0 1 Write a program that gets **two** words from the user and then displays a message saying if the first word can be created using the letters from the second word or not.

For example:

- The word EAT can be formed from the word ATE as the first word uses one E, one A and one T and the second word also contains one of each of these letters.
- The word EAT can be formed from the word HEART as the second word contains one E, one A and one T which are the letters needed to form the first word.
- The word TO can be formed from the word POSITION as the second word contains one T and (at least) one O which are the letters needed to form the first word.
- The word MEET cannot be formed from the word MEAT as the second word only contains one E and two Es are needed to form the first word.

You may assume that the user will only enter words that consist of upper case letters.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 1 . 1 Your PROGRAM SOURCE CODE.

[12 marks]

0 1 . 2 SCREEN CAPTURE(S) showing the result of testing the program by entering:

- the word NINE followed by the word ELEPHANTINE.
- the word NINE followed by the word ELEPHANT.

[1 mark]

0 2 This question refers to the subroutine `PlayGame`.

The program is to be changed so that it does not always display the same message when the user enters a command that is not recognised.

Adapt the subroutine `PlayGame` so that sometimes the existing message `"Sorry, you don't know how to ***."` is displayed and sometimes the new message `"Sorry, I don't know what *** means."` is displayed. `***` denotes the command entered by the user.

The message to display should be selected randomly by the program and the probability of each of the messages being displayed should be equal. The new message should be laid out exactly as shown – all on one line with all the text, punctuation and spaces in the indicated places.

Test that the changes you have made work:

- run the **Skeleton Program**
- load the file **flag1.gme**
- enter the command `eat` a few times to show that your changes have worked and that the message displayed is randomly selected.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 2 . 1 Your PROGRAM SOURCE CODE for the amended subroutine `PlayGame`.

[4 marks]

0 2 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

0 3 This question refers to the subroutine `GetItem`.

Currently the player can carry any number of items. The game is to be changed so that the player cannot carry more than five items.

Change the subroutine `GetItem` so that if the player is already carrying five items and they try to get an item that they would normally be able to get then the item is not added to their inventory. An appropriate message should be displayed saying that they can't get that item because they are carrying too much. If they are carrying fewer than five items then the subroutine should add the item to the player's inventory.

Test that the changes you have made work:

- run the **Skeleton Program**
- load the file **flag1.gme**
- enter the command `get torch`
- then enter the command `get red die`
- then enter the command `open cupboard door`
- then enter the command `go east`
- and then enter the command `get book`.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 3 . 1 Your PROGRAM SOURCE CODE for the amended subroutine `GetItem`.

[7 marks]

0 3 . 2 SCREEN CAPTURE(S) showing the requested test. You must make sure that evidence for all parts of the requested test by the user is provided in the SCREEN CAPTURE(S).

[1 mark]

---

0 4 This question extends the functionality of the game by modifying the subroutine `PlayGame` and adding a new subroutine called `DropItem`.

An additional command, `drop`, is to be added to the game. When a player uses the `drop` command the item specified after `drop` should be moved from the player's inventory to the location the player is currently in unless the item has a status of fragile, in which case it should be removed from the game as it breaks when dropped.

A message should be displayed stating either that the item has now been dropped, that the item broke when it was dropped or that the item is not in their inventory.

What you need to do

Task 1
Create a new subroutine called `DropItem` that implements all the functionality required for the `drop` command.

Task 2
Modify the `PlayGame` subroutine so that a call is made to the `DropItem` subroutine when the `drop` command has been entered by the user.

Task 3
Test that the changes you have made work:

- run the **Skeleton Program**
- load the file **flag1.gme**
- enter the command `drop apple`
- then enter the command `drop jar`
- then enter the command `examine inventory`
- then enter the command `open green door`
- then enter the command `go north`
- then enter the command `go south`.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 4 . 1 Your PROGRAM SOURCE CODE for the amended subroutine `PlayGame` and the new subroutine `DropItem`.

[12 marks]

0 4 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

---

0 5 This question refers to the subroutine `PlayDiceGame`.

The dice game in the **Skeleton Program** is very simple – the player and the other character both roll a die and whoever gets the highest number wins.

The `PlayDiceGame` subroutine is to be changed to a more complex dice game. All other parts of the subroutine are to remain the same – no changes should be made to checking that it is possible to play the dice game or to what happens when the player wins, loses or draws the game.

In the new game the player rolls their die three times. The largest of the numbers they roll is multiplied by 100 and the second largest of the numbers they roll is multiplied by 10. The results of these two multiplications are added together along with the smallest of the numbers they rolled to give the player's total score.

The other character's die is also rolled three times. The second number they roll is multiplied by 10 and the third number they roll is multiplied by 100. The results of these two multiplications are added together along with the first number they rolled to give their total score.

As with the original dice game, whoever has the highest total score wins the game and if the scores are the same the result is a draw.

The program should display the result of each die roll for each player and the total scores obtained by both players.

New dice game examples

- The player rolls a 4, then a 2 and then a 3. Their total score is 4×100 + 3×10 + 2 = 432. The other character rolls a 5, then a 2, then a 3. Their total score is 3×100 + 2×10 + 5 = 325. The player's score is higher so they win the game.

- The player rolls a 2, then a 3 and then a 4. Their total score is 4×100 + 3×10 + 2 = 432. The other character rolls a 6, then a 1, then a 4. Their total score is 4×100 + 1×10 + 6 = 416. The player's score is higher so they win the game.

What you need to do

Task 1
Modify the subroutine `PlayDiceGame`.

Task 2
Test that the changes you have made work by playing the new dice game with the guard twice:

- run the **Skeleton Program**
- load the file **flag2.gme**
- enter the command `playdice guard`
- choose a minimum value of 1 and a maximum value of 6 for the player's die each time the die is rolled
- if the player wins the game take any item from the guard; if the guard wins the game and the randomly selected item to take from the player is the player's die then you will need to re-run the **Skeleton Program** and load the file **flag2.gme** again before completing the next part of the test
- enter the command `playdice guard`
- choose a minimum value of 1 and a maximum value of 6 for the player's die each time the die is rolled.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 5 . 1 Your PROGRAM SOURCE CODE for the amended subroutine `PlayDiceGame`.

[8 marks]

0 5 . 2 SCREEN CAPTURE(S) showing the tests described in **Task 2**.

[1 mark]

---

0 6 Write a program that asks the user how many numeric digits they would like to enter and then gets the user to enter that number of numeric digits.

The program should calculate and display the number of times the most frequently entered numeric digit was input.

> **Example**
>
> If the user says they are going to enter four digits and then enters the digits 3, 4, 5 and 3, the program should display the value 2 as the most frequently entered digit was 3 and that digit was entered twice.

If more than one numeric digit had the same frequency and was the most frequently entered then instead of displaying the frequency, a message saying "`Data was multimodal`" should be displayed.

A numeric digit is 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9

You may assume that the number that the user enters to state how many numeric digits there will be and the numeric digits entered by the user are all valid.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 6 . 1 Your PROGRAM SOURCE CODE.

[12 marks]

0 6 . 2 SCREEN CAPTURE(S) showing the result of testing the program by entering:

- the number 6 then the numeric digits 0, 1, 2, 1, 2 and 1
- the number 5 then the numeric digits 0, 1, 2, 2 and 1

[1 mark]

---

0 7 This question refers to the method `AddCompany` in the `Simulation` class.

The program is to be changed so that it checks that the company name entered is valid. The company must have a name and it cannot be the same as the name of an existing company. The program should keep doing these checks until a valid name for the company has been entered.

What you need to do

Task 1
Modify the method `AddCompany` so that an appropriate error message is displayed if the user presses the Enter key without entering a company name when they are asked to enter the name of the company.

Task 2
Modify the method `AddCompany` so that an appropriate error message is displayed if the user enters the name of a company that is already being used for a company in the simulation when they are asked to enter the name of the company.

You may use the method `GetIndexOfCompany` to find out if a company name has already been used, though you do not have to use this method in your answer.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- choose to use a normal size settlement
- choose to use the default companies
- choose the option to add a new company from the menu
- press the Enter key without entering a name for the new company
- type in the name `AQA Burgers` and press the Enter key
- type in the name `In Jest` and press the Enter key.

Note: you will get three marks for your program code if either Task 1 or Task 2 has been successfully completed and you will get five marks if both Task 1 and Task 2 have been successfully completed.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 7 . 1 Your PROGRAM SOURCE CODE for the amended method `AddCompany`.

[5 marks]

0 7 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

0 8 This question extends the functionality of the simulation by adding a new type of household, an affluent household, that eats out considerably more often than other households.

What you need to do

Task 1
Create a new class called `AffluentHousehold` that inherits from the `Household` class. The constructor for this new class should make a call to the constructor of the `Household` class and then set the value of `ChanceEatOutPerDay` to 1

Task 2
Modify the `AddHousehold` method in the `Settlement` class so that it creates an `AffluentHousehold` object instead of a `Household` object if the randomly-generated value for `X` is less than 100

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- choose to use a normal size settlement
- choose to use the default companies
- then choose the menu option to display the details of the households and check that there is a mixture of both types of household in the settlement.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 8 . 1 Your PROGRAM SOURCE CODE for the amended method `AddHousehold` and the new class `AffluentHousehold`.

[7 marks]

0 8 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

---

0 9 This question extends the Skeleton Program by adding the facility for a company to obtain a loan to increase their balance.

When a company chooses to get a loan, 10 000 is added to their balance. The company should only be allowed to get a loan if they do not already have a loan that has not been paid back. The user will be asked to specify the daily interest rate for the loan.

The company can pay back all or part of the loan at any time. The program should keep track of the amount of the loan that the company still needs to pay back.

Each time the simulation is advanced by a day, the company will have their balance reduced by an amount equivalent to the interest rate applied to the amount of the loan minus any repayments that have been made. **Figure 5** shows an example of the calculation.

Figure 5

A company has repaid 1500 of the loan meaning that it still owes 8500. The interest rate for the loan was 0.0005% and the company's current balance is 4000. The simulation is advanced by a day and the company's balance is reduced by 0.0425 to 3999.9575 (0.0425 is 0.0005% of 8500).

To get full marks on this question you will need to make use of encapsulation to hide attributes in the `Company` class.

What you need to do

Task 1
Add attributes to the `Company` class named `LoanBalance` and `InterestRate`. `LoanBalance` should be used to track the amount of the loan that the company needs to pay back. `InterestRate` should be used to store the interest rate at which the loan was taken.

Task 2
Modify the `ModifyCompany` method in the `Simulation` class so that there are options on the menu to get a loan and to pay back a loan.

Task 3
Modify the Skeleton Program so that if a company has a loan balance of 0 then it can take out a loan. When it does:

- 10 000 should be added to `Balance` and `LoanBalance`
- `InterestRate` should be set to the value specified by the user.

Task 4
Modify the Skeleton Program so that when a company chooses to pay back a loan the user is asked how much to pay back and the values of `LoanBalance` and `Balance` are reduced by this amount. You **do not** need to check that the value entered is sensible or that the company has a loan to pay back.

Task 5
Modify the `ProcessDayEnd` method in the `Company` class so that the company's daily interest for the loan is subtracted from the company's balance.

Task 6
Test that the changes you have made work:

- run the Skeleton Program
- choose to use a normal size settlement
- choose to use the default companies
- give the company AQA Burgers a loan with a percentage interest rate of 0.015
- advance the simulation
- get the company AQA Burgers to pay back 500 of the loan
- advance the simulation
- then choose the menu option to display the details of the companies.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 9 . 1 Your PROGRAM SOURCE CODE for the amended parts of your Skeleton Program. You should include the entire code for each method you have changed/created.

[10 marks]

0 9 . 2 SCREEN CAPTURE(S) showing the requested test. You only need to show the result of choosing the menu option to display the details of the companies. Evidence for the earlier parts of the test is not needed.

[1 mark]

---

1 0 Currently, the Skeleton Program creates a delivery route for a company based on the order in which a company's outlets have been created – the route starts with the oldest outlet, then goes to the second oldest outlet and so on until all the outlets have been included in the route.

The route produced specifies the order in which the outlets are visited when making deliveries – the longer the route, the higher the delivery cost for the company.

This question extends the Skeleton Program by using a greedy algorithm to try to reduce the total distance of the delivery route. A greedy algorithm is one that takes the choice which seems to be the best each time; it is not guaranteed to find the optimal solution.

Figure 6 shows how the greedy algorithm should work.

Figure 6

The greedy algorithm should:

1. start by adding the oldest outlet (the one in position 0 in `Outlets`) to the route
2. then find the company's outlet that is nearest (has the smallest distance) to the outlet that was last added to the route, and which has not yet been added to the route, and add this to the end of the route
3. repeat step 2 until all the company's outlets have been added to the route.

Figure 7 shows how the delivery cost is arrived at for the company Paltry Poultry which has four outlets. The outlet numbers (0–3) refer to the index of an outlet within the `Outlets` data structure for that company. This example may help you check your code works correctly.

Figure 7

- The oldest outlet (outlet 0) is at coordinates (800,390) within the settlement; this will be the first outlet on the route.
- From this outlet the closest outlet is outlet 2 which is at (820,370) so outlet 2 will be added as the second outlet on the route.
- From outlet 2 the closest outlet not yet in the route is outlet 3 which is at (800,600) so outlet 3 will be added as the third outlet on the route.
- From outlet 3 the closest outlet not yet in the route is outlet 1 which is at (400,390) so outlet 1 will be added as the fourth outlet on the route.
- The correct order that the outlets should be visited is 0, 2, 3, 1
- The delivery cost would be 6.96708

What you need to do

Task 1
Create a new method `GetOrderedListOfOutlets` in the `Company` class. The new method should return a list that represents the route created using the algorithm in **Figure 6**. Each outlet should be represented in the list by the index of its position in `Outlets`.

Task 2
Modify the `CalculateDeliveryCost` method in the `Company` class so that it uses the list returned by the new method `GetOrderedListOfOutlets` instead of the list returned by the method `GetListOfOutlets`.

Task 3
Test that the changes you have made work:

- run the **Skeleton Program**
- choose to use a normal size settlement
- choose to use the default companies
- choose the menu option to display the details of the companies.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

10.1 Your PROGRAM SOURCE CODE for the amended method `CalculateDeliveryCost` and for the new method `GetOrderedListOfOutlets`.

[11 marks]

10.2 SCREEN CAPTURE(S) showing the test described in **Task 3**. The screen capture should show the delivery cost for **AQA Burgers**.

[1 mark]

---

1 1 A Harshad number is a positive integer which is exactly divisible by the sum of its digits. The first twelve Harshad numbers are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12 and 18

- 36 is a Harshad number. The digits of 36 are 3 and 6; the sum of these digits is 9 as 3 + 6 = 9 and 36 is exactly divisible by 9 (36 ÷ 9 = 4)
- 300 is a Harshad number. The digits of 300 are 3, 0 and 0; the sum of these digits is 3 as 3 + 0 + 0 = 3 and 300 is exactly divisible by 3 (300 ÷ 3 = 100)
- 15 is not a Harshad number. The digits of 15 are 1 and 5; the sum of these digits is 6 as 1 + 5 = 6 and 15 is not exactly divisible by 6

Write a program that asks the user to enter a number, *n*, and will then calculate and display the *n*th Harshad number.

Example

If the user enters the number 12 then the program should calculate and display the twelfth Harshad number. The twelfth Harshad number is 18

You may assume that the number that the user enters will be a positive integer.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 1 . 1 Your PROGRAM SOURCE CODE.

[12 marks]

1 1 . 2 SCREEN CAPTURE(S) showing the result of testing the program by entering the number `600`

[1 mark]

---

1 2 This question refers to the subroutine `DestroyPiecesAndCountVPs` in the `HexGrid` class.

The victory point scoring system for the game is to be changed so that at the end of each turn both players gain additional victory points based on how many LESS pieces they have on the board.

What you need to do

Task 1
Modify the subroutine `DestroyPiecesAndCountVPs` so that if a piece has **not** been destroyed it checks to see if it is a LESS piece. If it is a LESS piece the number of victory points awarded to the player to whom that piece belongs should be increased by one.

Task 2
Test that the changes you have made work:

- run the Skeleton Program
- choose to load a game
- enter the filename **game1.txt**
- keep pressing the Enter key until **both** players have had a turn and the grid has been shown at the start of Player One's second turn.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 2 . 1 Your PROGRAM SOURCE CODE for the amended subroutine `DestroyPiecesAndCountVPs`.

[5 marks]

1 2 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

---

1 3 This question adds a new type of piece to the game, a ranger, that moves differently to the other pieces.

A ranger can move in the same way as a standard (serf) piece but can also move directly to any available forest tile in the grid if the ranger is currently in a forest tile. The cost of making this type of move is one fuel.

What you need to do

Task 1
Create a new class called `RangerPiece` that is a subclass of the `Piece` class. The constructor for this new class should make a call to the constructor of the `Piece` class and then set the value of `PieceType` to `R`.

Task 2
Create a subroutine `CheckMoveIsValid` in the new `RangerPiece` class that overrides the subroutine from the base class and allows a ranger piece to move in the way described.

Task 3
Modify the subroutine `AddPiece` in the `HexGrid` class so that it creates a new `RangerPiece` if `TypeOfPiece` is `Ranger`.

Task 4
Modify the subroutine `SetupDefaultGame` so that Player One has a ranger piece in tile 8 instead of a serf piece.

Task 5
Test that the changes you have made work:

- run the Skeleton Program
- choose the default game
- enter the command `move 8 12`
- enter the command `move 12 2`
- enter the command `move 2 3`
- then press the Enter key so that the grid is displayed showing the results of these commands.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 3 . 1 Your PROGRAM SOURCE CODE for the new class `RangerPiece` and the amended subroutine `AddPiece`.

[7 marks]

1 3 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

---

1 4 This question extends the Skeleton Program by adding a new command to the game that allows a player to burn lumber to turn it into fuel.

If a player uses the `burn` command when they have lumber in their supply a random integer between one and the amount of lumber they have in their supply is generated. The amount of lumber in their supply is decreased by this random integer with the amount of fuel in their supply being increased by the same amount.

If a player uses the `burn` command when they do not have any lumber in their supply the message `Cannot burn lumber` is displayed and no fuel is created.

What you need to do

Task 1
Modify the `CheckCommandIsValid` subroutine so that it returns `True` if the command was `burn`.

Task 2
Modify the `ExecuteCommand` subroutine in the `HexGrid` class so that when a player chooses the `burn` command it returns the string `Cannot burn lumber` if the player does not have any lumber in their supply.

If the player does have lumber in their supply it:

- generates a random integer between one and the amount of lumber in the player's supply
- decreases the amount of lumber in the player's supply by the random number generated
- increases the amount of fuel in the player's supply by the random number generated
- returns the string `Command executed`.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- choose the default game
- enter the command `burn`
- press the Enter key twice.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 4 . 1 Your PROGRAM SOURCE CODE for the amended subroutines `CheckCommandIsValid` and `ExecuteCommand`.

[8 marks]

1 4 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

---

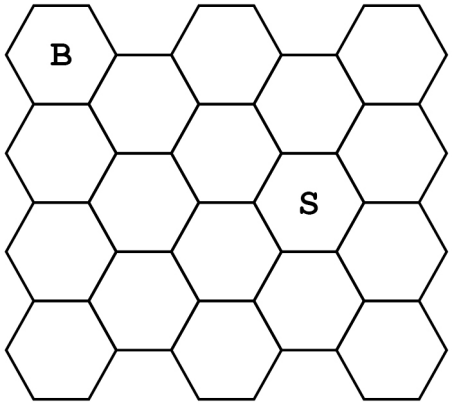
1 5 A new feature called 'fog of war' is to be added to the Skeleton Program. Fog of war means that each player will only be shown the location of a piece belonging to their opponent if that piece is near one of their own pieces. A piece is near another piece if it is two or fewer cells away on the grid.

The player will still be shown the terrain that is in all the tiles.

Figures 7 to **11** show an example of how the fog of war feature should work.

Figure 7 shows the current positions of Player One's pieces.

Figure 7



In **Figure 8** the shaded tiles are those that are two or fewer cells away from Player One's baron piece.

In **Figure 9** the shaded tiles are those that are two or fewer cells away from Player One's serf piece.

Figure 8

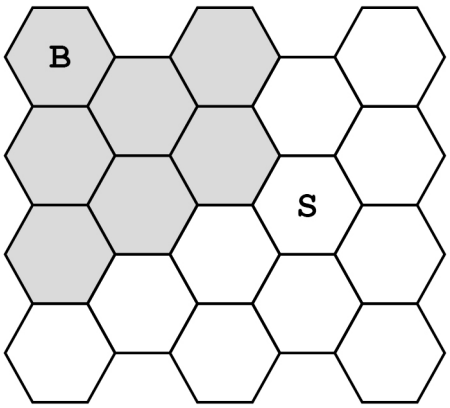


Figure 9

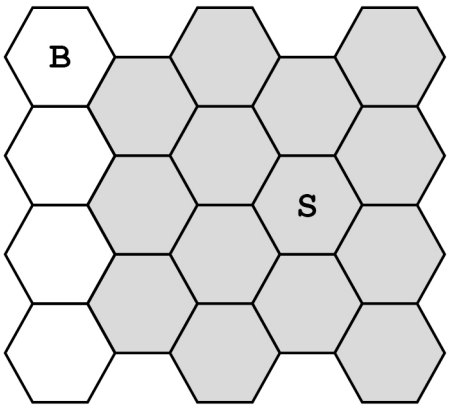


Figure 10 shows the positions of Player One's pieces and the positions of Player Two's pieces.

Figure 11 shows what Player One should see when they are shown the grid when the fog of war feature has been implemented.

- Player Two's baron piece can be seen because it is within two cells of Player One's serf piece.
- Player Two's LESS piece can be seen because it is within two cells of both Player One's serf piece and Player One's baron piece.
- Player Two's serf piece can be seen because it is within two cells of Player One's baron piece.
- Player Two's PBDS piece cannot be seen because it is not within two cells of any of Player One's pieces.

Figure 10

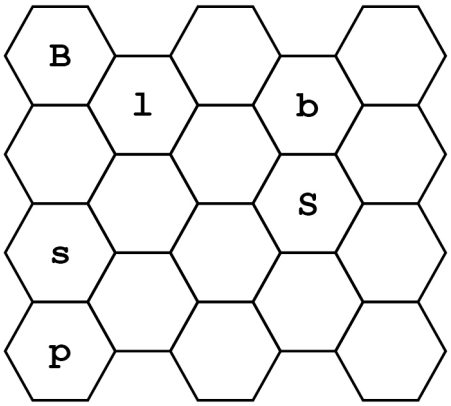
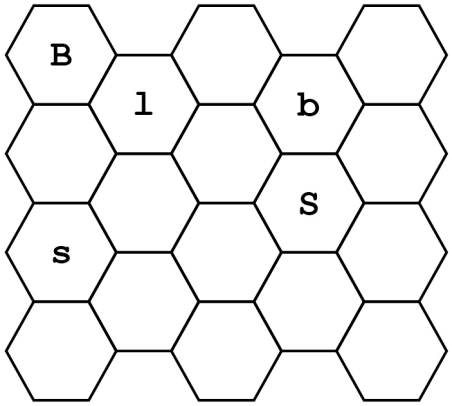


Figure 11



What you need to do

Task 1
Create a new subroutine `GetFogOfWar` in the `HexGrid` class.

The new subroutine should take the index of a tile in `Tiles` and return `False` if the player whose turn it is has any piece which is two or fewer cells away from that tile as the tile is not hidden because of fog of war. Otherwise, it should return `True` as this tile is hidden because of fog of war.

Task 2
Modify the `GetPieceTypeInTile` subroutine in the `HexGrid` class so that it uses the `GetFogOfWar` subroutine to determine if this tile is affected by fog of war.

If the contents of this tile would be hidden because of fog of war, it should return the string consisting of a single space character.

If the contents would not be hidden because of fog of war, the existing functionality of the `GetPieceTypeInTile` subroutine should not be changed.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- choose to load a game
- enter the filename **game1.txt**
- keep pressing the Enter key until the grid has been shown at the start of Player Two's first turn.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

15.1 Your PROGRAM SOURCE CODE for the amended subroutine `GetPieceTypeInTile` and for the new subroutine `GetFogOfWar`.

[13 marks]

15.2 SCREEN CAPTURE(S) showing the requested test. The screen capture should show all the output displayed by the program.

[1 mark]

---

1 6 Write a program that asks the user to enter a string. It should then change the order of the vowels in the string and display the result.

If there are *n* vowels in the string, the 1st vowel in the string should swap with the *n*th vowel in the string, the 2nd vowel in the string should swap with the (*n-1*)th vowel in the string, and so on.

The letters `a`, `e`, `i`, `o` and `u` are the only vowels.

Examples

If the user enters the string `horse` then the program should display the string `herso`.

If the user enters the string `goose` then the program should display the string `geoso`.

If the user enters the string `pinkfairyarmadillo` then the program should display the string `ponkfiaryarmidalli`.

If the user enters the string `nakedmolerat` then the program should display the string `nakedmolerat`.

If the user enters the string `lynx` then the program should display the string `lynx`.

If the user enters the string `pig` then the program should display the string `pig`.

You may assume the string that the user enters will only contain lowercase letters.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 6 1 Your PROGRAM SOURCE CODE.

[12 marks]

1 6 2 SCREEN CAPTURE(S) showing the results of three tests of the program by entering the strings `persepolis`, `darius` and `xerxes`.

[1 mark]

1 7 How many subroutines in the Skeleton Program access external data files?

[1 mark]

1 8 This question refers to the subroutine `GetDiscardOrPlayChoice` in the `Breakthrough` class.

The program is to be changed so that it checks that the choice entered after the player has chosen to use a card is valid. `D` and `P` are the only valid choices. The program should keep getting the player to enter a value until a valid choice has been made.

What you need to do

Task 1
Modify the subroutine `GetDiscardOrPlayChoice` so it checks that the value entered by the player is valid. An appropriate error message should be displayed if an invalid value is entered and the user should be made to enter another value.

Task 2
Test that the changes you have made work:

- run the Skeleton Program
- play a new game
- enter `U`
- enter `2`
- enter `L`
- enter `D`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 8 . 1 Your PROGRAM SOURCE CODE for the amended subroutine `GetDiscardOrPlayChoice`.

[4 marks]

1 8 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

1 9 This question extends the Skeleton Program so that it displays some information about the contents of the deck to the player.

The program needs to be modified so that it displays messages telling the player how many cards are in the deck and how many tool cards there are in the deck. A tool card is a pick, file, or key.

What you need to do

Task 1
Modify the `PlayGame` subroutine in the `Breakthrough` class so that a message is displayed telling the player how many cards there are in the deck.

This message should be displayed after the message telling the player what their current score is and before the contents of the player's hand are displayed.

Task 2
Create a subroutine `GetNumberOfToolCards` in the `CardCollection` class that calculates how many tool cards there are in the `Cards` data structure. It should return the calculated value to the calling routine.

Task 3
Modify the `PlayGame` subroutine in the `Breakthrough` class so that a message is displayed telling the player how many tool cards there are in `Deck`. This message should use the value returned by calling the `GetNumberOfToolCards` subroutine for `Deck`.

This message should be displayed after the message telling the player what their current score is and before the contents of the player's hand are displayed.

Task 4
Test that the changes you have made work:

- run the Skeleton Program
- play a new game
- enter `U`
- enter `2`
- enter `D`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 9 . 1 Your PROGRAM SOURCE CODE for the new subroutine `GetNumberOfToolCards` and the amended `PlayGame` subroutine.

[8 marks]

1 9 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

2 0 This question extends the Skeleton Program by allowing a player to use a blasting cap to complete any challenge on the current lock.

The player will only be able to do this once per game.

When the player chooses to use the blasting cap, the program should check if the player has already used the blasting cap. If they have not used the blasting cap, the program should:

- change the status of the blasting cap to show that it has been used
- ask the player to enter the position of the challenge on the current lock they would like to use the blasting cap on (1 for the first challenge, 2 for the second challenge, etc)
- check:
  - that the position entered is less than or equal to the number of challenges on the current lock
  - that the challenge in that position has not already been met.
- if both these checks are passed:
  - mark the challenge as being met
  - display a message saying the blasting cap was used successfully
  - display the details for the current lock.
- if either of these checks fail, then the blasting cap has been wasted and the game continues.

If the player tries to use the blasting cap when they have already used it then nothing changes and the game continues.

What you need to do

Task 1
Modify the `GetChoice` subroutine in the `Breakthrough` class so the message displayed shows the blasting cap option.

Task 2
Modify the `PlayGame` subroutine in the `Breakthrough` class so that there is a blasting cap that works in the way described.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- enter `L`
- choose to use a blasting cap
- choose to complete the third challenge on the current lock
- choose to use a blasting cap.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 0 . 1 Your PROGRAM SOURCE CODE for the amended subroutines `GetChoice` and `PlayGame`.

[9 marks]

2 0 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

2 1 This question adds a new type of difficulty card to the game, a trap.

When the player draws a trap card from the deck, it works as follows:

- if no challenges from the current lock have been met, the trap card works in the same way as a difficulty card and the player's choice to lose a key or discard five cards is executed
- if one or more challenges from the current lock have been met, one of the met challenges is randomly chosen and has its status changed so it is no longer met. This happens instead of executing the player's choice to lose a key or discard five cards from the deck.

Trap cards are used in the game instead of difficulty cards when the player chooses to load a game from a file. They are **not** used when the player chooses to play a new game.

What you need to do

Task 1
Create a new class called `TrapCard` that is a subclass of the `DifficultyCard` class.

The constructor for this new class should set the value of `CardNumber` to the value of the constructor's parameter and set the value of `CardType` to `Trp`.

Create a subroutine `Process` in the `TrapCard` class that overrides the subroutine from the `DifficultyCard` class and allows a trap card to work in the way described.

Task 2
Modify the `SetupCardCollectionFromGameFile` subroutine in the `Breakthrough` class so that it creates `TrapCard`s instead of `DifficultyCard`s.

Task 3
Modify the `GetCardFromDeck` subroutine in the `Breakthrough` class so that if the top card of the deck has a `CardType` of `Trp`, that card is treated in the same way as if the top card of the deck had a `CardType` of `Dif`.

When the top card of the deck has a `CardType` of `Trp`, the message "`Trap!`" should be displayed.

Task 4
Test that the changes you have made work:

- run the Skeleton Program
- enter `L`
- enter `U`
- enter `1`
- enter `P`
- enter `U`
- enter `1`
- enter `P`
- enter `D`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 1 1 Your PROGRAM SOURCE CODE for the amended subroutine `GetCardFromDeck`, the amended subroutine `SetupCardCollectionFromGameFile` and the new class `TrapCard`.

[12 marks]

2 1 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

2 2 Write a program that gets the user to enter a string. It should keep getting the user to enter a string until they enter a valid string. Each time they enter a string an appropriate message should be displayed telling them whether the string they entered is valid or not.

For a string to be valid:

- it must be between five and seven characters in length (inclusive)
- it must consist only of uppercase letters
- it must contain only unique characters (ie no character appears in the string more than once)
- the sum of the ASCII codes of the characters in the string must be between 420 and 600 (inclusive).

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 2 . 1 Your PROGRAM SOURCE CODE

[12 marks]

2 2 . 2 SCREEN CAPTURE(S) showing the results of testing the program by entering the strings `BOIL`, `BRAisE`, `ROAST`, `BLANCH` and `PRESSURECOOK`. You will need to execute your program more than once to test all of the strings.

[1 mark]

2 3 This question refers to the method `UseMoveOptionOffer` in the `Dastan` class.

The program is to be changed so that it checks the choice made by the player about which move option to replace is valid. A choice is valid if it is between 1 and 5 inclusive.

The program should keep getting the player to enter a value until a valid choice has been made. Each time an invalid value is entered an appropriate error message should be displayed.

What you need to do

Task 1
Modify the method `UseMoveOptionOffer` so it checks that the value entered by the player is valid. An appropriate error message should be displayed if an invalid value is entered and the user should be made to enter another value.

You may assume that the user will only enter a single numeric digit.

Task 2
Test that the changes you have made work:

- run the Skeleton Program
- enter 9
- enter 6
- enter 4

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 3 . 1 Your PROGRAM SOURCE CODE for the amended method `UseMoveOptionOffer`.

[4 marks]

2 3 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

2 4 This question extends the Skeleton Program by allowing a player to choose a bhukampa (earthquake).

After choosing a bhukampa the player can then choose a move option, choose to use the move option offer or choose a bhukampa again.

When a bhukampa is chosen the following steps are repeated **five** times:

- two **different** squares are randomly selected
- the positions on the board of those two squares are swapped.

The player's score is then reduced by 15 and the new state of the game is displayed.

What you need to do

Task 1
Create a new method in the `Dastan` class called `ProcessBhukampa` that swaps the squares in the way described.

Task 2
Modify the `PlayGame` method in the `Dastan` class so that if the player enters 8 when asked for their move option, the new method `ProcessBhukampa` is called, the current player's score is reduced by 15 and the new game state is displayed.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- enter 8

The results of your test should show at least one piece or kotla now being in a different location. If your test does not show this (because all the squares randomly selected by your program were empty), then keep doing this test until that is not the case.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 4 . 1 Your PROGRAM SOURCE CODE for the new method `ProcessBhukampa` and the amended method `PlayGame`.

[9 marks]

2 4 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

2 5 This question extends the Skeleton Program so that there is a new type of square – a gacaka (trap).

When a player makes a move that means a piece moves into a gacaka, the value of `PointsIfCaptured` of the piece is increased by 2 and a message saying "`Trap!`" is displayed. A player who has a piece in a gacaka has their score decreased by 3 each turn until the piece is moved out of the gacaka.

The location of the gacaka is not shown on the board.

What you need to do

Task 1
Create a new class called `Gacaka` that is a subclass of the `Square` class.

Create two methods `SetPiece` and `GetPointsForOccupancy` in the `Gacaka` class that allow a gacaka to work in the way described.

The method `SetPiece` in the `Gacaka` class should override the method from the `Square` class.

The method `GetPointsForOccupancy` in the `Gacaka` class should override the method from the `Square` class.

Task 2
Modify the `CreateBoard` method in the `Dastan` class so that row four, column four is a gacaka instead of a square.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- enter 3
- enter 24
- enter 44
- enter 1
- enter 54
- enter 44

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 5 . 1 Your PROGRAM SOURCE CODE for the new class `Gacaka`, the amended `CreateBoard` method and any other methods you have modified or created when answering this question.

[8 marks]

2 5 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

2 6 This question extends the program by telling the player how many legal moves they have in the current state of the game.

A legal move is when, using one of the first three move options in their move option queue, a player's piece will move to a square that contains an opponent's piece or to a square that does not contain a piece. There are 45 legal moves at the start of the game for the first player.

When answering this question, you should ignore the option to use the move option offer and the option to use a bhukampa (from Question **14**) as these do not count as moves.

What you need to do

Task 1
Create a new method called `GetNoOfPossibleMoves` in the `Player` class that takes `Board` as a parameter and returns the total number of possible legal moves for that player based on the contents of the squares in the `Board` list.

Task 2
Modify the `DisplayState` method in the `Dastan` class so that it displays the value returned by a call to `GetNoOfPossibleMoves` for the current player.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- enter `1`
- enter `22`
- enter `12`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 6 1 Your PROGRAM SOURCE CODE for the new method `GetNoOfPossibleMoves` and the amended method `DisplayState`.

[12 marks]

2 6 2 SCREEN CAPTURE(S) showing the requested test with the number of legal moves for the second player displayed.

[1 mark]

---

2 7 Write a program that gets the user to enter an integer. It should keep doing this until they enter a value greater than 0.

The program should then tell the user if they have entered a perfectly bouncy number, a bouncy number or a number that is not bouncy.

A **bouncy number** is a number that is not an increasing number and not a decreasing number.

An **increasing number** is one where each digit is greater than or equal to the previous digit in the number.

A **decreasing number** is one where each digit is less than or equal to the previous digit in the number.

A **perfectly bouncy number** is a bouncy number in which the number of digits that are followed by a larger digit is equal to the number of digits that are followed by a smaller digit.

Examples

- 13578 is not a bouncy number because it is an increasing number.
- 973 is not a bouncy number because it is a decreasing number.
- 98657 is a bouncy number.
- 1111 is not a bouncy number because it is both an increasing number and a decreasing number.
- 13421 is a perfectly bouncy number as exactly two digits are followed by a larger digit and there are also exactly two digits followed by a smaller digit.
- 1829361 is a perfectly bouncy number as exactly three digits are followed by a larger digit and there are also exactly three digits followed by a smaller digit.
- 13333331 is a perfectly bouncy number as there is exactly one digit followed by a larger digit and also exactly one digit followed by a smaller digit.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 7 1 Your PROGRAM SOURCE CODE

[12 marks]

2 7 2 SCREEN CAPTURE(S) showing the results of testing the program by entering the integers:

- `-3`
- `14982`
- `1234`

You will need to execute your program more than once to test all of the integers.

[1 mark]

2 8 This question refers to the method `AttemptPuzzle` in the `Puzzle` class.

The Skeleton Program is to be changed so that it checks the choice made by the user for the **column** number. A choice is valid if it is between one and the number of columns in the puzzle inclusive. You do **not** need to add checks for the row number.

The program should keep getting the user to enter a value until a valid choice has been made.

What you need to do

Task 1
Modify the method `AttemptPuzzle` so it checks that the value entered by the user is valid. If an invalid value is entered, the user should be made to enter another value.

Task 2
Test that the changes you have made work:

- run the Skeleton Program
- press the Enter key
- enter `1`
- enter `10`
- enter `4`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 8 . 1 Your PROGRAM SOURCE CODE for the amended method `AttemptPuzzle`.

[4 marks]

2 8 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

---

2 9 This question extends the Skeleton Program by tracking the user's average score and highest score for the puzzle.

After the user chooses not to do another puzzle, the program should display the user's highest score achieved on the puzzle and the user's average score. Your amended Skeleton Program should calculate the average and highest scores for any number of puzzle attempts.

Example
If the user:

- gets a score of 30 when they complete a puzzle
- decides to do another puzzle
- gets a score of 20 for the second puzzle
- decides not to do another puzzle

the program should display messages saying that their highest score was 30 and their average score was 25

What you need to do

Task 1
Modify the `Main` subroutine so that after the user decides not to attempt another puzzle their highest score for the puzzles attempted and their average score for the puzzles attempted are displayed.

Task 2
Test that the changes you have made work:

- run the Skeleton Program
- load the file `puzzle1`
- enter `5`
- enter `5`
- enter `X`
- enter `Y`
- load the file `puzzle1`
- enter `1`
- enter `4`
- enter `X`
- enter `N`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 9 1 Your PROGRAM SOURCE CODE for the amended subroutine `Main`.

[5 marks]

2 9 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

---

3 0 This question extends the Skeleton Program by giving the user an option to shift all the cells in one row in the puzzle one place to the left.

When shifting left, the first cell in the row will move to the last position in the row.

Figure 8 shows an example puzzle and **Figure 9** shows the result obtained from shifting all the cells in row 2 from the puzzle in **Figure 8** one place to the left.

Figure 8

	1	2	3	4	5	6	7	8
8			@					
7			@	@				
6								
5							@	
4								
3		@		@				
2	X	X		X				
1						@		

Figure 9

	1	2	3	4	5	6	7	8
8			@					
7			@	@				
6								
5							@	
4								
3		@		@				
2	X		X					X
1						@		

After the cells have been shifted, the user's score should be reduced by 20 and the new state of the grid and the user's new score should be displayed. No other changes to the user's score should be made.

Shifting the cells does not count as a turn. After choosing to shift cells, the user should carry on with the rest of their turn by selecting a cell to place a symbol in.

When answering this question, you should make sure your program code will work for any size of puzzle grid.

What you need to do

Task 1

Create a new method called `ShiftCellsInRowLeft` in the `Puzzle` class that takes an integer parameter that specifies the row number to use with the shift.

Each cell, in the row indicated by the parameter, should be moved in `Grid` so that it is one place to the left; the leftmost cell should shift to the end of the row.

Task 2
Modify the `AttemptPuzzle` method in the `Puzzle` class so that it gives the user the option to shift the cells in a row. Inside the iteration structure used to get a row number from the user your program should:

- display a modified message telling the user to enter a row number or to enter `0` to shift cells
- if the user enters `0`:
  - ask the user to enter the row number for the row to shift
  - call the method `ShiftCellsInRowLeft` with the number entered as a parameter
  - subtract 20 from the user's score
  - display the new grid
  - display the new score
  - set `Valid` to be false.

Task 3
Test that the changes you have made work:

- run the Skeleton Program
- load the file `puzzle2`
- enter `0`
- enter `1`
- enter `1`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

3 0 . 1 Your PROGRAM SOURCE CODE for the new method `ShiftCellsInRowLeft` and the amended method `AttemptPuzzle`.

[11 marks]

3 0 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

---

3 0 . 3 The movement of the cells within a row can be described using vectors.

State the 2-vector that describes the movement for all but the leftmost cell in the row.

[1 mark]

3 0 . 4 State the 2-vector that describes the movement for the leftmost cell in the row.

Your vector should work for any grid size.

[1 mark]

3 1 This question extends the Skeleton Program so that there is a new type of cell – a countdown cell.

A countdown cell has a timer associated with it that decreases by one each time the user tries to place a symbol in the puzzle grid.

The location of a countdown cell is shown by a numeric digit representing the current value of its timer. When its timer reaches zero, the cell's symbol changes to an @.

A countdown cell should be added to the board each time the user's score is increased. The location for the countdown cell should be an empty cell on the grid and selected randomly.

What you need to do

Task 1
Create a new class called `CountdownCell` that is a subclass of the `BlockedCell` class.

Create a constructor for the `CountdownCell` class that sets the initial value of the timer and makes sure that the symbol displayed for the cell will be the value of its timer. The first symbol that should be displayed for the timer value of a `CountdownCell` is `3`

Create a method `UpdateCell` in the `CountdownCell` class that:

- overrides the method from the base class
- decreases the value of the timer by one
- changes the symbol to @ when the timer has a value of zero.

Task 2
Modify the `AttemptPuzzle` method in the `Puzzle` class so that, if the amount to add to the score is greater than zero, it selects an empty cell in the grid at random and replaces that cell with a new `CountdownCell`.

Task 3
Modify the `AttemptPuzzle` method in the `Puzzle` class so that, immediately before it checks if the number of symbols left is zero, it calls the `UpdateCell` method for each cell in the grid.

Task 4
Test that the changes you have made work:

- run the Skeleton Program
- load the file `puzzle3`
- enter `1`
- enter `4`
- enter `X`
- place a `T` in an empty cell
- place a `T` in an empty cell
- place a `T` in an empty cell

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

3 1 1 Your PROGRAM SOURCE CODE for the new class `CountdownCell` and the amended `AttemptPuzzle` method.

[13 marks]

3 1 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]