the creation of a new `Taziz` class (Reinforcement) which

Create a new type of game square, the `Ta'ziz` (similar to the `Kotla`), which is a playing board (or slightly closer to player one if there are an even number of columns). The `Ta'ziz` should use the `NoOfRows` and `NoOfColumns` attributes to ensure the size of the board, regardless of the size of the board. Either player can occupy the `Ta'ziz` with their pieces. Two players cannot both occupy the `Ta'ziz` simultaneously. A second player will remove the first player piece from the board and takes ownership of the `Ta'ziz` on the board. If a player occupies the `Ta'ziz` for two turns by both players, the `Ta'ziz` will be removed from the board. If a player occupies the `Ta'ziz` for two turns by their opponent (entering the `Ta'ziz` is considered a player's first turn), the `Ta'ziz` will have zero cost. This gives a player a zero cost move, but risks sitting in the `Ta'ziz` to get it. Continuing to camp in the `Ta'ziz` after two turns gains no further advantage.

What you need to do

Task 1

Create a new class called `Taziz` which should inherit from the `Square` class.

- Add a new private attribute `CampedTurns` and initialise it to 0.
- Override the `SetPiece` and `RemovePiece` methods from the `Square` class. In the `SetPiece` method, adjust the `Taziz` symbol to an upper case 'X' if player one owns the `Ta'ziz` and a lower case 'x' if player two owns the `Ta'ziz`. You may assume that the player with a piece at the top – player one. When a player piece leaves the `Ta'ziz`, ownership of the `Ta'ziz` will change and the symbol set to a lower case 'x'. Each time the `Ta'ziz` is played, the `CampedTurns` should be reset back to zero.
- Override the new method `GetCampedTwoTurns` from the `Square` class. The `GetCampedTwoTurns` method should check the number of turns the `Ta'ziz` has been camped in the `CampedTurns` attribute and return true if it is = 4.

Task 2

Modify the `CreateBoard` method in the `Dastan` class to place an `Ta'ziz` on the middle of the board with a lower case 'x' symbol when the board is first created.

NOTE: The `Ta'ziz` should be correctly placed on the board even if the size of the board is odd. The `Ta'ziz` should take account of the number of columns and rows.

In the case where there are an even number of rows, the castle should be placed in the middle of the board. Also if there are an even number of columns then it should be slightly closer to player one. For a starting board this will place it on square 33, but it should work for any size of board.

The initial `Ta'ziz` does not belong to either player.

INSPECTION COPY

COPYRIGHT
PROTECTED



(TASK CONTINUES ON THE NEXT PAGE)

Task 3

Modify the `PlayGame` method so that if a move is legal the game should be
been camped in for two full turns and, if so, give the selected move to the p

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
with 1 attacking it using player two.
- after both players have had two turns, show a move option by player

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame`
- PROGRAM SOURCE CODE showing changes made to the `Created`
- PROGRAM SOURCE CODE showing the new `GetCampedTwoTurn`
`Square` class
- PROGRAM SOURCE CODE showing the new `Taziz` class
- SCREEN CAPTURE(S) showing the required output

INSPECTION COPY



COPYRIGHT
PROTECTED



Task 14

This question refers to the `PlayGame` method together with creation of a new `WeatherEventOccurs` method in the `Dastan` class. Additionally it involves `WeatherEvent` with the methods `CountDownComplete`, `SetWeatherLocation` and `GetWeatherLocation`.

The Weather Event has a 50% chance of appearing in any turn and can appear in any space on the board. On appearing on the board, both players are given a `WeatherEvent`. A `WeatherEvent` will destroy a player's piece on the same column as the `WeatherEvent`. After two turns for each player, the `WeatherEvent` strikes and any piece from that column is destroyed, including the `Kotla`.

Note: A `WeatherEvent` can only occur if a `WeatherEvent` is not already occurring.

What you need to do

Task 1

Create a new class `WeatherEvent` which should include new methods `CountDownComplete`, `SetWeatherLocation` and `GetWeatherLocation`. On instantiation, the `WeatherEvent` should have a countdown to count the number of game turns before the event occurs. Create a test to see if the countdown has expired. The `SetWeatherLocation` and `GetWeatherLocation` should set and get the location of the `WeatherEvent` on the board. Suitable output should be shown each turn to indicate how long until the `WeatherEvent` will occur.

Task 2

Create a new method called `WeatherEventOccurs` in the `Dastan` class which will create a `WeatherEvent` in a random empty square on the board. When the event has occurred, both players should know.

Task 3

Modify the `PlayGame` method in the `Dastan` class to test to see if a `WeatherEvent` has occurred so if the `WeatherEvent` countdown has expired. If it has, use the `WeatherEventOccurs` method to create a `WeatherEvent` piece (from either player) from the same column as the `WeatherEvent`, and the pieces 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 same column as the weather event over the next two turns.
- show the board during the countdown to the `WeatherEvent` and after the event has occurred showing the pieces from both players removed from the board.

Evidence you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing the new `WeatherEventOccurs` method
- PROGRAM SOURCE CODE showing the new `WeatherEvent` class
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 15

This question refers to the PlayGame method together with modification of and CreatePieces methods and creation of three new private methods, CanPlaceBarrier and CheckManhattanDistance in the Dastan class. Additionally, a new public method ContainsBarrier in the Square class and the creation of a new class Barrier which inherits from Square.

Create a new game piece called a Barrier. On creation of the board each player would like to place their Barrier on the board. The Barrier is 3 squares wide and 3 squares high. A Barrier cannot be placed on a position occupied by a normal piece or an opponent's Barrier. A Barrier can be moved, but it cannot be placed or jumped over by either player.

Some moves, however, do not move in a straight line, for example the Jazz move. The direct move would be through the Barrier which is not allowed. A move around the Barrier, however, is possible which is, therefore, allowed. Use the Manhattan distance to determine if there is a move route possible around the edge of the Barrier.

Manhattan distance is a heuristic function for calculating distance between two points on a grid. In the case of Dastan it is calculated by counting the sum of the number of squares horizontally (or vice versa) between a player starting location and the target location. This is shown in Fig 2 below.

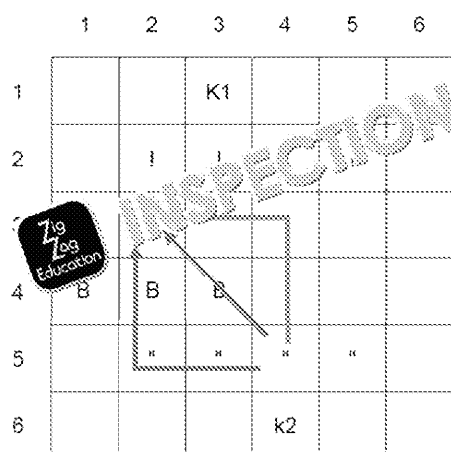


Fig 1

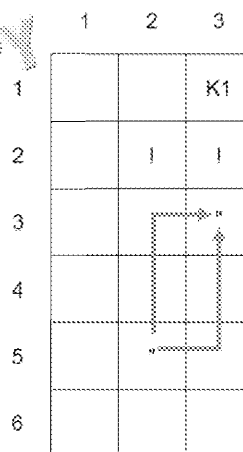


Fig 2

What you need to do

Task 1

- Create a new class Barrier which should inherit from the Square class. It should be assigned an owner and given the symbol of a capital 'B' if it belongs to player one and lowercase 'b' if it belongs to player two.
- Create a new public method ContainsBarrier in the Square class which returns true if a Barrier has been placed in that square.

(TASK CONTINUES ON THE NEXT PAGE)

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 2

- i) Modify the `CheckSquaresValid` method to check if the square being asked where to place the piece is valid. That is, check if the square is within the bounds of the board and that a piece cannot occupy it or attempt to move it.
- ii) Create a new method `CheckBarriersValid` in the `Dastan` class which checks if a Barrier being placed by a player fits within the bounds of the board squares.
- iii) Create a new method called `CreateBarrier` in the `Dastan` class which creates a Barrier onto the board. The Barrier will always be horizontal and the centre square will be the square being asked where to place the Barrier.

Task 3

- i) Create a new method called `CheckManhattanDistance` in the `Dastan` class which calculates the Manhattan distance between two squares. The distance is the number of paths from a starting square reference to a finishing square reference. This is calculated by moving horizontally then down the finishing column and also down the starting column then horizontally to the 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 be used in `CheckPlayerMove` using a logical AND to set the value of the variable `isValidMove`.

Note: This should be used for all moves even if they are too short to potentially be able to go round.

Task 4

Test that the changes 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 `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `CheckSquaresValid` and `CreatePieces` methods in the `Dastan` class
- PROGRAM SOURCE CODE for the new private `CheckBarriersValid` and `CheckManhattanDistance` methods in the `Dastan` class
- PROGRAM SOURCE CODE showing changes made to the `Square` class
- PROGRAM SOURCE CODE showing the new `Barrier` class
- SCREEN CAPTURE(S) showing the required test

**COPYRIGHT
PROTECTED**



Dastan

Programming Tasks (Extension)

Extension 1

Introduce a health scoring system for pieces. Each piece (except the Kotla) has health points. Each time a piece is landed on, it incurs damage, reducing its health points. Each time a piece's health is reduced. When a piece reaches 0 health points, it is removed from the board. Only one piece can attack another at one time. When a piece is being attacked, the attacker's player symbol should be shown on the left of the piece and the target piece's player symbol on 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's starting location and the finishing location as shown in **Fig 1**.

An attack from a move from position 1 in the move queue reduces health by 1 point. An attack from position 3 in the move queue reduces health by 3 points. This shows how far away the opponent is from the attacker. This is the sum of the rows and columns between the attacker and the target. An attack from further away, therefore, incurs a greater level of damage.

Extension 2

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

The 'Throw bomb' option should be available through the MoveOffer menu. If the bomb is thrown to an opponent's square, the opponent takes ownership of the bomb back or moves it to a safe square. Each turn carries a 10% chance of the bomb detonating. If the bomb is thrown to a player's square, the bomb loses ownership from either player and remains at this location until a player moves to the square containing the bomb and is able to throw it. Each turn carries a 10% chance of the bomb detonating. If a player piece is captured while holding the bomb, the ownership of the bomb transfers to the opponent of the square.

INSPECTION COPY

COPYRIGHT
PROTECTED



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'. When the player can select two move options to execute sequentially. The player 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...' reference for move option 1. Both moves must be legal. The program should use error handling for entering illegal references and allow them to re-enter.

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

On entering a legal multi-move, the game should move the selected player piece. 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.

**COPYRIGHT
PROTECTED**



Extension 5

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

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

A piece in 'mole' mode can move around underneath the board using normal movement rules. It can be captured by an opponent piece on the surface of the board. Once the piece is captured, the piece in 'mole' mode, the submenu should change to now give the player the option to move the piece after moving it. If a piece in 'mole' mode resurfaces in a square where an opponent piece is, the current player captures that opponent piece. Once a player resurfaces a 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 deep.

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

Extension 6

Introduce an option to 'preview a move' before making it. Add a new option to the move menu. On selecting this option, a player can select any move from position to position. A valid player piece and a valid square reference is selected. The player is then shown a 'preview' copy of the board which shows an opponent piece on all the squares which the selected move option can move to. The current player's piece is shown on the starting square.

The player should then be given the option to enter in a valid 'move to...' square reference. If a valid 'move to...' square reference is selected, the game should make the move.

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

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

**COPYRIGHT
PROTECTED**



Extension 7

Introduce a new option at the start of the game to allow the players to place different formations prior to the game starting. Players can choose from any of the following options.

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

Once the players have selected their chosen starting positions, the game starts.

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

'Khandaq' Option (Trench)

	1	2
1		
2		
3		
4	"	
5		
6		

'Itifaq' Option (Trench)

	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		

'Khanja' Option (Corner Kick)

Extension 8

Introduce the concept of an 'Al Amlaq' (Giant), which is formed when a player combines their **own** player pieces. A Giant is shown as 's' for player 1 and 'g' for player 2.

Once a Giant has been created by combining a player Mirza with a normal player, it remains as a Giant for the rest of the game.

A Giant can move around the board using the same move options as a normal player. It needs to land within one square (in any direction) of any opposition piece, opponent Kotla and Mirza.

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

INSPECTION COPY

COPYRIGHT
PROTECTED



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 like. A player can only add chainmail to two pieces during the game. The chainmail covers the front and sides of a piece and therefore a piece's symbol doesn't change when it has the chainmail. A piece with chainmail has a forward-facing barrier which means that the piece can only be attacked from the back. An opponent piece can be one square in front of a current player piece, it cannot be adjacent to a piece with chainmail – it must approach the chainmailed piece from either side or behind.

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



Extension 10

Introduce the load and save features to the game. Add new options 'L' and 'S' to the main menu. Selecting these options to load a previously saved game or save the current game.

The load and save submenus should give the user the opportunity to enter a file name. The program should have appropriate error handling to prevent it from attempting to load a file that does not exist or saving to an invalid location. The program should store appropriately separated values to store all of the program data required to rebuild a game. Appropriate error handling should be included when a game is being rebuilt to ensure that the data is all valid within the bounds of the board.

Extension 11

Introduce a new feature to allow the size of the playing board and pieces to be changed. Give the user the option to choose the size of the board. The dimensions should be even; however, appropriate error handling should be included to prevent the board from being larger than 10 x 10. (Appropriate formatting needs introducing on a 10 x 10 board to make the lines line up correctly.)

For boards of 6 to 8 columns wide, ensure that both player Kotlas are placed in the top and bottom rows of the board. A 7 column wide board should have 5 pieces per player. A 6 column 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, which should be placed in either of the Kotla squares at 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		I	I
3			
4			
5			
6			
7			
8		"	"
9			K2

**COPYRIGHT
PROTECTED**



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 and move the piece in square 25 and move to square 31 which is one square forward followed by two squares to the left looking at the board from the point of view of the player.

1
2
3
4
5
6

Extension 13

Introduce the concept of an 'Muraqib' (Meerkat Lookout piece). At the start of the game, player 1 has the opportunity to place their Muraqib on any empty square on the board. The Muraqib is represented by an 'M' symbol and player 2 Muraqib is represented by an 'm' symbol.

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

A Muraqib cannot be captured. If either player lands on the square containing the Muraqib, the Muraqib disappears down into its burrow underneath the board. While it is in its burrow, it belongs to of any threatening moves. When the player piece occupying the square moves 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 player's queue. The Aqrab can only be applied to one piece per player. Once applied, the piece symbol is changed to 'A' for a player 1 piece or 'a' for a player 2 piece. A piece chosen to be an Aqrab is added to the board; however, when it is one square away from an opponent piece (in any direction), the piece becomes paralysed and cannot move. This makes it vulnerable to being captured by the Aqrab itself.

The Aqrab, however, can still be captured by any piece which can move from the square it is on (in any direction). If the Aqrab moves away from a piece which it has paralysed, it is 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 menu. When a player selects the Quantum Mirza option, the player should be asked for the board location of the piece to swap. The piece must belong to the current player. The program should use the board to ensure that a valid piece is selected. The program should then swap the locations of the two pieces. The player turn should then continue as normal.

COPYRIGHT
PROTECTED



Dastan

Exam-style Questions (Mark Scheme)

Q	Suggested Solution	Marks
1	<p>(a) $O(n^2)$</p> <p>(b) 1 mark for each point</p> <ul style="list-style-type: none"> ... is 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 	<p>1 mark</p> <p>3 marks</p>
2	<p>(a) 1 mark for each point</p> <ul style="list-style-type: none"> You may mistype/misspell one of them ... which could mean that the code develops a logic or runtime error <p>(b) 1 mark for each point</p> <ul style="list-style-type: none"> One possible solution would be to define string literals as constants (1 mark) ... resulting in a syntax error with an undefined identifier before running the program, rather than a logic or runtime error 	<p>2 marks</p> <p>2 marks</p>
3	<p>(a) Every square in the board is treated as a square [1 mark] but some of them may be Kotla (which inherit from Square) [1 mark] so they can call the overridden method on the Kotla to polymorphism, meaning the object Square and Kotla can use the same method identifiers but can have a different result or return value.</p>	3 marks
4	<p>(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]</p> <p>(b) Change the calculation for player one [1 mark] to $(\text{NoOfColumns}+1) \text{ DIV } 2$ [1 mark] which will round up for odd numbers [1 mark] but round down for even numbers [1 mark]</p>	<p>2 marks</p> <p>4 marks</p>
5	<p>(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]</p>	3 marks
6	<p>(a) A queue is more appropriate because move options are added to the end of the queue but could not be added to the bottom 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]</p>	2 marks

INSPECTION COPY

COPYRIGHT
PROTECTED



Q	Suggested Solution	Marks
6	<p>(b) A circular queue would need a head variable [1 mark] and a tail variable [1 mark]...</p> <div style="text-align: center;"> </div> <p>... so that when an item is added to the queue, the rear pointer could be incremented or wrapped back around to 0 if it is 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]</p>	4 marks
7	<p>(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]</p>	3 marks
8	<p>(a) One dimension could be the row [1 mark] and the second dimension could be the column [1 mark]</p>	2 marks
	<p>(b) An array is static so the amount of memory used will not change and the board size is fixed so this is appropriate</p>	1 mark
9	<p>(a) Metadata describes the data in a file [1 mark] Possible examples (any sensible answer will do)</p> <ul style="list-style-type: none"> • Board size (resolution) • Number of pieces for each player 	2 marks
10	<p>(a) This is not polymorphism because each of the five methods creates a MoveOption object [1 mark] which is the same class and 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].</p>	2 marks
	<p>(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]</p>	2 marks
11	<p>(a) 1 mark for each point</p> <ul style="list-style-type: none"> • base() is used to refer to the base class object • and call the base constructor 	2 marks
	<p>(b) Overriding</p>	1 mark
	<p>(c) 1 mark for each point</p> <ul style="list-style-type: none"> • To provide multiple implementations • of a method with the same name • selecting which version to run based on the number and type of parameters passed • within the same class definition 	3 marks

COPYRIGHT
PROTECTED



Q	Suggested Solution	Marks
12	a) 1 mark for each point <ul style="list-style-type: none"> 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 correct priority section then they join the queue at the front of the next lower priority or at the back of the queue with a higher priority Items are still taken from the front of the entire queue and can join at the back of the appropriate sub-section 	4 marks
13	(a) 1 mark for each point <ul style="list-style-type: none"> The name describes the purpose of the variable which makes the code easier for programmers to read/understand/follow 	2 marks
14	(a) 1 mark for each point <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> Integer division returns a whole number Floating point division returns a decimal value with a decimal point 	2 marks
	(b) It has two values, true or false	1 mark

COPYRIGHT
PROTECTED

INSPECTION COPY



Dastan

Programming Tasks (Mark Scheme)

Task 1

Coding

- Create a new method `CreateCustomPlayers` which allows the user to enter in names until the names are different. [1 mark]

Example Solution

Modify constructor in Dastan:

```
public Dastan(int R, int C, int NoOfPieces)
{
    //CHANGE
    CreateCustomPlayers(); //Q1
    //CHANGE
    CreateMoveOptions();
}
```

New private method:

```
//CHANGE
private void CreateCustomPlayers() //Q1
{
    Console.WriteLine("Enter in the name for player one: ");
    Players.Add(new Player(Console.ReadLine(), 1));

    bool NameInvalid = true;
    string PlayerTwoName = "";
    while (NameInvalid)
    {
        Console.WriteLine("Enter in the second player name: ");
        PlayerTwoName = Console.ReadLine();
        if (Players[0].GetName() == PlayerTwoName)
        {
            Console.WriteLine("You can't have that name - it's already taken by player 1");
        }
        else
        {
            NameInvalid = false;
        }
    }
    Players.Add(new Player(PlayerTwoName, -1));
}
//END CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Testing:

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

```
Enter in the name for player one:
Tom
Enter in the second player name:
Tom
You can't have that name - it has already been taken by player 1
Enter in the second player name:
Tom
You can't have that name - it has already been taken by player 1
Enter in the second player name:
Victoria
```

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

```
Move option offer: jazair
Tom
Score: 100
Move option queue: 1. ryott 2. chowkidar 3. cuirassier 4. faujdar 5. jazair
Turn: Tom
Choose move option to use from queue (1 to 3) or 9 to take the offer:
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 2

Coding

- Adding a new MoveOptionOffer to the CreateMoveOptionOffer method, and adding the move option to both players in the parameter set correctly. [1 mark]
- Adding a Faris to the CreateMoveOption method, a method which calls the method. [1 mark]
- Create a new method CreateFarisMoveOption which correctly uses the Direction and the possible positions for the Faris. [1 mark]

Example Solution:

Changes to CreateMoveOptionOffer:

```
private void CreateMoveOptionOffer()
{
    //CHANGE
    MoveOptionOffer.Add("faris"); //Q2
    //END CHANGE
```

Changes to CreateMoveOption:

```
private MoveOption CreateMoveOption(string Name, int Direction)
{
    //CHANGE
    if (Name == "faris")
    {
        return CreateFarisMoveOption(Direction); //Q2
    }
    else if (Name == "chowkidar")
    {
        return CreateChowkidarMoveOption(Direction);
    }
    //END CHANGE
```

Code for CreateFarisMoveOption:

```
//CHANGE
private MoveOption CreateFarisMoveOption(int Direction) //Q2
{
    MoveOption NewMoveOption = new MoveOption("faris");
    Move 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(-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(1 * Direction, 2 * Direction);
    NewMoveOption.AddToPossibleMoves(NewMove);
    return NewMoveOption;
}
//END CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to CreateMoveOptions:

```
private void CreateMoveOptions()
{
    Players[0].AddToMoveOptionQueue(CreateMoveOption("ryott")
    //CHANGE
    Players[0].AddToMoveOptionQueue(CreateMoveOption("faris")
    //END CHANGE
    Players[0].AddToMoveOptionQueue(CreateMoveOption("chowk")
    Players[0].AddToMoveOptionQueue(CreateMoveOption("cuira")
    Players[0].AddToMoveOptionQueue(CreateMoveOption("faujd")
    Players[0].AddToMoveOptionQueue(CreateMoveOption("jazai")
    Players[1].AddToMoveOptionQueue(CreateMoveOption("ryott")
    //CHANGE
    Players[1].AddToMoveOptionQueue(CreateMoveOption("faris")
    //END CHANGE
    Players[1].AddToMoveOptionQueue(CreateMoveOption("chowk")
    Players[1].AddToMoveOptionQueue(CreateMoveOption("jazai")
    Players[1].AddToMoveOptionQueue(CreateMoveOption("faujd")
    Players[1].AddToMoveOptionQueue(CreateMoveOption("cuira")
}
```

Testing:

- Displaying the Faris move option correctly in the player one queue and moving a 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. cuirassier 5. faujdar 6. cu

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2
Enter the square containing the piece to move (row number followed by column number):
Enter the square to move to (row number followed by column number): 43
New score: 101

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. faris 3. chowkidar 4. jazair 5. faujdar 6. cu

Turn: Player Two

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 3

Coding

- Adding a new MoveOptionOffer to the CreateMoveOptionOffer method, and adding the move option to both players in the parameter set correctly. [1 mark]
- Adding a Sarukh to the CreateMoveOption method - a move option which calls the method. [1 mark]
- Create a new method CreateSarukhMoveOption, which correctly uses the Direction to return the possible positions for the Sarukh. [1 mark]

Example Solution:

Changes to CreateMoveOptionOffer:

```
private void CreateMoveOptionOffer()
{
    //CHANGE
    MoveOptionOffer.Add("sarukh"); //Q3
    //END CHANGE
```

Changes to CreateMoveOption:

```
private MoveOption CreateMoveOption(string Name, int Direction)
{
    //CHANGE
    if (Name == "sarukh")
    {
        return CreateSarukhMoveOption(Direction); //Q3
    }
    else if (Name == "chowkidar")
    {
        return CreateChowkidarMoveOption(Direction);
    }
    //END CHANGE
```

Code for CreateSarukhMoveOption:

```
//CHANGE
private MoveOption CreateSarukhMoveOption(int Direction) //Q3
{
    MoveOption NewMoveOption = new MoveOption("sarukh");
    Move NewMove = new Move(0, -1 * Direction);
    NewMoveOption.AddToPossibleMoves(NewMove);
    NewMove = new Move(1 * Direction, -1 * Direction);
    NewMoveOption.AddToPossibleMoves(NewMove);
    NewMove = new Move(2 * Direction, 0);
    NewMoveOption.AddToPossibleMoves(NewMove);
    NewMove = new Move(1 * Direction, 1 * Direction);
    NewMoveOption.AddToPossibleMoves(NewMove);
    NewMove = new Move(0, 1 * Direction);
    NewMoveOption.AddToPossibleMoves(NewMove);
    return NewMoveOption;
}
//END CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to CreateMoveOptions:

```
private void CreateMoveOptions()
{
    Players[0].AddToMoveOptionQueue(CreateMoveOption("ryott"))
    //CHANGE
    Players[0].AddToMoveOptionQueue(CreateMoveOption("sarukh"))
    //END CHANGE
    Players[0].AddToMoveOptionQueue(CreateMoveOption("chowkidar"))
    Players[0].AddToMoveOptionQueue(CreateMoveOption("cuirassier"))
    Players[0].AddToMoveOptionQueue(CreateMoveOption("faujdar"))
    Players[0].AddToMoveOptionQueue(CreateMoveOption("jazair"))
    Players[0].AddToMoveOptionQueue(CreateMoveOption("ryott"))
    //CHANGE
    Players[1].AddToMoveOptionQueue(CreateMoveOption("sarukh"))
    //END CHANGE
    Players[1].AddToMoveOptionQueue(CreateMoveOption("chowkidar"))
    Players[1].AddToMoveOptionQueue(CreateMoveOption("jazair"))
    Players[1].AddToMoveOptionQueue(CreateMoveOption("faujdar"))
    Players[1].AddToMoveOptionQueue(CreateMoveOption("cuirassier"))
}
```

Testing:

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

1 2 3 4 5 6

1			K1		
2					
3					
4					
5					
6			k2		

Move option queue: 1. ryott 2. sarukh 3. chowkidar 4. cuirassier 5. faujdar

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2

Enter the square containing the piece to move (row number followed by column number): 42

Enter the square to move to (row number followed by column number): 42

New score: 181

1 2 3 4 5 6

1			K1		
2					
3					
4					
5					
6			k2		

Move option queue: 1. ryott 2. sarukh 3. chowkidar 4. jazair 5. faujdar

Turn: Player Two

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

INSPECTION COPY

COPYRIGHT
PROTECTED



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 WafrAwarded

Example Solution

Changes to Player:

```
class Player
{
    private string Name;
    private int Direction, Score;
    private MoveOptionQueue Queue = new MoveOptionQueue();
    //CHANGE
    private bool WafrAwarded = false; //Q4
    //END CHANGE
```

```
//CHANGE
public bool GetWafrAwarded() //Q4
{
    return WafrAwarded;
}
public void SetWafrAwarded() //Q4
{
    WafrAwarded = true;
}
//END CHANGE
```

Code for AwardWafr:

```
//CHANGE
private bool AwardWafr() //Q4
{
    if (RGen.Next(0, 4) == 0) //25% chance of returning true.
    {
        return true;
    }
    else
    {
        return false;
    }
}
//END CHANGE
```

Changes to PlayGame:

```
public void PlayGame()
{
    bool GameOver = false;
    while (!GameOver)
    {
        DisplayState();
        bool SquareIsValid = false;
        //CHANGE
        int Choice = 0; //Q4
        bool Wafr = false;
        if (AwardWafr() && !CurrentPlayer.GetWafrAwarded())
        {
            Console.WriteLine("You have been offered a Wafr!");
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

        Console.WriteLine("You can select any move from your offer");
        Wafr = true;
        CurrentPlayer.SetWafrAwarded();
    }
    if (Wafr)
    {
        do
        {
            Console.WriteLine("Choose move option to use from offer");
            Choice = Convert.ToInt32(Console.ReadLine());
        }
        while (Choice < 1 || Choice > 5);
    }
    else
    {
        do
        {
            Console.WriteLine("Choose move option to use from offer: ");
            Choice = Convert.ToInt32(Console.ReadLine());
            if (Choice == 9)
            {
                UseMoveOptionOffer();
                DisplayState();
            }
        }
        while (Choice < 1 || Choice > 3);
    }
    //END CHANGE
    int StartSquareReference = 0;
    while (!SquareIsValid)
    {
        StartSquareReference = GetSquareReference("contain");
        SquareIsValid = CheckSquareIsValid(StartSquareReference);
    }
    int FinishSquareReference = 0;
    SquareIsValid = false;
    while (!SquareIsValid)
    {
        FinishSquareReference = GetSquareReference("to move");
        SquareIsValid = CheckSquareIsValid(FinishSquareReference);
    }
    bool MoveLegal = CurrentPlayer.CheckPlayerMove(Choice, FinishSquareReference);
    if (MoveLegal)
    {
        int PointsForPieceCapture = CalculatePieceCapturePoints(Choice, FinishSquareReference);
        //CHANGE
        if (!Wafr) //Q4
        {
            CurrentPlayer.ChangeScore(-(Choice + (2 * (FinishSquareReference - StartSquareReference))));
        }
        else
        {
            Console.WriteLine();
            Console.WriteLine("Using wafr move at zero cost");
        }
    }
    //END CHANGE
}

```

**COPYRIGHT
PROTECTED**



Testing:

- Show player one being awarded a Wafr and selecting a move from position 4 or without incurring a cost. [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. faujdar

Turn: Player One

You have been offered a Wafr!
You can select any move from your queue for free this turn.
Choose move option to use from queue (1 to 5): 4
Enter the square containing the piece to move (row number followed by column number): 21
Enter the square to move to (row number followed by column number): 21

Using Wafr move at zero cost
New score: 105

  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  5.

Turn: Player Two
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 5

Coding

- Change PlayGame to give new menu option 8 and reduction of player score by 5 [1 mark]
- 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 GetJustQueueAsString which calls in GetQueueAsString method in Player. [1 mark]

Example Solution

Changes to PlayGame:

```
//CHANGE
do
{
    //CHANGE
    Console.WriteLine("Choose move option to use from the offer or 8 to spy on your opponent's queue");
    Choice = Convert.ToInt32(Console.ReadLine());
    if (Choice == 8)
    {
        Console.WriteLine("Selecting this option costs 5 points");
        Environment.NewLine();
        Player Opponent;
        if (CurrentPlayer.SameAs(Players[0]))
        {
            Opponent = Players[1];
        }
        else
        {
            Opponent = Players[0];
        }
        Console.WriteLine(Opponent.GetName() + "'s queue");
        Console.WriteLine(Opponent.GetJustQueueAsString());
        Environment.NewLine();
        CurrentPlayer.ChangeScore(-5);
        DisplayState();
    }
    else if (Choice == 9)
    {
        UseMoveOptionOffer();
        DisplayState();
    }
}
//END CHANGE
}
```

Changes to Player:

```
//CHANGE
public string GetJustQueueAsString()
{
    return Queue.GetQueueAsString();
}
//END CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Testing:

- Display new menu option. Player one to select option 8 to view player two's queue

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

Move option offer: jazair

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

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 8 to spy on y
Selecting this option costs 5 score points.

Player Two's queue is:
1. ryott  2. chowkidar  3. jazair  4. faujdar  5. cuirassier

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

Move option offer: jazair

Player One
Score: 95
Move option menu: 1. ryott  2. chowkidar  3. cuirassier  4. faujdar  5. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 8 to spy on y
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 6

Coding

- Create a new method GetValidInt which returns the user input if the user enters a valid integer or in range. It should print out a suitable message and force the user to re-enter a valid integer or in range. [1 mark]
- 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 square input. [1 mark]
- Change UseMoveOptionOffer to use the GetValidInt method for choosing whether to take the offer. It should include a range of 1 to 5 to prevent an invalid queue choice. [1 mark]

Example Solution

Code for GetValidInt:

```
//CHANGE
private int GetValidInt(string MessageToDisplay, int OptionalRange)
{
    bool ValidInput = false;
    int UserInput = -1;
    while (!ValidInput)
    {
        Console.WriteLine(MessageToDisplay);
        try
        {
            UserInput = Convert.ToInt32(Console.ReadLine());
            if (OptionalRange != 0)
            {
                if (UserInput > 0 && UserInput <= OptionalRange)
                {
                    ValidInput = true;
                }
                else
                {
                    Console.WriteLine("Value out of range - try again");
                }
            }
            else
            {
                ValidInput = true; //Condition where range handling not needed
            }
        }
        catch
        {
            Console.WriteLine("Not a valid input - try again");
        }
    }
    return UserInput;
}
//END CHANGE
```

Changes to PlayGame:

```
//CHANGE
Choice = GetValidInt("Choose move option to use or take the offer: "); //Q6

if (Choice == 9)
{
    UseMoveOptionOffer();
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED




```

        DisplayState();
    }
    else if (Choice < 1 || Choice > 3)
    {
        Console.WriteLine("That is not a valid option. Enter 1, 2, 3 or 9 to take the offer:" + Environment.NewLine);
    }
}
while (Choice < 1 || Choice > 3)
{
    int StartSquareReference = 0;
    while (!SquareIsValid(StartSquareReference))
    {
        StartSquareReference = GetSquareReference("continue to enter the start square reference: ");
        SquareIsValid = CheckSquareIsValid(StartSquareReference);
        if (!SquareIsValid)
        {
            Console.WriteLine("You must enter a valid square reference");
        }
    }
    int FinishSquareReference = 0;
    SquareIsValid = false;
    while (!SquareIsValid)
    {
        FinishSquareReference = GetSquareReference("to enter the finish square reference: ");
        SquareIsValid = CheckSquareIsValid(FinishSquareReference);
        if (!SquareIsValid)
        {
            Console.WriteLine("You must enter a valid square reference");
        }
    }
    bool MoveLegal = CurrentPlayer.MovePlayerMove(Choice, FinishSquareReference);
}
//END CHANGE

```

Changes to GetSquareReference:

```

//CHANGE
private int GetSquareReference(string Description) //Q6
{
    int SelectedSquare;
    SelectedSquare = GetValidInt("Enter the square " + Description + " followed by column number: ");
    return SelectedSquare;
}
//END CHANGE

```

Changes to UseMoveOptionOffer:

```

//CHANGE
private void UseMoveOptionOffer() //Q6
{
    int ReplaceChoice;
    ReplaceChoice = GetValidInt("Enter the move option from 1 to 5: ", 5);
    CurrentPlayer.MoveOptionQueueWithOffer(ReplaceChoice);
    CreateMoveOptionOffer(MoveOptionOffer[MoveOptionOfferPosition]);
    CurrentPlayer.ChangeScore(-(10 - (ReplaceChoice * 2)));
    MoveOptionOfferPosition = RGen.Next(0, 5);
}
//END CHANGE

```

**COPYRIGHT
PROTECTED**



Testing:

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

```
  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. faujdar  5. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer:
1
Enter the square containing the piece to move (row number followed by column number):
28
You must enter a valid square
Enter the square containing the piece to move (row number followed by column number):
22
Enter the square to move to (row number followed by column number):
22
You must enter a valid square
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  |  |  |  |  |  |

Move option offer: jazair

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

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
Value out of range - try again.
Choose the move option from your queue to replace (1 to 5):
1
```

INSPECTION COPY

COPYRIGHT
PROTECTED

INSPECTION COPY



Task 7

Coding

- Adding the ChoiceOptionsLeft attribute to Player with getter method. Initial value is 3 [1 mark]
- Create a new method DecreaseChoiceOptionsLeft in Player which decreases the ChoiceOptionsLeft by 1 [1 mark]
- Change PlayGame to test if the player has used all of their move options and if so, display a message and end the game [1 mark]
- Change UseMoveOptionOffer to call DecreaseChoiceOptionsLeft for the chosen move option from the menu AND advise the player how many move options they have left [1 mark]

Example Solution

Changes to Player.cs:

```
class Player
{
    private string Name;
    private int Direction, Score;
    private MoveOptionQueue Queue = new MoveOptionQueue();
    //CHANGE
    private int ChoiceOptionsLeft = 3; //Q7
    //CHANGE
```

```
//CHANGE
public void DecreaseChoiceOptionsLeft() //Q7
{
    ChoiceOptionsLeft--;
}
public int GetChoiceOptionsLeft() //Q7
{
    return ChoiceOptionsLeft;
}
//END CHANGE
```

Changes to PlayGame.cs:

```
do
{
    //CHANGE
    if (CurrentPlayer.GetChoiceOptionsLeft() > 0)
    {
        Console.WriteLine("Choose move option to use from menu and take the offer: ");
    }
    else
    {
        Console.WriteLine("Choose move option to use from menu and take the offer: ");
    }
    Choice = Convert.ToInt32(Console.ReadLine());
    if (CurrentPlayer.GetChoiceOptionsLeft() > 0 && Choice > 0)
    {
        UseMoveOptionOffer(Choice);
        DisplayScore();
    }
    else
    {
        Console.WriteLine("You have no move options left. Game over.");
    }
} while (true);
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to UseMoveOptionOffer:

```
private void UseMoveOptionOffer()
{
    int ReplaceChoice;
    Console.WriteLine("Choose the move option from your queue to replace");
    ReplaceChoice = Convert.ToInt32(Console.ReadLine());
    CurrentPlayer.UpdateMoveOptionQueue(); Offer(ReplaceChoice);
    CreateMoveOption(MoveOptionOffer, MoveOptionOfferPosition);
    CurrentPlayer.GetDirection();
    CurrentPlayer.ChangeDir((10 - (ReplaceChoice * 2)));
    MoveOptionOffer.Position = RGen.Next(0, 5);
    //CHANGE
    CurrentPlayer.DecreaseChoiceOptionsLeft(); //Q7
    Console.WriteLine("You have " +
        Convert.ToString(CurrentPlayer.GetChoiceOptionsLeft()) +
        " options left.");
    //END CHANGE
}
```

Testing:

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

```
1 2 3 4 5 6
-----
1 | | | | | |
2 | | | | | |
3 | | | | | |
4 | | | | | |
5 | | | | | |
6 | | | | | |
-----

Move option offer: jazair

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

Turn: Player One

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): 2
You have 2 move options left.

1 2 3 4 5 6
-----
1 | | | | | |
2 | | | | | |
3 | | | | | |
4 | | | | | |
5 | | | | | |
6 | | | | | |
-----

Move option offer: jazair

Player One
Score: 34
Move option queue: 1. jazair 3. cuirassier 4. faujdar 5. j

Turn: Player One

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Turn: Player One

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): 2

You have 1 move options left.

	1	2	3	4	5	6
1				K1		
2						
3						
4						
5						
6				k2		

Move option of jazair

Player One

Score: 86

Move option queue: 1. jazair 2. faujdar 3. cuirassier 4. faujdar 5. j

Turn: Player One

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): 3

You have 0 move options left.

	1	2	3	4	5	6
1				K1		
2						
3						
4						
5						
6				k2		

Move option offer: f

Player One

Score: 82

Move option queue: 1. jazair 2. faujdar 3. jazair 4. faujdar 5. jazai

Turn: Player One

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

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 8

Coding

- Writing the `ResetQueueBackAfterUndo` method which calls the `ResetQueueBack` and successfully pops the item from the end of the queue and returns it to its original position.
- Asking the player if they would like to undo after they have played their move and return the choice to the `PlayGame` method.
- Correctly handling the undo to deduct 5 points and reset the board and queue.

Example Solution

Changes to `Player`:

```
//CHANGE
public void ResetQueueBackAfterUndo(int Position) //Q8
{
    Queue.ResetQueueBack(Position);
}
//END CHANGE
```

Changes to `MoveOptionQueue`:

```
//CHANGE
public void ResetQueueBack(int Position) //Q8
{
    MoveOption RearElement = Queue[Queue.Count - 1];
    Queue.RemoveAt(Queue.Count - 1);
    Queue.Insert(Position - 1, RearElement);
}
//END CHANGE
```

Changes to `PlayGame`:

```
if (MoveLegal)
{
    //CHANGE
    int StartScore = CurrentPlayer.GetScore(); //Q8
    //END CHANGE
    int PointsForPieceCapture = CalculatePieceCapture(Choice, CurrentPlayer);
    CurrentPlayer.ChangeScore(-(Choice + (2 * (Choice - 1))));
    CurrentPlayer.UpdateQueueAfterMove(Choice);
    UpdateBoard(StartSquareReference, FinishSquareReference, Choice);
    UpdatePlayerScore(PointsForPieceCapture);
    Console.WriteLine("New score: " + CurrentPlayer.GetScore());
    //CHANGE
    int UndoScore = StartScore - CurrentPlayer.GetScore();
    Console.WriteLine("Would you like to undo this move? yes/no");
    string UndoChoice = Console.ReadLine().ToLower();
    if (UndoChoice == "yes")
    {
        CurrentPlayer.ChangeScore(UndoScore - 5);
        UpdateBoard(FinishSquareReference, StartSquareReference, Choice);
        CurrentPlayer.ResetQueueBackAfterUndo(Choice);
    }
    else
    {
        if (CurrentPlayer.ScoreAs(Players[0]))
        {
            CurrentPlayer = Players[1];
        }
        else
        {
            CurrentPlayer = Players[0];
        }
        GameOver = CheckIfGameOver();
    }
}
//END CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



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 continue. [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. faujdar 5. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2
Enter the square containing the piece to move (row number followed by column number): 2
Enter the square to move to (row number followed by column number): 31
New score: 101

Would you like to undo this move at the cost of 5 score points? yes/no
yes
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. faujdar 5. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
Enter the square containing the piece to move (row number followed by column number): 2
Enter the square to move to (row number followed by column number): 32
New score: 99

Would you like to undo this move at the cost of 5 score points? yes/no
no

```

```

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 5. cuirassier

Turn: Player Two

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

```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 9

Coding

- CreateMoveOptionOffer has been modified to append "sahm" as a move option to the Name parameter of "sahm" in the CreateMoveOption method. [1 mark]
- 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 turn. [1 mark]
- Correctly removing all the pieces in the Sahm line of fire from the board (except the Sahm). [1 mark]
- Correctly awarding points for removed/destroyed pieces (even if a piece was not the Sahm). [1 mark]

Example Solution

Changes to CreateMoveOptionOffer:

```
private void CreateMoveOptionOffer()
{
    //CHANGE
    MoveOptionOffer.Add("sahm"); //Q9
    //END CHANGE
    MoveOptionOffer.Add("jazair");
    MoveOptionOffer.Add("chowkidar");
    MoveOptionOffer.Add("cuirassier");
    MoveOptionOffer.Add("ryott");
    MoveOptionOffer.Add("faujdar");
}
```

Code for CreateSahmMoveOption:

```
//CHANGE
private MoveOption CreateSahmMoveOption(int Direction)
{
    MoveOption NewMoveOption = new MoveOption("sahm");
    Move NewMove = new Move(0, 0);
    NewMoveOption.AddToPossibleMoves(NewMove);
    return NewMoveOption;
}
//END CHANGE
```

Changes to CreateMoveOption:

```
private MoveOption CreateMoveOption(string Name, int Direction)
{
    //CHANGE
    if (Name == "sahm") //Q9
    {
        return CreateSahmMoveOption(Direction);
    }
    else if (Name == "chowkidar")
    {
        //...
    }
    //END CHANGE
}
```

Changes to PlayGame:

```
while (!SquareIsValid)
{
    StartSquareReference = GetSquareReference("cont...");
    SquareIsValid = CheckSquareIsValid(StartSquareReference);
}
//CHANGE
if (CurrentPlayer.ChoiceIsSahm(Choice)) //Q9
{
    if (CurrentPlayer.GetSahmUsed())
    {
        Console.WriteLine("You have already used your Sahm. Please select an alternative move.");
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED




```

else
{
    int PointsForPieceCapture = CalculateSahmMove(Choice);
    CurrentPlayer.SetSahmUsed();
    CurrentPlayer.UpdateQueueAfterMove(Choice);
    UpdatePlayerScore(PointsForPieceCapture);
}
}
else
{
    int FinishSquareReference = 0;
    SquareIsValid = false;
    while (!SquareIsValid)
    {
        FinishSquareReference = GetSquareReference(Choice);
        SquareIsValid = CheckSquareIsValid(FinishSquareReference);
    }
    bool MoveLegal = CurrentPlayer.CheckPlayerMove(Choice, StartSquareReference, FinishSquareReference);
    if (MoveLegal)
    {
        int PointsForPieceCapture = CalculatePieceCapturePoints(FinishSquareReference, StartSquareReference);
        CurrentPlayer.ChangeScore(-(Choice + (2 * (PointsForPieceCapture))));
        CurrentPlayer.UpdateQueueAfterMove(Choice);
        UpdateBoard(StartSquareReference, FinishSquareReference);
        UpdatePlayerScore(PointsForPieceCapture);
        Console.WriteLine("New score: " + CurrentPlayer.Score);
        Environment.NewLine);
    }
}
//END CHANGE

```

Code for CalculateSahmMove:

```

//CHANGE
private int CalculateSahmMove(int StartSquareReference)
{
    Console.WriteLine("Sahm Deployed!");
    StartRow = StartSquareReference / 10;
    StartCol = StartSquareReference % 10;
    int SahmPointsCollector = 0;
    int RowCounter = StartRow;
    int Rowdifference = StartRow;
    if (CurrentPlayer.GetDirection() == 1)
    {
        Rowdifference = 6 - StartRow;
    }
    for (int i = 0; i < Rowdifference; i++)
    {
        int TargetSquareReference = int.Parse(RowCounter.ToString() + i.ToString());
        if (Board[GetIndexOfSquare(TargetSquareReference)].Color == CurrentPlayer.Color && !Board[GetIndexOfSquare(TargetSquareReference)].IsOccupied)
        {
            if (!CurrentPlayer.SameAs(Board[GetIndexOfSquare(StartSquareReference)].GetPieceInSquare().GetBelongsToPlayer()))
            {
                RaaketPointsCollector += CalculatePieceCapturePoints(TargetSquareReference, StartSquareReference);
                Board[GetIndexOfSquare(TargetSquareReference)].IsOccupied = true;
            }
        }
        if (CurrentPlayer.GetDirection() == 1)
        {
            RowCounter += 1;
        }
        else
        {
            RowCounter -= 1;
        }
    }
}

```

**COPYRIGHT
PROTECTED**



```

    }
    Console.WriteLine("Points Collected from Sahm: " + Convert.ToInt32(SahmPointsCollector));
    return SahmPointsCollector;
}
//END CHANGE

```

Code for SahmMoveSelected:

```

//CHANGE
public bool SahmMoveSelected(int Choice) //Q9
{
    if (Choice != 9)
    {
        if (Game[Choice - 1].GetName() == "sahm")
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}
//END CHANGE

```

Changes to Player:

```

class Player
{
    private string Name;
    private int Direction, Score;
    private MoveOptionQueue Queue = new MoveOptionQueue();

    //CHANGE
    private bool SahmUsed; //Q9
    //CHANGE

    public Player(string N, int D)
    {
        Score = 100;
        Name = N;
        Direction = D;
    }

    //CHANGE
    public void SetSahmUsed() //Q9
    {
        SahmUsed = true;
    }

    public bool GetSahmUsed() //Q9
    {
        return SahmUsed;
    }

    public bool ChoiceIsSahm(int Choice) //Q9
    {
        if (Queue.SahmMoveSelected(Choice))
        {
            return true;
        }
        return false;
    }
}
//END CHANGE

```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



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		1	1	1	
3					
4					
5	-	-	-	-	
6			k2		

Move option offer: sahm

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

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2
Enter the square containing the piece to move (row number followed by column number): 22
Enter the square to move to (row number followed by column number): 33
New score: 101

1 2 3 4 5 6

1			K1		
2		1	1	1	
3					
4					
5	-	-	-	-	
6			k2		

Move option offer: sahm

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

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 (1 to 5): 1

1 2 3

1		K1	
2		1	
3		1	
4			
5	-	-	-
6		k2	

Move option offer: ryott

Player Two
Score: 92
Move option queue: 1. sahm 2. chowkidar 3. jazair 4. faujdar 5. cuirassier

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
Enter the square containing the piece to move (row number followed by column number): 53
Sahm Deployed!
Points Collected from Sahm: 2

1 2 3 4 5 6

1			K1		
2			1	1	
3					
4					
5	-	-	-	-	
6			k2		

Move option offer: ryott

Player One
Score: 101
Move option queue: 1. ryott 2. cuirassier 3. faujdar 4. jazair 5. chowkidar

Turn: Player One

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 10

Coding

- Adding 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 from the square.
- Ensuring that the players turn ends after choosing 7 ('k' with in the example of PieceSacrificed variable). [1 mark]

Example Solution

Changes to PlayGame?

```
put public void PlayGame()
{
    bool GameOver = false;
    while (!GameOver)
    {
        DisplayState();
        bool SquareIsValid = false;
        int Choice;
        //CHANGE
        int TargetSquareReference = 0; //Q10
        bool PieceSacrificed = false;
        do
        {
            Console.WriteLine("Choose move option to use from the menu: 1 to 6 to move a piece, 7 to create a Kotla, 8 to replace a piece with a new Kotla, 9 to sacrifice a piece or 0 to end the game. Enter your choice:");
            Choice = Convert.ToInt32(Console.ReadLine());
            if (Choice == 7)
            {
                Console.WriteLine("Please enter the square reference of the square you want to sacrifice a piece on (e.g. a1, converted to a Kotla");
                int TargetSquareReference = 0;
                while (!SquareIsValid)
                {
                    TargetSquareReference = GetSquareReferenceFromUser();
                    Console.WriteLine("Please enter the square reference of the square you want to replace with a new Kotla");
                    SquareIsValid = CheckSquareIsValid(TargetSquareReference);
                    if (Board[GetIndexOfSquare(TargetSquareReference)] == null || Board[GetIndexOfSquare(TargetSquareReference)] != PieceType.Kotla)
                    {
                        Console.WriteLine("That is not a valid square reference. Please try again.");
                        SquareIsValid = false;
                    }
                }
                Board[GetIndexOfSquare(TargetSquareReference)] = new Kotla(CurrentPlayer, "K");
                Square S;
                if (CurrentPlayer.GetName() == "Player One")
                {
                    S = new Kotla(CurrentPlayer, "K");
                }
                else
                {
                    S = new Kotla(CurrentPlayer, "k");
                }
                Board[GetIndexOfSquare(TargetSquareReference)] = S;
                SquareIsValid = false;
                PieceSacrificed = true;
            }
            else if (Choice == 9)
            {
                UseMoveOptionOffer();
                DisplayState();
            }
        }
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

    }
    while (Choice < 1 || Choice > 3 && !PieceSacrificed)
    if (!PieceSacrificed)
    {
        int StartSquareReference = 0;
        while (!SquareIsValid)
        {
            StartSquareReference = GetSquareReference("co
            SquareIsValid = CheckSquareIsValid(StartSqu
        }
        int FinishSquareReference = 0;
        SquareIsValid = false;
        while (!SquareIsValid)
        {
            FinishSquareReference = GetSquareReference(
            SquareIsValid = CheckSquareIsValid(FinishSq
        }
        bool MoveLegal = CurrentPlayer.CheckPlayerMove(
        StartSquareReference, FinishSquareReference);
        if (MoveLegal)
        {
            int PointsForPieceCapture =
            CalculatePieceCapturePoints(FinishSquareRef
            CurrentPlayer.ChangeScore(-(Choice + (2 * (
            CurrentPlayer.UpdateQueueAfterMove(Choice);
            UpdateBoard(StartSquareReference, FinishSqu
            UpdatePlayerScore(PointsForPieceCapture);
            Console.WriteLine("New score: " + CurrentPl
            Environment.NewLine);
        }
    }
}
//END CHANGE

```

Testing:

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



The screenshot displays a game interface with a board and player information. The board is a 6x6 grid with columns labeled 1 to 6 and rows labeled 1 to 6. The board contains several pieces: 'K1' at (1,1), 'K' at (2,1), 'k2' at (6,1), and 'k' at (5,1). The interface shows the following text:

```

Move option offer: jazair

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

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 7 to replace a piece w
Select one of your players to make the ultimate sacrifice and be converted to a Kotla
Enter the square containing the piece to replace with a new Kotla (row number followed by colum

```

The board is shown again with the same pieces. The interface then shows the following text:

```

Move option offer: jazair

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

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer or 7 to replace a piece w

```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 11

Coding

- Adding option 6 to the menu which brings up a list of options to modify the queue.
- Displaying a suitably formed menu as per the question/example code. [1 mark]
- Validating that the user entered an option from the menu correctly before proceeding.
- Option 1 correctly reverses the player's queue in a method inside MoveOptionQueue.
- Option 2 correctly swaps queue with the opponent's without breaking encapsulation that shouldn't be exposed. [1 mark]
- Option 3 correctly swaps the first and last elements of your queue. [1 mark]
- Option 4 correctly moves a MoveOption to the front of the queue. [1 mark]
- Option 5 costs 1 point and the other cost any points but the other options all cost 3 points.

Example Solution

Changes to PlayGame:

```
while (!GameOver)
{
    DisplayState();
    bool SquareIsValid = false;
    int Choice;
    do
    {
        //CHANGE
        Console.WriteLine("Choose move option to use from the offer or 6 to modify your queue options: ");
        Choice = Convert.ToInt32(Console.ReadLine());
        if (Choice == 6)
        {
            ModifyQueueOptions();
            DisplayState();
        }
        if (Choice == 9)
        {
            UseMoveOptionOffer();
            DisplayState();
        }
        //END CHANGE
    }
    while (Choice < 1 || Choice > 3);
}
```

Code for ModifyQueueOptions:

```
//CHANGE
private void ModifyQueueOptions() //Q11
{
    bool Valid = false;
    int UserChoice = 0;
    while (!Valid)
    {
        Console.WriteLine();
        Console.WriteLine("Select from the Queue menu choice");
        Console.WriteLine("1: Reverse your Queue \t\t\t\t\t");
        Console.WriteLine("2: Swap Queues with your Opponent");
        Console.WriteLine("3: Swap the first and last items in");
        Console.WriteLine("4: Move a MoveOption of your choice");
        Console.WriteLine("5: Nothing, make a normal move \t");
        Console.WriteLine();
        UserChoice = Convert.ToInt32(Console.ReadLine());
        if (UserChoice > 0 && UserChoice < 6)
        {
            Valid = true;
        }
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

        {
            Valid = true;
        }
    }
    switch (UserChoice)
    {
        case 1:
            CurrentPlayer.ReversePlayerQueue();
            CurrentPlayer.ChangeScore(-3);
            break;
        case 2:
            MoveOptionQueue TempPlayer1Queue = Players[0].GetMoveOptionQueue();
            MoveOptionQueue TempPlayer2Queue = Players[1].GetMoveOptionQueue();
            Players[0].ReplaceQueue(TempPlayer2Queue);
            Players[1].ReplaceQueue(TempPlayer1Queue);
            Console.ReadLine();
            CurrentPlayer.ChangeScore(-3);
            break;
        case 3:
            CurrentPlayer.SwapFirstAndLast();
            CurrentPlayer.ChangeScore(-3);
            break;
        case 4:
            CurrentPlayer.MoveItemToFront();
            CurrentPlayer.ChangeScore(-3);
            break;
        case 5:
            Console.WriteLine("No change made - returning to menu");
            break;
    }
}
//END CHANGE

```

Changes to MoveOptionQueue:

```

//CHANGES TO MOVEOPTIONQUEUE
public void ReverseQueue() //Q11
{
    Queue.Reverse();
}
public void SwapFirstAndLast() //Q11
{
    MoveOption FirstItem = Queue[0];
    MoveOption LastItem = Queue[Queue.Count - 1];
    Queue.RemoveAt(0);
    Queue.Insert(0, LastItem);
    Queue.RemoveAt(Queue.Count - 1);
    Queue.Add(FirstItem);
}
public void MoveItemToFront(int Position) //Q11
{
    MoveOption Temp = Queue[Position];
    Queue.RemoveAt(Position);
    Queue.Insert(0, Temp);
}
//END CHANGE

```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Changes to Player:

```
//CHANGE
public void ReplaceQueue(MoveOptionQueue NewQueue)
{
    Queue = new MoveOptionQueue();
    Queue = NewQueue;
}
public MoveOptionQueue GetMoveOptions()
{
    return Queue;
}
public void SwapPlayerQueue() //Q11
{
    Queue.ReverseQueue();
}
public void SwapFirstAndLast() //Q11
{
    Queue.SwapFirstAndLast();
}
public void MoveItemToFront() //Q11
{
    int SelectedOption = 0;
    do
    {
        Console.WriteLine("Select a MoveOption to move to the front of the queue. Select an option 1 to 5:" + Environment.NewLine);
        Console.WriteLine(Queue.GetQueueAsString());
        SelectedOption = Convert.ToInt32(Console.ReadLine());
    }
    while (SelectedOption < 1 || SelectedOption > 5);
    Queue.MoveItemToFront(SelectedOption - 1);
}
//END CHANGE
```

Testing:

- Showing a queue of options 1-4 working. [1 mark]
- Showing the remaining three options working. [1 mark]
- Showing option 5 and the scoring working correctly. [1 mark]

INSPECTION COPY

COPYRIGHT
PROTECTED



	1	2	3	4	5	6
1				K1		
2			1	1	1	1
3						
4						
5			"	"	"	"
6					k2	

Move option offer: jazair

Player One

Score: 100

Move option queue: 1. chowkidar 2. cuirassier 3. faujdar 4. jazair 5. ryott

Turn: Play

Choose move option to use from queue (1 to 3) or 9 to take the offer or 6 to modify

Select from the Queue menu choices below.

- 1: Reverse your Queue 3 points
- 2: Swap Queues with your Opponent 3 points
- 3: Swap the first and last items in your queue 3 points
- 4: Move a MoveOption of your choice to the front of your Queue 3 points
- 5: Nothing, make a normal move 0 points

1

	1	2	3	4	5	6
1				K1		
2			1	1	1	1
3						
4						
5			"	"	"	"
6					k2	

Move option offer: jazair

Player One

Score: 97

Move option queue: 1. faujdar 2. cuirassier 3. chowkidar 4. ryott 5. jazair

Turn: Play

Choose move option to use from queue (1 to 3) or 9 to take the offer or 6 to modify

Select from the Queue menu choices below.

- 1: Reverse your Queue 3 points
- 2: Swap Queues with your Opponent 3 points

COPYRIGHT
PROTECTED



- 3: Swap the first and last items in your queue 3 points
- 4: Move a MoveOption of your choice to the front of your Queue 3 points
- 5: Nothing, make a normal move 0 points

2

	1	2	3	4	5	6
1				K1		
2		1	1	1	1	
3						
4						
5		"	"	"	"	
6				k2		

Move option



Player One

Score: 94

Move option queue: 1. ryott 2. chowkidar 3. jaisir 4. faujdar 5. cuirassier

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 6 to modify

Select from the Queue menu choices below.

- 1: Reverse your Queue 3 points
- 2: Swap Queues with your Opponent 3 points
- 3: Swap the first and last items in your queue 3 points
- 4: Move a MoveOption of your choice to the front of your Queue 3 points
- 5: Nothing, make a normal move 0 points

3

	1	2	3	4	5	6
1				K1		
2		1	1	1	1	
3						
4						
5		"	"	"	"	
6				k2		



COPYRIGHT
PROTECTED



COPYRIGHT
PROTECTED



Move option offer: jazair

Player One

Score: 91

Move option queue: 1. cuirassier 2. chowkidar 3. jazair 4. faujdar 5. ryott

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 6 to modify

Select from the Queue menu choices below.

- 1: Reverse your Queue 3 points
- 2: Swap Queues with your Opponent 3 points
- 3: Swap the first and last items in your queue 3 points
- 4: Move a MoveOption of your choice to the front of your Queue 3 points
- 5: Nothing, make a normal move 0 points

4

Select a MoveOption to move to the front of your queue. Select an option 1 to 5:

1. cuirassier 2. chowkidar 3. jazair 4. faujdar 5. ryott

4

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Move option offer: jazair

Player One

Score: 88

Move option queue: 1. faujdar 2. cuirassier 3. chowkidar 4. jazair 5. ryott

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 6 to modify

Select from the Queue menu choices below.

- 1: Reverse your Queue 3 points
- 2: Swap Queues with your Opponent 3 points
- 3: Swap the first and last items in your queue 3 points
- 4: Move a MoveOption of your choice to the front of your Queue 3 points
- 5: Nothing, make a normal move 0 points

5

No change made - returning to main menu

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Move option offer: jazair

Player One

Score: 88

Move option queue: 1. cuirassier 2. chowkidar 3. jazair 4. faujdar 5. ryott

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer or 6 to modify

Task 12

Coding

- Creating and storing the number of pieces correctly in the new protected attribute
- Adding a call to CheckReincarnation in the correct place. [1 mark]
- Creating CountNormalPieces to correctly return the number of pieces excluding [1 mark]
- Correctly detecting when a piece reaches the opponent's back row. [1 mark]
- Having a condition to only allow reincarnation if the player has fewer pieces than
- Correctly handling the reincarnation in the player's own back row and checking

Example Solution

Changes to Dastan constructor:

```
protected Random RGen = new Random();
//CHANGE
protected int NoOfPieces; //Q12
//END CHANGE

//CHANGE
public Dastan(int R, int C, int NumberOfPieces) //Q12
{
    NoOfPieces = NumberOfPieces; //Q12
    //END CHANGE
    Players.Add(new Player("Player One", 1));
    Players.Add(new Player("Player Two", -1));
}
```

Changes to PlayGame:

```
bool MoveLegal = CurrentPlayer.CheckPlayerMove(Choice,
FinishSquareReference);
if (MoveLegal)
{
    //CHANGE
    CheckReincarnation(FinishSquareReference);
    //END CHANGE
    int PointsForPieceCapture = CalculatePieceCapturePoints;
    CurrentPlayer.ChangeScore(-(Choice + (2 * (Choice - 1))));
    CurrentPlayer.UpdateQueueAfterMove(Choice);
    UpdateBoard(StartSquareReference, FinishSquareReference);
    UpdatePlayerScore(PointsForPieceCapture);
    Console.WriteLine("New score: " + CurrentPlayer.GetScore());
}
```

Code for CheckReincarnation:

```
//CHANGE
private void CheckReincarnation(int SquareReference)
{
    int Row = SquareReference / 16;
    if (CurrentPlayer == Players[0])
    {
        if (Row == NoOfRows && CountNormalPieces() < NoOfPieces)
        {
            Console.WriteLine("Congratulations, you have earned the right to be reincarnated!");
            bool Valid = false;
            while (!Valid)
            {
                Console.WriteLine("Which column would you like to be reincarnated on?");
            }
        }
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

        int ReincarnationCol = Convert.ToInt32(Console.ReadLine());
        if (Board[GetIndexOfSquare(10 + ReincarnationCol)] != null)
        {
            Console.WriteLine("The square must be empty");
        }
        else
        {
            Board[GetIndexOfSquare(10 + ReincarnationCol)] =
                new Piece("piece", Players[0], 1, "!");
            Valid = true;
        }
    }
    else
    {
        if (Row == 1 && CountNormalPieces() < NoOfPieces)
        {
            Console.WriteLine("Congratulations, you have earned a new piece!");
            bool Valid = false;
            while (!Valid)
            {
                Console.WriteLine("Which column would you like to be reincarnated on?");
                int ReincarnationCol = Convert.ToInt32(Console.ReadLine());
                if (Board[GetIndexOfSquare(6 * 10 + ReincarnationCol)].GetPieceInSquare() != null)
                {
                    Console.WriteLine("The square must be empty");
                }
                else
                {
                    Board[GetIndexOfSquare(6 * 10 + ReincarnationCol)] =
                        new Piece("piece", Players[1], 1, "\\");
                    Valid = true;
                }
            }
        }
    }
}
//END CHANGE

```

Code for CountNormalPieces:

```

//CHANGE
private int CountNormalPieces() //Q12
{
    int Pieces = 0;
    foreach (Square S in Board)
    {
        Piece PieceInSquare = S.GetPieceInSquare();
        if (PieceInSquare != null)
        {
            if (PieceInSquare.GetBelongsTo().SameAs(CurrentPlayer) &&
                PieceInSquare.GetTypeOfPiece() == "piece")
            {
                Pieces += 1;
            }
        }
    }
    return Pieces;
}
//END CHANGE

```

**COPYRIGHT
PROTECTED**



Testing:

- Correctly showing the moves as requested in the tests, specifically including the 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:
Player One
Score: 100
Move option queue: 1. ryott 2. chowkidar 3. cuirassier 4. faujdar 5. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
Enter the square containing the piece to move (row number followed by column number): 5
Enter the square to move to (row number followed by column number): 61
Congratulations, you have earned a reincarnation!
Which column would 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 5. cuirassier

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
Enter the square containing the piece to move (row number followed by column number): 2
Enter the square to move to (row number followed by column number): 11
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. jazair 5. ryott

Turn: Player One

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

```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 13

Coding

- Putting the Ta'ziz in the correct place regardless of board size. [1 mark]
- Having a mechanism that correctly counts the number of turns that the Ta'ziz has camped for.
- Resetting the CampedTurns attribute if the square becomes empty or changes colour.
- Allowing the player to make a move that costs zero points when they have held it for two turns.
- Showing the correct 'A' and 'a' symbols when the Ta'ziz is occupied by overriding the ToString() method.
- Correctly resetting the symbol for the Ta'ziz to 'x' when a player leaves by overriding the ToString() method.

Example Solution

Changes to Board:

```
for (int Row = 1; Row <= NoOfRows; Row++)
{
    for (int Column = 1; Column <= NoOfColumns; Column++)
    {
        if (Row == 1 && Column == NoOfColumns / 2)
        {
            S = new Kotla(Players[0], "K");
        }
        //CHANGE
        else if (Row == NoOfRows / 2 && Column == NoOfColumns / 2)
        {
            S = new Taziz("x");
        }
        //END CHANGE
    }
}
```

Changes to PlayGame:

```
if (MoveLegal)
{
    if (ChoiceForPieceCapture == CalculatePieceCapture)
    {
        //CHANGE
        CurrentPlayer.UpdateQueueAfterMove(Choice);
        UpdateBoard(StartSquareReference, FinishSquareReference);
        if (!Board[GetIndexOfSquare(Convert.ToInt32((NoOfColumns / 2).ToString()))].GetCampedTwoTurns())
        {
            CurrentPlayer.ChangeScore(-(Choice + (2 * (NoOfColumns / 2))));
        }
        else
        {
            Console.WriteLine("You have camped for two turns. This move is for free!");
        }
        //END CHANGE
        UpdatePlayerScore(PointsForPieceCapture);
        Console.WriteLine("New score: " + CurrentPlayer.GetScore());
    }
}
```

Changes to Square

```
//CHANGE
public virtual bool GetCampedTwoTurns() //Q13
{
    return false;
}
//CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Code for Taziz:

```
//CHANGE
class Taziz : Square //Q13
{
    private int CampedTurns = 0;
    public Taziz(string S) : base()
    {
        Symbol = S;
    }
    public override void SetPiece(Piece P)
    {
        base.SetPiece(P);
        P.BelongsTo().GetBelongsTo();
        P.BelongsTo().GetName() == "Player Two")
        Symbol = "a";
        CampedTurns = 0;
    }
    else
    {
        Symbol = "A";
        CampedTurns = 0;
    }
}
public override Piece RemovePiece()
{
    Symbol = "x";
    CampedTurns = 0;
    Piece PieceToReturn = PieceInSquare;
    PieceInSquare = null;
    return PieceToReturn;
}

public override bool GetCampedTurns()
{
    if (CampedTurns == 0)
    {
        CampedTurns = 0;
        return true;
    }
    else
    {
        if (PieceInSquare != null)
        {
            CampedTurns++;
        }
        return false;
    }
}
}
//END CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Testing:

- Show the Ta'ziz 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		x			
4					
5	"	"	"	"	
6			k2		

Move option d

Player One

Score: 100

Move option queue: 1. ryott 2. chowkidar 3. cuirassier 4. faujdar 5. jazair

Turn: Player One

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

Enter the square containing the piece to move (row number followed by column number): 33

Enter the square to move to (row number followed by column number): 33

New score: 104

1 2 3 4 5 6

1			K1		
2					
3		A1			
4					
5	"	"	"	"	
6			k2		

Move option offer: jazair

Player Two

Score: 100

Move option queue: 1. ryott 2. chowkidar 3. jazair 4. faujdar 5. cuirassier

Turn: Player Two

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

Enter the square containing the piece to move (row number followed by column number): 43

Enter the square to move to (row number followed by column number): 43

New score: 104

1 2 3 4 5 6

.....(steps while player 1 is camping not shown)

INSPECTION COPY

COPYRIGHT
PROTECTED



Enter the square to move to (row number followed by column number): 45
New score: 108

	1	2	3	4	5	6
1				K1		
2						
3			A1			
4			"		"	
5		"			"	
6				k2		

Move option offer: jaz

Player One
Score: 105

Move option queue: 1. chowkidar 2. faujdar 3. jazair 4. ryott 5. cuirassier

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
Enter the square containing the piece to move (row number followed by column number): 34
Enter the square to move to (row number followed by column number): 36
You have camped for two whole turns - well done. This move is for free!
New score: 110

	1	2	3	4	5	6
1				K1		
2						
3			A1			
4			"		"	
5		"			"	
6				k2		

Move option offer: jazair

Player Two
Score: 108

Move option queue: 1. jazair 2. faujdar 3. cuirassier 4. ryott 5. chowkidar

Turn: Player Two

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

INSPECTION COPY

COPYRIGHT
PROTECTED

INSPECTION COPY



Task 14

Coding

- Using a method to track the weather event (this is the WeatherEvent variable in the code)
- Having the countdown timer allow precisely two complete turns from when it is announced
- Announcing to the players when the weather event started with a 2 turns warning
- Destroying all pieces in the same column as the weather event at when the timer expires
- Destroying a Kotla in the weather event column when the timer expires. (1 mark)
- Correctly selecting a random empty square (1 mark)
- Creating a WeatherEvent class with GetWeatherLocation and SetWeatherLocation methods
- Implementing CountdownComplete so that it returns an appropriate value including the column number when the timer expires. (1 mark)

Example Solution

Changes to PlayGame:

```
public void PlayGame()
{
    bool GameOver = false;
    //CHANGE
    WeatherEvent RandomWeatherEvent = null;    //Q14
    //END CHANGE
    while (!GameOver)
```

```
    while (!SquareIsValid)
    {
        FinishSquareReference = GetSquareReference("to be destroyed");
        SquareIsValid = CheckSquareIsValid(FinishSquareReference);
    }
    //CHANGE
    if (RandomWeatherEvent == null)    //Q14
    {
        RandomWeatherEvent = WeatherEventOccurs();
    }
    else
    {
        if (RandomWeatherEvent.CountDownComplete())
        {
            int ColToDestroy = RandomWeatherEvent.GetWeatherLocation();
            for (int Row = 1; Row < NoOfColumns + 1; Row++)
            {
                if (Board[GetIndexOfSquare(Row * 10 + ColToDestroy)].GetPieceInSquare() != null)
                {
                    Board[GetIndexOfSquare(Row * 10 + ColToDestroy)].SetPieceInSquare(null);
                }
                if (Board[GetIndexOfSquare(Row * 10 + ColToDestroy)].GetPieceInSquare() == null)
                {
                    Board[GetIndexOfSquare(Row * 10 + ColToDestroy)].SetPieceInSquare(Kotla);
                }
            }
        }
    }
    //CHANGE
    bool MoveLegal = CurrentPlayer.CheckPlayerMove(Choice, FinishSquareReference);
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Code for WeatherEventOccurs:

```
//CHANGE
private WeatherEvent WeatherEventOccurs() //Q14
{
    WeatherEvent RandomWeatherEvent;
    if (RGen.Next(0, 2) == 0) //50% chance of returning true
    {
        bool WeatherEventPlaced = false;
        while (!WeatherEventPlaced)
        {
            int Row = RGen.Next(1, 7);
            int Col = RGen.Next(1, 7);
            string RandomSquare = int.Parse(Row.ToString()) + Col.ToString();
            if (Board[GetIndexOfSquare(RandomSquare)].GetPiece() == null)
            {
                RandomWeatherEvent = new WeatherEvent(RandomSquare);
                WeatherEventPlaced = true;
                Console.WriteLine("A Weather event has happened. Please move out of it's way!");
                return RandomWeatherEvent;
            }
        }
        ///This scenario can only happen if the board is completely full
        ///but the compiler needs to have a return on all code paths
        Console.WriteLine("There are no locations on the board available for a weather event");
        return null;
    }
    else
    {
        return null;
    }
}
//END CHANGE
```

Code for WeatherEvent:

```
//CHANGE
class WeatherEvent //Q14
{
    private int CountdownTimer = 3;
    private int SquareReference;
    public WeatherEvent(int WeatherSquareReference)
    {
        SquareReference = WeatherSquareReference;
        Console.WriteLine("A weather event has occurred at location " +
            Convert.ToString(WeatherSquareReference));
        Console.WriteLine("After two complete turns, all pieces on the board will be destroyed.");
    }

    public void SetWeatherLocation(int WeatherSquareReference)
    {
        SquareReference = WeatherSquareReference;
    }

    public int GetWeatherLocation()
    {
        return SquareReference;
    }

    public bool CountdownComplete()
    {
        if (CountdownTimer == 0)
        {
            Console.WriteLine("All pieces on the board have been destroyed.");
        }
    }
}
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



```

    {
        Console.WriteLine("The weather event destroys all the pieces in the column where the weather event occurred. " +
            Convert.ToString(SquareReference % 10));
        return true;
    }
    else
    {
        CountdownTimer--;
        if (CountdownTimer > 1)
        {
            Console.WriteLine("The weather event at location: " +
                Convert.ToString(SquareReference) + " will occur after " +
                CountdownTimer + " more turn(s).");
        }
        else
        {
            Console.WriteLine("The weather event at location: " +
                Convert.ToString(SquareReference) + " will occur now.");
        }
        return false;
    }
}
//END CHANGE

```

Testing:

- Having at least one piece owned by each player in the column where the weather event occurred. [1 mark]
- Showing all pieces in the column destroyed. [1 mark]



1 2 3 4 5 6

Move option offered: 1. ryott 2. chowkidar 3. cuirassier 4. faujdar 5. jazair

Player One
Score: 188

Turn: Player One

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

Enter the square containing the piece to move (row number followed by column number): 22

Enter the square to move to (row number followed by column number): 32

A weather event has occurred at location: 15

After two complete turns, all pieces in the same column will be destroyed.

A weather event has happened. You have 2 turns to move out of it's way!

New score: 184

1 2 3 4 5 6

Move option offered: jazair

Player Two
Score: 188

Move option queue: 1. ryott 2. chowkidar 3. jazair 4. faujdar 5. cuirassier

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2

Enter the square containing the piece to move (row number followed by column number): 54

Enter the square to move to (row number followed by column number): 45

The weather event at location: 15 will occur after one more turn

New score: 181

(steps during weather event countdown not shown)

COPYRIGHT
PROTECTED



Turn: Player Two

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

Enter the square containing the piece to move (row number followed by column number): 53

Enter the square to move to (row number followed by column number): 43

The weather event at location: 15 will occur next turn

New score: 185

	1	2	3	4	5	6
1				K1		
2						
3						
4						
5						
6					k2	

Move option offer: jazair

Player One

Score: 188

Move option queue: 1. cuirassier 2. faujdar 3. jazair 4. ryatt 5. chawkidar

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2

Enter the square containing the piece to move (row number followed by column number): 32

Enter the square to move to (row number followed by column number): 33

The weather event destroys all the piece in the column: 5

New score: 189

	1	2	3	4	5	6
1				K1		
2						
3						
4						
5						
6					k2	

Move option offer: jazair

Player Two

Score: 185

INSPECTION COPY

COPYRIGHT
PROTECTED



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 mark]
- Modifying CheckSquareIsValid to return false if the square contains a barrier. [1 mark]
- Creating CheckBarrierIsValid which checks the squares for the barrier. [1 mark]
- Creating CheckBarrierIsValid which checks that all the squares for the barrier are valid. [1 mark]
- Creating PlaceBarrier which successfully creates a barrier 3 squares wide on the board. [1 mark]
- Modifying CreateGame with suitable input messages that calls CheckBarrierIsValid and PlaceBarrier for each player. [1 mark]
- Creating CheckManhattanDistance and modifying PlayGame to call that as well as CheckSquareIsValid for the line starting MoveLegal=. [1 mark]
- Inside CheckManhattanDistance, swapping the start column/row and end column/row and reducing the number of checks which need to be made. [1 mark]
- Inside CheckManhattanDistance, iterating along the row and column and vice versa until a move has only been attempted. [1 mark]
- Inside CheckManhattanDistance, correctly iterating along the row and column for all combinations of up, down, left and right (with and without vertical/horizontal moves) and all possible move orientations. [1 mark]

Example Solution

Code for Barrier:

```
//CHANGE
class Barrier : Square //Q15
{
    public Barrier(Player P, Symbol S) : base(P, S)
    {
        BelongsToSymbol(S);
    }
}
//END CHANGE
```

Changes to Square:

```
//CHANGE
public virtual bool ContainsBarrier() //Q15
{
    if (Symbol == "B" || Symbol == "b")
    {
        return true;
    }
    else
    {
        return false;
    }
}
//END CHANGE
```

Changes to CheckSquareIsValid (Dastan class):

```
private bool CheckSquareIsValid(int SquareReference, bool Symbol)
{
    if (!CheckSquareInBounds(SquareReference))
    {
        return false;
    }
}
//CHANGE
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

        if (Board[GetIndexOfSquare(SquareReference)].ContainsBarrier)
        {
            return false;
        }
    }
    //END CHANGE

```

Code for CheckBarrierIsValid (Dastan class):

```

//CHANGE
private bool CheckBarrierIsValid(int SquareReference) //Q15
{
    for (int i = SquareReference - 1; i <= SquareReference + 1; i++)
    {
        if (!CheckSquareInBounds(i))
        {
            return false;
        }
        Piece PieceInSquare = Board[GetIndexOfSquare(i)].GetPiece();
        if (PieceInSquare != null || Board[GetIndexOfSquare(i)].ContainsBarrier)
        {
            return false;
        }
    }
    return true;
}
//END CHANGE

```

Code for PlaceBarrier (Dastan class):

```

//CHANGE
private void PlaceBarrier(Player P, int BarrierMiddleReference)
{
    int BarrierStartReference = BarrierMiddleReference - 1;
    int BarrierEndReference = BarrierMiddleReference + 1;
    for (int i = BarrierStartReference; i <= BarrierEndReference; i++)
    {
        Barrier B = new Barrier(P, i);
        Board[GetIndexOfSquare(i)] = B;
    }
}
//END CHANGE

```

Changes to CreatePieces (Dastan class):

```

CurrentPiece = new Piece("mirza", Players[1], 5, "2");
Board[GetIndexOfSquare(NoOfRows * 10 + (NoOfColumns / 2 + 1))] = CurrentPiece;

//CHANGE
for (int i = 0; i < 2; i++) //Q15
{
    bool Valid = false;
    while (!Valid)
    {
        DisplayBoard();
        Console.WriteLine("Placing barrier piece for " + P.Name);
        Console.WriteLine("A barrier is 3 squares wide. It will occupy 3 squares. \nSelect the location for the middle square.");
        int BarrierLocationReference = Convert.ToInt32(Console.ReadLine());
        int StartRow = BarrierLocationReference / 10;
        int StartColumn = BarrierLocationReference % 10;
        bool ValidLocation = CheckBarrierIsValid(BarrierLocationReference);
        if (ValidLocation)
        {
            PlaceBarrier(P, BarrierLocationReference);
        }
    }
}

```

INSPECTION COPY

**COPYRIGHT
PROTECTED**




```

        if (i == 0)
        {
            PlaceBarrier(Players[i], BarrierLocation);
            Valid = true;
        }
        else
        {
            PlaceBarrier(Players[i], BarrierLocation);
            Valid = true;
        }
    }
    else
    {
        Console.WriteLine("That is not a valid location");
    }
}
//END CHANGE

```

Changes to PlayGame (Dastan class):

```

while (!SquareIsValid)
{
    FinishSquareReference = GetSquareReference("to");
    SquareIsValid = CheckSquareIsValid(FinishSquareReference);
}
//CHANGE
bool MoveLegal = CurrentPlayer.CheckPlayerMove(Choice, FinishSquareReference) && CheckManhattanDistance(StartSquareReference, FinishSquareReference); //Q15
if (MoveLegal)
{
    int PointsForPieceCapture = CalculatePieceCapturePoints(Choice, FinishSquareReference);
    CurrentPlayer.ChangeScore(-(Choice + (2 * (Choice - 1))));
    CurrentPlayer.UpdateQueueAfterMove(Choice);
    UpdateBoard(StartSquareReference, FinishSquareReference);
    UpdatePlayerScore(PointsForPieceCapture);
    Console.WriteLine("New score: " + CurrentPlayer.Score);
    Environment.NewLine;
}
else
{
    Console.WriteLine(Environment.NewLine + "That is not a valid move");
    Environment.NewLine;
}
//END CHANGE

```

Code for CheckManhattanDistance (Dastan class):

```

//CHANGE
public bool CheckManhattanDistance(int StartSquareReference, int FinishSquareReference)
{
    int StartRow = StartSquareReference / 10;
    int StartColumn = StartSquareReference % 10;
    int FinishRow = FinishSquareReference / 10;
    int FinishColumn = FinishSquareReference % 10;
    bool RouteAVerticalClear = true;
    bool RouteAHorizontalClear = true;
    bool RouteBVerticalClear = true;
    bool RouteBHorizontalClear = true;

    if (StartRow > FinishRow) //If required, swap the rows
        around so we only have to check one way
    {

```

**COPYRIGHT
PROTECTED**



```

        int Temp = StartRow;
        StartRow = FinishRow;
        FinishRow = Temp;
    }
    if (StartColumn > FinishColumn)
    {
        int Temp = StartColumn;
        StartColumn = FinishColumn;
        FinishColumn = Temp;
    }
    for (int i = StartRow; i <= FinishRow; i++)
    {
        int TargetSquareReference = int.Parse(i.ToString() + FinishColumn.ToString());
        if (Board[GetIndexOfSquare(TargetSquareReference)] != 0)
        {
            RouteAVerticalClear = false;
        }
        TargetSquareReference = int.Parse(i.ToString() + FinishRow.ToString());
        if (Board[GetIndexOfSquare(TargetSquareReference)] != 0)
        {
            RouteBVerticalClear = false;
        }
    }
    for (int i = StartColumn; i <= FinishColumn; i++)
    {
        int TargetSquareReference = int.Parse(StartRow.ToString() + i.ToString());
        if (Board[GetIndexOfSquare(TargetSquareReference)] != 0)
        {
            RouteBHorizontalClear = false;
        }
        TargetSquareReference = int.Parse(FinishRow.ToString() + i.ToString());
        if (Board[GetIndexOfSquare(TargetSquareReference)] != 0)
        {
            RouteAHorizontalClear = false;
        }
    }
    return (RouteAVerticalClear && RouteAHorizontalClear) | (RouteBVerticalClear && RouteBHorizontalClear);
}
//END CHANGE

```

INSPECTION COPY

COPYRIGHT
PROTECTED



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 mark]
- 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 turn is not correct (right to left and bottom to top). [1 mark]

INSPECTION COPY

1 2 3 4 5 6

1			k1		
2					
3					
4					
5					
6			k2		

Placing barrier piece for Player One
A barrier is 3 squares wide. You cannot jump it, move it or occupy it.
Select the location for the middle piece of the barrier:
34

1 2 3 4 5 6

1			k1		
2					
3		B	B	B	
4					
5					
6			k2		

Placing barrier piece for Player Two
A barrier is 3 squares wide. You cannot jump it, move it or occupy it.
Select the location for the middle piece of the barrier:
42

1 2 3 4 5 6

1			k1		
2					
3		B	B	B	
4	b	b	b		
5					
6			k2		

Move option offer: jazair

Player One
Score: 109
Move option queue: 1. ryott 2. chowkidar 3. cuirassier 4. faujdar 5. jazair

Turn: Player One

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

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
Enter the square containing the piece to move (row number followed by column number): 2A
Enter the square to move to (row number followed by column number): 4B

INSPECTION COPY

INSPECTION COPY

COPYRIGHT
PROTECTED



Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
 Enter the square containing the piece to move (row number followed by column number): 24
 Enter the square to move to (row number followed by column number): 46
 New score: 96

	1	2	3	4	5	6
1				K1		
2						
3			B	B	B	
4	b	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 5. cuirassier

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer: 3
 Enter the square containing the piece to move (row number followed by column number): 53
 Enter the square to move to (row number followed by column number): 31

That is not a valid move

	1	2	3	4	5	6
1				K1		
2						
3			B	B	B	
4	b	b	b	b		
5						
6				K2		

Move option offer: ryott

Player One

Score: 96

Move option queue: 1. chowkidar 2. cuirassier 3. faujdar 4. jazair 5. jazair

Turn: Player One

Choose move option to use from queue (1 to 3) or 9 to take the offer: 2
 Enter the square containing the piece to move (row number followed by column number): 45
 Enter the square to move to (row number followed by column number): 45

That is not a valid move

	1	2	3	4	5	6
1				K1		
2						
3			B	B	B	
4	b	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 5. cuirassier

Turn: Player Two

Choose move option to use from queue (1 to 3) or 9 to take the offer: 1
 Enter the square containing the piece to move (row number followed by column number): 52
 Enter the square to move to (row number followed by column number): 42
 Enter the square to move to (row number followed by column number): 51
 New score: 104

	1	2	3	4	5	6
1				K1		
2						
3			B	B	B	
4	b	b	b	b		
5						
6				K2		

Move option offer: ryott

Player One

Score: 96

Move option queue: 1. ryott 2. cuirassier 3. faujdar 4. jazair 5. jazair

Turn: Player One

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Name

ZigZag Education supporting

A Level AQA Computer Science Paper

Summer 2023



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:

INSPECTION COPY

COPYRIGHT
PROTECTED



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)

INSPECTION COPY

COPYRIGHT
PROTECTED



Exam-style Programming Task

Answer all questions. Remember to save this document

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

INSPECTION COPY

COPYRIGHT
PROTECTED



Preview of Questions Ends Here

This is a limited inspection copy. Sample of questions ends here to avoid students previewing questions before they are set. See contents page for details of the rest of the resource.

Preview of Answers Ends Here

This is a limited inspection copy. Sample of answers ends here to stop students looking up answers to their assessments. See contents page for details of the rest of the resource.