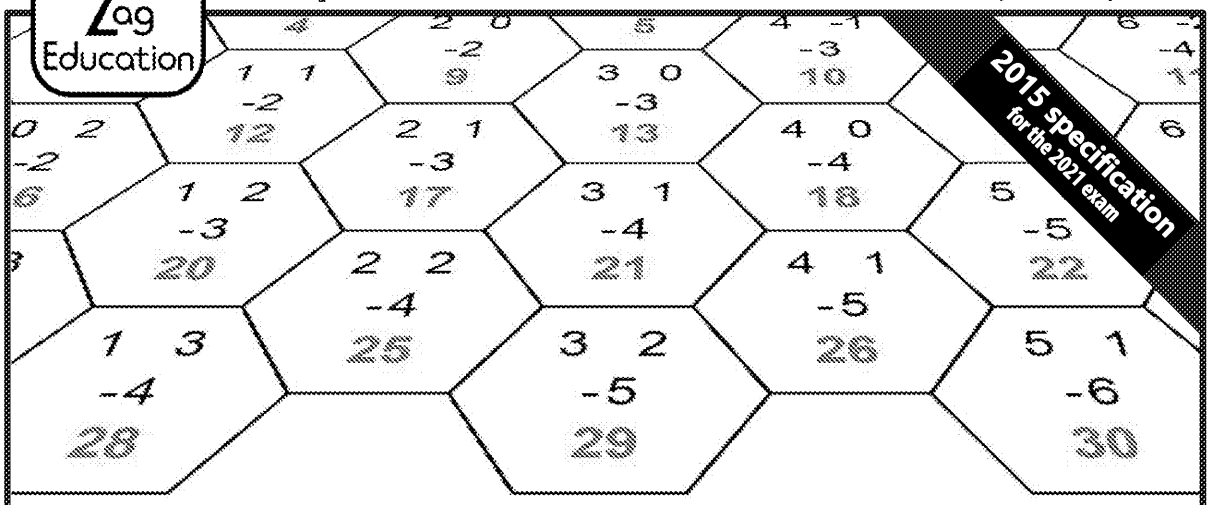
Zig Zag Education

Computer Science

A Level | AQA | 7517

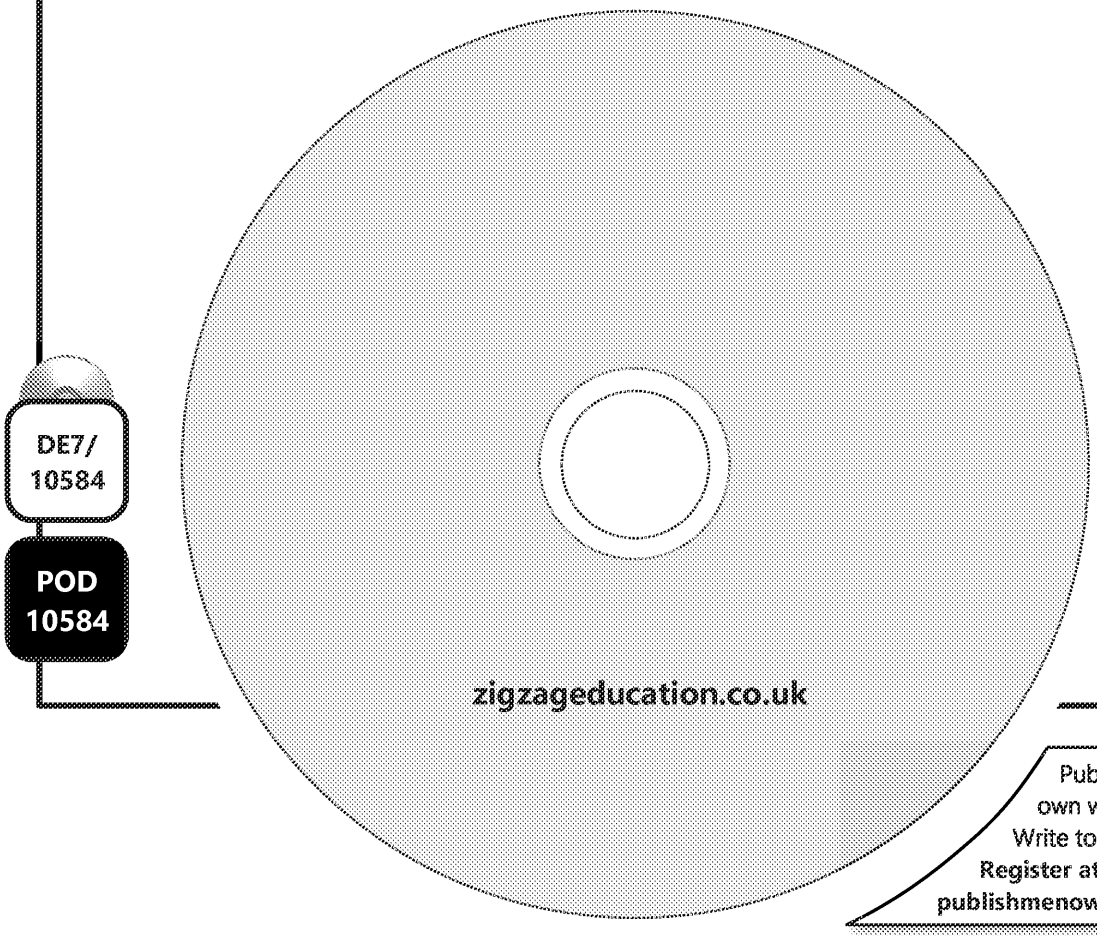


PAPER 1 EXAM RESOURCE PACK 2021

HEX BARON

for A Level AQA Computer Science

C# EDITION



DE7/
10584

POD
10584

zigzageducation.co.uk

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

Contents

Teacher's Introduction

This resource pack is designed to help you support your students taking the **A Level Computer Science Paper 1** exam. It is based on the ***Hex Baron*** preliminary material (C#) – for examination summer 2021.

On the CD, you will find the following:

`HexBaron`	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: Copy the entire `HexBaron` 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:

- `c06a-ClassDiagramTasksMS.pdf`
- `c06b-ClassDiagramFull.pdf`
- `c07-TheoryQuestionsMS.pdf`
- `c08-ProgrammingTasksMS.pdf`

The resource pack consists of the following:

① **Pre-release Commentary** including a general explanation of how the program works (text and video), plus a more detailed, technical overview* describing each subroutine class and method in turn.

> ***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 Diagram Tasks**

Three class diagram tasks for students to complete while getting to grips with the skeleton program. A mark scheme is provided via the *Solutions* web page as a password-protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the skeleton program, like Section C in the exam. 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 version are available format. Suggested answers are provided via the *Solutions* web page as a password-protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test, like Section D in the exam. 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.

HEX BARON – Commentary

Hex Baron is a two-player game in which the main objective is to gain as many Victory Points (VPs) as possible, normally by killing your opponent's Baron. All adjacent hexes count as a distance of 1 and are legal moves, although for some pieces this will cost 2 fuel. It is best to start by creating a LESS and then you will have enough lumber to be able to spawn another Serf and start digging for fuel.

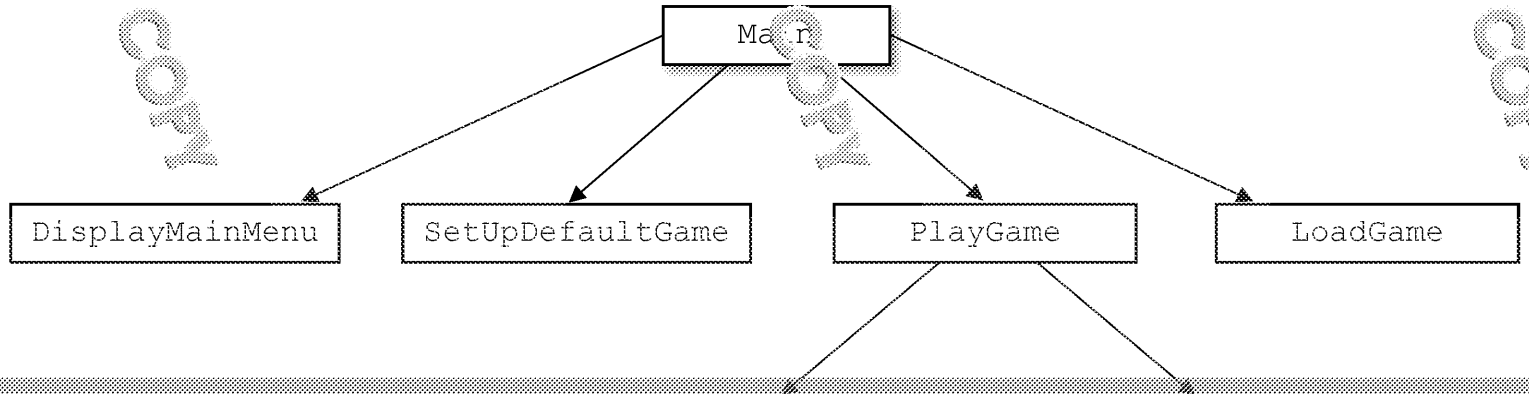
In the game you have to enter three moves before any of them are validated, so if you make a mistake with the first move you could end up losing your entire turn of all three moves as it might affect your second and third move if you thought you'd achieved something that had in fact failed.

Please watch the video explanation for a better understanding of the game mechanics.

Please note that throughout this resource, subroutines are considered to be part of the main program, and methods and attributes belong to classes.

Subroutines

Subroutines (Main Program): Hierarchy Diagram



Subroutines (All)

Name	Data	Description
`CheckCommandIsValid`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Bo	Depending on the first string in the list `items`, this calls one of the other `Check.` `mandFormat` subroutines.
`CheckMoveCommandFormat`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are three eleme n the `items` list 2) The second and third ite are strings containing integers Otherwise it returns False.
`CheckStandardCommandFormat`	***Parameters:*** `items` (list of str gs) ***Return value(s):*** Boolean	Returns True if the following condit r are met: 1) There are two elements in the `items` list 2) The second item is a string co ining an integer Otherwise it returns False.
`CheckUpgradeCommandFormat`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions re met: 1) There are three elements in the `items` list 2) The third item is a string containing integer 3) The second item is either less or pbds (any case) Otherwise it returns False.
`DisplayEndMessages`	***Parameters:*** `player1`, `player2` (Player objects) ***Return value(s):*** -	Displays the following for both players: 1) Their remaining resources 2) Their VPs Then, displays the winner.

Name	Data	Description
`Main`	***Parameters:*** *-* ***Return value(s):*** *-*	Calls `DisplayMainMenu` and processes the result to either quit, load a game by calling `LoadGame` or play the default game by calling `SetUpDefaultGame` and then calling `PlayGame` to play the game.
`PlayGame`	***Parameters:*** `player1` (`Player` object) `player2` (`Player` object) `grid` (`HexGrid` object) ***Return value(s):*** *-*	This alternates between Player 1 and Player 2 and always ensures that Player 2 has an equal number of turns even if Player 1 has just killed their Baron. For each player's turn, the board is printed out and then three commands are read in and validated (by calling `CheckCommandIsValid`) and then executed (by calling `ExecuteCommand` on the `HexGrid` object for the game – stored in the variable `Grid`). After each command, the `UpdateLumber`, `UpdateFuel` and `RemoveTileFromSupply` (if necessary) methods are called on the `Player` object for that player's turn. Once the commands have been executed, places are destroyed, VPs are assigned and a check is made to see whether the game is over. Before the game moves on to the next player's turn, the status of both players (resources and VPs) is displayed. If the game is over and Player 2 has played, then `DisplayEndMessages` is called to display the final scores

`BaronPiece` (inherits from `Piece`)

Name	Data	Description
`BaronPiece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Calls the super constructor (`Piece`) and passes `player1` as an argument. Initialises the following protected attributes: • `pieceType` to "B" • `VPValue` to 10
`CheckMoveIsValid` (public) *Override*	***Parameters:*** `distanceBetweenTiles` (int), `startTerrain` (string), `endTerrain` (string) ***Return value(s):*** `fuelCostOfMove` (int)	If the value of the parameter `distanceBetweenTiles` is 1 then it returns the value of the protected attribute `fuelCostOfMove` otherwise it returns -1.

`HexGrid`

Name	Data	Description
`HexGrid` (constructor)	***Parameters:*** `n` (int) ***Return value(s):*** -	Initialises the following protected attributes: • `size` from parameter `n` • `player1Turn` to True • `tiles` to an empty list • `pieces` to an empty list It also calls the private methods `SetUpTiles` and

Name	Data	Description
`CheckPieceAndTileAreValid` (private)	***Parameters:*** `tileToUse` (int) ***Return value(s):*** Boolean	Returns True if there is a piece belonging to the current player in `tileToUse`, otherwise it returns False.
`CheckTileIndexIsValid` (private)	***Parameters:*** `tileToCheck` (int) ***Return value(s):*** Boole	Returns True if the parameter `tileToCheck` is a valid index of the protected attribute `tile`
`CreateBottomLine` (private)	***Parameters:*** - ***Return value(s):*** `line` (st	Returns a string containing the bottom line of the `HexGrid` (as displayed at the start of each tu hen playing the game).
`CreateEvenLi` (private)	***Parameters:*** `firstEvenL` (Boolean) ***Parameters passed by ref:*** `listPositionOfTile` (int) ***Return value(s):*** `line` (string)	Returns a string containing the c t string of an even line of the `HexGrid` (as displayed at the t of each turn when playing the game). `listPositio Tile` is used to select the correct tile from the `tiles` list. I is hen incremented to select the next tile in the list to be dis d. It is passed by reference so that the `createEvenLi` and `createOddLine` methods can be alte ately called. `firstEvenLine` is used to draw the fi tile of the first line correctly
`CreateOddLine` (private)	***Parameters passed by ref:*** `listPositionOfTile` (int) ***Return value(s):*** `line` (string)	Returns a string containing the correct strin of an odd line of the `HexGrid` (as displayed at the start of each n when playing the game). `listPositionOfTile` is used o select the correct tile from the `tiles` list. It is then incremented to select the next tile in the list to be displayed. It is passed by reference so that the `createEvenLine` and `createOddLine` methods can be alternately called.
`CreateTopLine` (private)	***Parameters:*** - ***Return value(s):*** `line` (string)	Returns a string containing the top line of the `HexGrid` (as displayed at the start of each turn when playing the game)

Name	Data	Description
`ExecuteCommand` (public)	***Parameters:*** `items` (list of strings), `fuelAvailable` (int), `lumberAvailable` (int), `piecesInSupply` (int) ***Parameters passed by ref:*** `fuelChange` (int), `lumberChange` (int), `supplyChange` (int) ***Return value(s):*** `status` (string)	Depending on the first element of `items`, it either calls `ExecuteMoveCommand`, `ExecuteSpawnCommand`, `ExecuteUpgradeCommand` or `ExecuteCommandInTile`. Each of these private methods return values is used to create a suitable status message and determine the figures for the return values of `fuelChange`, `lumberChange` and `supplyChange` (which were initialised to 0).
`ExecuteCommandInTile` (private)	***Parameters:*** `items` (list of strings) ***Parameters passed by ref:*** `fuel` (int), `lumber` (int) ***Return value(s):*** `status` (Boolean)	Checks whether there is a `Piece` belonging to the player in the `Tile` specified as the second string in `items`, if not then False is returned It then uses `HasMethod` to determine whether the `Piece` in the `Tile` has a `saw` or `dig` command as specified by the first string in the `items` list and calls that method. If it was a `saw` the referenced parameter of `lumber` is set to 1 if the piece is in a `Tile` with forest terrain. If it was a dig and more than 5 fuel was returned then it sets the terrain of the `Tile` to a field using the `SetTerrain` method. The method then returns True and the `fuel` or `lumber` gained from digging or sawing or it returns False and the values of 0 for fuel and 0 for lumber if the command could not be executed

Name	Data	Description
`ExecuteSpawnCommand` (private)	***Parameters:*** `items` (list of strings), `lumberAvailable` (int), `piecesInSupply` (int) ***Return value(s):*** `lumberCost` (int)	Checks to see whether the player has at least 1 `pieceInSupply` and 3 `lumber` and that the `Tile` specified as the second string in the `items` list is empty, otherwise it returns -1. The method then checks to see that the player's own Baron is a neighbour and then spawns a new Serf in the `Tile`, appends it to the attribute `pieces` and returns 3, otherwise it returns -1.
`ExecuteUpgradeCommand` (private)	***Parameters:*** `items` (list of strings), `lumberAvailable` (int) ***Return value(s):*** `lumberCost` (int)	If the tile specified as the third item in the `items` list is available and contains a Serf belonging to the player whose turn it is and that player has at least 5 lumber available then it is upgraded to a `LESSPiece` or `PBDSPiece` according to the second string in `items` and 5 is returned as the cost, otherwise -1 is returned.
`GetGridAsString` (public)	***Parameters:*** `p1Turn` (Boolean) ***Return value(s):*** `gridAsString` (string)	Initialises the local variable `listPositionOfTile` to 0 and uses that to loop through the `Tiles` in the grid calling the private methods `CreateTopLine`, `CreateOddLine`, `CreateEvenLine` and `CreateBottomLine` as needed to form a string containing the entire `HexGrid`. `listPositionOfTile` is passed as by reference to the `createEvenLine` and `createOddLine` subroutines to increment through the `tiles` list.
`GetPieceTypeInTile` (public)	***Parameters:*** `ID` (int) ***Return value(s):*** `pieceType` (string)	If there is no `Piece` in the `Tile` specified by `ID` then this returns " " (string with a single space), otherwise it returns the result of the `GetPieceType` method which is called on the `Piece` in the `Tile`.

Name	Data	Description
`SetUpNeighbours` (private)	***Parameters:*** - ***Return value(s):*** -	For each `Tile` on the `HexGrid`, it loops through all of the `Tiles` on the `HexGrid` and adds any `Tile` that is a distance of 1 away (as determined by the `GetDistanceToTileT` method in the `Tile` class) to its list of neighbours by calling the `AddToNeighbours` method in the `Tile` and passing an argument of the `Tile` that is 1 away.
`SetUpTiles` (private)	***Parameters:*** - ***Return value(s):*** -	Loops through and creates all of the tiles needed for the `HexGrid` according to the protected attribute `size` and adds them to the protected attribute `tiles`.

`LESSPiece` (inherits from `Piece`)

Name	Data	Description
`LESSPiece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Calls the super constructor (`Piece`) and passes `player1` as an argument. Initialises the following protected attributes: • `pieceType` to "L" • `VPValue` to 3
`CheckMoveIsValid` (public) *Override*	***Parameters:*** `distanceBetweenTiles` (int), `startTerrain` (string), `endTerrain` (string) ***Return value(s):*** [illegible]	If the value of the parameter `distanceBetweenTiles` is 1 and `startTerrain` is not a forest then if the move begins or ends in a peat bog it returns `fuelCostOfMove` x2 and if the move doesn't begin or end in a peat bog then it returns `fuelCostOfMove` otherwise it returns -1.

`PBDSPiece` (inherits from `Piece`)

Name	Data	Description
`PBDSPiece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Calls the super constructor (`Piece`) and passes `player1` as an argument. Initialises the following protected attributes: • `fuelCostOfMove` to 2 • `pieceType` to "P" • `VPValue` to 2
`CheckMoveIsValid` (public) *Override*	***Parameters:*** `distanceBetweenTiles` (int), `startTerrain` (string), `endTerrain` (string) ***Return value(s):*** `fuelCostOfMove`	If the value of the parameter `distanceBetweenTiles` is 1 and `startTerrain` is not a peat bog then it returns the value of the protected attribute `fuelCostOfMove` otherwise it returns -1.
`Dig` (public)	***Parameters:*** `terrain` (string) ***Return value(s):*** `fuel` (int)	If `terrain` is a peat bog then it has a 90% chance of returning 1 and a 10% chance of returning 5, otherwise it returns 0.

`Piece`

Name	Data	Description
`Piece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Initialises the following protected attributes: • `belongsToPlayer1` to `player1` • `connectionsToDestroy` to 2 • `destroyed` to False

`DestroyPiece` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** -	Sets the value of the protected attribute `destroyed` to True.
`GetBelongsToPlayer1` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `belongsToPlay` (Boolean)	Returns the value of the protected attribute `belongsToPlayer1`.
`GetConnec sNeededToDestroy` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `connectionsToDe oy` (int)	Returns the value of the cted attribute `connectionsToDestroy`
`GetPieceType` (lic) *Virtual*	***Parameters:*** - ***Return value(s):*** `piece e` (string)	If the attribute `belongsTo yer1` is True then it returns the value of the attribu `ieceType` otherwise it returns the lower case value.
`GetVPs` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `VPValue` (in)	Returns the value of the protecte ttribute `VPValue`.
`HasMethod` (public) *Virtual*	***Parameters:*** `methodName` (ng) ***Return value(s):*** `callable` (B lean)	If the method name specified as a p meter exists in the object then it returns True, otherwise it returns False.

Player

Name	Data	Description
`Player` (constructor)	***Parameters:*** `n` (string), `v` (int), `f` (int), `l` (int), `t` (int) ***Return value(s):*** -	Initialises the following protected attributes: • `name` from parameter `n` • `VPs` from parameter `v` • `fuel` from parameter `f` • `lumber` from parameter `l` • `piecesInSupply` from parameter `t`
`AddToVPs` (public) *Virtual*	***Parameters:*** `n` (int) ***Return value(s):*** -	Increments the attribute `VPs` by `n`.
`GetFuel` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `fuel` (int)	Returns the value of the attribute `fuel`.
`GetLumber` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `lumber` (int)	Returns the value of the attribute `lumber`.
`GetName` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `name` (string)	Returns the value of the attribute `name`.
`GetPiecesInSupply` (public)	***Parameters:*** - ***Return value(s):*** `piecesInSupply` (int)	Returns the value of the attribute `piecesInSupply`.
`GetStateString` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** string	Returns a string containing the `VPs`, the `piecesInSupply`, the `lumber` and the `fuel`.
`GetVPs` (public) *Virtual*	***Parameters:*** - ***Return value(s):*** `VPs` (int)	Returns the value of the attribute `VPs`.
`RemoveTileFromSupply`	***Parameters:*** -	Decrements the `piecesInSupply` attribute by 1.

Tile

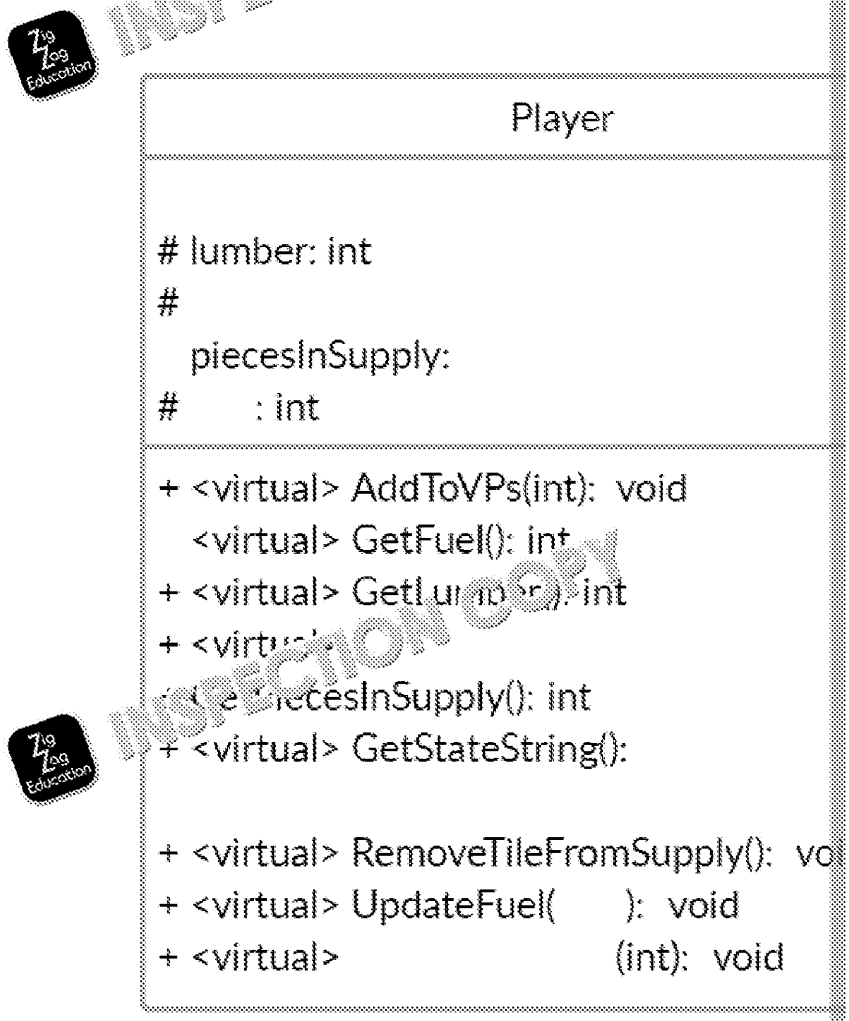
Name	Data	Description
`Tile` (constructor)	***Parameters:*** `xcoord` (int), `ycoord` (int), `zcoord` (int) ***Return value(s):*** -	Initialises the following protected attributes: • x from parameter `xc` • y from parameter `yc` • z from parameter `zcoor` • `terrain` to " " (a string c[illegible]aining a single space) • `pieceInTile` to null • `neighbours` to an empty li[illegible]
`AddToNeighbour` (public)	***Parameters:*** `n` (Tile) ***Return value(s):*** -	Adds the tile `n` to the end of the list s[illegible]or[illegible]d in the protected attribute `neighbours`.
`GetDistanceToTile` (public)	***Parameters:*** `t` (Tile) ***Return value(s):*** `distance` (int)	Returns the maximum of the absolute d[illegible]ffe[illegible]ence in `x`, `y` and `z` between the current tile and `t`.
`GetNeighbours` (public)	***Parameters:*** - ***Return value(s):*** `neighbours` (list of `Tile`s)	Returns the value of the protected attribute `neighbours`.
`GetPieceInTile` (public)	***Parameters:*** - ***Return value(s):*** `pieceInTile` (`Piece` object or null)	Returns the value of the protected attribute `p[illegible]ceInTile`.
`GetTerrain` (public)	***Parameters:*** - ***Return value(s):*** `terrain` (string)	Returns the value of the protected attribute `terrain`.
`Getx` (public)	***Parameters:*** - ***Return value(s):*** `x` (int)	Returns the value of the protected attribute `x`.

HEX BARON – Class Diagram Tasks

Task 1

A partially-complete UML class diagram for the `Player` class is shown be

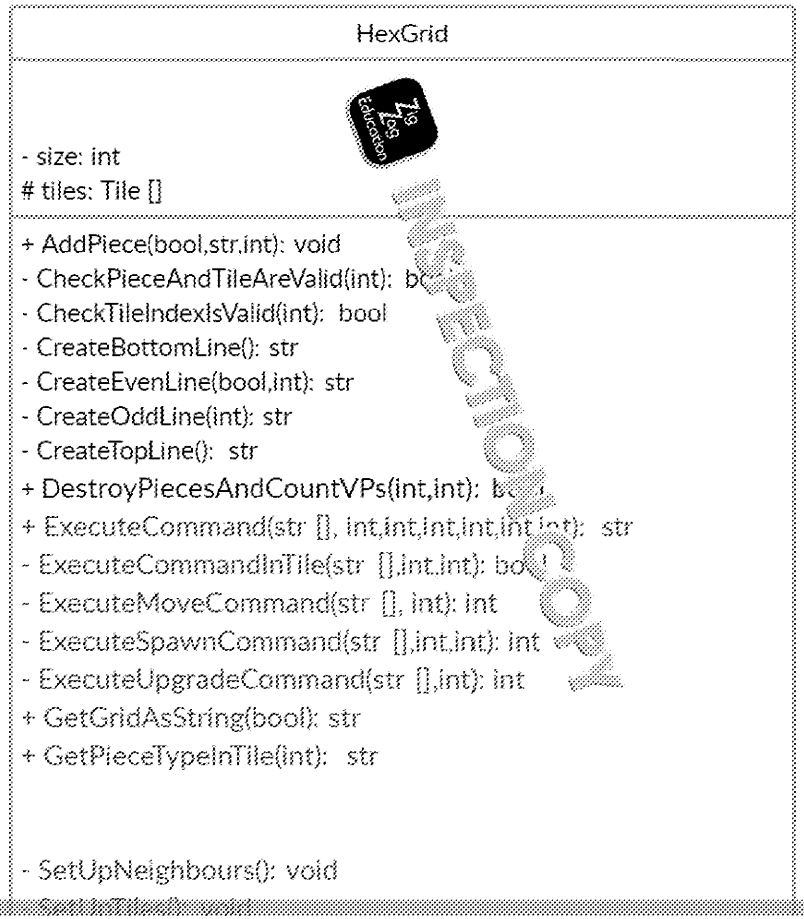
Fill in the missing details.

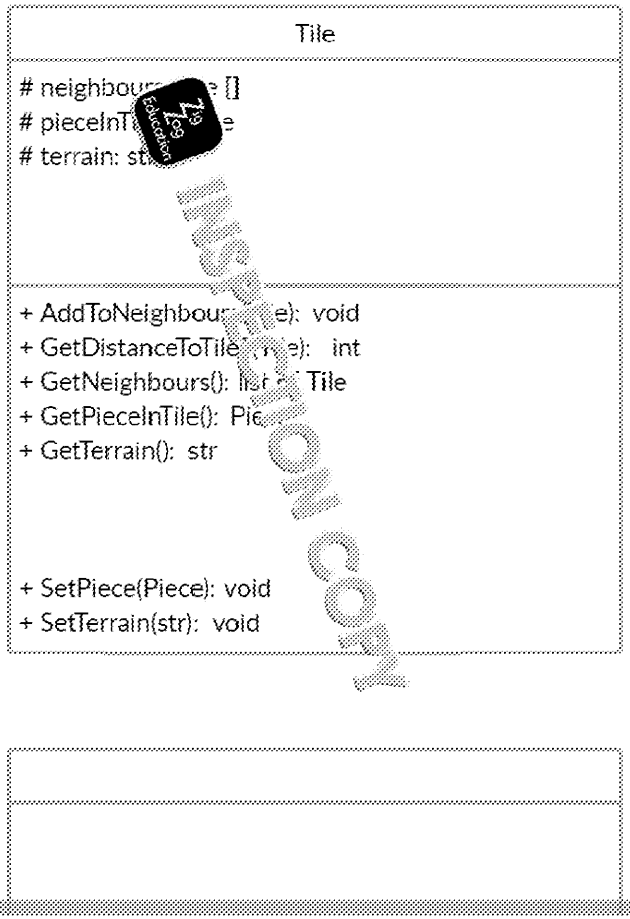


Task 2 (5 marks)

A partially-complete UML class diagram is shown.

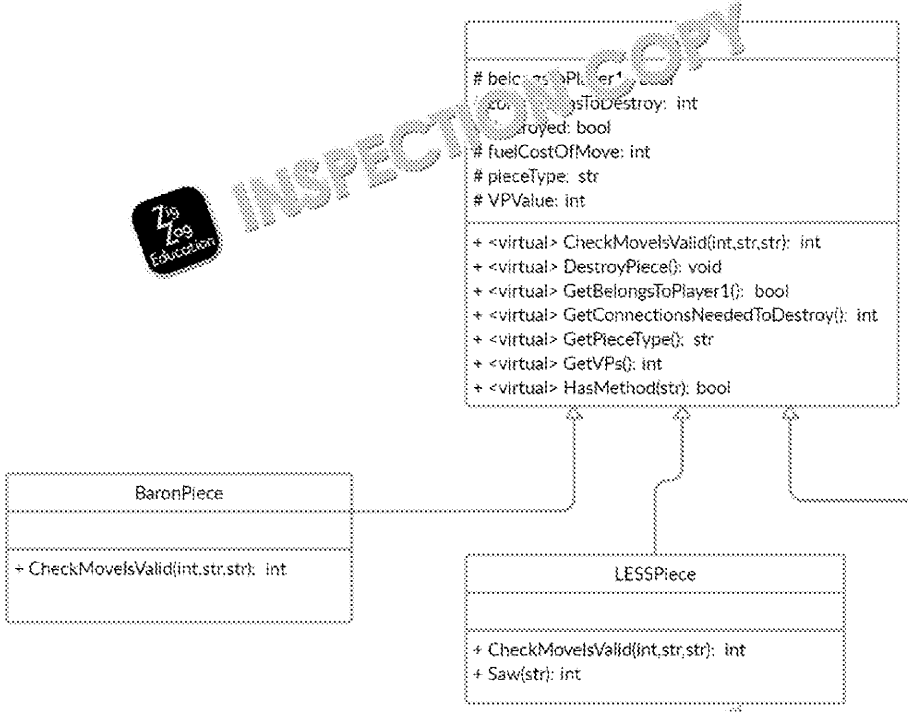
Fill in the missing details, including any relationships between the classes shown.





Task 3

There are two pieces of information missing from the UML class diagram belo



Explain what are they and why are they normally omitted?

HEX BARON – Theory Questions

These questions refer to the preliminary material and require you to lo
but do not require any additional programming.

1 This question refers to the `PlayGame` subroutine in the main program

The subroutine does not currently print out the actual player names wh
at the start of each turn; it just prints 'Player One' and 'Player Two'

a) How could this be changed so that it would print out the names us
meth[illegible] to access the protected attribute `Name` from the `Player` class?

.....

b) This is an example of using an accessor method. Why are access
in object-oriented programming?

.....

2 This question refers to the `PlayGame` subroutine in the main program

In the subroutine, the variable `Player1Turn` is set to True to indicat
Player One's turn. How does the game process Player Two's turn?

.....

.....

.....

3 This q[illegible] refers to the `HasMethod` method within the `Piece` cla
`Execu[illegible]ommandInTile` method in the `HexGrid` class.

Explain how `HasMethod` is used in the `ExecuteCommandInTile`

.....

.....

.....

4 This question refers to the `GetDistanceToTileT` method in the `T`
`GetDistanceToTileT` is used to calculate the distance from the c

a) Explain how this calculation is performed

.....

.....

.....

.....

.....

4 b) Give a worked example of this calculation if the current tile has coo
and the destination tile has coordinates (4,-4,0). All coordinates ar

.....

.....

.....

.....

5 The `B` `Piece`, `LESSPiece` and `PBDSPiece` classes all inherit

a) Explain what is meant by 'inheritance'.

.....

.....

b) Why isn't there a 'SerfPiece' class?

.....

.....

6 This question refers to the `BaronPiece` class a[illegible] `Piece` class.

The `BaronPiece` class overrides the public method `CheckMoveIs`

a) Explain what is mea[illegible] 'overriding'.

.....

.....

.....

b) Explain why the method was overridden in this case.

.....

.....

.....

7 Big-O notation is used to measure the time complexity of algorithms.

a) Examine the `[illegible]Neighbours` method in the `HexGrid` class

.....

b) Which other method is used as a measure of the complexity of alg

.....

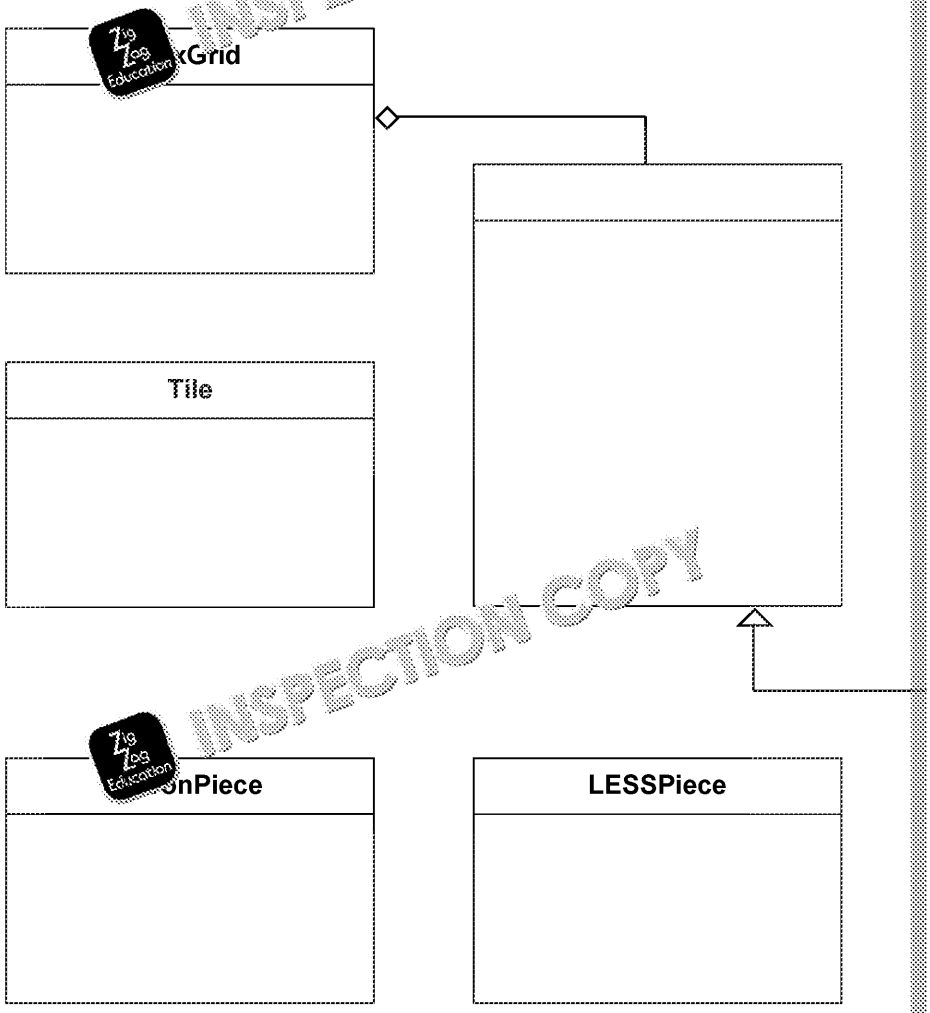
8 The skeleton program uses a Comma Separated Values (CSV) file as
saved game. Explain why a CSV file is suitable for cross-platform appl

.....

.....

9 Class diagrams can be represented using Unified Modeling Language

Add the missing class names and relationships to the diagram below:



10 This question refers to the `ExecuteUpgradeCommand` method of t

a) What does a return value of -1 from this method mean?

.....

.....

b) Wh es a return value of 5 from this method mean?

.....

.....

11 This question refers to the `Dig` method of the `PBDSPiece` class and `ExecuteCommandInTile` method of the `HexGrid` class.

When a 'dig' command is issued by a player, it is possible that the tile w from a peat bog to a field and that they will receive 5 fuel instead of the

Explain how this eventuality is processed in the code with specific refe used, function calls made and return values.

.....

.....

.....

.....

.....

.....

12 This question refers to the `DestroyPiecesAndCountVPs` method to the `Piece` class.

For a piece to be destroyed in the game, it needs to have two connect neighbours (i.e. pieces in two of the tiles that share a side with it).

a) Explain how the `DestroyPiecesAndCountVPs` method uses the protected attribute `ConnectionsToDestroy` of the Piece c

.....

.....

.....

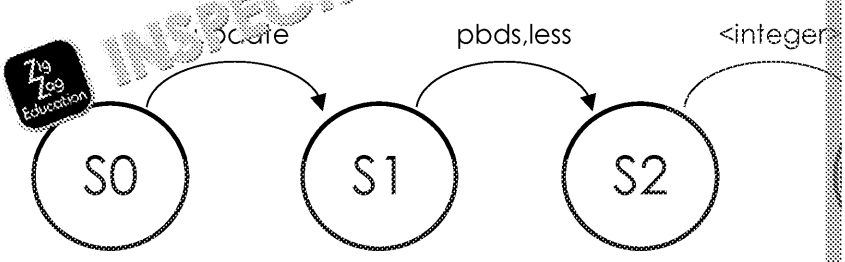
b) Why is polymorphism not possible for private attributes?

.....

.....

13 This question refers to the `CheckUpgradeCommandFormat` subro

Currently the validation performed by this subroutine could be represe (note that the 0 or higher condition is not currently checked in the code answer).



13 Where <integer> is any valid (0 or higher) integer.

a) Write down the regular expression that is the equivalent of this FSM

.....

b) Improve your regular expression so that it only accepts integers fro

.....

c) What is the definition of a [illegible] language?

.....

14 This question refers to the `GetGridAsString` method of the `HexG`

a) State the identifier for a local variable used in this method.

.....

b) What are the advantages of using local variables?

.....

.....

.....

.....

c) This method uses one priv[illegible] and two protected attributes. What is between a private and a protected attribute?

.....

.....

.....

15 This question refers to the `ExecuteSpawnCommand` method of the

Explain how the method works, including reference to how it meets the spawning a new Serf (it must be on an empty square adjacent to that pl

.....

.....

.....

.....

.....

.....

.....

.....

HEX BARON – Theory Questions

These questions refer to the preliminary material and require you to l
but do not require any additional programming.

1 This question refers to the `PlayGame` su[illegible] in the main program

The subroutine does not curr[illegible] out the actual player names wh
at the start of each t[illegible] prints out 'Player One' and 'Player Two'

a) Ho[illegible]d [illegible]e corrected so that it would print out the names us
me[illegible] the protected attribute `Name` from the `Player` class?

b) This is an example of using an accessor method. Why are access
in object-oriented programming?

2 This question refers to the `PlayGame` subroutine in the main program

In the subroutine, the variable `Player1Turn` is set to True to indicat
Player One's turn. How does the game process Player Two's turn?

3 This question refers to the `HasMethod` method within the `Piece` cla
`ExecuteCommandInTile` method in the `HexC[illegible]d` class.

Explain how `HasMethod` is used in the `[illegible]xecuteCommandInTile`

4 This qu[illegible] the `GetDistanceToTileT` method in the `T`
`GetDi[illegible]ceToTileT` is used to calculate the distance from the c

a) Explain how this calculation is performed.

b) Give a worked example of this calculation if the current tile has coo
and the destination tile has coordinates (4,-4,0). All coordinates ar
(x,y,z) form.

5 The `BaronPiece`, `LESSPiece` and `PBDSPiece` classes all inherit

a) Explain what is meant by 'inheritance'.

b) Why isn't there a 'SerfPiece' class?

6 This question refers to the `Ba[illegible]Piece` class and the `Piece` class.

The `Ba[illegible]Pi[illegible]` class overrides the public method `CheckMoveIs`

a) Exp[illegible]hat is meant by 'overriding'.

b) Explain why the method was overridden in this case.

7 Big-O notation is used to measure the time complexity of algorithms.

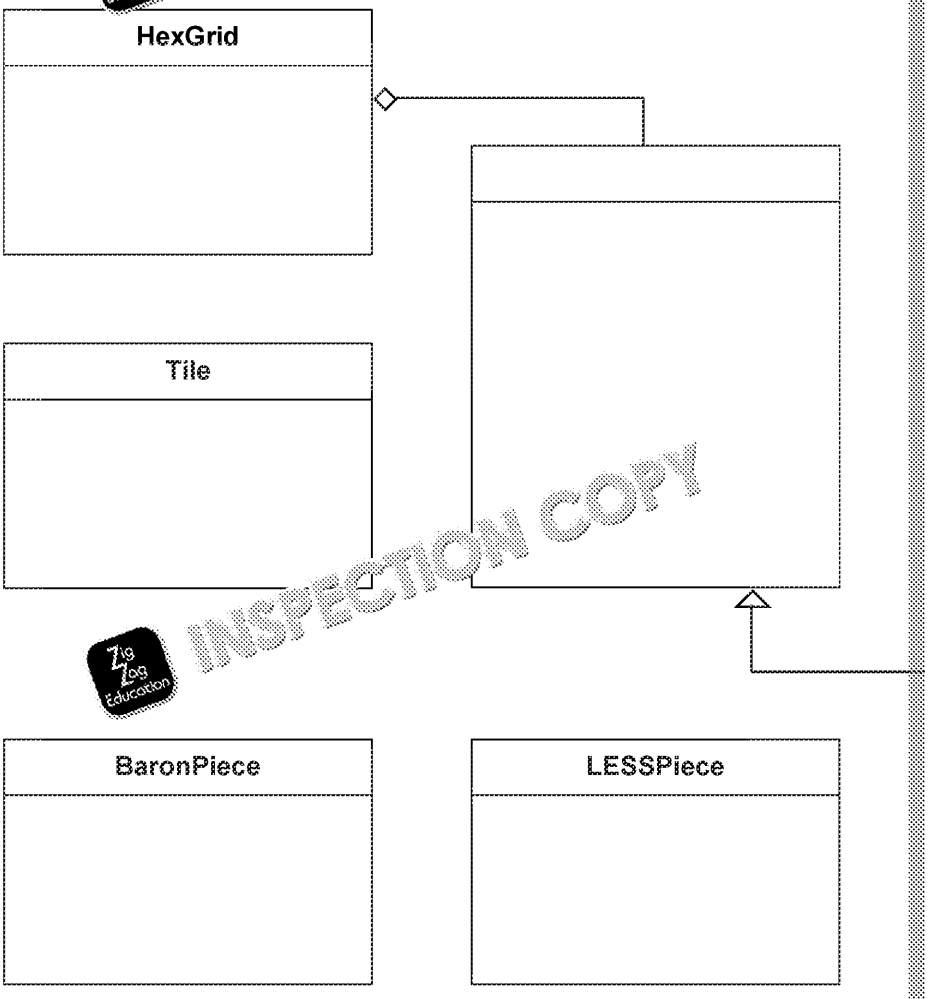
a) Examine the `SetUpNeighbours` method in the `HexGrid` class

b) Which other method is used as a measure of the complexity of alg

8 The skeleton program uses a Comma Separated Values (CSV) file as
saved game. Explain why a CSV file is suitable for a cross-platform appli

9 Class diagrams can be represented using Unified Modeling Language

Add the missing class names and relationships to the diagram below:



10 This question refers to the `ExecuteUpgradeCommand` method of t

a) What does a return value of -1 from this meth[illegible]ean?

b) What does a return value of 5 from thi[illegible]od mean?

11 This question refe[illegible] method of the `PBDSPiece` class and
`Execu[illegible]rinTile` method of the `HexGrid` class.

When a [illegible] command is issued by a player, it is possible that the tile
from a peat bog to a field and that they will receive 5 fuel instead of the

Explain how this eventuality is processed in the code with specific refe
used, function calls made and return values.

12 This question refers to the `DestroyPiecesAndCountVPs` method to the `Piece` class.

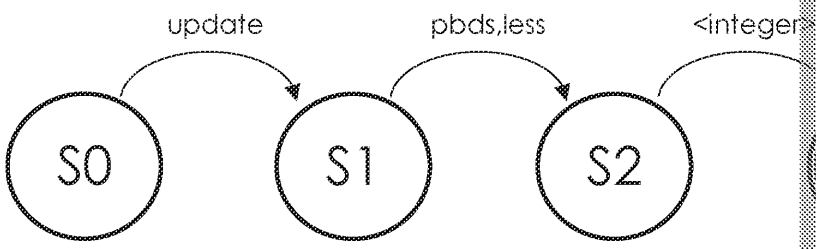
For a piece to be destroyed in the game, it needs to have two connecti neighbours (i.e. pieces in two of the tiles that share a side with it).

a) Explain how the `DestroyPiecesAndCountVPs` method uses p the protected attribute `ConnectionsToDestroy` of the Piece c

b) Why is polymorphism not possible for private attributes?

13 This q[illegible] to the `CheckUpgradeCommandFormat` subro

Currently, the validation performed by this subroutine could be represe (note that the 0 or higher condition is not currently checked in the code answer).



Where <integer> is any valid (0 or higher) integer.

a) Write down the regular expression that is the equivalent of this FSM

b) Improve your regular expression so that it only accepts integers fro

c) What is the [illegible] of a regular language?

14 This question refers to the `GetGridAsString` method of the `HexG`

a) State the identifier for a local variable used in this method.

b) What are the advantages of using local variables?

c) This method uses one private and two protected attributes. What is between a private and a protected attribute?

15 This question refers to the `ExecuteSpawnCommand` method of the

Explain how the method works, including referen[illegible] how it meets the spawning a new Serf (it must be on an empty square adjacent to that p

HEX BARON – Programming Tasks

Task 1

This question refers to the **PlayGame** subroutine in the main program.

The commands input by the player are already converted to lower case, w
insensitive. However sometimes a player may additionally or accidentally
or end of the command – currently this often causes an invalid command
code that converts them to lower case to also remove any leading or trailin
own subroutine for this, or a built-in subroutine.

Test that the change you have made works:

a) Choose menu option 1 to run the default game and then enter the
(they are in quotes as the spaces are important to type):

- "move 8 16 "
- "move 8 16"
- " upgrade pbds 16 "

b) Show a screen copy of entering the commands and the response
point at which it prints out the Player One current state line.

Evidence that you need to provide:

a. You PROGRAM SOURCE CODE for the amended subroutine **Pl**
subroutines or code that you wrote.

b. SCREEN CAPTURE(S) showing the required test.

Task 2

This question refers to the `SetUpDefaultGame` subroutine in the main

At the moment, the game sets the default names of the players to 'Player
`SetUpDefaultGame` subroutine. Change this subroutine to prompt for
and then use the values entered to construct two players.

Test that the changes you have written work:

a) Ch me option 1 to run the default game and then enter "To
and y" as the name for Player Two.

b) Show a screen copy of entering the commands and the responses
including the prompt for the first player to enter their commands.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Se`

b. SCREEN CAPTURE(S) showing the required test.

Task 3

This question refers to the `PlayGame` and `SetUpDefaultGame` subro
the `Player` class and to the `DestroyPiecesAndCountVPs` method

Normally when a piece is destroyed it depletes your s y chain and mov
losing the game as the number of pieces that you c make is limited. Th
that if a piece is destroyed then you g e piece in your supply chain.

- Add a method t t `Player` class that will allow the `PiecesIn`
be ec

- Modify the `DestroyPiecesAndCountVPs` method to take Pla
parameters and pass them in from `PlayGame` in both places that

- Further modify the `DestroyPiecesAndCountVPs` method so
`AddPiecesInSupply` method that you created for the appropri
additional piece in their supply every time one of their pieces is des

- Modify the `SetUpDefaultGame` subroutine so that the existing
are replaced with the following six:

```
grid.AddPiece(true, "Baron", 0);
grid.AddPiece(true, "Serf", 7);
grid.AddPiece(true, "Serf",
grid.AddPiece(false, "Bar
grid.AddPiece(fal "Serf", 15);
grid.AddPie "Serf", 23);
```

Test that t nges you have made work:

a) Choose menu option 1 to run the default game and then enter the
   - move 13 10
   - move 10 14
   - move 14 19

b) Show a screen copy of entering the commands and the responses
including the prompt for the second player to enter their commands

Evidence that you need to provide:

a. Your PROGRAM SOURCE C r the amended subroutine `Pl`
`DestroyPiece CountVPs` method of the `HexGrid` class
`Play` c

b. SC N CAPTURE(S) showing the required test.

Task 4

This question refers to the **SetUpDefaultGame** subroutine in the main
ExecuteSpawnCommand method in the **HexGrid** class.

Change the rules of the game so that each player can have a maximum of
time. There should be an additional output message generated saying tha
exceed max pieces.'

- Modify the private method **ExecuteSpawnCommand** of the **Hex**
- Modify the **SetUpDefaultGame** subroutine so that the existing
are replaced with the following eight:

```
grid.AddPiece(true, "Baron", 0);
grid.AddPiece(true, "Serf", 8);
grid.AddPiece(true, "Serf", 24);
grid.AddPiece(true, "Serf", 5);
grid.AddPiece(true, "Serf", 2);
grid.AddPiece(true, "Serf", 3);
grid.AddPiece(false, "Baron", 31);
grid.AddPiece(false, "Serf", 23);
```

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the
   - spawn 4
   - move 4 0
   - quit

b) Show a screen copy of entering the commands and the responses
point at which it prints out the Player One current state line.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended **ExecuteSpa**
HexGrid class.

b. SCREEN CAPTURE(S) showing the required test.

Task 5

This question refers to the `PlayGame` subroutine in the main program, an
`GetGridAsIndices` in the `HexGrid` class.

Add an additional command to the game which will pri ut the hex grid a
hex; these should be in the top half of each he and the board should be t
display. The new command, "hexes" sh d n count as one of the three
the results should be displayed im ately upon entry, i.e. the grid shoul
numbers displayed as soon as Enter is pressed. The player should be abl
their comm ai have three normal commands.

- Create a new method for the `HexGrid` class called `GetGridAs`
string containing the grid with all of the index numbers in the correc
on the current `GetGridAsString` method. You may wish to du
`CreateOddLine` and `CreateEvenLine` methods.

- Modify the `PlayGame` subroutine to implement a call to the new m
enters the command "hexes" and then request the normal comman

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the

- move 8 16
- hexes
- move 16 24
- hexes
- hexes
- og pbds 24

*(Repeating the "hexes" command demonstrates that the command
interpreted as one of the 3 user commands)*

b) Show a screen copy of entering the commands and the responses
including the prompt for the second player to enter their commands
hex grid with the numbers and the printout of the board after Playe
move.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the ed subroutine `Pl`
`GetGridAsIndices` method of he `Hexgrid` class and any ne
versions of `CreateOddL` and `CreateEvenLine` added to

b. SCR E(S) showing the required test.

Task 6

This question refers to the `PlayGame` and `SetUpDefaultGame` subro

Change the move command so that if you move the same piece three time
in cost of 1 fuel (in total). For example, if you moved a Serf to a field (from
and then to a field, it would only cost 4 fuel inst[illegible]. If you moved a Ba
only cost 2 fuel instead of 3. They should be able to do the triple move eve
up on 0.

- Mo[illegible]e `PlayGame` subroutine to check whether three valid mo
exe[illegible]by the player and refund them one fuel.
- Modify the `SetUpDefaultGame` subroutine so that Player One

```
player1 = new Player("Player One", 0, 2, 10,
```

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the
   - move 0 4
   - move 4 9
   - move 9 17

b) Show a screen copy of entering the co[illegible] and the responses
point at which it prints out the Pla[illegible] current state line.

Evidence you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Pl`

b. SCREEN CAPTURE(S) showing the required test.

Task 7

This question refers to the `PlayGame` subroutine in the main program and `ExecuteCommandInTile` method and a new method, `MakeField`,

Change the saw and dig commands so that if you saw or dig the same loc the same turn then you get 5 of that resource and the terrain reverts to a fi chance of getting 5 fuel when you dig so that you always get only 1.

- Modify the `ExecuteCommandInTile` to remove the possibility o field and 5 fuel. There is no need to modify the `Dig` method of the `H`
- Add a method to the `HexGrid` class called `MakeField` which ha the tile which is to be made into a field.
- Modify the `PlayGame` subroutine to check whether three dig or sa executed in a single turn by either player and give them the extra r a field by calling the newly added `MakeField` method in the `Hex`

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the

Player One	• move 8 16 • move 16 24 • upgrade pbds 24
Player Two	• move 23 27 • move 27 30 • upgrade pbds 30
Player One	• dig 24 • dig 24 • dig 24
Player Two	• saw 30 • saw 30 • saw 30

b) Show a screen copy of the final board after all of the commands ha the Player One and Player Two current states that are shown imm print.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Pl` `ExecuteCommandInTile` and `MakeField` methods of the `He`

b. SCREEN CAPTURE(S) showing the required test.

Task 8

This question refers to the `CheckCommandIsValid` and `SetUpDefau` main program, the creation of a new subroutine `CheckDowngradeComm` method, **`ExecuteDowngradeCommand`** as well as the **`ExecuteComm`** the **`HexGrid`** class.

Introduce a new downgrade comm[illegible]at a PDBS or LESS can be do of 1 lumber. The syntax for the command is "downgrade NUM" where NU to be downgraded.

- Mo[illegible]e **`CheckCommandIsValid`** subroutine to allow for and downgrade command.
- Create a new subroutine `CheckDowngradeCommandFormat` w `CheckCommandIsValid` subroutine.
- Modify the `ExecuteCommand` method of `HexGrid` to call a new `ExecuteDowngradeCommand`.
- Create a new private method in `HexGrid` called `ExecuteDowng`
- Modify the `SetUpDefaultGame` subroutine so that `Player1` S

```
grid.AddPiece(true, "Serf", 16);
```

Test that the changes you have made [illegible]

a) Choose menu [illegible] to run the default game and then enter the
   - [illegible]ove 16 24
   - [illegible]pgrade pbds 24
   - downgrade 24

b) Show a screen copy of entering the commands and the responses point after it prints out the updated board.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Ch` new subroutine **`CheckDowngradeCommandFormat`**, the `Exec` **`HexGrid`** class and the new code for the **`Ex[illegible]teDowngradeC`** **`HexGrid`**.

b. SCREEN CAPTURE/S[illegible]g the required test.

Task 9

This question refers to the `SetUpDefaultGame` subroutine in the main
`BaronPiece` class.

Change the rules for the Baron so that it needs three connections to kill it.

- Modify the `BaronPiece` class so that it can only be de
- Modify the `SetUpDefaultGame` subroutine so that the four start
the Baron is seven:

```
grid.AddPiece(true, "Baron", 0);
grid.AddPiece(true, "Serf", 15);
grid.AddPiece(true, "Serf", 27);
grid.AddPiece(true, "Serf", 25);
grid.AddPiece(false, "Baron", 19);
grid.AddPiece(false, "Serf", 8);
grid.AddPiece(false, "Serf", 2);
```

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the

Player One	• move 25 21 • move 21 18 • move 18
Player Two	• move 8 5 • move 5 1 • move 1 4

b) Show a screen copy of entering the commands for the entire game

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended `BaronPiece`

b. SCREEN CAPTURE(S) showing the required test.

Task 10

This question refers to the `CheckCommandIsValid` subroutine in the m
`ExecuteCommand` method and a new `ExecuteSalvageCommand` m

Develop the salvage command. Any of your own Ser[illegible]BDS or LESS pie
costs three moves to do so (e.g. it has to be th[illegible]s command that you ty
add 1 to your supply chain, give you [illegible]er and remove the piece from t
command is salvage NUM, w[illegible]s the location on the board of the

- Mo[illegible]e `CheckCommandIsValid` subroutine to allow for and
sal[illegible]ommand.
- Modify the `ExecuteCommand` method of `HexGrid` to call a new
`ExecuteSalvageCommand`.
- Create a new private method in `HexGrid` called `ExecuteSalva`

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the
    - salvage 31
    - salvage 4
    - salvage 8

b) Show a screen copy of enteri[illegible]commands and the response f
point at which it print[illegible]updated board.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine Ch
the `ExecuteCommand` and `ExecuteSalvageCommand` meth

b. SCREEN CAPTURE(S) showing the required test.

Task 11

This question refers to the `PlayGame` subroutine in the main program and to

Introduce three chances for each player so that if they enter an invalid move o executed for some reason, they can correct it, but only three times per game. them to re-enter the invalid move and deduct 1 from their chances. The rema out each time a player uses up a chance and at the start of each player's turn

Also, as the commands and replacements could now be confusing, make sure t for a command or a replacement command they are told which command numb

- Modify the `Player` class to create an attribute (`Chances`) for their t
- Modify the `PlayGame` subroutine so that it allows the player to enter a command that is invalid or could not be executed for some reason.
- Create three new methods for the `Player` class to control access to t `DeductChance`, `GetChances` and `ResetChances`.
- By way of example and clarity, the testing output for Player One's first t

```
Enter command 1: move
Enter command 2: move 8 8
Enter command 3: upgrade less 28
Command number 1 is an invalid command
You have 2 chances remaining.
Enter new command 1: move 8 16
Command 1: Command executed
Command 2: That move can't be done
Enter new command 2: move 16 20
```

Test that the changes you have made work:

a) Choose menu option 1 for the default game and then enter the follo between each player.

Player One	• move • move 8 8 • upgrade less 28 • move 8 16 • move 16 20 • move 20 28 • Enter *(for Player Two's turn)*
Player Two	• move 23 27 • move 27 30 • upgrade less 30 • *Enter (for Player One's turn)*
Player One	• upgrade less 28 • saw 28 • saw 29 • *Enter (for Player Two's turn)*

b) Show screenshots of entering the commands and all the responses

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended `PlayGame` subro

b. SCREEN CAPTURE(S) showing the required test.

Task 12

This question refers to the `CheckCommandIsValid` and `SetUpDefau` new `CheckSpawnCommandFormat` subroutine in the main program, to the `Piece` class and to the `ExecuteSpawnCommand`, `DestroyPiec` `GetPieceTypeInTile` and `ExecuteMoveCom d` methods in the

Develop a Bomb piece. The bomb can o be spawned by the Baron but It costs 10 lumber to spawn and 2 f each space that it moves regard

When the bomb exp s destroy all pieces in the hexes adjacent to bomb. It w lo two pieces are adjacent to it (as per normal attacks ability.

Once the bomb has moved five spaces it is immobilised and becomes 'pri (even in the middle of a move) one of the hexes surrounding the bomb wil bomb will not explode if it is primed next to an existing piece (but of course existing pieces), only if another piece enters a hex next to it, kind of like a

The command is a modification of the spawn command, so now spawn ca argument of bomb, e.g. spawn bomb NUM would spawn a bomb at NUM i Baron, and spawn NUM would operate as normal. The bomb has a VP va explodes, the opposing player receives 5 VPs.

- Modify the `CheckCommandIsValid` subroutine to make the ne correctly.
- Create a new subroutine called `Check a CommandFormat` `CheckCommandIsValid` to m e sure that the new spawn bom
- Modify the private m od `ExecuteSpawnCommand` so that it w the pl er to lumber or report an error if they don't have enough lu
- Cre new `BombPiece` class for the bomb.
- Modify the method `DestroyPiecesAndCountVPs` so that if a explodes and destroys the surrounding pieces too.
- Modify the method `GetPieceTypeInTile` for `HexGrid` so tha Serf when the board is displayed.
- Modify the private method `ExecuteMoveCommand` so that it che next to a primed bomb, and explodes the bomb if it did.
- Modify the `Piece` class to add an `Explode` method so that it can the turn in case a bomb was set off during the turn.
- Modify the `SetUpDefaultGame` su to fir so that both players

```
player1 = new Pl      ayer One", 0, 30, 30,
player2 = n      yer("Player Two", 1, 30, 30,
```

TASK CONTINUES ON THE NEXT PAGE

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the

Player One	• spawn bomb 4 • move 4 9 • move 9 13
Player Two	• move 23 19 • move 19 • [illegible] 11 14
Player One	• move 13 18 • move 18 22 • move 22 27
Player Two	• move 31 23 • move 14 18 • move 18 22

b) Show a screen copy of entering the commands and the responses point at which it prints out that Player One is the winner.

c) Choose menu option 1 to run the default game and then enter the

Player One	• spawn bomb 4 • move 4 9 • move 9 13
Player Two	• move 23 19 • upgrad[illegible]s • [illegible]
Player One	• move 13 18 • move 18 22 • move 22 27
Player Two	• saw 19 • saw 19 • saw 19

d) Show a screen copy of entering the commands and the responses point at which it prints out that Player One is the winner.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the modified `CheckCommand` `CheckSpawnCommand` subroutines, the new `BombPiece` `Piece` class and modified methods `ExecuteSpawnCommand` `DestroyAllPiecesAndCountVPs`, `GetPieceTypeInTile` and `ExecuteMoveCommand` of the `HexGrid` class.

b. SCREEN CAPTURE(S) showing the required test.

Task 13

This question refers to the **PlayGame**, **CheckUpgradeCommandForm**
SetUpDefaultGame subroutines in the main program, to the new **Wiz**
ExecuteUpgradeCommand and new **ResetWizards** methods in the

Introduce a new piece, a Wizard which can teleport a distance of three spa
this costs 5 fuel. It can also move one square for a cost of 1 fuel. The piec
upgrading a Serf using the new command upgrade wiz NUM, where NUM
containing a Serf. To move a wizard, you simply use the move command
is two or three squares then it teleports, if it is one square then it moves no
three squares then it is an invalid move. The wizard should be displayed w
for Player Two and should have a VP value of 3.

- Create a new **WizardPiece** class.
- Modify the **CheckUpgradeCommandFormat** subroutine to allow command.
- Modify the method **ExecuteUpgradeCommand** to create the ne
- Add a new method to the **HexGrid** class called **ResetWizards** commands at the end of each turn.
- Modify the **PlayGame** subroutine to call the new **ResetWizards** either side and the new line).
- Modify the **SetUpDefaultGame** subroutine so that Player One s

```
player1 = new Player("Player One", 0, 20, 10,
```

Test that the changes you have made work:

a) Ch[illegible] option 1 to run the default game and then enter the

Player One	• upgrade wiz 8 • move 8 13 • move 13 18
Player Two	• move 23 19 • move 19 11 • upgrade less 11
Player One	• move 18 8 • move 18 13 • move 13 8

b) Show a screen copy of entering the comm[illegible] and the responses point at which it prints out the Player One current state line.

Evidence that you need to provide:

a. You[illegible]OGRAM SOURCE CODE for the amended subroutines **P** **CheckUpgradeCommandFormat**, the **ExecuteUpgradeCo** **ResetWizards** methods of the **HexGrid** class and the new **Wi**

b. SCREEN CAPTURE(S) showing the required test.

Task 14

This question refers to the **PlayGame**, **CheckCommandIsValid**, **CheckUpgrade**
SetUpDefaultGame subroutines in the main program, to the new **SCFWPiece** c
to the **ExecuteUpgradeCommand**, **ExecuteCommand** and new **ExecuteSupply**
HexGrid class.

Introduce a new piece, a supply chain factory wo[illegible] the same cost of 5 l
syntax for the upgrade is "upgrade scfw N[illegible]" where NUM is the location of th
created, the piece is displayed as [illegible] Player1 and f for Player2. A new comm
NUM", which will take all [illegible] your moves (so it can only be the first comm
fuel and will add 1 [illegible] supply chain; NUM is the location of the factory. Th
The factory [illegible] be moved once created.

- Create the new **SCFWPiece** class.
- Modify the **CheckCommandIsValid** subroutine to validate the supply c
- Modify the **CheckUpgradeCommandFormat** subroutine so that it can vali
- Modify the private method **ExecuteUpgradeCommand** of the **HexGrid** c
SCFWPiece to be created.
- Modify the method **ExecuteCommand** of the **HexGrid** class to allow for
calling a new private method, **ExecuteSupplyCommand**.
- Create the new private method **ExecuteSupplyCommand** in the **HexGr**
supply command.
- Modify the **PlayGame** subroutine to only allow the supply command to
three and terminate input for the other two comman[illegible]s if received as th
to call the **AddPieceToSupplyChain** metho[illegible].
- Create a new **AddPieceToSupply[illegible]ain** method in the **Player** class t
supply chain.
- Modify the **SetUpDefaultGame** subroutine so that Player1 starts with 3

```
pl[illegible] new Player("Player One", 0, 30, 30,
```

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the follo

Player One	• move 8 16 • upgrade scfw 16 • supply 16
Player Two	• move 23 19 • move 19 11 • upgrade less 11
Player One	• supply 16

b) Show a screen copy of entering the commands and all the responses

Evidence that you need to provide:

a. You[illegible]GRAM SOURCE CODE for the amended subroutines **Play**
and **CheckUpgradeCommandFormat**, the **ExecuteUpgradeCommand**, **Ex**
ExecuteSupplyCommand methods of the **HexGrid** class, the new **SCFW**
AddPieceToSupplyChain method in the **Player** class.

b. SCREEN CAPTURE(S) showing the required test.

Task 15

This question refers to the `DisplayMainMenu`, `LoadGame` and new `C` the main program. The question also relies on using the `GetGridAsIn` including any methods called by that method. Therefore it is best to begin codebase that you ended Task 5 with.

Introduce a new command to the main menu that will allow a custom game should be able to enter the grid size as a choice of 6, 8 (the current size) or displayed for them to check – this is just a reuse of the code from task 5 and task 5 will be the starting point for this task), the location of a Baron for each of any additional pieces that they wish to add. They should be able to define and peat bogs and define the starting resources (including supply chain) for need to perform any validation on the data entry except for the grid size (w

The prompts that should be given in this new subroutine (in order) are:

- "Please enter the grid size (6, 8 or 10): "
- "Please enter all the locations of peat bogs separated by commands
- "Please enter all the locations of forests separated by commands wit
- "Please enter the location of the Baron for Player 1: "
- "Please enter the location of the Baron for Player 2: "
- "Please enter the starting amount of fuel for Player 1: "
- "Please enter the starting amount of fuel for Player 2: "
- "Please enter the starting amount of lumber for Player 1: "
- "Please enter the starting amount of lumber for Player 2: "
- "Please enter the starting amount of pieces for Player 1: "
- "Please enter the starting amount of pieces for Player 2: "
- "Please enter the starting amount of VPs for Player 1: "
- "Please enter the starting amount of VPs for Player 2: "
- "Enter piece to add for Player One (S=Serf, L=LESS, P=PBDS) or D
- "Enter piece to add for Player Two (S=Serf, L=LESS, P=PBDS) or D
- "What file would you like to save the file as (please include the ex
- "Would you like to play the game that you've created and saved? "

If a piece is chosen by either player, the following prompt should be given:

- "Enter the index of the hex where that piece should be placed: "

Note: There is a bug in the `LoadGame` subroutine that you will need to fix: of the terrain being a field. This needs to be corrected so that it is possible

The game should automatically save in a file (it should prompt for the nam players should have the opportunity to play it immediately if they wish with name. For reference, see the **game1.txt** file supplied with the pre-release as follows (it is all comma-separated values):

Line 1: text for Player One's name, Player One's VPs, Player One's fue Player One's supply chain
Line 2: text for Player Two's name, Player Two's VPs, Player Two's fue Player Two's supply chain
Line 3: [illegible]
Line 4: t[illegible] list all separated by commas in index order
Line 5+: one line for each piece on the board with the format: 1 or 2 to in piece name (case sensitive), index (board location)

TASK CONTINUES ON THE NEXT PAGE

- Modify the `DisplayMainMenu` subroutine to add option 3. Crea
- Modify the Main subroutine so that it calls a new subroutine `Creat` player if they would like to load the game and then loads the game the call `LoadGame` so that it accepts the file name as a parameter
- Create a new subroutine called `CreateGame` which will create th and return two values, `Success` (as a Boolean) and `FileName` (created).
- Modify the `LoadG` subroutine to correct the bug as described a

Test that the changes you have made work:

a) Choose menu option 3 to create a game and then enter the followi prompt:

- 8
- 24,25,6,7
- 4,5,26,27
- 0
- 31
- 12
- 12
- 15
- 15
- 5
- 5
- 0
- 12
- D
- S
- 19
- D
- test.csv
- y

b) Show a screen copy of entering the commands and the responses point at which it prints out the Player One current state line.

c) Show the contents of the file **test.csv** that was created.

Evidence that you need to provide:

a. You PROGRAM SOURCE CODE for the amended subroutines `Lo` `DisplayMainMenu` and the new subroutine `CreateGame` in th

b. SCREEN CAPTURE(S) showing the required test.

HEX BARON – Extension Tasks

These challenges are presented without solutions, and offer to further exp
skeleton program.

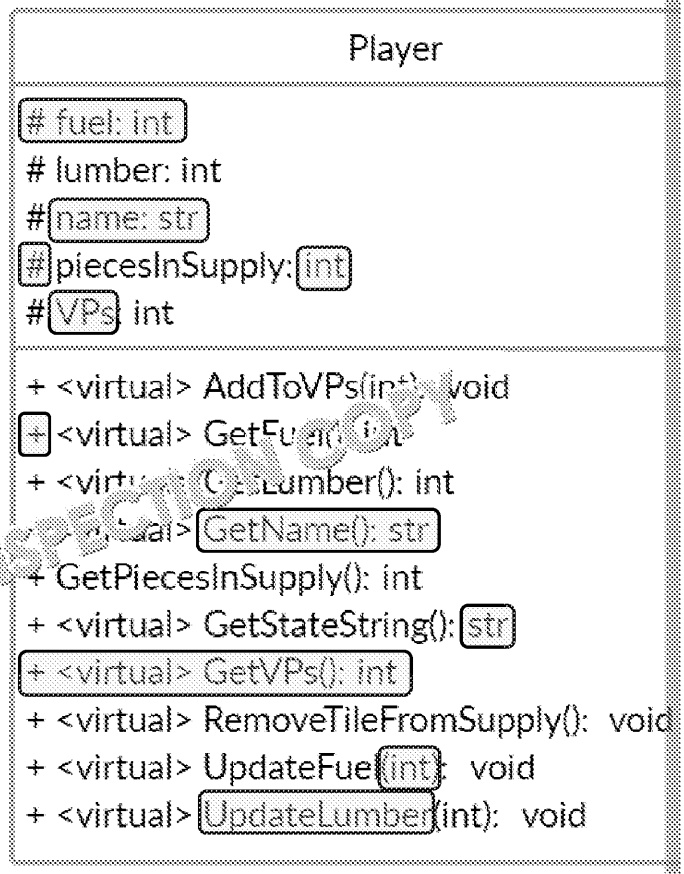
1. Introduce a GOD mode in which resources a[illegible]nite for both side
exited (using the command "godmode on" or "godmode off") then t
rules and the resources revert [illegible]ir values before godmode was
2. Create a computer [illegible]o follow player
3. Mak[illegible]e g[illegible]ore user-friendly so that each time any command
des[illegible]n is give, e.g. 5 lumber deducted for upgrading PBDS in h
con[illegible]ions from hexes 12 & 17 and was destroyed in hex 20.
4. Introduce a new level of upgrade so that pbds and less pieces can
and less pieces for a cost of 10 lumber but will now generate 2 res
command.
5. Introduce a new rule so that connections from your own pieces don
6. Introduce an assassin piece that can use a kill command once per
serf, the kill command will kill the Baron if it is next to it.
7. Introduce a poison piece command so that each player can poison
poisoned piece is killed the victim gains VPs instead of the killer.
8. Introduce a random game command from the main menu where th
resources are randomised (but fairly with suitable rules for distribut
9. Introduce a new blocking/wall piece that requ[illegible]s three connections
invincible for three turns).
10. Change the rules so that th[illegible]or[illegible]ction check for killing a piece is
instead of at the end[illegible]the turn.
11. Add th[illegible]ab[illegible]o modify the terrain (for a cost), or have a new piec
12. Intr[illegible]ranged units that have connections only two edges away
13. Introduce a mini serf (or pleb) piece that only costs 1 but cannot be
automatically killed if it ends the turn next to an enemy. It also does
supply chain to spawn.
14. Have an advanced game mode whereby there is one trap square r
piece landing in it will be destroyed. Alternatively, give each player
trap on any field not adjacent to the opponent's Baron, once per ga
15. Change the rules so that you can choose to have an immovable Ba
lumber and 1 extra fuel every turn.
16. Introduce a game timer so that each player has 180 seconds (or a
constant or inputted variable) to enter their moves. Failure to comp
in the player forfeiting the remainder of their m[illegible]s; whatever they
executed (if possible), after which it w[illegible]o[illegible]e the other player's
17. Add hit points for pieces to[illegible]an[illegible]e mechanic of when/how piec
18. Change the gam[illegible]the coordinates system is removed and
by th[illegible]un[illegible]exes that you need to move through. The IDs sh
he[illegible]on[illegible]ng out / maintaining which tiles are neighbours.
19. Change the game rules so that the game is terminated immediatel
At the moment if Player 1 destroys Player 2's Baron, they still get t
20. Change the victory rules at the end of the game so that once the g
VP for all of their remaining pieces according to the VP value for th

HEX BARON – Class Diagram Tasks (MS)

Task 1

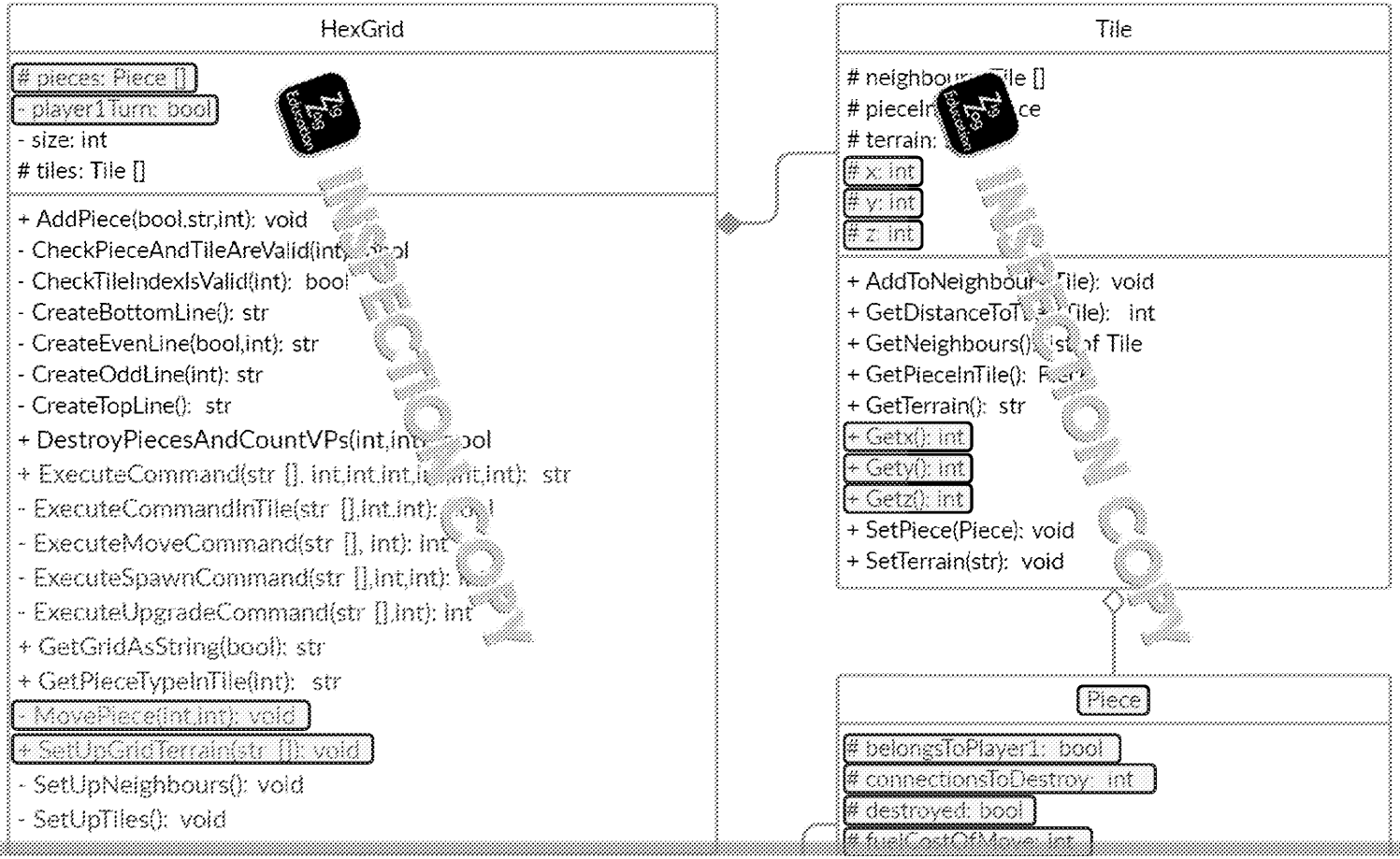
- 1 mark for completing the attributes correctly (as highlighted)
- 1 mark for completing the methods correctly (as highlighted)

A. Any order (it does not have to be alphabetical)



Task 2 (5 marks)

- 1 mark each for correctly completing the HexGrid and Tile classes
- 2 marks for correctly completing the Piece class (deduct 1 mark for a mostly correct attempt)
- 1 mark for adding the missing relationships



Task 3

- The constructors; because the convention is that they are not includ
  include them for automatic code generation).

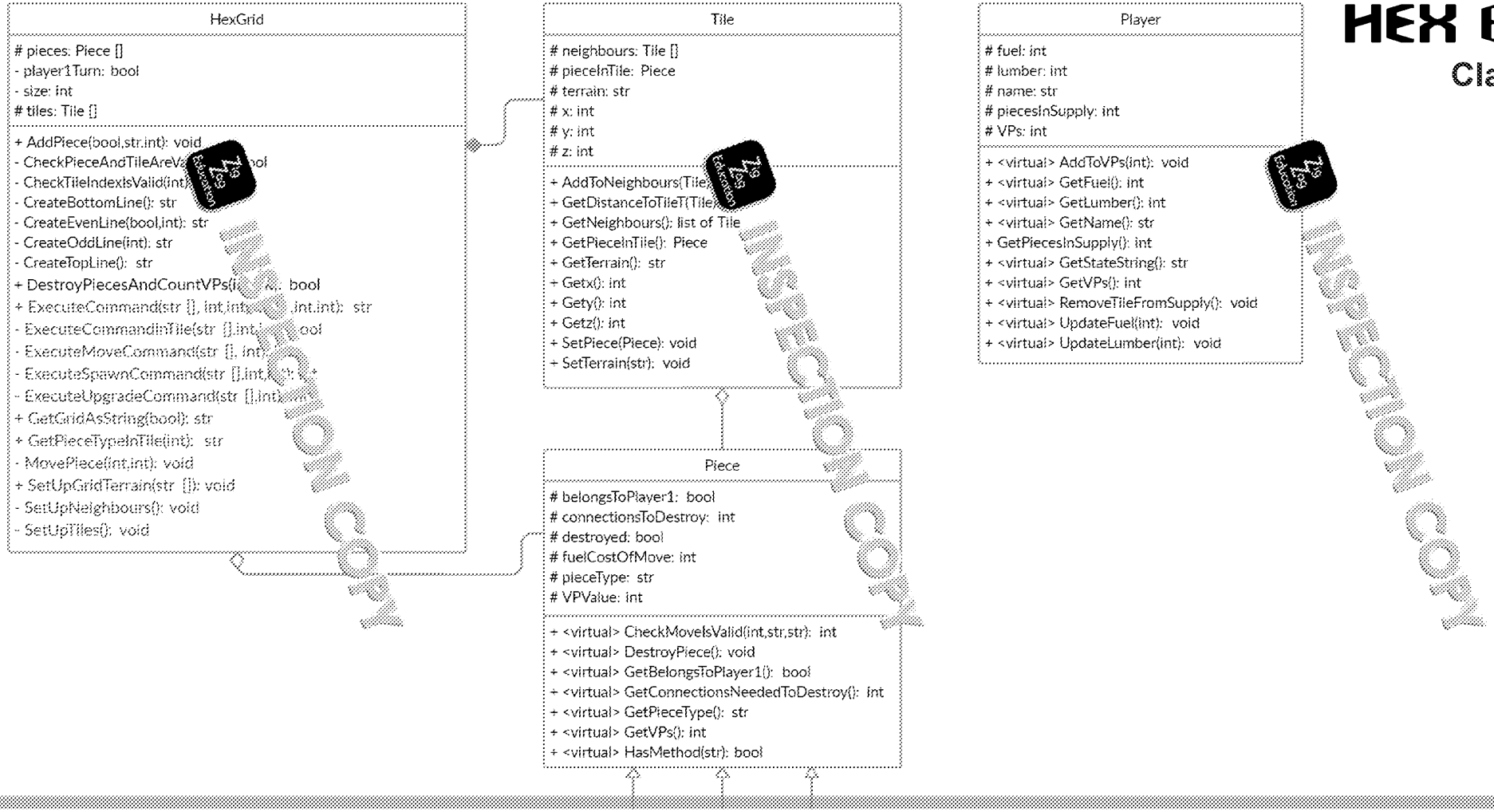
  1 mark for naming constructors and 1 mark for the explanation

- Methods that override methods in their parent class; because the pre
  same name) in the child class serves that it has overridden the meth
  (although some UML diagram tools use a <<override>> stereotype o

  1 ma naming override (or overridden methods) and 1 mark for t

HEX BARON

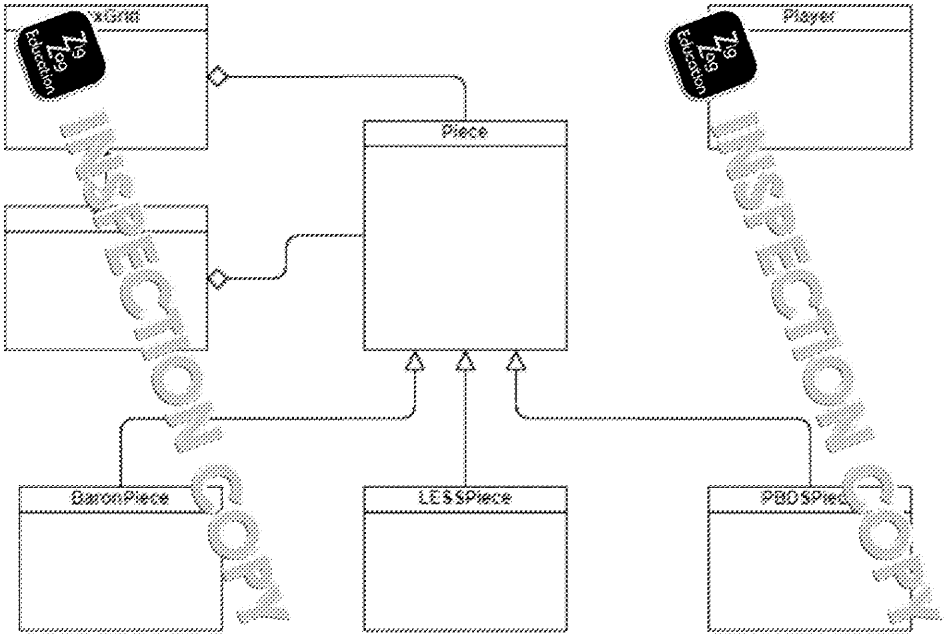
Class Diagram



HEX BARON – Theory Questions (MS)

(Total Marks: 50)

Question		Suggested Solution	Marks	Guidance
1	a	By using `Player1.GetName()` and `Player2.GetName()`	1 mark	• A. any sensible mention of `GetName()`
	b	To provide an interface to private or protected attributes	1 mark	• A. either private or protected • A. access or another similar word for interface
2		The variable `Player1Turn` is toggled using not `Player1Turn`; Player One always gets their turn if Player One ends the game as the game can only end when `Player1Turn` is True	2 marks	• 1 mark for each point
3		`HasMethod` is used to check whether the command that the player typed in is available in the class for the piece in the tile	2 marks	• A. to retrieve the method in the `Piece` class if it exists
4	a	It returns the maximum of three possible numbers: The absolute difference between the x coordinate of the Tile T and itself; The absolute difference between the y coordinate of the Tile T and itself; The absolute difference between the z coordinate of the Tile T and itself	3 marks	• 1 mark for the first point • 1 mark for any of the other three points or 2 marks for all of the last three points • MAX 2 if no mention of absolute (or equivalent)
	b	First, `max(abs(2-4),abs(-3-(-4)))` which is `max(2,1)` = 2 then `max(2,abs(0-0))` which is `max(2,0)` = 0	2 marks	• 1 mark for each max calculation if correct • MAX 1 mark if just `max(2,1,0)` or similar short answer
5	a	Inheritance is where you create a new class that gains all the attributes and methods of its parent	1 mark	• A. similar words or meanings, e.g. behaviours
	b	Because the implementation for this is already in the `Piece` class	1 mark	• A. answers that refer to `Piece` as a concrete class or talk about avoiding it being an abstract class

Question		Suggested Solution	Marks	Guidance
8		Because it is a standard format; That can be used/read easily on all platforms	1 mark	• MAX 1 mark, accept any reasonable point
9			4 marks	• 1 mark for both missing class names (`Piece` and `PBDSP`) • 1 mark for the composition from `HexGrid` to `Tile` • 1 mark for the aggregation from `Tile` to `Piece` • 1 mark for the two inheritance arrows from `BaronPiece` and `LESSPiece` to `Piece` • R. incorrect arrowhead or arrowheads on the wrong end
10	a	The upgrade was unsuccessful	1 mark	• A. any similar statement
	b	The upgrade was successful AND 5 lumber should be deducted from the player	1 mark	• Only accept answers that make both points
11		In the `Dice` method, a random number is generated;	4 marks	• A. blank or space for field

Question		Suggested Solution	Marks	Guidance
13	a	upgrade (pbds|less) (0|[1-9][0-9]*)	3 marks	• 1 mark for upgrade and (pbds|less) • 1 mark for integers including ones starting with 0, such as 01, which are not strictly valid (e.g. [0-9]*) or 2 [illegible] for only strictly positive integers incl[illegible]0
	b	Upgr[illegible] (pbds|less) (0|[1-9][0-9]?)	1 mark	• 1 mark for any [illegible]ression that only allows integers from 0–[illegible] • DPT problem wit[illegible]tegers starting with 0, e.g. 01 from part a
	c	One that [illegible]a be represented by a FSM (with no outputs)	1 mark	• A. one that can be r[illegible]presented by a regular expression
14	a	`GridAsSt[illegible]ng`	1 mark	• A. `count`
	b	Their scope is [illegible]mited to the current subroutine; So that means that you can test the subroutine in isolation; And, therefore, al[illegible]w people to use the subroutine through its inte[illegible]e only so they do not n[illegible]' to concern themselves with the implementatio[illegible]	3 marks	• A. any valid point about [illegible]ope being the subroutine or a related advantage (e.g. no need to worry that there might [illegible]e changes elsewhere) • A. equivalent ideas • A. method or function for subr[illegible]utine
	c	A private attribute is only available within the class where it is defined; Whereas a protected attribute can also be accessed by sub-classes	2 marks	• A. children for sub-classes • A. equivalent points presented from a different perspective
15		GROUP 1 Marks: It checks to see whether there is at least 1 piece in supply; and at least 3 lumber;	6 marks	MAX 3 from Group 1 Marks MAX 4 from Group 2 Marks MAX 6 marks total

HEX BARON – Programming Tasks (MS)

Task 1 (2 marks)

Coding:

- 1 mark for st...ng the leading and trailing spaces from all three inp... commands.

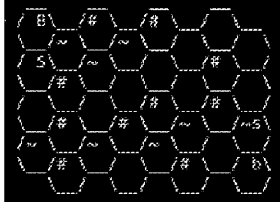
For example:

```
commands.Add(Conso... ReadLine().ToLower().Trim());
```

A. Solutions which ma...ually strip off as many spaces as there are, but no... so...utions that only check for a single space and then remov... i...

Testing: 1 mark for show... that the move command is now valid the first time a... that the upgrade can happen successfully, despite the le... space. For example:

```
1. Default game
2. Load game
Q. Quit

Enter your choice: 1
Player One current state - VPs: 0   P...  in supply: 5   Lumber: 10   Fuel: 10
Player Two current state - VPs: 1   P...  in supply: 5   Lumber: 10   Fuel: 10
```



Task 2 (3 marks)

Coding:

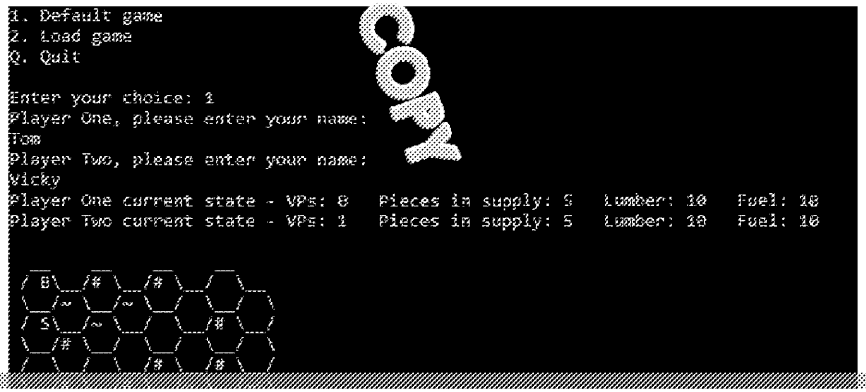
- 1 mark for getting the players' names to be entered by using a **suitable prompt**.
- 1 mark for passing through the names entered correctly to the constructor for the relevant Player objects.

For example:

```
Console.WriteLine("Player One, please enter your name: ");
string player1Name = Console.ReadLine();
Console.WriteLine("Player Two, please enter your name: ");
string player2Name = Console.ReadLine();
player1 = new Player(player1Name, 0, 10, 10, 5);
player2 = new Player(player2Name, 1, 10, 10, 5);
```

A. Solutions which include the ReadLine() command directly within the new Player constructor parameters.

Testing: 1 mark for showing a suitable prompt for the names for each player AND using those names when printing out the states and asking for the entry of the commands. For example:



Task 3 (4 marks)

Coding:

- 1 mark for creating the AddPiecesInSupply method correctly to the Player class.
- 1 mark for passing through player1 and player2 to both calls to DestroyPiecesAndCountVPs and adding the two parameters to the method.
- 1 mark for calling the AddPiecesInSupply method for the correct player so that it adds one piece to their supply.

For example:

AddPiecesInSupply method added to the Player class:

```
public void AddPiecesInSupply(int n)          // Task 3 - New method added
{
    piecesInSupply += n;
}
```

Both calls to DestroyPiecesAndCountVPs:

```
if (gameOver)
{
    grid.DestroyPiecesAndCountVPs(ref player1VPsGained, ref player2VPsGained, ref player1, ref player2);
}
else
{
    gameOver = grid.DestroyPiecesAndCountVPs(ref player1VPsGained, ref player2VPsGained, ref player1, ref player2);
}
```

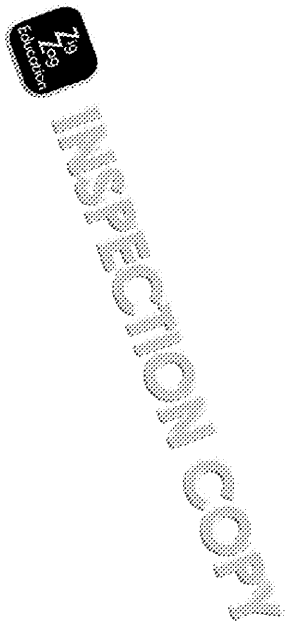
Adding parameters to the DestroyPiecesAndCountVPs method:

```
public bool DestroyPiecesAndCountVPs(ref int player1VPs, ref int player2VPs, ref Player player1, ref Player player2)
```

Modifications to the DestroyPiecesAndCountVPs method:

Testing: 1 mark for showing entry of all three commands correctly and a printout of the pieces in supply and board as shown.

For example:



Task 4 (4 marks)

Coding:

- 1 mark for counting the number of pieces on the board for the current player correctly.
- 1 mark for printing out the message 'Spawn attempted to exceed max pieces' when the number of pieces for the player is 6 or more (award even if the first mark calculated the number of pieces incorrectly).
- 1 mark for returning -1 when the number of pieces for the player is 6 or more AND putting this code before the `Piece newPiece = new Piece(playerlTurn);` line of code (anywhere before it in the method).

For example:

```
        if (!canAaronIsNeighbour)
        {
            return -1;
        }
        //########## Start of new Task 4 code
        int PlayerPieces = pieces.Where(PlayerPiece => PlayerPiece.GetBelongsToPlayer1() == playerlTurn).Count();
        if (PlayerPieces >= 6)
        {
            Console.WriteLine("Spawn attempted to exceed max pieces");
            return -1;
        }
        //########## End of new Task 4 code
        Piece newPiece = new Piece(playerlTurn);
        pieces.Add(newPiece);
        tiles[tileToUse].SetPiece(newPiece);
        return 3;
    }
```

```
1. Default game
2. Load game
Q. Quit

Enter your choice: 1
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 10   Fuel: 10
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 10   Fuel: 10
```

A. Solutions which use a loop instead of a LINQ.

Task 5 (9 marks)

Coding:

- 1 mark for calling the new method when the 'hexes' command is entered.
- 1 mark for making sure that another command can be entered and that the 'hexes' command did not count as one of the three commands (no matter how many times they use it).
- 1 mark for using a variable to track the tile index and incrementing it throughout the process.
- 1 mark for creating the string correctly for the top and bottom lines (only if the format is correct).
- 1 mark for creating the string with the correct index numbers for the even lines (even if the format is off).
- 1 mark for creating the string with the correct index numbers for the odd lines (even if the format is off).

For example:

Modifications to the PlayGame subroutine:

```
do
{
    Console.WriteLine(grid.GetGridAsString(player1Turn));
    if (player1Turn)
        Console.WriteLine(player1.GetName() + " state your three commands, pressing enter after each one.");
    else
        Console.WriteLine(player2.GetName() + " state your three commands, pressing enter after each one.");
    for (int count = 1; count <= 3; count++)
    {   //##############################################################Start of new Task 5 code
        Console.Write("Enter command: ");
        string command = Console.ReadLine().ToLower();            //Task 5 - New Input added for Hexes Command
        while (command == "hexes")
        {
            Console.WriteLine(grid.getGridAsIndicies());
            Console.Write("Enter command: ");
            command = Console.ReadLine().ToLower();
        }
```

```
        {
            listPositionOfTile += 1;
            gridAsString += CreateEvenLineIndices(false, ref listPositionOfTile, ref index);
            listPositionOfTile += 1;
            gridAsString += CreateOddLineIndices(ref listPositionOfTile, ref index);
        }
        return gridAsString + CreateBottomLine();
    }
```

Code for the new C[illegible]OddLineIndices method:

```
    private string CreateOddLineIndices(ref int listPositionOfTile, ref int index)      //Task 6 new method to draw
    {                                                                                   //boxes with indices showing
        string line = "";
        for (var count = 1; count <= size / 2; count++)
        {
            if (count > 1 & count < size / 2)
            {
                line += @"\  /";
                listPositionOfTile += 1;
                line += String.Format("{0,-2}", index);
                index += 1;
            }
            else if (count == 1)
            {
                line += @"\__/" + String.Format("{0,-2}", index);
                index += 1;
            }
        }
        line += @"\__/";
        listPositionOfTile += 1;
        if (listPositionOfTile < tiles.Count())
        {
```

Code for the new CreateEvenLineIndices method:

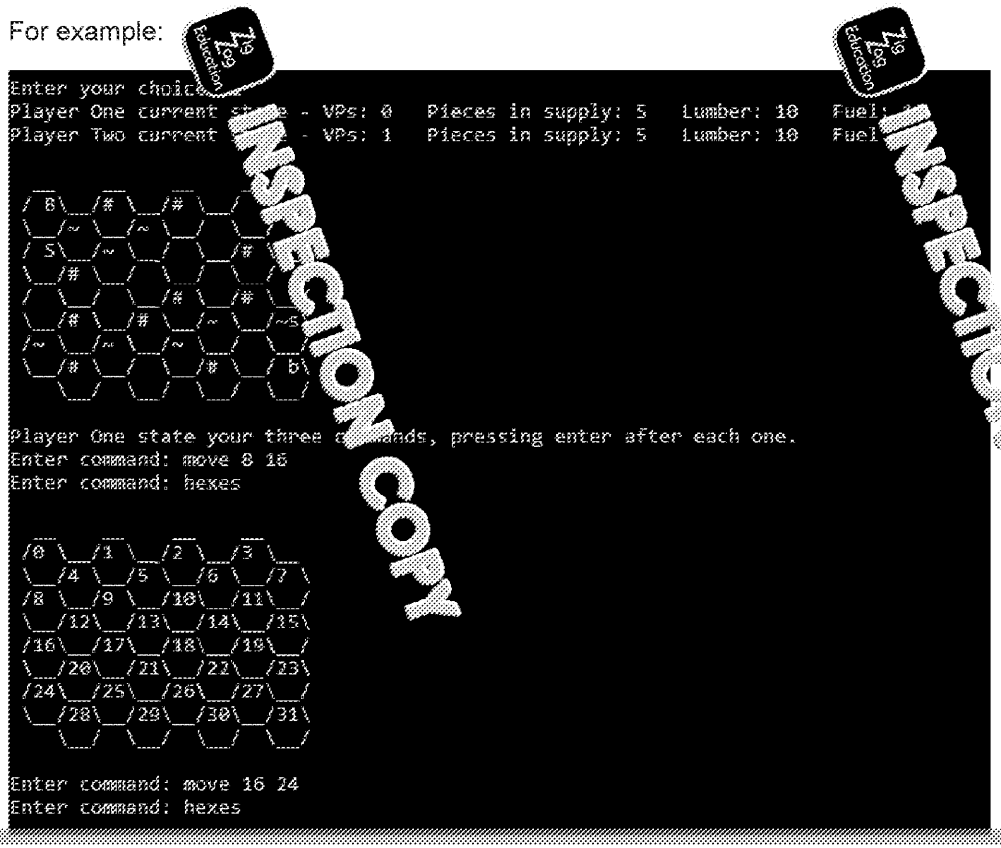
```
private string CreateEvenLineIndices(bool firstEvenLine, ref int listPositionOfTile, ref int index) //Task 5 - New method to
draw even line hexes with indices showing
{
    string line = " /" + String.Format("{0,-2}", index);
    listPositionOfTile += 1;
    for (var count = 1; count <= size / 2 - 1; count++)
    {
        listPositionOfTile += 1;
        line += @"\__/" + String.Format("{0,-2}", index);
        index += 1;
    }
    if (firstEvenLine)
    {
        line += @"\__" + Environment.NewLine;
    }
    else
    {
        line += @"\__/" + Environment.NewLine;
    }
    return line;
}
```

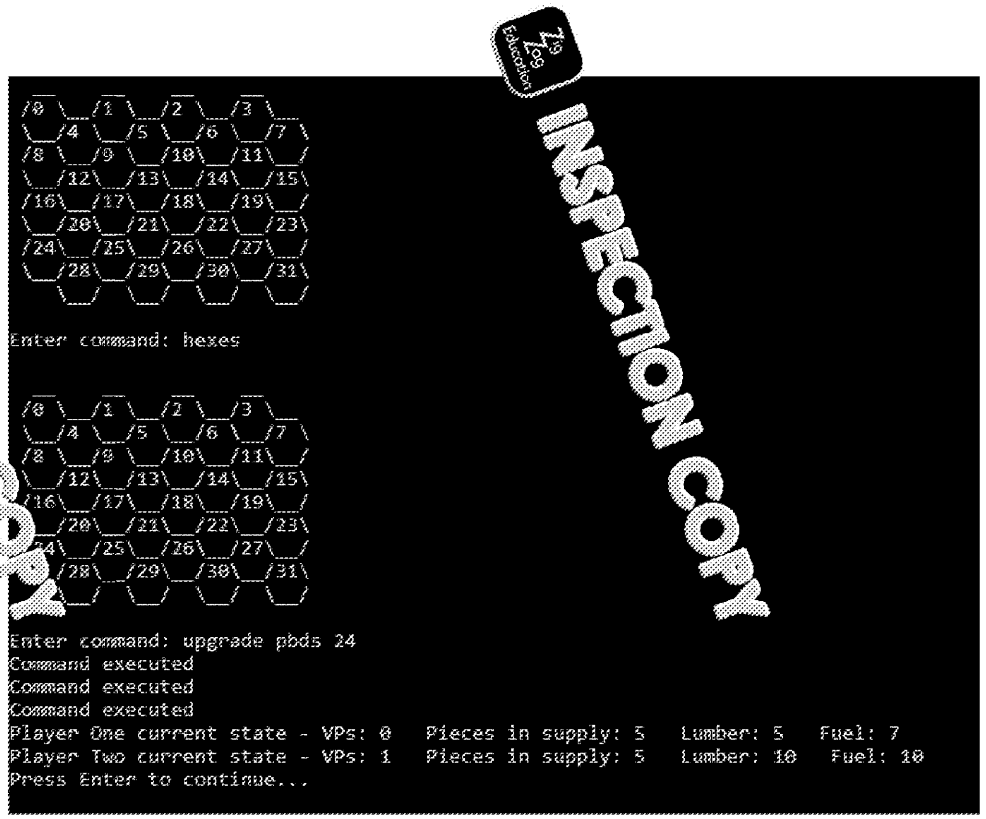
A. Alternative solutions that simply print out the grid without modifying or reusing existing code, just award the marks based on the mark scheme above, but allow outputting of the correct line instead of having to store it in a variable.

Testing:

- 1 mark for printing out the first line of the grid correctly formatted with the numbers 0–3 shown for the index numbers.
- 1 mark for printing out the whole of the grid with all the numbers correctly from 0–31 even if the formatting is off.
- 1 mark for printing out the whole of the grid, correctly formatted with the correct numbers for each hex.

For example:





Task 6 (4 marks)

Coding:

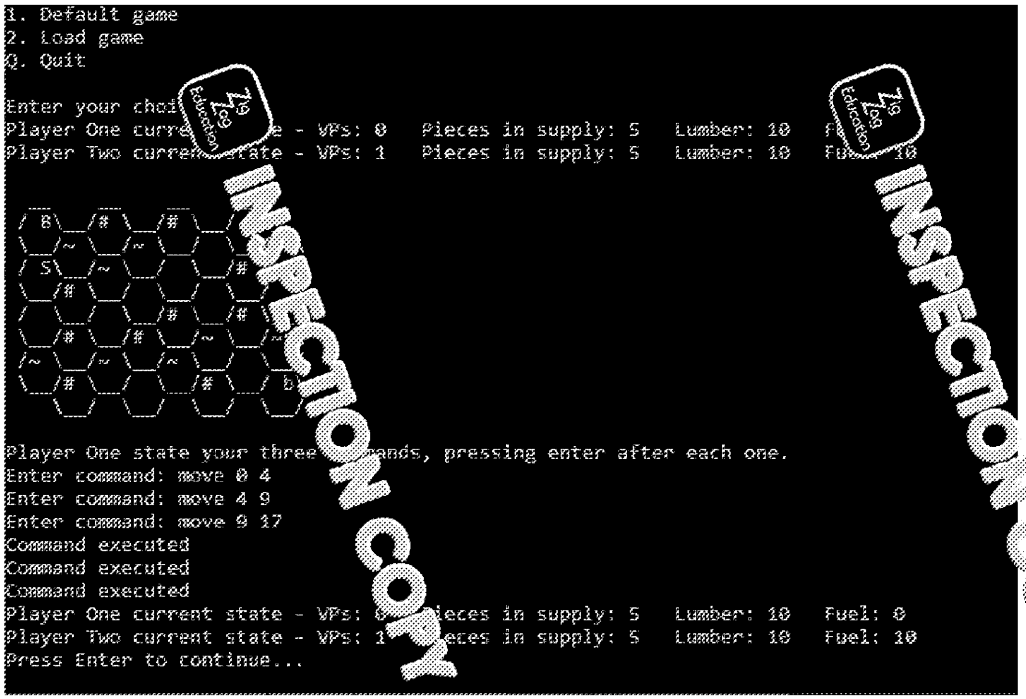
- 1 mark for initialising a counter to 0 before the for C in Commands loop and then incrementing it inside the loop (even if logic is otherwise incorrect).
- 1 mark for using the correction criteria for a selection structure to check that the player has made three valid move commands.
- 1 mark for giving Player 1 additional fuel before the third move is made (or discounting the cost of the third move).

For example:

```
        Console.Write("Enter command: ");
        commands.Add(Console.ReadLine().ToLower());
    }
    //################################################Start of new Task 6 code
    int validMoveCommands = 0;
    foreach (var c in commands)
    {
        List<string> items = new List<string>(c.Split(' '));
        var validCommand = CheckCommandIsValid(items);
        if (!validCommand)
            Console.WriteLine("Invalid command");
        else
        {
            if (items[0] == "move")      //Task 6 - New code to count the number of move commands and change fuel usage
            {
                validMoveCommands += 1;
                if (validMoveCommands == 3)
                {
                    if (player1Turn)
                    {
                        player1.UpdateFuel(1);
                    }
                    else
```

Testing: 1 mark for showing the commands entered correctly and that the fuel for Player One is now 0 in the printout of their state.

For example:



Task 7 (7 marks)

Coding:

- 1 mark for correctly removing the code from ExecuteCommandInTile.
- 1 mark for initialising variables to track the valid dig and saw commands.
- 1 mark for recording that the three commands happened at the same location (before awarding the extra resource and generating the field).
- 1 mark for correctly awarding the extra 2 resources for 3 consecutive dig or saw commands (even if they didn't check the location).
- 1 mark for calling MakeField under the correct circumstances (3 valid saws or digs in the same location in the same turn).
- 1 mark for correctly creating the MakeField method.

For example:

Modifications to ExecuteCommandInTile:

```
if (thePiece.HasMethod(items[0]))
{
    string methodToCall = items[0];
    Type t = thePiece.GetType();
    System.Reflection.MethodInfo method = t.GetMethod(methodToCall);
    object[] parameters = { tiles[tileToUse].GetTerrain() };
    if (items[0] == "Saw")
    {
        lumber += Convert.ToInt32(method.Invoke(thePiece, parameters));
    }
    else if (items[0] == "Dig")
    {
        fuel += Convert.ToInt32(method.Invoke(thePiece, parameters));
        //##########################################################Start of new Task 7 code
        //Task 7 - Removed the code which generates a field
        //############################################################End of new Task 7 code
    }
    return true;
```

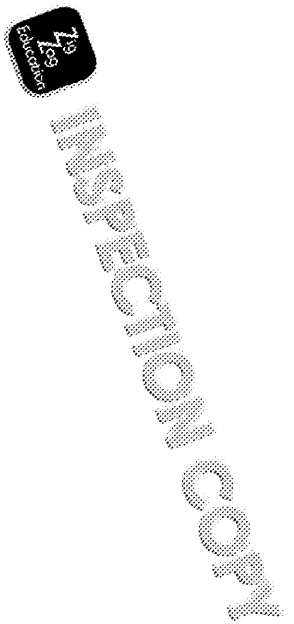
```
{
    Console.Write("Enter command: ");
    commands.Add(Console.ReadLine().ToLower());
}
//##################################################################Start of new Task 7 code
int validDigCommands = 0;
int validSawCommands = 0;
string digSawLocation = "";
foreach (var c in commands)
{
    List<string> items = new List<string>(c.Split(' '));
    validCommand = CheckCommandIsValid(items);
    if (!validCommand)
        Console.WriteLine("Invalid command");
    else
    {
        if (digSawLocation == "")
        {
            digSawLocation = items[1];
        }
        if (items[0] == "saw" && items[1] == digSawLocation)
        {
            validSawCommands += 1;
        }
        else if (items[0] == "dig" && items[1] == digSawLocation)
        {
            validDigCommands += 1;
        }
        int fuelChange = 0;
        int lumberChange = 0;
        int supplyChange = 0;
        string summaryOfResult;
        if (player1Turn)
        {
            summaryOfResult = grid.ExecuteCommand(items, ref fuelChange, ref lumberChange, ref supplyChange,
```

```
                player2.RemoveTileFromSupply();
            }
        }
        if (validSawCommands == 3)
        {
            if (player1Turn)
            {
                player1.UpdateLumber(2);
            }
            else
            {
                player2.UpdateLumber(2);
            }
            grid.MakeField(Convert.ToInt32(digSawLocation));
        }
        else if (validDigCommands == 3)
        {
            if (player1Turn)
            {
                player1.UpdateFuel(2);
            }
            else
            {
                player2.UpdateFuel(2);
            }
            grid.MakeField(Convert.ToInt32(digSawLocation));
        }
        Console.WriteLine(summaryOfResult);
    }
}
//##########################################################End of new Task 7 code
```

Code for new MakeField method:

Testing: 1 mark for showing the final board after all 12 commands have been executed including the state of both players.

For example:



Task 8 (7 marks)

Coding:

- 1 mark for correctly modifying the CheckCommandIsValid subroutine to recognise downgrade and call the new subroutine.
- 1 mark for creating the new subroutine CheckDowngradeCommandFormat and having it return True when there are two items and the second one is an integer, and otherwise returning False.
- 1 mark for modifying the ExecuteCommand method to add the new downgrade command and call the new ExecuteDowngradeCommand method.
- 1 mark for correctly processing the return value from the new method and ensuring that the player will receive a refund of 1 lumber.
- 1 mark for creating the ExecuteDowngradeCommand method and having it check that the piece in the specified tile is either a PLANK or a LESS.
- 1 mark for correctly converting the piece to a Serf and updating the Piece list.

For example:

Code for modified CheckCommandIsValid subroutine:

```
            case "upgrade":
                {
                    return CheckUpgradeCommandFormat(items);
                }
            //#########################################################Start of new Task 8 code
            case "downgrade":
                {
                    return CheckDowngradeCommandFormat(items);
                }
            //#########################################################End of new Task 8 code
        }
    }
```

Code for new CheckDowngradeCommandFormat subroutine:

```
    public static bool CheckDowngradeCommandFormat(List<string> items)
    {
        int result;
```

Code for modified ExecuteCommand method:

```
        case "upgrade":
            {
                int lumberCost = ExecuteUpgradeCommand(items, lumberAvailable);
                if (lumberCost < 0)
                    return "Upgrade not possible";
                lumberChange = -lumberCost;
                break;
            }
        ##########################################Start of new Task 3 code
        case "downgrade":
            {
                int lumberCost = ExecuteDowngradeCommand(items, lumberAvailable);
                if (lumberCost == 0)
                {
                    return "Downgrade not possible";
                }
                lumberChange = -lumberCost;
                break;
            }
        ##########################################End of new Task 3 code
    }
```

Code for new ExecuteDowngradeCommand method:

```
private int ExecuteDowngradeCommand(List<string> items, int lumberAvailable)
{
    int tileToUse = Convert.ToInt32(items[1]);
    if (!CheckPieceAndTileAreValid(tileToUse) || lumberAvailable < 1)
    {
        return 0;
    }
    else
    {
        Piece thePiece = tiles[tileToUse].GetPieceInTile();
        if (!(thePiece.GetPieceType().ToUpper() == "P" || thePiece.GetPieceType().ToUpper() == "L"))
```

Testing: 1 mark for showing the commands entered correctly and the error message as above along with the system error message saying that spawning did not occur.

For example:

Task 9 (3 marks)

Coding:

- 1 mark for correctly overriding the protected attribute ConnectionsToDestroy in the BaronPiece class.

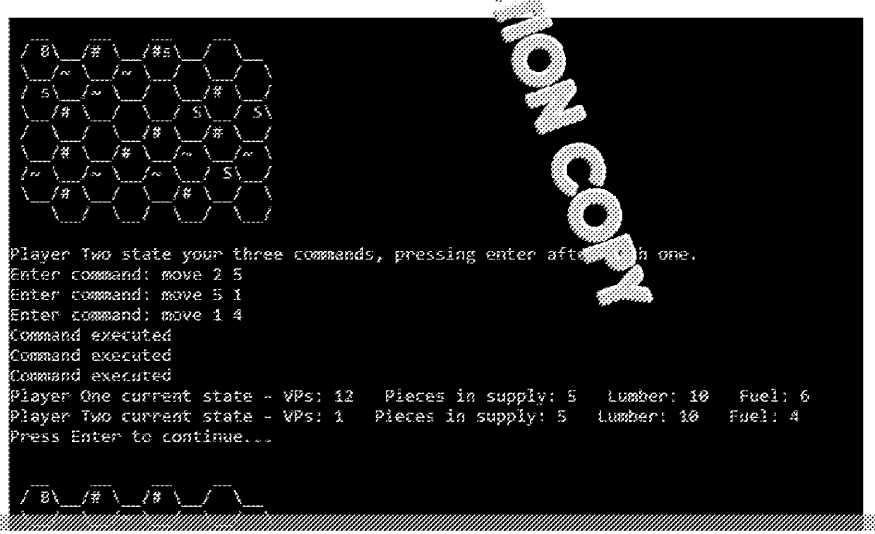
For example:

```
    public BaronPiece(bool player1)
        : base(player1)
    {
        connectionsToDestroy = 3;       //Task 9
        pieceType = "B";
        VPValue = 10;
    }
```

Testing:

- 1 mark for showing the Player Two Baron was destroyed.
- 1 mark for showing the Player Two Serfs were destroyed but that the Player One Baron was not.

For example:



Task 10 (4 marks)

Coding:

- 1 mark for modifying CheckCommandIsValid to add 'salvage' to the list of commands with a standard format.
- 1 mark for printing out the message 'Salvaging was not possible' (or similar) when the command attempts to salvage an empty hex or the Baron or one of the other player's
- 1 mark fo ctly removing the piece from the board when it is ed.

For example:

Code for modified C CommandIsValid subroutine:

```
            case "salvage":                 //Task 10 - additional option added in for salvage
                {
                    return CheckStandardCommandFormat(items);
                }
            case "upgrade":
                {
                    return CheckUpgradeCommandFormat(items);
                }
            }
        }
        return false;
```

Code for modified ExecuteComma method:

```
        case "dig":
            {
                if (!ExecuteCommandInTile(items, ref fuelChange, ref lumberChange))
                {
                    return "Couldn't do that";
                }
                else
```

Code for new ExecuteSalvageCommand method:

```
private int ExecuteSalvageCommand(List<string> items)
{
    int tileToUse = Convert.ToInt32(items[1]);
    if (!CheckPieceAndTileAreValid(tileToUse))
    {
        return 0;
    }
    Piece ThePiece = tiles[tileToUse].GetPieceInTile();
    bool player1SalvagablePiece = true, player2SalvagablePiece = true;
    if (player1Turn && (ThePiece.GetPieceType() != "P" && ThePiece.GetPieceType() != "L"))
    {
        player1SalvagablePiece = false;
    }
    if (!player1Turn && (ThePiece.GetPieceType() != "P" && ThePiece.GetPieceType() != "L"))
    {
        player2SalvagablePiece = false;
    }
    if (!player1SalvagablePiece || !player2SalvagablePiece)
    {
        return 0;
    }
    ThePiece.DestroyPiece();
    return -5;
}
```

A. Solutions which are similar to Task 8 and have their own case condition and checking function but only give the same mark as if they solved the problem the short way.

Testing: 1 mark for showing the commands entered correctly and the error message as above along with the existing error message saying that salvaging was not possible (twice).

For example:

Task 11 (10 marks)

Coding:

- 1 mark for making the modifications to the Player class so that it now has a private chances attribute and three methods to access it: one that will subtract one from the number of chances, one that will return the number of chances and one that will reset the number of chances to 3.
- 1 mark for setting the chances to 3 at the start of each game for both players.
- 1 mark for allowing a command to be re-entered if it is of the incorrect format and deducting one chance for this (even if it only works in one place).
- 1 mark for allowing a command to be re-entered if it is of the correct format but could not be executed (e.g. move 8 8) and deducting one chance for this (even if it only works in one place).
- 1 mark for making sure that you always have exactly 3 chances.
- 1 mark for printing out an error message every time a command is entered that is invalid or could not be executed correctly.
- 1 mark for ensuring that all command entry prompts and all command error messages have an accurate command number associated with them (from 1 to 3).

For example:

Code for modified Player class:

```
class Player
{
    protected int piecesInSupply, fuel, VPs, lumber, chances;      //Task 11 - chances property also added
    protected string name;

    public Player(string n, int v, int f, int l, int t)
    {
        name = n;
        VPs = v;
        fuel = f;
        lumber = l;
        piecesInSupply = t;
        ResetChances();                 //Task 11 - Chances set to 3 in the constructor
    }
    //##########################################################Start of new Task 11 code
    public void DeductChance()
```

Code for modified PlayGame subroutine:

```
public static void PlayGame(Player player1, Player player2, HexGrid grid)
{//##########################################################Start of new Task 11 code
    bool gameOver = false;
    bool player1Turn = true;
    bool validCommand = false;
    List<string> commands = new List<string>();
    player1.ResetChances();          //Task 11
    player2.ResetChances();          //Task 11
    Console.WriteLine("Player One current state - " + player1.GetStateString());
    Console.WriteLine("Player Two current state - " + player2.GetStateString());
    do
    {
        Console.WriteLine(grid.GetGridAsString(player1Turn));
        Player currentPlayer;
        if (player1Turn)
        {
            currentPlayer = player1;
        }
        else
        {
            currentPlayer = player2;
        }
        Console.WriteLine(currentPlayer.GetName() + " you have " + currentPlayer.GetChances() + " chances remaining.");
        Console.WriteLine(currentPlayer.GetName() + " state three commands, pressing enter after each.");
        for (int count = 1; count <= 3; count++)
        {
            Console.Write("Enter command " + Convert.ToString(count) + ":");
            commands.Add(Console.ReadLine().ToLower());
        }
        for (int CommandNo = 0; CommandNo < commands.Count(); CommandNo++)
        {
            List<string> items = new List<string>(commands[CommandNo].Split(' '));
            validCommand = CheckCommandIsValid(items);
```

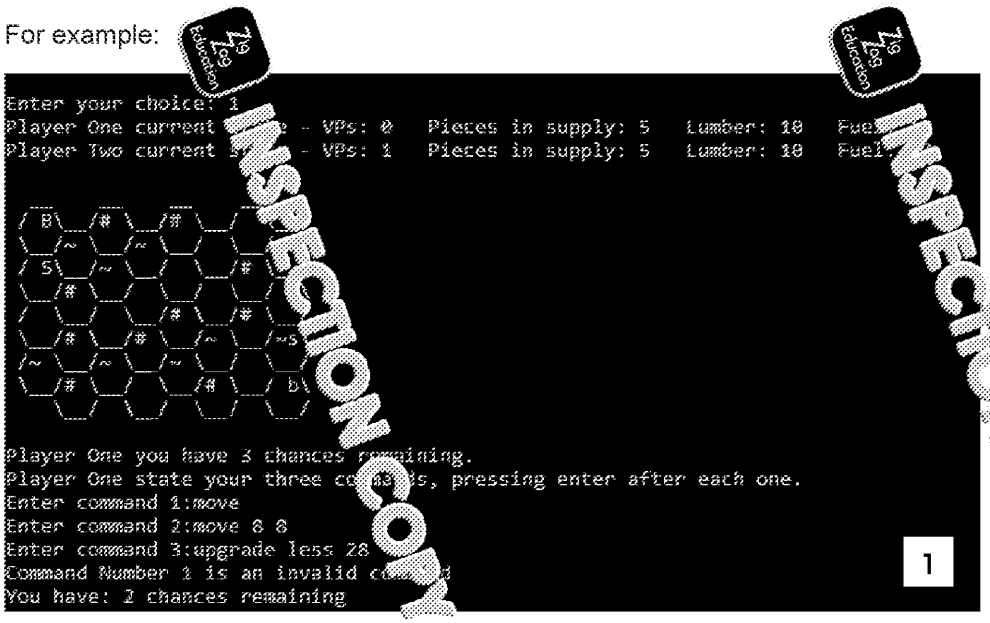
```
        Console.WriteLine("Command number " + Convert.ToString(CommandNo + 1) + " is an invalid command");
    }
    bool commandExecuted = false;
    while (!commandExecuted && validCommand)
    {
        int fuelChange = 0;
        int lumberChange = 0;
        int supplyChange = 0;
        string summaryOfResult = grid.ExecuteComm...(items, ref fuelChange, ref lumberChange, ref supplyChange,
            currentPlayer.GetFuel(), currentPlayer.GetLumber(), currentPlayer.GetPle...InSupply());
        currentPlayer.UpdateLumber(lumberCh...e);
        currentPlayer.UpdateFuel(fuelChange);
        if (supplyChange == 1)
        {
            currentPlayer.RemoveTileFromSuppl...();
        }
        Console.WriteLine("Command " + Convert...String(CommandNo + 1) + ": " + summaryOfResult);
        if (summaryOfResult == "Command executed")
        {
            commandExecuted = true;
        }
        else if (currentPlayer.GetChances() > 0)
        {
            Console.Write("Enter new command " + Co...rt.ToString(CommandNo + 1) + ": ");
            string command = Console.ReadLine().ToLo...();
            currentPlayer.DeductChance();
            items = new List<string>(command.Split(' '));
            validCommand = CheckCommandIsValid(items);
            if (!validCommand && (currentPlayer.GetChances() == 0))
            {
                Console.WriteLine("Command number " + Convert.ToString(CommandNo + 1) + " is an invalid command");
            }
        }
        while (!validCommand && (currentPlayer.GetChances() > 0));
```

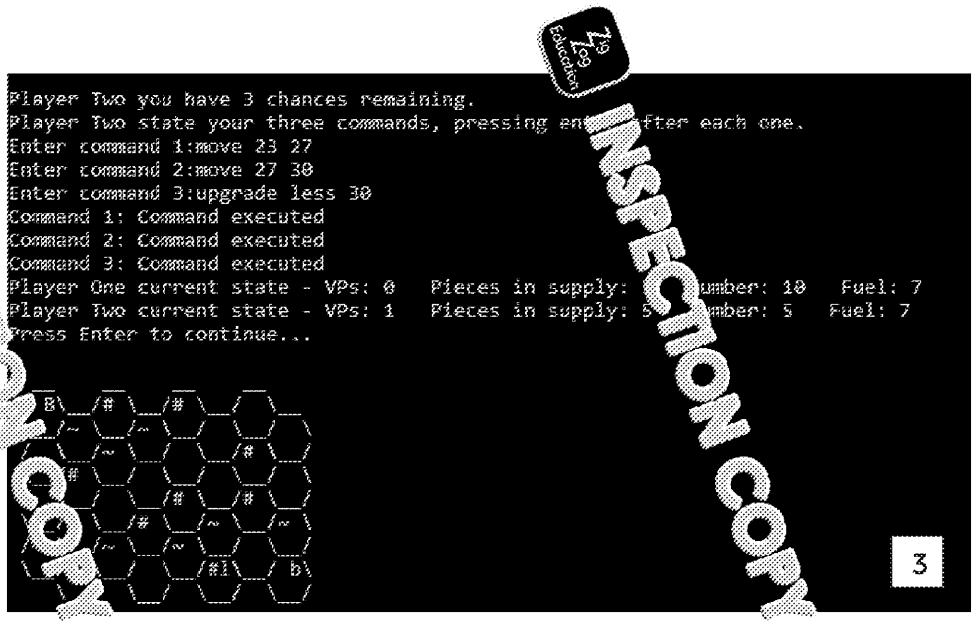
```
        {
            commandExecuted = true;
            validCommand = true;
            break;
        }
    }
    else
    {
        break;
    }
}
Console.Clear();
playerTurn = !playerTurn;
int player1VPsGained = 0;
int player2VPsGained = 0;
if (gameOver)
{
    grid.DestroyPiecesAndCountVPs(ref player1VPsGained, ref player2VPsGained);
}
else
{
    gameOver = grid.DestroyPiecesAndCountVPs(ref player1VPsGained, ref player2VPsGained);
}

player1.AddToVPs(player1VPsGained);
player2.AddToVPs(player2VPsGained);
Console.WriteLine("Player One current state -- " + player1.GetStateString());
Console.WriteLine("Player Two current state -- " + player2.GetStateString());
Console.Write("Press Enter to continue...");
Console.ReadLine();
```

Testing:

- 1 mark for showing the commands entered correctly and the commands and error messages all being numbered clearly (even if the numbering is incorrect).
- 1 mark for displaying the number of chances correctly each turn and persisting them between turns (e.g. Player One's second turn shows 0 chances remaining).
- 1 mark for displaying the final error message correctly and not letting Player One have a fourth chance.

For example:





Task 12 (13 marks)

Coding:

- 1 mark for modifying the CheckCommandIsValid method to process the new format for the spawn bomb variant of the spawn command.
- 1 mark for correctly checking the format of the new spawn command (ideally using the CheckSpawnCommandFormat method they wrote).
- 1 mark for modifying the spawn command in ExecuteSpawnCommand so that it will correctly spawn a bomb and charge 10 lumber (do not award if they don't check for the availability of the lumber).
- 1 mark for having a new BombPiece class and setting the FuelCostOfMove to 2.
- 1 mark for having a mechanism to count the number of moves until primed and then detecting when the bomb is primed.
- 1 mark for having a method to destroy the bomb when it is killed due to no connections (prior to being primed).
- 1 mark for having a mechanism for exploding the bomb when it is triggered and destroying all the surrounding pieces (award even if this is done at the end of the turn).
- 1 mark for correctly destroying the bomb the moment that it is triggered and not at the end of the turn.
- 1 mark for correctly awarding all of the VPs for the explosion.
- 1 mark for making the bomb display as a Serf piece for the appropriate player.

For example:

Code for modified CheckCommandIsValid subroutine:

```
        case "move":
            {
                return CheckMoveCommandFormat(items);
            }
        //##########################################################Start of new Task 12 code
        case "dig":
        case "saw":
            {
                return CheckStandardCommandFormat(items);
            }
```

Code for new CheckSpawnCommandFormat subroutine:

```
public static bool CheckSpawnCommandFormat(List<string> items)
{
    if (items.Count == 2)
    {
        return CheckStandardCommandFormat(items
    }
    {
        try
        {
            if (items[1] == "bomb")
            {
                int result = Convert.ToInt32(items[2]);
            }
        }
        catch
        {
            return false;
        }
        return true;
    }
}
```

Code for modified private Execute[illegible]awnCommand method:

```
private int ExecuteSpawnCommand(List<string> items, int lumberAvailable, int piecesInSupply)
{
    int tileToUse;
    int spawnCost;
    if (items.Count == 2)
    {
        tileToUse = Convert.ToInt32(items[1]);
```

```
        return -1;
    }
    bool ownBaronIsNeighbour = false;
    List<Tile> listOfNeighbours = new List<Tile>(tiles[tileToUse].GetNeighbours());
    foreach (var n in listOfNeighbours)
    {
        ThePiece = n.GetPieceInTile();
        if (ThePiece != null)
        {
            if (playerlTurn && ThePiece.GetPieceType() == "B" || !playerlTurn && ThePiece.GetPieceType() == "b")
            {
                ownBaronIsNeighbour = true;
                break;
            }
        }
    }
    if (!ownBaronIsNeighbour)
    {
        return -1;
    }
    if (items[1] == "bomb")
    {
        Piece newPiece = new BombPiece(playerlTurn);
        pieces.Add(newPiece);
        tiles[tileToUse].SetPiece(newPiece);
    }
    else
    {
        Piece newPiece = new Piece(playerlTurn);
        pieces.Add(newPiece);
        tiles[tileToUse].SetPiece(newPiece);
    }
    return spawnCost;
}
```

```
    if (distanceBetweenTiles == 1)
    {
        if (movesBeforePrimed > 0)
        {
            movesBeforePrimed -= 1;
            return fuelCostOfMove;
        }
        return -1;
    }
    public override void DestroyPiece(Tile tileLocation,
    {
        if (!destroyed)
        {
            collectionsToDestroy = 0;
            movesBeforePrimed = -1; // as the bomb is no longer primed
            List<Tile> neighbours = tileLocation.GetNeighbours();
            foreach (Tile neighbour in neighbours)
            {
                Piece tilePiece = neighbour.GetPieceInTile();
                if (tilePiece != null)
                {
                    tilePiece.Explode();
                }
            }
            destroyed = true;
        }
    }
    public override bool Explode(Tile TileLocation, ref int p1VPs, ref int p2VPs, ref bool baronDestroyed)
    {
        if (!destroyed)
        {
            List<Tile> neighbours = TileLocation.GetNeighbours();
            foreach (Tile neighbour in neighbours)
            {
```

```
                }
                tilePiece.DestroyPiece();
                neighbour.SetPiece(null);
            }
        }

        destroyed = true;
        tileLocation.SetPiece(null);

        return baronDestroyed;
    }
    public override bool Primed()
    {
        return movesBeforePrimed == 0;
    }
}
```

Code for modified DestroyPiecesAndCountVPs method:

```
public bool DestroyPiecesAndCountVPs(ref int player1VPs, ref int player2VPs)
{
    bool baronDestroyed = false;
    List<Tile> listOfTilesContainingDestroyedPieces = new List<Tile>();
    foreach (var t in tiles)
    {
        if (t.GetPieceInTile() != null)
        {
            List<Tile> listOfNeighbours = new List<Tile>(t.GetNeighbours());
            int noOfConnections = 0;
            foreach (var n in listOfNeighbours)
            {
                if (n.GetPieceInTile() != null)
                {
                    noOfConnections += 1;
                }
```

```
            if (thePiece.GetPieceType().ToUpper() == "B")
            {
                baronDestroyed = true;
            }
        }
        listOfTilesContainingDestroyedPieces.Add(t);
        if (thePiece.GetBelongsToPlayer
        {
            player2VPs += thePiece.GetV
        }
        else
        {
            player1VPs += thePiece.GetVPs();
        }

    }
    foreach var t in listOfTilesContainingDestroyedPie
    {
        t.Set
    }
    return baron
}
```

Code for modified GetPieceType
 method:

```
public string GetP
{
    Piece thePiece = tiles[ID].GetPieceInTile();
    if (thePiece == null)
    {
        return " ";
    }
    else
```

Code for modified ExecuteMoveCommand method:

```
private int ExecuteMoveCommand(List<string> items, int fuelAvailable)
{
    int startID = Convert.ToInt32(items[1]);
    int endID = Convert.ToInt32(items[2]);
    if (!CheckPieceAndTileAreValid(startID) || !CheckTileIndexIsValid(endID))
    {
        return -1;
    }
    Piece thePiece = tiles[startID].GetPieceInTile();
    if (tiles[endID].GetPieceInTile() != null)
    {
        return -1;
    }
    int distance = tiles[startID].GetDistanceToTile(tiles[endID]);
    int fuelCost = thePiece.CheckMoveIsValid(distance, tiles[startID].GetTerrain(), tiles[endID].GetTerrain());
    if (fuelCost == -1 || fuelAvailable < fuelCost)
    {
        return -1;
    }
    MovePiece(endID, startID);
    List<Tile> neighbours = tiles[endID].GetNeighbours();
    foreach (Tile neighbour in neighbours)
    {
        Piece pieceInTile = neighbour.GetPieceInTile();
        if (pieceInTile != null)
        {
            if (pieceInTile.GetPieceType().ToUpper() == "X" && pieceInTile.Primed())
            {
                pieceInTile.DestroyPiece(neighbour);
            }
        }
    }
    return fuelCost;
```

```
    }
    //########################################################Start of new Task 12 code
    public virtual void DestroyPiece(Tile tileLocation = null) //Task 12 - new overridden method for the BombPiece class
    {
        destroyed = true;
    }

    public void Explode()
    {
        connectionsToDestroy = 0;
    }
    public virtual bool Explode(Tile tileLocation, ref int p1VPs, ref int p2VPs, ref bool baronDestroyed) //Task 12 -
    overridden new method for the BombPiece class
    {
        return true;
    }

    public virtual bool Primed()
    {
        return true;
    }
    //########################################################End of new Task 12 code
}
```

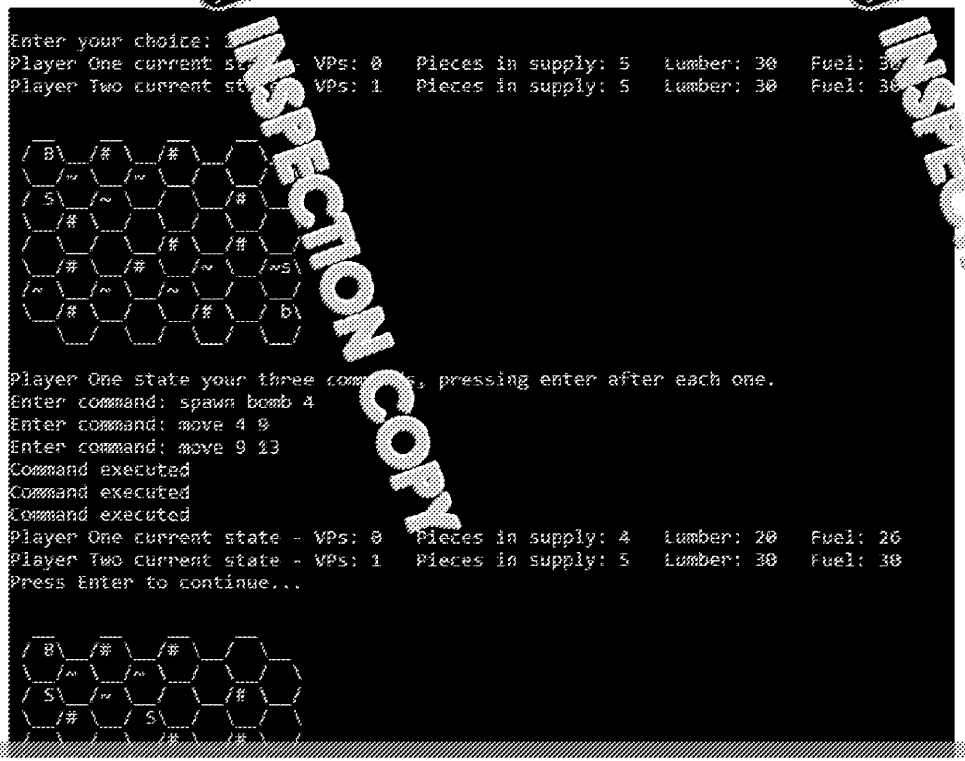
A. Solutions which solve the problem using a different approach and fulfil all of the criteria on the mark scheme, award full marks. When awarding partial marks the criteria can be interpreted slightly loosely if that seems reasonable given the nature of the alternative solution.

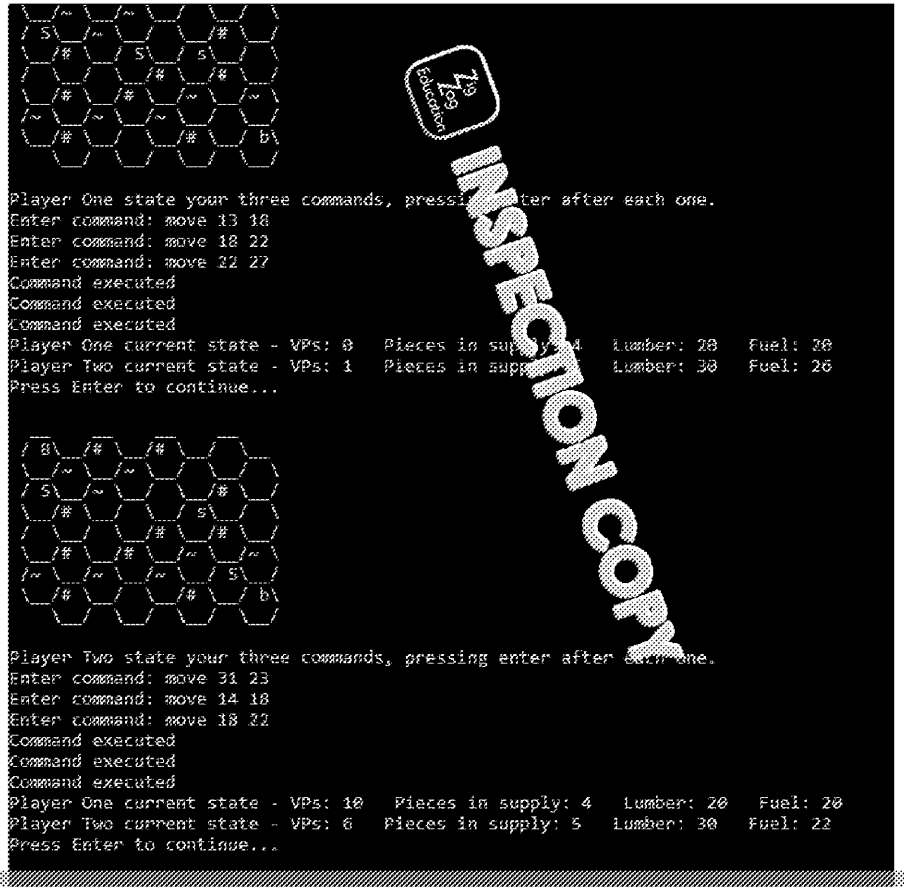
R. Solutions that break OOP by accessing private and protected attributes in ugly or unreasonable ways, or solutions that unnecessarily provide direct access to things when they could be solved in a more suitable way using the OOP hierarchy, encapsulation and polymorphism.

Testing:

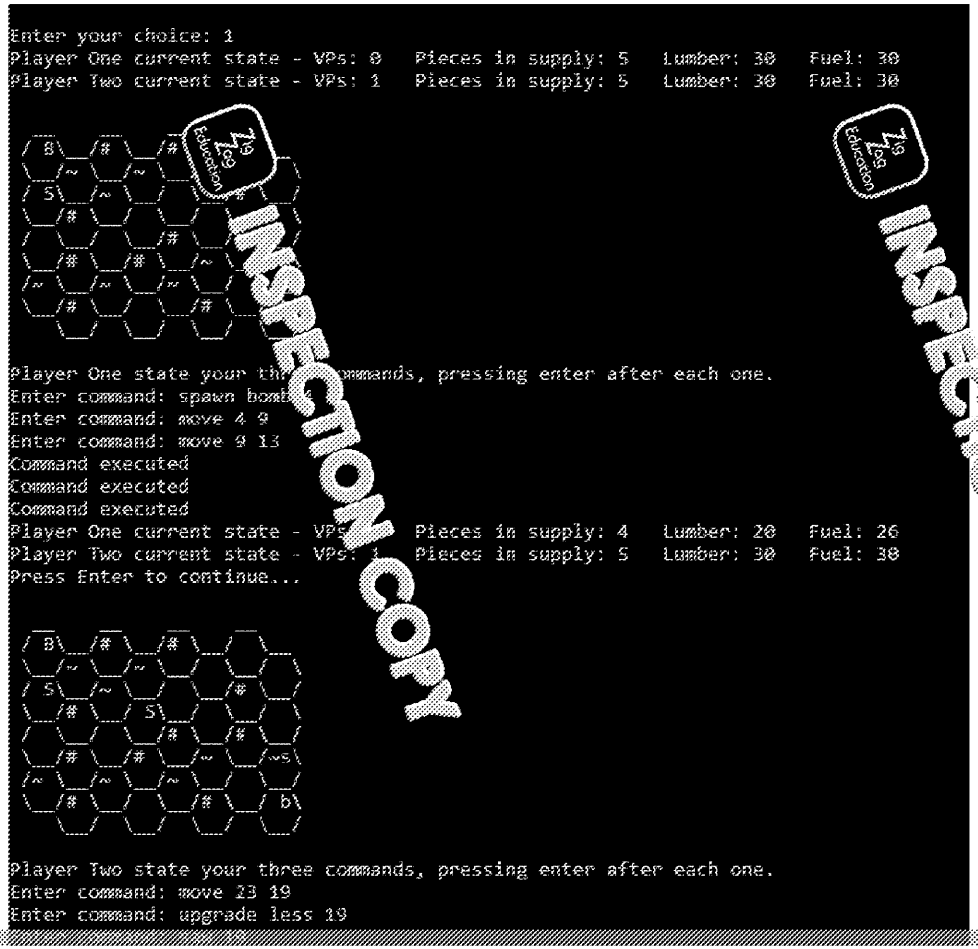
- 1 mark for showing the commands entered correctly and all of the output from the computer (for both games).
- 1 mark for Player One winning the first game with the correct VP totals.
- 1 mark for Player One winning the second game with the correct VP totals.

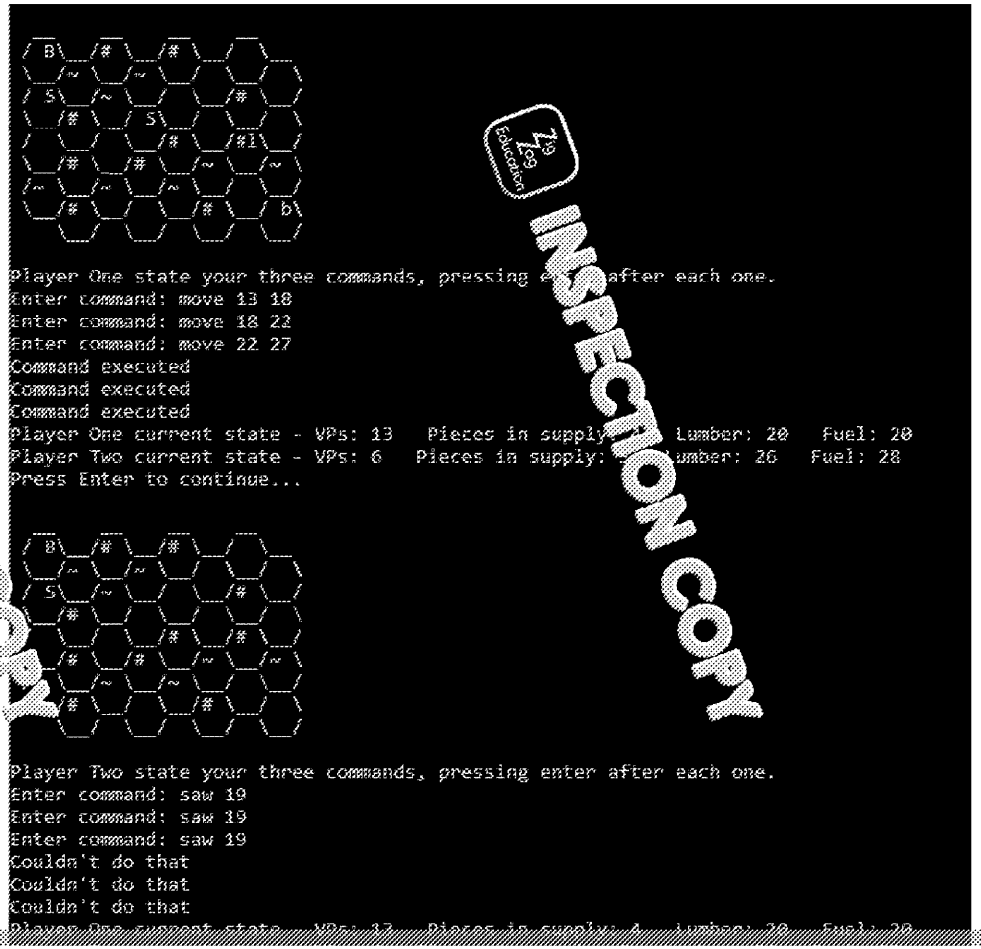
For example – [illegible] One:





For example – Game Two:





Task 13 (10 marks)

Coding:

- 1 mark for creating the WizardPiece class with a VP value of 3.
- 1 mark for overriding the CheckMoveIsValid method to implement the teleport.
- 1 mark for creating a new attribute to track the use of the teleport.
- 1 mark for having a ResetTeleport mechanism that is activated at the end or beginning of each turn.
- 1 mark for changing ExecuteSpawnCommand to enable a wizard piece to be spawned.
- 1 mark for changing CheckUpgradeCommandFormat to enable the new spawn command to be validated.
- 1 mark for allowing a teleport exactly once per turn (only if there is enough fuel).
- 1 mark for getting teleport working for 2 or 3 square distances (even if they can do it multiple times in a turn) and charging 5 fuel but allowing moves of 1 for 1 fuel and blocking moves of 4 or more.

For example:

Code for the new WizardPiece class:

```
class WizardPiece : Piece
    {
        bool telePorted = false;
        public WizardPiece(bool player1)
            : base(player1)
        {
            pieceType = "W";
            VPValue = 3;
            ResetTeleport();
        }
        public override int CheckMoveIsValid(int distanceBetweenTiles, string startTerrain, string endTerrain)
        {
            if (distanceBetweenTiles == 1)
            {
                return fuelCostOfMove;
```

Code for modified CheckUpgradeCommandFormat subroutine:

```
public static bool CheckUpgradeCommandFormat(List<string> items)
{
    int result;
    if (items.Count == 3)
    {
        if (items[1].ToUpper() != "LESS" && items[1].ToUpper() != "PBDS" && items[1].ToUpper() != "WIZ")      //task 13
            return false;

        try
        {
            result = Convert.ToInt32(items[2]);
        }
        catch
        {
            return false;
        }
        return true;
    }
    return false;
}
```

Code for modified ExecuteUpgradeCommand method:

```
if (items[1] == "pbds")
{
    thePiece = new PBDSPiece(player1Turn);
}
//##########################################################start of new task 13 code
else if (items[1] == "less")
{
    thePiece = new LESSPiece(player1Turn);
}
else
{
```

Code for new method ResetWizards in the HexGrid class:

```
public void ResetWizards()
{
    foreach (Tile t in tiles)
    {
        Piece pieceInTile = t.GetPieceInTile();
        if (pieceInTile != null)
        {
            if (pieceInTile.GetPieceType().ToUpper() == "W")
            {
                Type x = pieceInTile.GetType();
                System.Reflection.MethodInfo method = x.GetMethod("ResetTeleport");
                object[] parameters = { };
                method.Invoke(pieceInTile, parameters);
            }
        }
    }
}
```

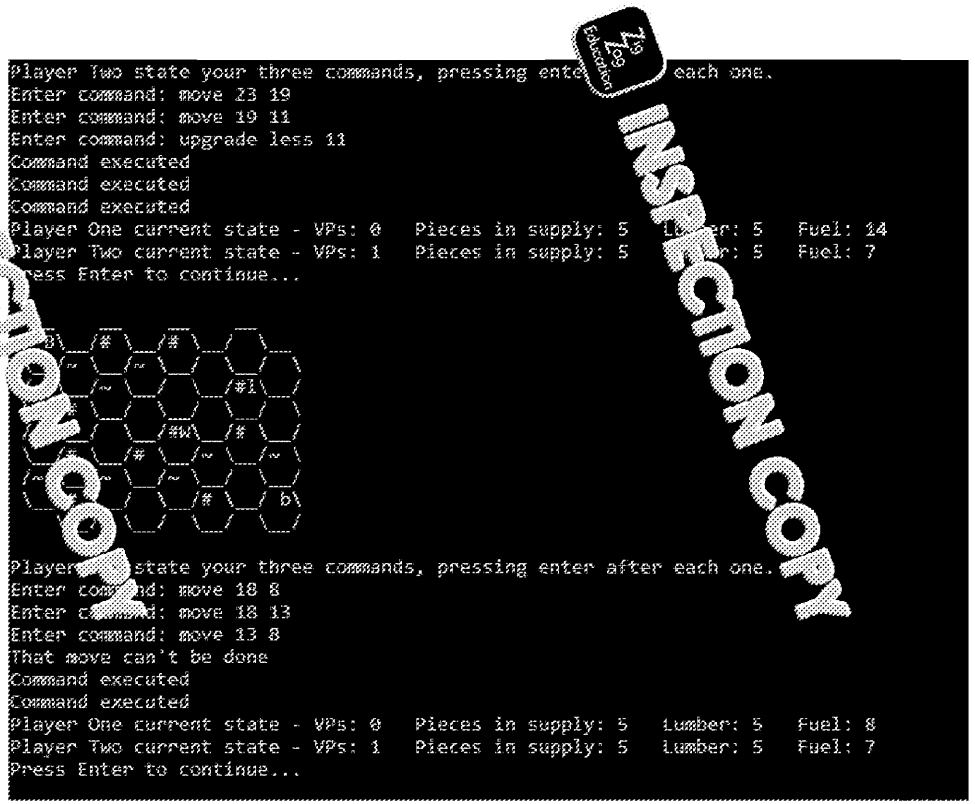
Code for new call to ResetWizards in the PlayGame subroutine:

```
grid.ResetWizards();          //Task 13 - Resetting the wizard teleports back to 3
commands.Clear();
playerlTurn = playerlTurn;
int player1VE = ined = 0;
```

Testing:

- 1 mark for showing the commands entered correctly and the error message for the move 18 8 command and then correctly executing the following two commands.
- 1 mark for correctly teleporting the piece to location 13 (and then onto 18) in the second board printout.

For example:



Task 14 (10 marks)

Coding:

- 1 mark for creating the new SCFWPiece class and setting the VPValue to 3.
- 1 mark for overriding the CheckMoveIsValid method (or any other reasonable way of disabling the move command for this piece).
- 1 mark for implementing the supply command in the SCFWPiece class and checking that there is enough fuel and lumber.
- 1 mark for making it so that the pieces in supply, lumber and fuel are updated correctly when the supply command is run.
- 1 mark for implementing the upgrade scfw command and allowing it to run.
- 1 mark for displaying the new piece as 'F' for Player One and charging the lumber for the upgrade (and making sure it's available).
- 1 mark for checking whether the first command entered is a supply command and terminating the loop early if it is.

For example:

Code for new SCFWPiece class:

```
class SCFWPiece : Piece
    {
        public SCFWPiece(bool player1)
            : base(player1)
        {
            pieceType = "F";
            VPValue = 3;
        }
        public override int CheckMoveIsValid(int distanceBetweenTiles, string startTerrain, string endTerrain)
        {
            return -1;
        }
        public int Supply(int fuelAvailable, int lumberAvailable)
        {
            if (fuelAvailable >= 5 && lumberAvailable >= 5)
            {
                return -5;
```

Code for the modified CheckCommandIsValid subroutine:

```
public static bool CheckCommandIsValid(List<string> items, bool supplyFirst)          //Task 14
{
    if (items.Count > 0)
    {
        switch (items[0])
        {
            case "move":
                {
                    return CheckMoveCommandFormat(items);
                }
            case "dig":
            case "saw":
            case "spawn":
            case "supply":                  //Task 14 - New section added for the supply command
                {
                    bool validCommand = CheckStandardCommandFormat(items);
                    if (items[0] == "supply" && !supplyFirst)
                    {
                        return false;
                    }
                    else
                    {
                        return validCommand;
                    }
                }
            case "upgrade":
                {
                    return CheckUpgradeCommandFormat(items);
                }
        }
    }
```

```
            result = Convert.ToInt32(items[2]);
        }
        catch
        {
            return false;
        }
        return true;
    }
    return false;
}
```

Code for modified method ExecuteUpgradeCommand of the HexGrid class:

```
private int ExecuteUpgradeCommand(List<string> items, int lumberAvailable)
{
    int tileToUse = Convert.ToInt32(items[2]);
    if (!CheckPieceAndTileAreValid(tileToUse) || lumberAvailable < 5 || !(items[1] == "pbds" || items[1] == "less" ||
    items[1] == "ecfw"))  //Task 14 - addition ecfw option added
    {
        return -1;
    }
    else
    {
        Piece thePiece = tiles[tileToUse].GetPieceInTile();
        if (thePiece.GetPieceType().ToUpper() != "S")
        {
            return -1;
        }
        thePiece.DestroyPiece();
        if (items[1] == "pbds")
        {
            thePiece = new PBDSPiece(player1Turn);
        }
        //##########################################################Start of new Task 14 code
        else if (items[1] == "less")
```

Code for modified ExecuteCommand method:

```
        case "dig":
            {
                if (!ExecuteCommandInTile(items, ref fuelChange, ref lumberChange))
                {
                    return "Couldn't do that";
                }
                break;
            }
        //##########################################################Start of new Task 14 code
        case "supply":
            if (!executeSupplyCommand(items, ref fuelChange, ref lumberChange, ref supplyChange, fuelAvailable, lumberAvailable))
            {
                return "Couldn't do that";
            }
            break;
        //##########################################################End of new Task 14 code
```

Code for new ExecuteSupplyCommand method:

```
private bool executeSupplyCommand(List<string> items, ref int fuelChange, ref int lumberChange, ref int supplyChange,
int fuelAvailable, int lumberAvailable)
{
    int tileToUse = Convert.ToInt32(items[1]);
    int fuelAndLumberChange = 0;
    if (!CheckPieceAndTileAreValid(tileToUse))
    {
        fuelChange = 0;
        lumberChange = 0;
        return false;
    }
    Piece thePiece = tiles[tileToUse].GetPieceInTile();
    items[0] = items[0][0].ToString().ToUpper() + items[0].Substring(1);
```

```
            {
                return false;
            }
        }
        return false;
    }
```

Code for modified PlayGame subroutine:

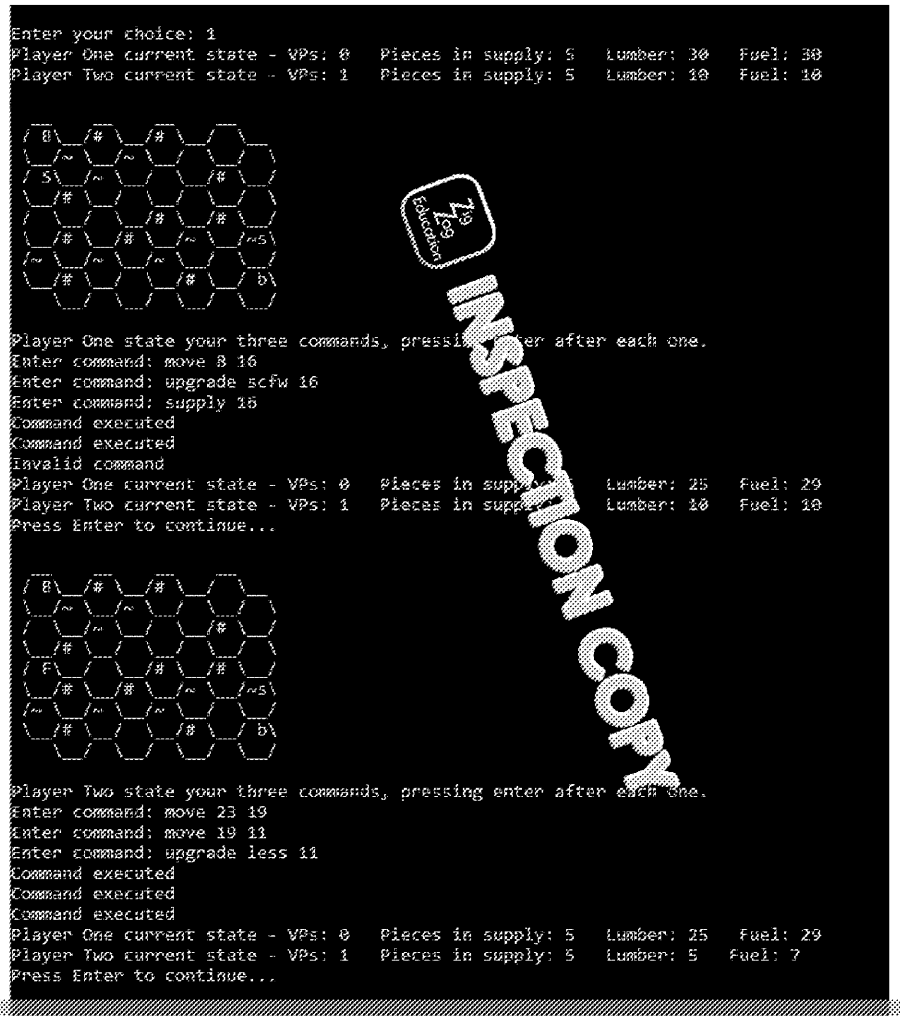
```
public static void PlayGame(Player player1, Player player2, HexGrid grid)
{
    bool gameOver = false;
    bool player1Turn = true;
    bool validCommand;
    List<string> commands = new List<string>();
    bool supplyFirst = false;                    //Task 14 - additional variable added
    Console.WriteLine("Player One current state - " + player1.GetStateString());
    Console.WriteLine("Player Two current state - " + player2.GetStateString());
    do
    {
        Console.WriteLine(grid.GetGridAsString(player1Turn));
        if (player1Turn)
            Console.WriteLine(player1.GetName() + " state your three commands, pressing enter after each one.");
        else
            Console.WriteLine(player2.GetName() + " state your three commands, pressing enter after each one.");
        for (int count = 1; count <= 3; count++)
        {
            Console.Write("Enter command: ");
            //##########################################################Start of new Task 14 code
            string command = Console.ReadLine().ToLower();
            commands.Add(command);
            List<string> items = new List<string>(command.Split(' '));
            if (items[0] == "supply" && count == 1)
            {
                supplyFirst = true;
```

```
int supplyChange = 0;
string summaryOfResult;
if (player1Turn)
{
    summaryOfResult = grid.ExecuteCommand(items, ref fuelChange, ref lumberChange, ref supplyChange,
        player1.GetFuel(), player1.GetLumber(), player1.GetPiecesInSupply());
    player1.UpdateLumber(lumberChange);
    player1.UpdateFuel(fuelChange);
    if (supplyChange == 1)
    {
        player1.RemoveTileFromSupply();
    }
    else if (supplyChange == -1)
    {
        player1.addPieceToSupplyChain();
    }
}
else
{
    summaryOfResult = grid.ExecuteCommand(items, ref fuelChange, ref lumberChange, ref supplyChange,
        player2.GetFuel(), player2.GetLumber(), player2.GetPiecesInSupply());
    player2.UpdateLumber(lumberChange);
    player2.UpdateFuel(fuelChange);
    if (supplyChange == 1)
    {
        player2.RemoveTileFromSupply();
    }
    else if (supplyChange == -1)
    {
        player1.addPieceToSupplyChain();
    }
}
Console.WriteLine(summaryOfResult);
//##########################################################################################
```

Testing:

- 1 mark for showing that the upgrade scfw command worked correctly.
- 1 mark for showing that the first supply 16 command didn't work, with a suitable error message.
- 1 mark for showing that the second supply 16 command worked successfully and that input was terminated when it was entered.

For example:



Task 15 (10 marks)

Coding:

- 1 mark for modifying the main menu to contain a new option 3 for creating a game.
- 1 mark for changing the LoadGame subroutine so that it accepts the FileName as a parameter correctly and doesn't ask for it to be input inside the subroutine, but asks in the subroutine prior to calling LoadGame.
- 1 mark for correctly returning the Success and FileName parameters and using them to ask whether the user would like to start a game and then passing through the appropriate return value as the FileName to LoadGame.
- 1 mark for validating the grid size so that the user can only enter 6, 8 or 10.
- 1 mark for printing out the board correctly, showing all the index numbers for each location.
- 1 mark for accepting the terrain input correctly and using it to generate a string to write to the file (even if not printed out to the screen).
- 1 mark for accepting the starting values for each player (VPs, Fuel, Lumber and Supply Chain) and writing them to the file.
- 1 mark for accepting more than one piece being entered (in addition to the Baron) for each player.

For example:

DisplayMainMenu subroutine:

```
        public static void DisplayMainMenu()
        {
            //##########Start of new Task 15 code
            Console.WriteLine("1. Default game");
            Console.WriteLine("2. Load game");
            Console.WriteLine("3. Create game");
            Console.WriteLine("Q. Quit");
            Console.WriteLine();
            Console.Write("Enter your choice: ");
            //##########End of new Task 15 code
        }
```

Main subroutine:

```
            PlayGame(player1, player2, grid);
        }

        else if (choice == "3")
        {
            string fileName = "";
            bool success = CreateGame(out fileN
            if (success)
            {
                string answer;
                Console.WriteLine("Would you like to play the game that you've just created and saved? ");
                answer = Console.ReadLine().Trim().ToLower();
                if (answer == "y" || answer == "yes
                {
                    fileLoaded = LoadGame(out player1, out player2, out grid, fileName);
                    if (fileLoaded)
                    {
                        PlayGame(player1, player2, grid);
                    }


                }
                else
                {
                    Console.WriteLine("Sorry, but the game could not be created");
                }

            }
            //#############################################################End of new Task 15 code
        }
```

LoadGame subroutine:

```
    public static bool LoadGame(out Player player1, out Player player2, out HexGrid grid, string fileName)
    {
        //declare local items
```

```
            grid = new HexGrid(gridSize);
            List<string> t = new List<string>(myStream.ReadLine().Split(','));
            //##############################################################Start of new Task 15 code
            if (t[t.Count() - 1] == "")
            {
                t[t.Count() - 1] = " ";
            }
            //##############################################################End of new Task 15 code
            grid.SetUpGridTerrain(t);
```

CreateGame subroutine:

```
public static bool CreateGame(out string fileName)
{
    Console.WriteLine("Welcome to the Hex Baron Game Creator");
    Console.WriteLine("-------------------------------------");
    Console.WriteLine();
    Console.WriteLine("Please enter the grid size (6, 8 or 10): ");
    string gridSize = Console.ReadLine();
    while (gridSize != "6" && gridSize != "8" && gridSize != "10")
    {
        Console.WriteLine("Not valid, please enter 6, 8 or 10 for the grid size: ");
        gridSize = Console.ReadLine();
    }
    List<string> terrainList = new List<string>();
    for (int i = 0; i < Math.Floor(Math.Pow(Convert.ToInt32(gridSize), 2)) / 2; i++)
    {
        terrainList.Add(" ");
    }
    HexGrid grid = new HexGrid(Convert.ToInt32(gridSize));
    grid.SetUpGridTerrain(terrainList);
    Console.WriteLine(grid.getGridAsIndicies());
    Console.WriteLine("Please enter all the locations of peat bogs separated by commas");
    string listOfPeatBogsFromPlayer = Console.ReadLine();
```

```
            newTerrain.Add("#");
        }
        else
        {
            newTerrain.Add(" ");
        }
    }
    SetUpGridTerrain(newTerrain);
    p1Baron = 0;
    p2Baron = 0;
    Console.WriteLine("Please enter the location of the Baron for Player 1: ");
    p1Baron = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the location of the Baron for Player 2: ");
    p2Baron = Convert.ToInt32(Console.ReadLine());
    grid.ApplyPiece(true, "Baron", p1Baron);
    grid.ApplyPiece(false, "Baron", p2Baron);
    Console.WriteLine(grid.GetGridAsString(true));
    int p1Fuel, p2Fuel, p1Lumber, p1Supply, p1VP, p2VP;
    Console.WriteLine("Please enter the starting amount of fuel for Player 1: ");
    p1Fuel = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of fuel for Player 2: ");
    p2Fuel = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of lumber for Player 1: ");
    p1Lumber = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of lumber for Player 2: ");
    p2Lumber = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of spares in the supply chain for Player 1: ");
    p1Supply = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of spares in the supply chain for Player 2: ");
    p2Supply = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of VPs for Player 1: ");
    p1VP = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please enter the starting amount of VPs for Player 2: ");
    p2VP = Convert.ToInt32(Console.ReadLine());
    List<string> gridAsStringList = grid.GetGridAsStringList();
```

```
        Console.WriteLine("Enter in the index of the hex where that piece should be placed: ");
        location = Console.ReadLine();
        switch (nextPiece)
        {
            case "S":
                pieceName = "Serf";
                break;
            case "L":
                pieceName = "Lord";
                break;
            case "P":
                pieceName = "Pawn";
                break;
        }
        List<string> newHex = new List<string>() { pieceName, location };
        Pieces.Add(newHex);
        d.AddPiece(true, pieceName, Convert.ToInt32(location));
    }
    nextPiece = "";
    while (nextPiece != "S")
    {
        Console.WriteLine("Enter Piece to add for Player AGS = Serf, L = Lord, P = Pawn) or S for Done");
        nextPiece = Console.ReadLine().ToUpper();
        if (nextPiece != "S")
        {
            string location = "";
            string pieceName = "";
            Console.WriteLine("Enter in the index of the hex where that piece should be placed: ");
            location = Console.Readline();
            switch (nextPiece)
            {
                case "S":
                    pieceName = "Serf";
                    break;
```

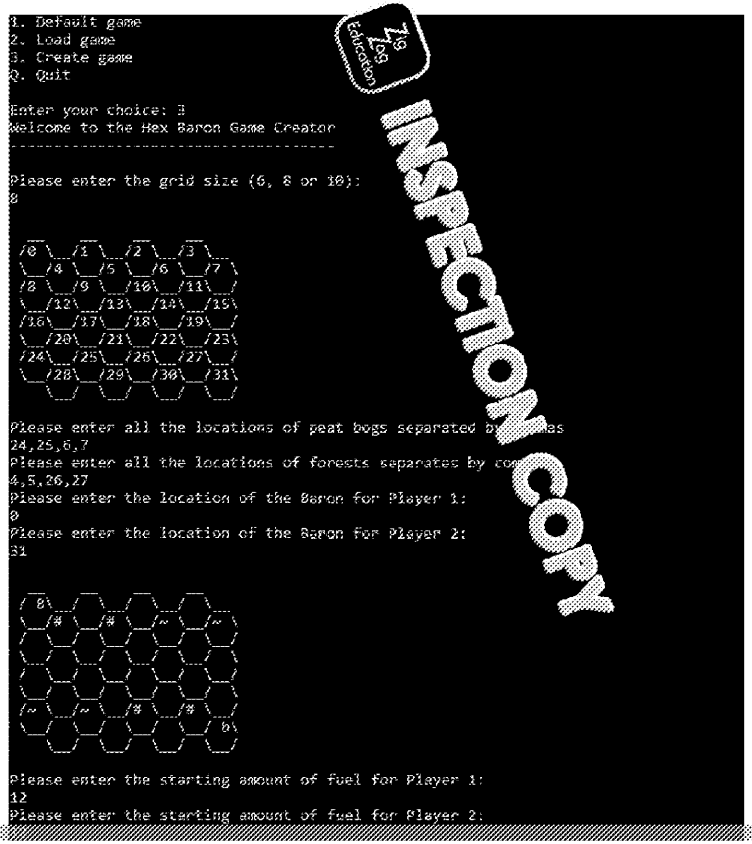
```
        fileName = Console.ReadLine();
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(fileName, true))
        {
            file.WriteLine("Player One," + Convert.ToString(p1VP) + "," + Convert.ToString(p1Fuel) + "," +
            Convert.ToString(p1Lumber) + "," + Convert.ToString(p1Supply));
            file.WriteLine("Player Two," + Convert.ToString(p2VP) + "," + Convert.ToString(p2Fuel) + "," +
            Convert.ToString(p2Lumber) + "," + Convert.ToString(p2Supply));
            file.WriteLine(gridSize);
            file.WriteLine(String.Join(",", newTerr
            foreach (List<string> piece in p1Pieces)
            {
                file.WriteLine("1," + piece[0] + "," +
            }
            foreach (List<string> piece in p2Pieces)
            {
                file.WriteLine("2," + piece[0] + "," + piece[1]);
            }
        }
        return true;
    }
```

A. Solutions which have alternative input mechanisms or file printing mechanisms/structures as long as the code works.

Testing:

- 1 mark for showing the commands entered correctly and appropriate prompts, including the board printout showing the hex index numbers and the game starting correctly at the end.
- 1 mark for the test.csv file appearing exactly as shown.

For example:



Name	

ZigZag Education supporting

A Level AQA Computer Science Pap

Summer 2021

HEXBARON

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

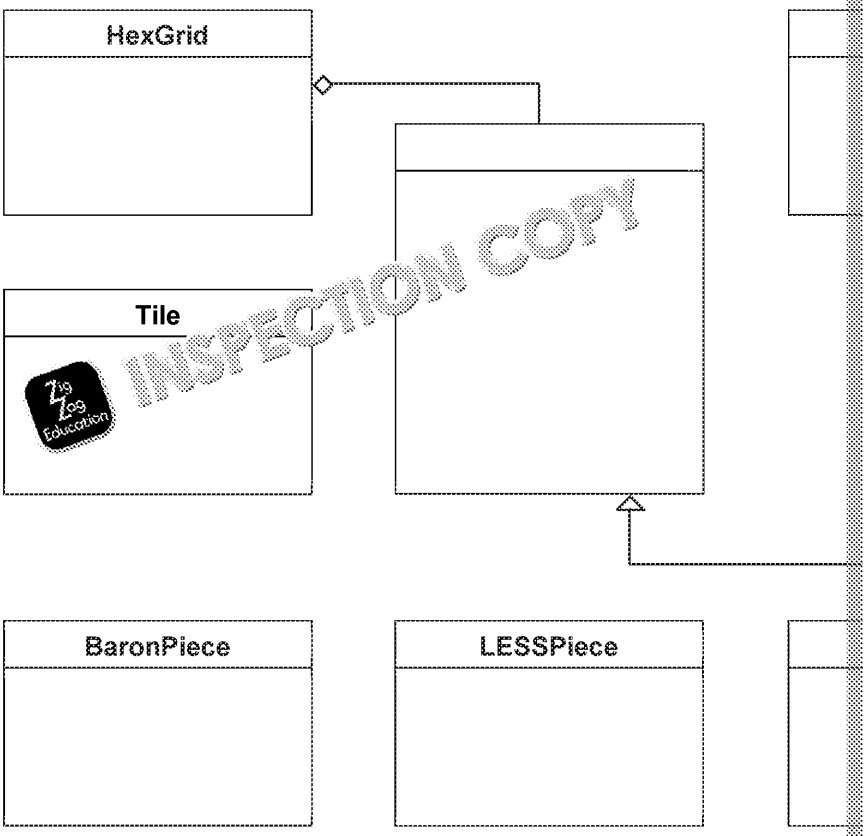
- You will need:
  - access to a computer
  - access to a printer
  - access to appropriate software
  - electronic copies of the required skeleton code
  - EAD (Electronic Answer Document)

Total marks:

Programming Theory Questio

Answer all questions. Remember to save this document

Q		Ans
1	(a)	
	(b)	
2		
3		
4	(a)	
	(b)	
5	(a)	
	(b)	
6	(a)	
	(b)	
7	(a)	
	(b)	
8		

9		
10	(a)	
	(b)	
11		
12	(a)	
	(b)	
13	(a)	
	(b)	
	(c)	
14	(a)	
	(b)	
	(c)	
15		

Programming Tasks

Answer all questions. Remember to save this document

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