

**Programming Tasks Part 1**

These questions require you to load the **Skeleton Program** and to make programming changes to it.

*Note that any alternative or additional code changes that you deemed appropriate to make must also be evidenced   
– ensuring that it is clear where in the Skeleton Program those changes have been made.*

# Task 1

**Task 1** **Marks:** 2

This question refers to the **Dastan** class.

Introduce new functionality at the point at which both players are instantiated that allows players to have custom names set by the users. Ensure that players cannot both have the same name. This code will replace the two lines that currently create the players with a single call to a new private 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. Include 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 the second player 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 re-prompted, enter ‘Tom’ again and then at the next prompt, enter ‘Victoria’.
* show the game using one of the custom names to address the player in the main game menu.

**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

# Task 2

**Task 2** **Marks:** 4

This question refers to the **CreateMoveOptionOffer**, **CreateMoveOption** and **CreateMoveOptions** 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 moves similarly to a knight in chess – either two squares forward/backwards and one square left/right or oppositely two squares left/right and one square forward/backwards. You should demonstrate the use of the Direction parameter.

A picture containing text, crossword puzzle

Description automatically generated**What you need to do**

**Task 1**

1. Add new functionality into the **CreateMoveOptionOffer** & **CreateMoveOption** methods to perform a Faris move.
2. Modify the **CreateMoveOptions** method to add the Faris after the Ryott for both players.
3. 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 skeleton program.
* play two turns, showing both players making legal Faris moves.

**Evidence that you need to provide:**

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

# Task 3

**Task 3** **Marks:** 4

Develop a new move option called a ‘Sarukh’ (Rocket). The Sarukh move option moves forward in a rocket shape. You should demonstrate the use of the Direction parameter.

**What you need to do**

A picture containing text, crossword puzzle

Description automatically generated**Task 1**

1. Add new functionality into the **CreateMoveOptionOffer**, **CreateMoveOption** and **CreateMoveOptions** methods to perform a Sarukh move.
2. Modify the **CreateMoveOptions** method to add the Sarukh after the Ryott for both players.
3. 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

# Task 4

**Task 4** **Marks:** 5

This question refers to the **PlayGame** method in the **Dastan** class and creation of a new method **AwardWafr** in the **Dastan** class, and **GetWafrAwarded** and **SetWafrAwarded** together with a new attribute **WafrAwarded** in the **Player** class.

Create a ‘Wafr’ award (Abundance) which can be applied to either player once per game. The Wafr has a 25% chance of being awarded to a player on their turn. On receipt of the Wafr, the player has the option of ANY move from their move queue rather than just being able to select from the first three items. The Wafr award removes the move cost for the move the player selects for that turn.

**Note:** If the player makes an invalid move then they ‘lose’ their Wafr and get no value from it. Also the player should not be able to ‘take the offer’ if a Wafr is awarded.

**What you need to do**

**Task 1**

1. Create a new method in the **Dastan** class called **AwardWafr**. This method should have a 25% chance of returning true.
2. Add a new private attribute to the **Player** class called **WafrAwarded**. Include accessor and mutator (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 to allow any move option in the queue to be selected. Ensure that there is no score adjustment for playing a move, and update the value of the attribute to ensure that they cannot receive another Wafr.

**Task 3**

Test that the changes you have made work:

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

**Evidence that you need to provide:**

* PROGRAM SOURCE CODE showing changes made to the PlayGame method of the Dastan class, creation of a new method AwardWafr in the Dastan class
* PROGRAM SOURCE CODE showing changes made to the Player class and creation of the new methods GetWafrAwarded, SetWafrAwarded together with one new attribute WafrAwarded
* SCREEN CAPTURE(S) showing the required test

# Task 5

**Task 5** **Marks:** 5

This question refers to the **PlayGame** method of the **Dastan** class and the creation of a new method **GetJustQueueAsString** in the **Player** class.

Introduce a new option 8 to the main game playing menu. On selecting this option, a player can look at their opponent’s queue to spy what move options their opponent might be considering next. Spying on an opponent’s queue, however, carries a cost of 5 points from the player’s score. After spying on an opponent’s queue, the player’s turn should continue as normal.

**What you need to do**

**Task 1**

Create a new method in the **Player** class called **GetJustQueueAsString** which uses 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 option 8 to the main game playing menu. If the user selects this option, display the move option queue for the opposing player.   
  
(**Hint:** You can check the current player using the **SameAs** method and then pick the other player.) Subtract 5 from the current player score and display the game state again allowing the player to continue their turn as normal.

**Task 3**

Test that the changes you have made work:

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

**Evidence that you need to provide:**

* PROGRAM SOURCE CODE showing changes made to the PlayGame method and of the Dastan class
* PROGRAM SOURCE CODE showing new method GetJustQueueAsString in the Player class
* SCREEN CAPTURE(S) showing the required test

# Task 6

**Task 6** **Marks:** 5

This question refers to the **PlayGame** method together with the modification of **GetSquareReference**, **UseMoveOptionOffer** methods and creation of a new method **GetValidInt** in the **Dastan** class.

Currently the game has a number of areas where it does not handle erroneous user input. Introduce error handling into the **PlayGame**, **GetSquareReference** and **UserMoveOptionOffer** methods to prevent unhandled exceptions from occurring if the user inputs data in an incorrect data type. Allow the user 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 the move is valid; the player should still have a wasted turn if the move is invalid, the purpose of this is to stop the program from crashing.

**What you need to do**

**Task 1**

Create a new private method called **GetValidInt** in the **Dastan** class which checks if the user input is a valid integer. If the input is invalid, allow the user to keep trying again without penalty.

**Task 2**

Modify the **GetSquareReference** method to use the new **GetValidInt** method to test for erroneous user input. Add an error message if the user enters an invalid square.

**Task 3**

Modify the **UseMoveOptionOffer** method to use the new **GetValidInt** method to test for erroneous user input and test to confirm that the user input is within the correct range.

**Task 4**

Test that the changes you have made work:

* run the skeleton program.
* from the main game playing menu, enter ‘help’ as your choice and show a suitable error message. Then choose move 1.
* For player one, enter a square of 19 and show the error message. Then choose square 22.
* For player two, select option 9 to take the offer move and choose position 8. Show the error message.

**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 UseMoveOptionOffer method
* PROGRAM SOURCE CODE showing the creation of new GetValidInt method
* SCREEN CAPTURE(S) showing the required test

# Task 7

**Task 7** **Marks:** 5

This question refers to the **PlayGame** and **UseMoveOptionOffer** methods in the **Dastan** class and the creation of a new attribute **ChoiceOptionsLeft** along with accessor and mutator (getter/setter) methods **DecreaseChoiceOptionsLeft** and **GetChoiceOptionsLeft** in the **Player** class.

Currently a player can repeatedly select option 9 from the main game playing menu, filling their queue with new move options. Introduce a limit so that a player can only ‘accept the offer’ from the Move Option menu three times in a game. Each time a player accepts the offer, advise them of how many selections they have left and remove the menu for that player once they have used it three times.

**What you need to do**

**Task 1**

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

1. Initialise **ChoiceOptionsLeft** to 3.
2. Create a new accessor method called **GetChoiceOptionsLeft** which returns the value of the attribute **ChoiceOptionsLeft**.
3. Create a new mutator method called **DecreaseChoiceOptionsLeft** which decrements the **ChoiceOptionsLeft** attribute.

**Task 2**

Modify the **PlayGame** method to test the number of options the player has left so that they can only use three during the game.

1. Modify the **PlayGame** method so that if the player has used up all their option choices, option 9 will no longer be available to the player.
2. Modify the **UseMoveOptionOffer** method so that when a move option is selected by the player, the number of options available to them decreases by one. Advise the player how many option 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 positions 1 to 4 in the player one queue.
* show the removal of option 9 from the main game playing menu and show that it does nothing if 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

# Task 8

**Task 8** **Marks:** 5

This question refers to the **PlayGame** method of the **Dastan** class and creation of new methods **ResetQueueBack** in the **MoveOptionQueue** class and **ResetQueueBackAfterUndo** in the **Player** class.

Introduce a new option that allows a player to undo their last move (after they have seen the result of it and before the next player makes their move), undoing any score gained or lost in that move and returning the game to its previous state. Undoing a move costs a player 5 points. After undoing a move, 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.

1. 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 in the queue. The method should take one parameter, **Position**, which is the place to which the last element of the queue will be restored.
2. Create a new method in the Player class called **ResetQueueBackAfterUndo**. This method should call the newly created **ResetQueueBack** method on the **Queue** attribute in the **Player** class. The method should take one parameter, **Position**, which is the choice that the player made from the menu.

**Task 2**

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

1. If a move is legal, store the player score prior to the move.
2. After displaying the board as a result of the move, give the player the option to undo it.
3. If they choose to undo then: return the player score to the stored pre-move score, subtract 5 points and restore the board and the player’s queue back to their pre-move states.

**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 and play a ‘Ryott’.
* show the game board after the undo and the score set correctly and that player one can choose a new move.

**Evidence that you need to provide:**

* PROGRAM SOURCE CODE showing changes made to the PlayGame method in the Dastan class
* 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

# Task 9

**Task 9** **Marks:** 7

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

It also refers to a new attribute **SahmUsed** in the **Player** class along with creating two new methods, **GetSahmStatus** and **SetSahmUsed**, which operate as the accessor and mutator (getter/setter) methods for the newly created **SahmUsed** attribute.

Chart, line chart

Description automatically generatedImplement a new ‘Sahm’ move option (Arrow). The Sahm can only be fired once in a game per player and is fired instead of a piece moving. A Sahm can be fired by any piece at any position on the board. The Sahm fires in a straight line forwards from the player, all the way across the board, destroying any opponent piece(s) in its way except a Kotla, which is strong enough to withstand an attack and protect any piece inside it. The Sahm is only made available to a player through the MoveOptionOffer 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.

**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.

1. Modify the **CreateMoveOptionOffer** method to offer the new ‘Sahm’ move first.
2. Create the new private **CreateSahmMoveOption** method to allow the player to select which piece fires the Sahm and add only one possible new move Move(0,0) for this method.

**Note:** The move should not actually move the piece anywhere, i.e. 0 rows and columns.

1. Modify the **CreateMoveOption** method to handle Sahm.

**Task 2**

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

1. Add a new **SahmUsed** attribute in the **Player** class which is initialised to False.
2. Create two new methods, **GetSahmStatus** and **SetSahmUsed**, which operate as the accessor and mutator (getter/setter) methods for the newly created **SahmUsed** attribute.
3. Create a method **ChoiceIsSahm** method which takes a parameter and checks if the move option chosen is a Sahm move, whereupon it returns True.

**(TASK CONTINUES ON THE NEXT PAGE)**

**Task 3**

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

1. Modify **PlayGame** to call the new method **ChoiceIsSahm** and only ask for the start square if it is.
2. Create a new private method in the Dastan class called **CalculateSahmMove** which will calculate the points for a Sahm move and destroy the pieces that are hit (unless they are in a Kotla).
3. Modify **PlayGame** to so that it calls the new method **CalculateSahmMove** to get the points for the Sahm move and destroys the relevant pieces. It should also call the **SetSahmUsed** method 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 22 as the ‘from’ and square 33 as the ‘to’ to diagonally move one piece in front of another player one piece in the Kotla column.
* select 9 from menu for player two to accept the offer. Choose 1 to put it in position 1 and then choose option 1 to select the Sahm move. Choose the piece on square 53 to fire the Sahm and show the updated board with both player one pieces removed from the board by the Sahm fired by player two, but not the Mirza 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 **CreateMoveOptionOffer** and **CreateMoveOption** methods
* PROGRAM SOURCE CODE showing the creation of new **CreateSahmMoveOption**, **ChoiceIsSahm** and **CalculateSahmMove** methods
* PROGRAM SOURCE CODE showing changes made to the **Player** class
* SCREEN CAPTURE(S) showing the required test

# Task 10

**Task 10** **Marks:** 5

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 option, a player can select one of their own pieces to destroy and replace with a second Kotla. A new Kotla can only be placed in the square in which the piece was sacrificed. A player can only replace one of their own pieces. Replacing a piece with a Kotla should use up a player turn and they should not score any points for that turn.

**What you need to do**

**Task 1**

Modify the **PlayGame** method in the **Dastan** class to introduce a new option 7 into the main game playing menu. Allow the player to select a piece which they would like to replace with a new Kotla. Use validation to ensure that the user can only select one of their pieces and it cannot be the Kotla. On confirmation, replace the piece with a second Kotla assigned to the correct team.

**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 square for the new Kotla.
* show the Kotla being placed correctly in square 22, a valid square, and assigned to player one.

**Evidence that you need to provide:**

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