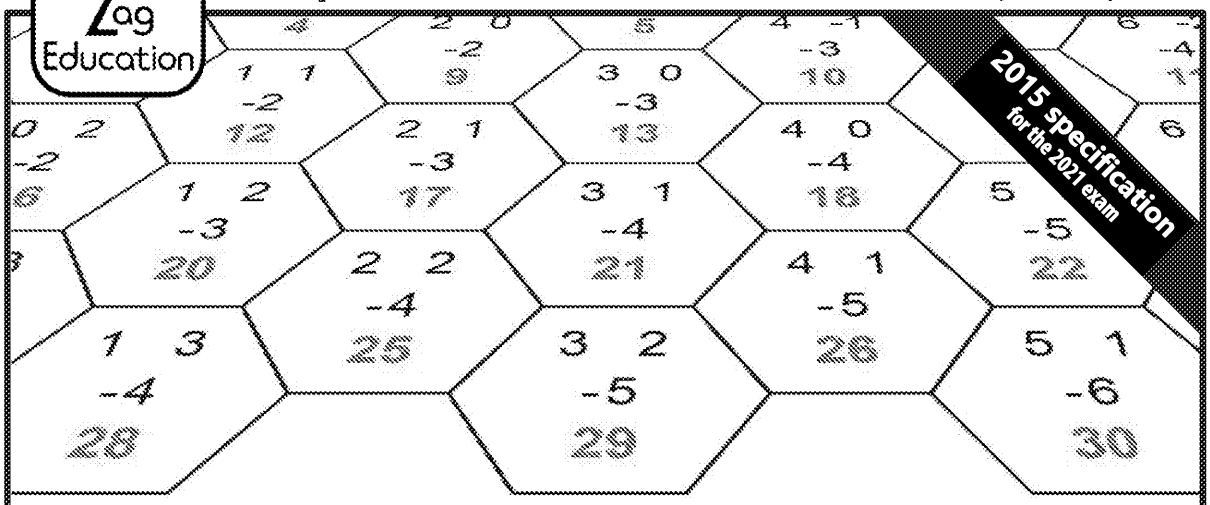
Zig Zag Education

Computer Science

A Level | AQA | 7517

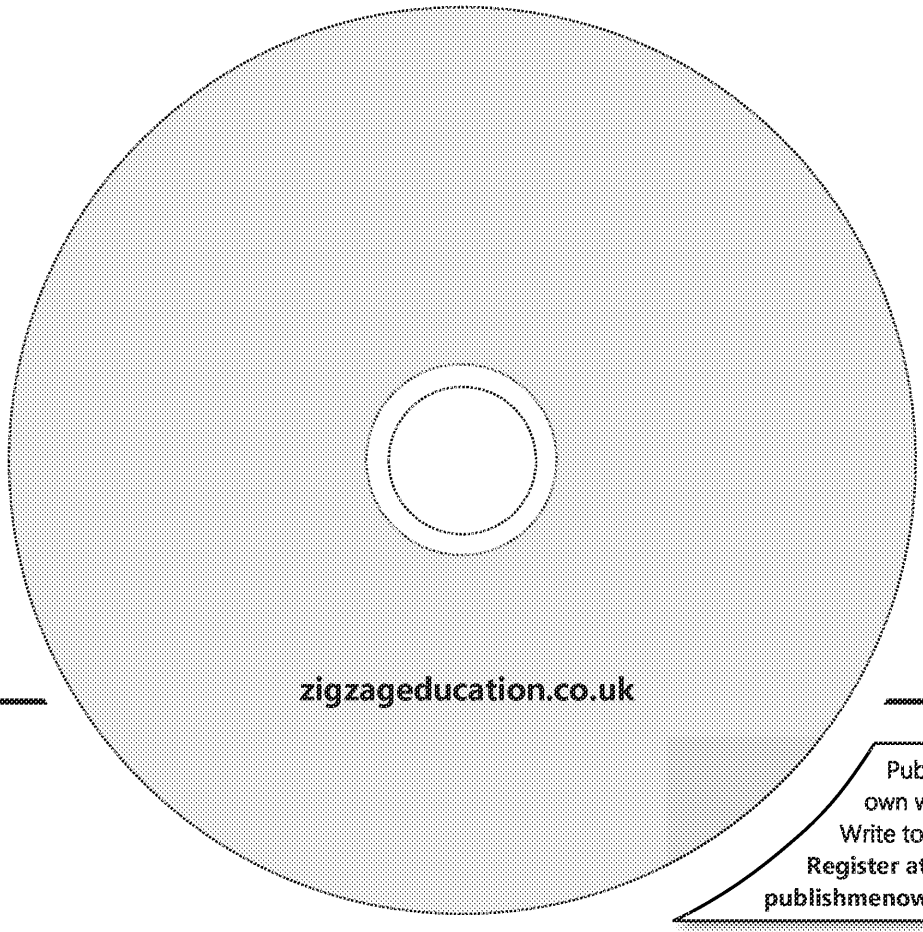


PAPER 1 EXAM RESOURCE PACK 2021

HEX BARON

for A Level AQA Computer Science

VB.NET EDITION



DE10/
10587

POD
10587

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 (VB.NET) – 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:

- `vb06a-ClassDiagramTasksMS.pdf`
- `vb06b-ClassDiagramFull.pdf`
- `vb07-TheoryQuestionsMS.pdf`
- `vb08-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.

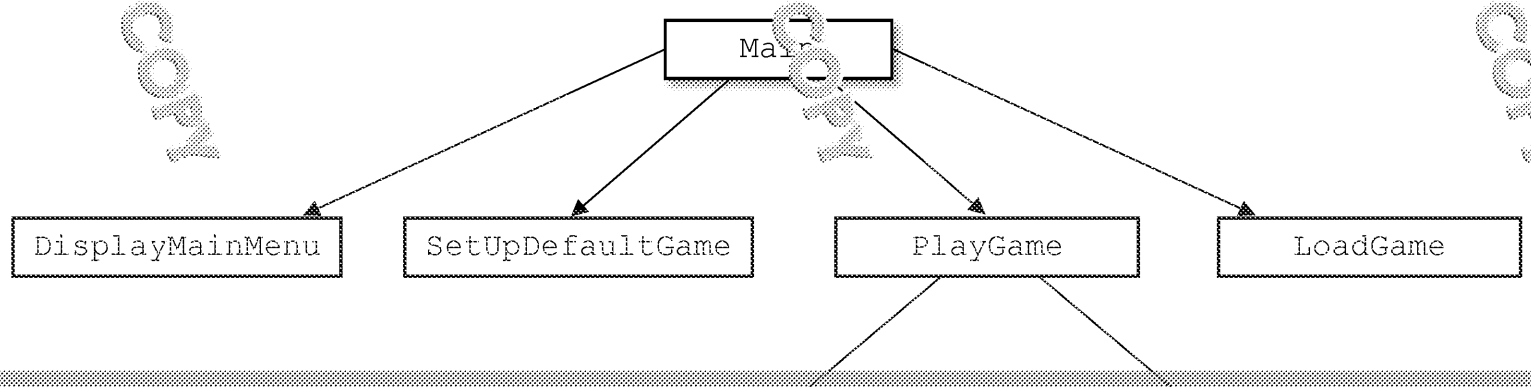
In the game you have to enter three moves before any of them are validated, so if you make a mistake with the first move you could end up losing your entire turn of all three moves as it might affect your second and third move if you thought you'd achieved something that had in fact failed.

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



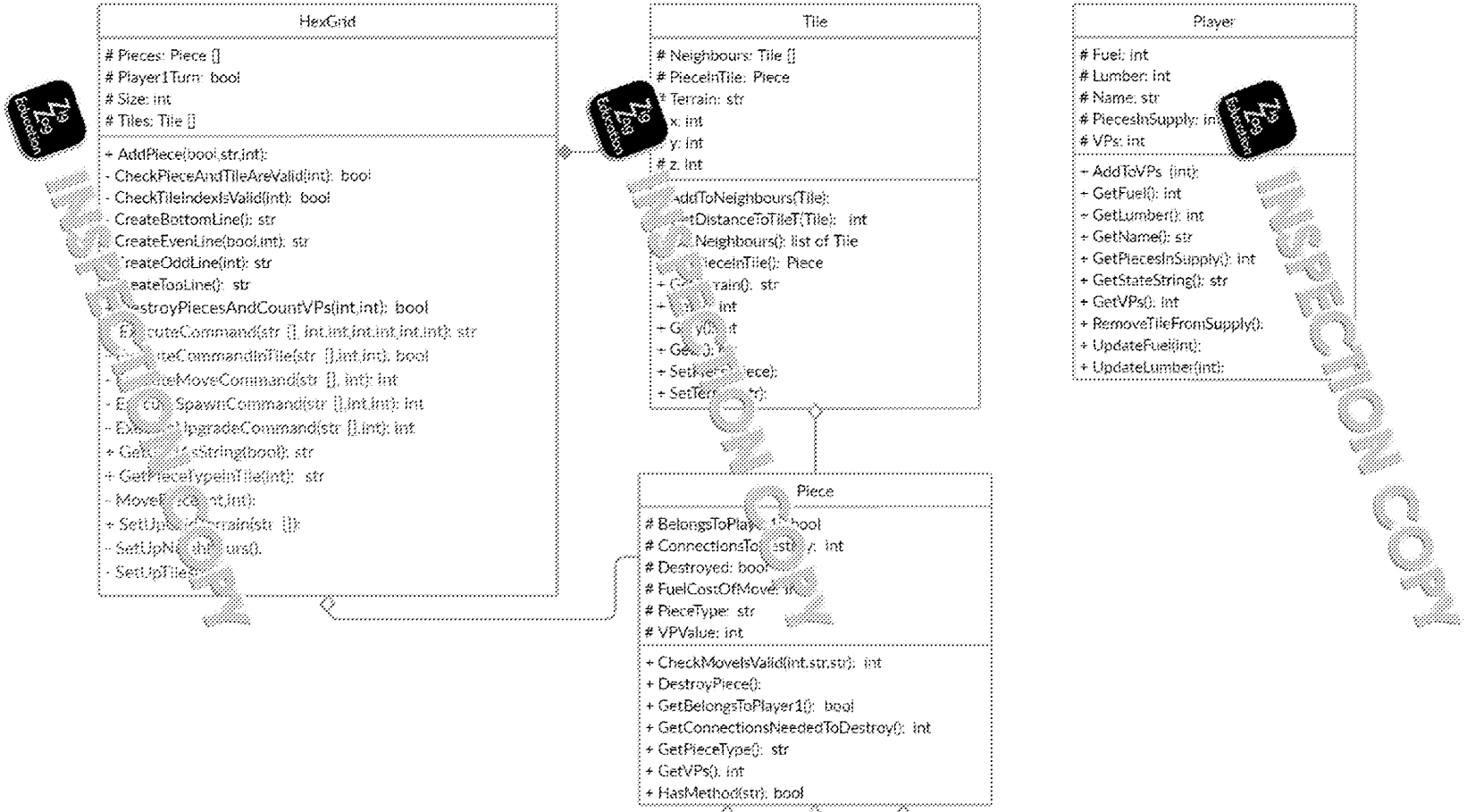
Subroutines (All)

* next to a parameter indicates that it is passed *by reference* (all other parameters are passed *by value)*

Name	Data	Description
`CheckCommandIsValid`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Depending on the first string in the list `Items`, this calls one of the other `Check...CommandFormat` subroutines.
`CheckMoveCommandFormat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are three elements in the `Items` list 2) The second and third items are strings containing integers Otherwise it returns False.
`CheckStandardCommandFormat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are two elements in the `Items` list 2) The second item is a string containing an integer Otherwise it returns False.
`CheckUpgradeCommandFormat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are three elements in the `Items` list 2) The third item is a string containing an 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

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

Classes



`BaronPiece` (inherits from `Piece`)

Name	Data	Description
`New` (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)	***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`

* next to a parameter indicates that it is passed *by reference* (all other parameters are passed *by value)*

Name	Data	Description
`New` (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:*** `FirstEvenLine` (Boolean), `*ListOfPositionOfTile`	Returns a string containing the co t string of an even line of the `HexGrid` (as displayed at the t of each turn when playing the game).
	Return value(s): `Line` (string)	
`CreateOddLine` (pr e)	***Parameters:*** `*ListOfPositionOfTile` (int),	Returns a string containing the correct ng of an odd line of the `HexGrid` (as displayed at the start of n turn when playing the game).
	Return value(s): `Line` (string)	
`CreateTopLine` (private)	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the top line of e `HexGrid` (as displayed at the start of each turn when pla in the game).
`DestroyPiecesAndCount Ps` (public)	***Parameters:*** `*Player1VPs` (int), `*Player2VPs` (int)	Loops through every `Tile` in the `HexGrid` and checks for any `Pieces` that need to be destroyed by counting the number of connections for each `Piece` and comparing it to the number of connections needed to destroy the `Piece` in question.
	Return value(s): `BaronDestroyed` (Boolean),	For each piece that is destroyed, track the VPs for the relevant player and also whether their Baron was destroyed or not.

Name	Data	Description
`ExecuteCommandInTile` (private)	***Parameters:*** `Items` (list of strings), `*Fuel` (int), `*Lumber` (int) ***Return value(s):*** `Stat` boolean),	Checks whether there is a `Piece` belonging to the player in the `Tile` specified as the second string in `Items`, if not then `Fuel` and `Lumber` are set to 0 and False returned. It then uses `HasMethod` to ine whether the `Piece` in the `Tile` has a `saw` or `dig` command as specified by the first string in the `Items` list and calls t at method. If it was a dig and more than 5 fuel was returned the sets the terrain of the `Tile` to a field using the `SetTer n` method. The method then returns True and s `Fuel` or `Lumber` to the amount gained from digging or sawin returns False and sets `Fuel` and `Lumber` to 0 if the comma could not be executed.
`ExecuteMoveComman` (private)	***Parameters:*** `Items` (list of strings), `FuelAvailable` (int) ***Return value(s):*** `FuelCost` (int)	Checks whether there is a `Piece` belo g to the player in the `Tile` specified as the second string in `I ems` and that the `Tile` specified in the third string of `Item` empty, if not then -1 is returned straight away. Otherwise, it then checks that the move is allo ed by calling the `CheckMoveIsValid` method on the `Piec` the first `Tile` and if there is enough `Fuel` available and the move was valid then it calls `MovePiece` to execute the move and returns `FuelCost` (normally 1 or 2), otherwise it returns -1.
`ExecuteSpawnCommand` (private)	***Parameters:*** `Items` (list of strings),	Checks to see whether the player has at least 1 `PieceInSupply` and 3 `Lumber` and that the `Tile` specified

Name	Data	Description
`GetGridAsString` (public)	***Parameters:*** `PlTurn` (Boolean) ***Return value(s):*** `GridAsString` (string)	Uses the private attribute `ListPositionOfTile` 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`
`GetPieceTypeOnTile` (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`.
`MovePiece` (private)	***Parameters:*** `NewIndex` (int), `OldIndex` (int) ***Return value(s):*** -	Sets the `Piece` at `NewIndex` (in the `Tiles` attribute) to the same as the `Piece` (or None) at `OldIndex` by calling the `SetPiece` method on the tile, then sets the `Piece` at the `OldIndex` to None, again by calling the `SetPiece` method on the `Tile`.
`SetUpGridTerrain` (public)	***Parameters:*** `ListOfTerrain` (list of strings) ***Return value(s):*** -	Loops through the `ListOfTerrain` and calls `SetTerrain` for each `Tile` in the protected attribute `Tiles` (which is a list of all the `Tiles` on the `Grid`) in the same order.
`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 the list of neighbours by calling the `AddToNeighbours` method on the `Tile` and passing an argument of the `Tile` that is 1 away.

`LESSPiece` (inherits from `Piece`)

Name	Data	Description
`New` (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)	***Parameters:*** `DistanceBetweenTiles` (int), `StartTerrain` (string), `EndTerrain` (string) ***Return value(s):*** `FuelCostOfMove` (int)	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.
`Saw` (public)	***Parameters:*** `Terrain` (string) ***Return value(s):*** `Lumber` (int)	If `Terrain` is a forest it returns 1, otherwise it returns 0.

`PBDSPiece` (inherits from `Piece`)

Name	Data	Description
`New` (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

Piece

Name	Data	Description
New (constructor)	***Parameters:*** Player1 (Boolean) ***Return value(s):*** -	Initialises the following protected attributes: • BelongsToPlayer1 to Player1 • ConnectionsToDestroy to 2 • Destroyed to False • FuelCostOfMove to 1 • PieceType to "S" • VPValue to 1
CheckMoveIsValid (public)	***Parameters:*** DistanceBetweenTiles (int), StartTerrain (string), EndTerrain (string) ***Return value(s):*** FuelCostOfMove (int)	If the value of the parameter DistanceBetweenTiles is 1 then if the move begins or ends in a peat bog it returns FuelCostOfMove ×2 and if the move doesn't begin or end in a peat bog then it returns FuelCostOfMove otherwise it returns -1.
DestroyPiece (public)	***Parameters:*** - ***Return value(s):*** -	Sets the value of the protected attribute Destroyed to True.
GetBelongsToPlayer1 (public)	***Parameters:*** - ***Return value(s):*** BelongsToPlayer1 (Boolean)	Returns the value of the protected attribute BelongsToPlayer1.
GetConnectionsNeededToDestroy (public)	***Parameters:*** - ***Return value(s):*** ConnectionsToDestroy (int)	Returns the value of the protected attribute ConnectionsToDestroy.
GetPieceType (public)	***Parameters:*** -	If the attribute BelongsToPlayer1 is True then it

`Player`

Name	Data	Description
`New` (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)	***Parameters:*** `n` (int) ***Return value(s):*** -	Increments the attribute `VPs` by `n`.
`GetFuel` (public)	***Parameters:*** - ***Return value(s):*** `Fuel` (int)	Returns the value of the attribute `Fuel`.
`GetLumber` (public)	***Parameters:*** - ***Return value(s):*** `Lumber` (int)	Returns the value of the attribute `Lumber`.
`GetName` (public)	***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)	***Parameters:*** - ***Return value(s):*** `StateString` (string)	Returns a string containing the `VPs`, the `PiecesInSupply`, the `Lumber` and the `Fuel`.
`GetVPs` (public)	***Parameters:*** - ***Return value(s):*** `VPs` (int)	Returns the value of the attribute `VPs`.
`RemoveTileFromSupply`	***Parameters:*** -	Decrements the `PiecesInSupply` attribute by 1

Tile

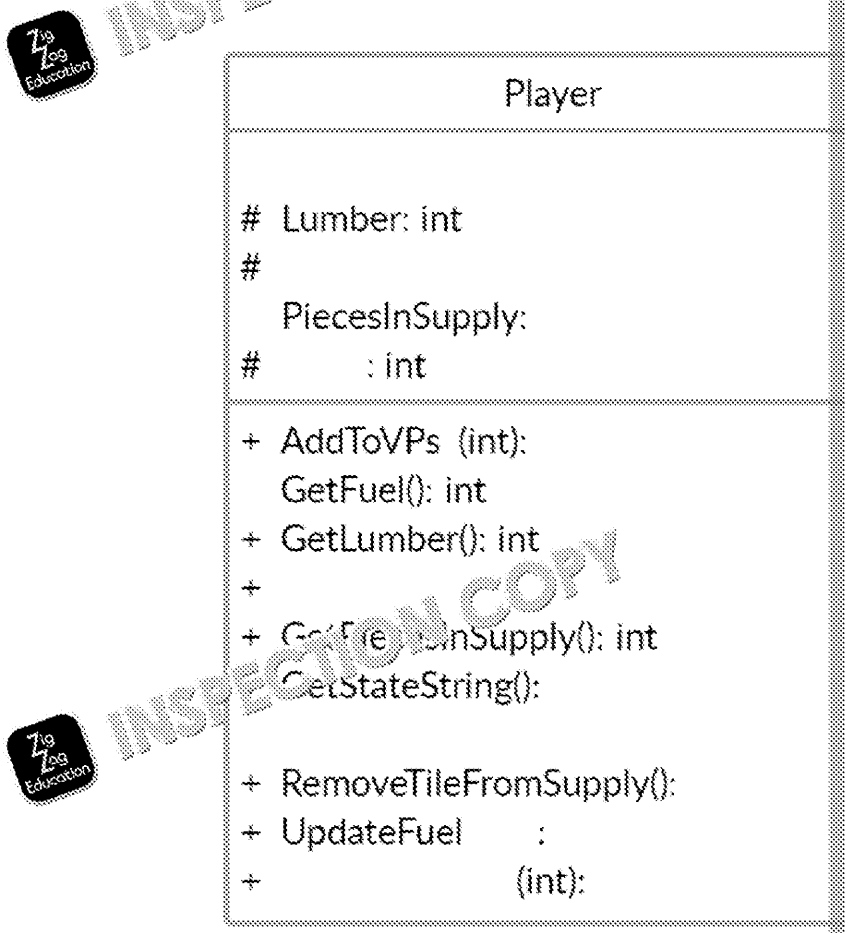
Name	Data	Description
New (constructor)	***Parameters:*** xcoord (int), ycoord (int), zcoord (int) ***Return value(s):*** -	Initialises the following protected attributes: • x from parameter xcoord • y from parameter ycoord • z from parameter zcoord • Terrain to " " (a string containing a single space) • PieceInTile to None
AddToNeighbours (public)	***Parameters:*** N (Tile object) ***Return value(s):*** -	Adds the tile N to the end of the list stored in the protected attribute Neighbours.
GetDistanceToTile (public)	***Parameters:*** T (Tile object) ***Return value(s):*** Distance (int)	Returns the maximum of the absolute difference in x, y and z between the current tile and T.
GetNeighbours (public)	***Parameters:*** - ***Return value(s):*** Neighbours (list of Tiles)	Returns the value of the protected attribute Neighbours.
GetPieceInTile (public)	***Parameters:*** - ***Return value(s):*** PieceInTile (Piece object or None)	Returns the value of the protected attribute PieceInTile.
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.
Gety (public)	***Parameters:*** -	Returns the value of the protected attribute y.

HEX BARON – Class Diagram Tasks

Task 1

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

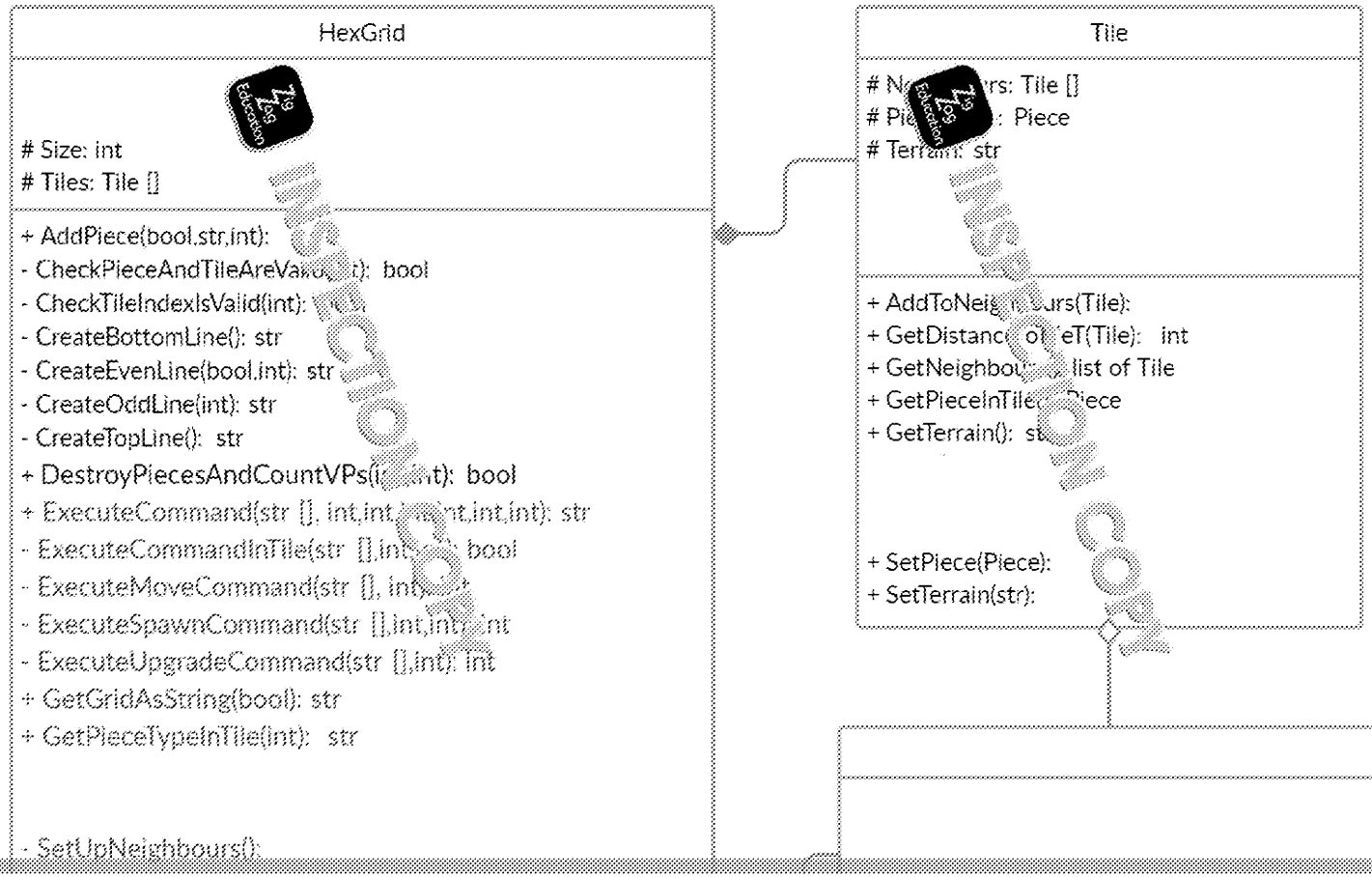
Fill in the missing details.



Task 2 (5 marks)

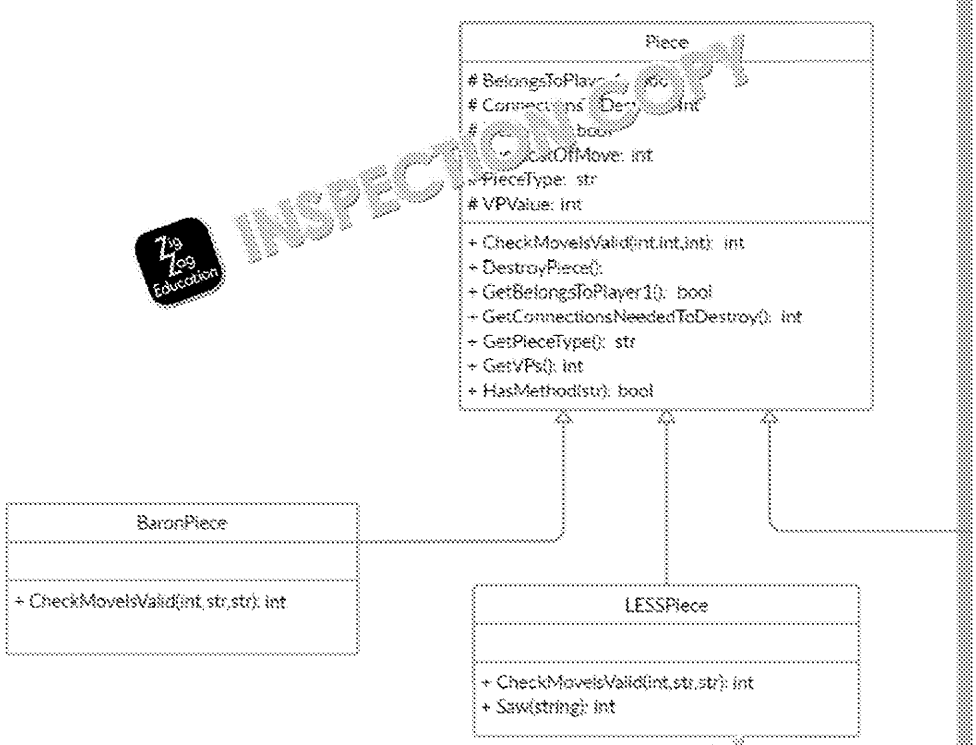
A partially-complete UML class diagram is shown.

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 b



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 l
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 ou[t] [a]ctual player names wh
at the start of each turn; it just pri[nts] 'Player One' and 'Player Two'

a) How could this be [m]odified so that it would print out the names us
meth[od] to [a]ccess the 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[uestion] r[efers] to the `HasMethod` method within the `Piece` cla
`Execu[teC]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 d[i]stance from the c

a) Explain how this calculation is perform[ed]

.....

.....

.....

.....

.....

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[illegible]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 and the Piece class.

The BaronPiece class overrides the public method CheckMoveIs

a) Explain what 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 [illegible]ighbours method in the HexGrid class

.....

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

.....

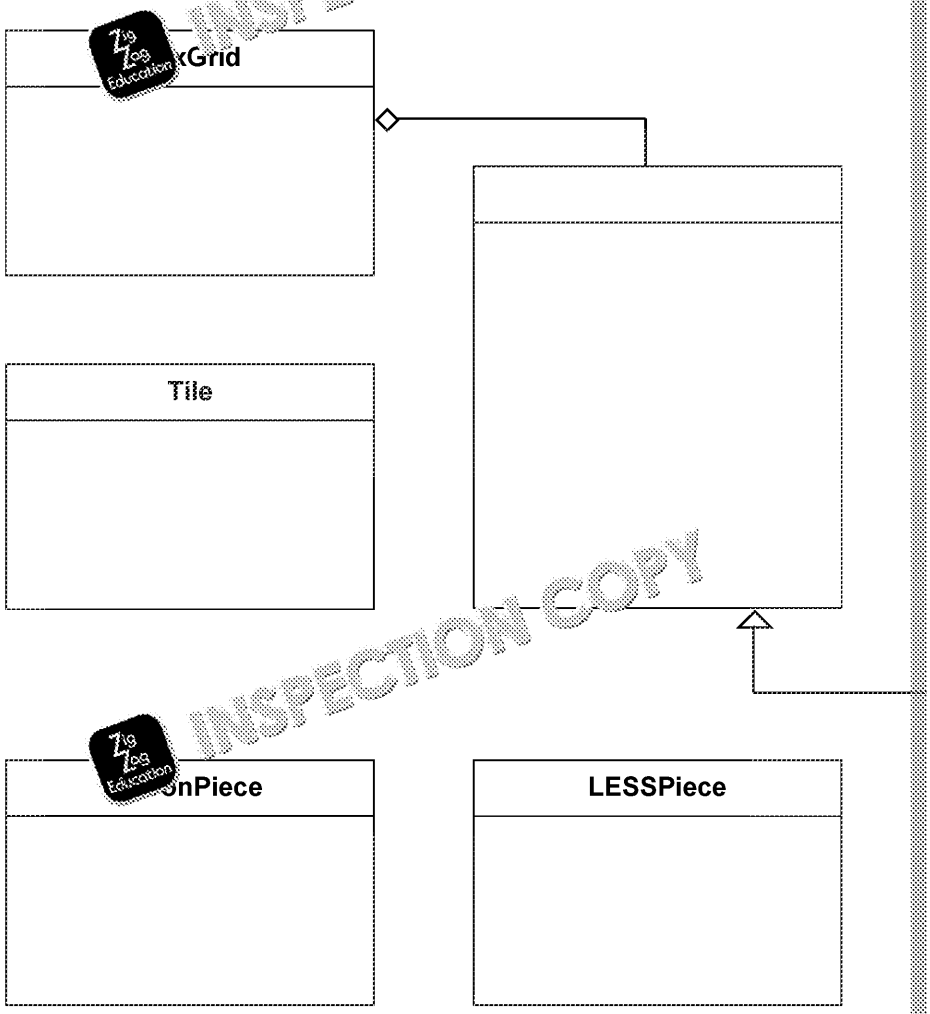
8 The skeleton program uses a Comma Separated Values (CSV) file as
saved game. Explain why a CSV file is suitable for cross-platform appl

.....

.....

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 s 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 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

.....

.....

.....

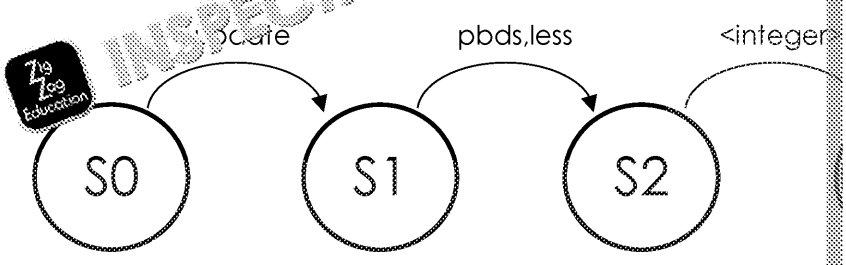
b) Why is polymorphism not possible for private attributes?

.....

.....

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 p

.....

.....

.....

.....

.....

.....

.....

.....

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]ld [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.

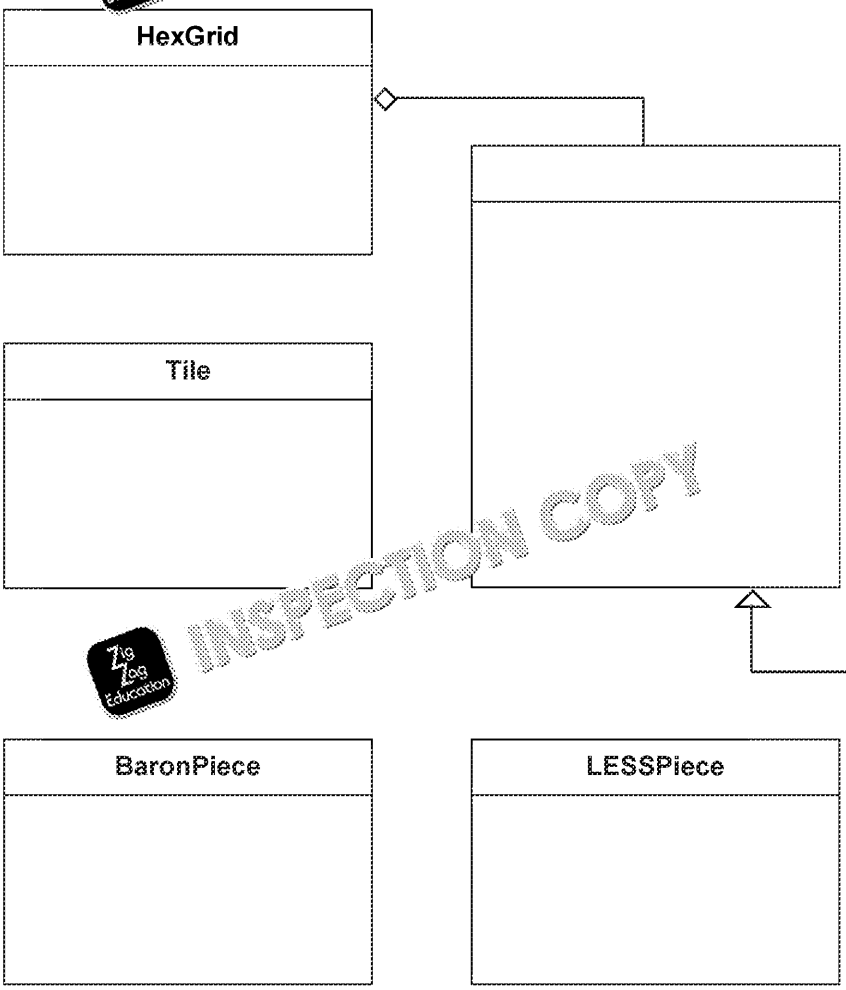
a) Examine the `SetUpNeighbours` method in the `HexGrid` class

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 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 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
`Exec[illegible]rainTile` 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.

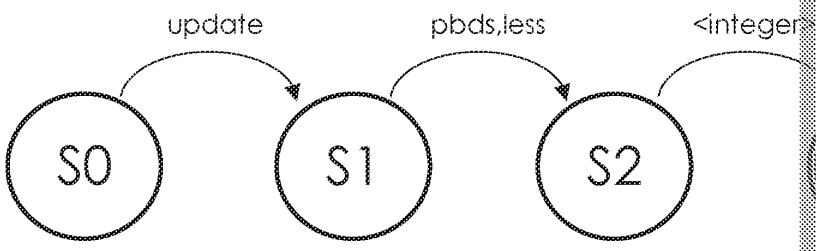
For a piece to be destroyed in the game, it needs to have two connect neighbours (i.e. pieces in two of the tiles that share a side with it).

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).



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 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 referencing 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 t
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 made work:

a) Ch me option 1 to run the default game and then enter "To
and " 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 ly 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 the **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 EN 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 6
- 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 out 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 not count as one of the three
the results should be displayed im diately 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

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 amended subroutine `Pl`
`GetGridAsIndices` method of the `HexGrid` class and any ne
versions of `CreateOddLine` and `CreateEvenLine` added to

b. SCREEN CAPTURE(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 tim
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 B
only cost 2 fuel instead of 3. They sho[illegible] to do the triple move ev
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 = Player("Player One", 0, 2, 10, 5)
```

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 an
`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
field.
- Mo `Dig` method of the `PBDSPiece` to remove the possibil
generating 5 fuel.
- 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 [illegible]
Player Two	• [illegible] • move 27 30 • upgrade less 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`
`Exe ndInTile` and `MakeField` methods of the `He`
mo `Dig` method of the `PBDSPiece` class.

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 comma[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 wor[illegible]

a) Choose menu opti[illegible] to run the default game and then enter the
   - [illegible]ow 24
   - [illegible]grade 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 **`ExecuteDowngradeC`** **`HexGrid`**.

b. SCREEN CAPTURE(S) s[illegible]howing 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 19
Player Two	• move 2 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
Salvaging a piece will add 1 to your supply ch[illegible]g[illegible] you 5 lumber and r
board. The format of the command is [illegible]ge NUM, where NUM is the lo
piece to salvage.

- Mo[illegible]e [illegible]`ckCommandIsValid` 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 th

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

```
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 wor

a) Choose menu option 4 to run the default game and then enter the follo between each

Pla e	• 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 a screen capture showing the commands and all the responses

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended `PlayGame` subrou
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 only be spawned by the Baron but It costs 10 lumber to spawn and 2 lumber for each space that it moves regard

When the bomb explodes it will destroy all pieces in the hexes adjacent to bomb. It will do so if 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 will 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 new correctly.
- Create a new subroutine called **`CheckSpawnCommandFormat`** **`CheckCommandIsValid`** to make sure that the new spawn bom
- Modify the private method **`ExecuteSpawnCommand`** so that it wi the player a number or report an error if they don't have enough lu
- Create a 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`** subroutine so that both players

```
Player1 = Player("Player One", 0, 30, 30, 5)
Player2 = Player("Player Two", 1, 30, 30, 5)
```

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] • [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 [illegible]fied `CheckComman` `CheckSpawnCommandForm`[illegible]ubroutines, the new `BombPie` `Piece` class and the [illegible]ethods `ExecuteSpawnComman` `DestroyPi`[illegible]`AccountVPs`, `GetPieceTypeInTile` an of t[illegible]`x`[illegible]`a` 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 sp
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 T 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 = Player("Play  ", 0, 20, 10, 5)
```

Test that the changes you have made work:

a) Ch me 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 and the responses point at which it prints out the Player One rent state line.

Evidence that you need to provide:

a. You PROGRAM 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**
to the **ExecuteUpgradeCommand**, **ExecuteCommand** and new **ExecuteSupply**
HexGrid class.

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

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

```
Pl[illegible] = Player("Player One", 0, 30, 30, 5)
```

Test that the changes you have made work:

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

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 th[illegible]on [illegible]nds and all the responses

Evidence that you n[illegible]d to [illegible]ovide:

a. You[illegible]GRAM SOURCE CODE for the amended subroutines **Play**[illegible]
and **CheckUpgradeCommandFormat**, the **ExecuteUpgradeCommand**, **Ex**[illegible]
ExecuteSupplyCommand methods of the **HexGrid** class, the new **SCFW**[illegible]
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 single digit, 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 name 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: "

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

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:	grid size
Line 4:	the index 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 `Crea` 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

Test that the changes you have made work:

a) Choose menu option 3 to create a game and then enter the follow 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 `L` `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]ro[illegible]player
3. Mak[illegible]e g[illegible]ore user-friendly so that each time any comman
des[illegible]n is give, e.g. 5 lumber deducted for upgrading PBDS in
con[illegible]ons 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 do
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]three connection
invincible for three turns).
10. Change the rules so that th[illegible]or[illegible]ection 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 s
he[illegible]o[illegible]ing 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

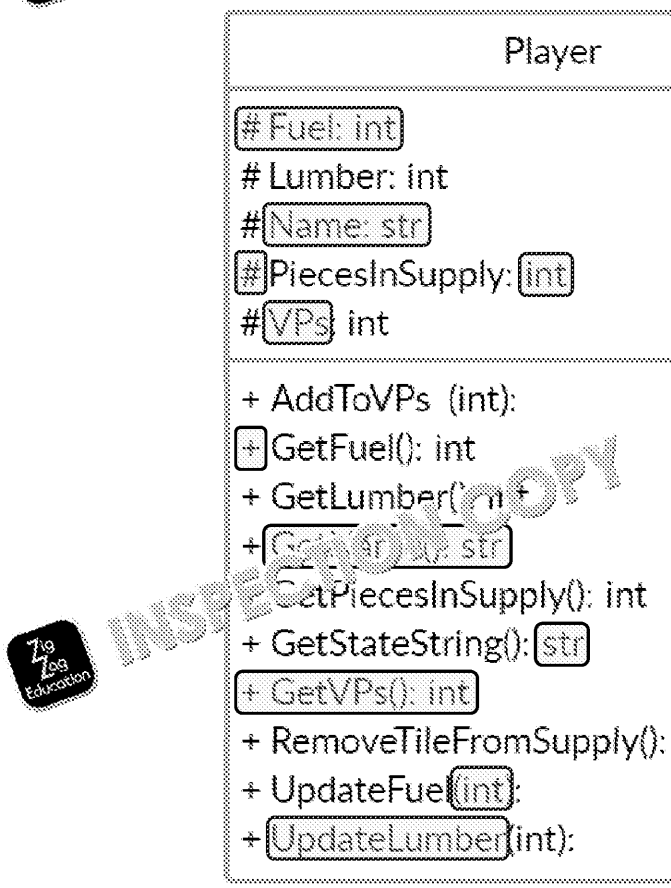
HEX BARON – Class Diagram Tasks (MS)

Task 1

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

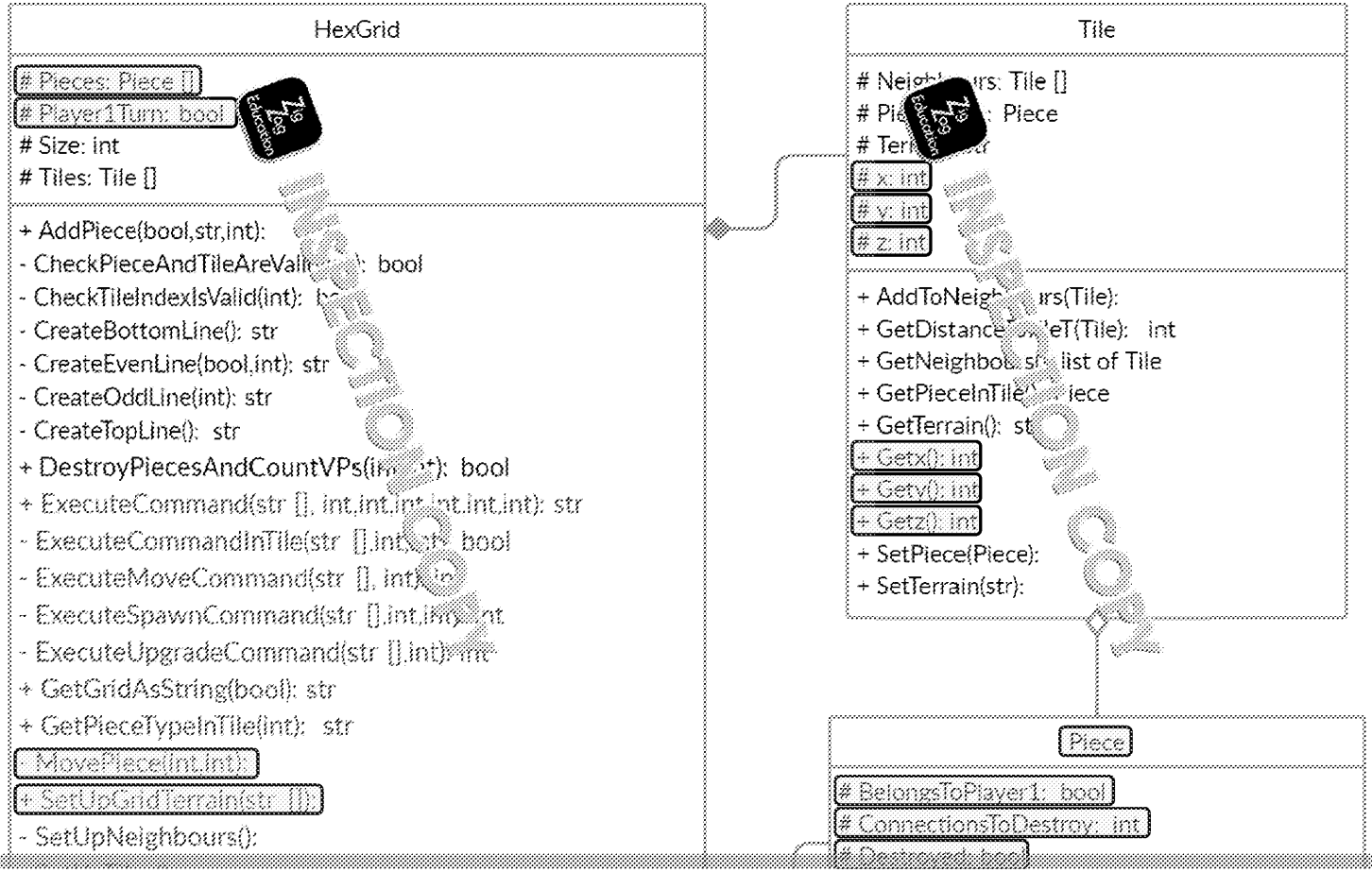
A. Use of void where there is no return value or there are no parameters

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).

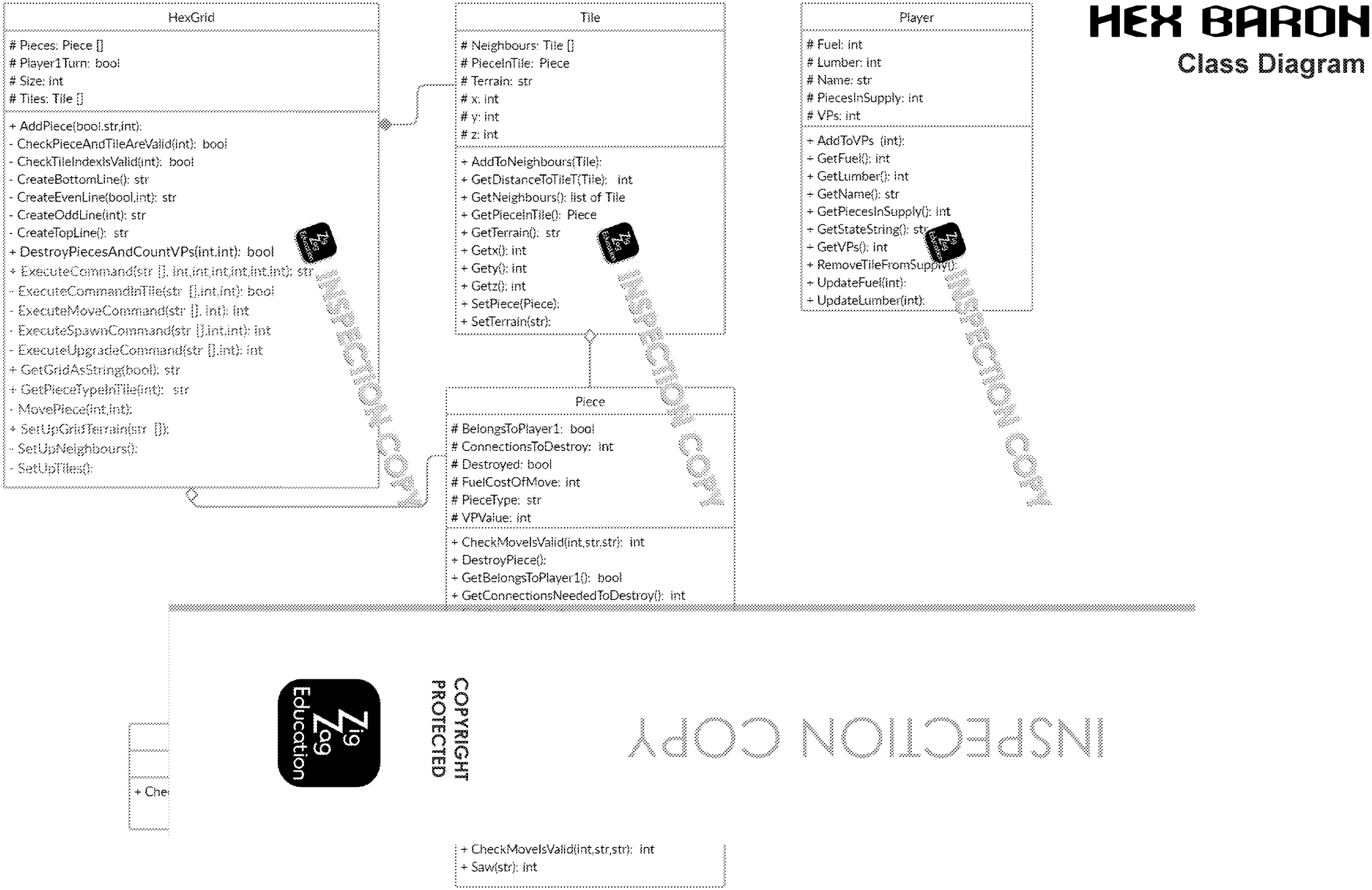
  1 mark for naming constructors and 1 mark for the explanation

- 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

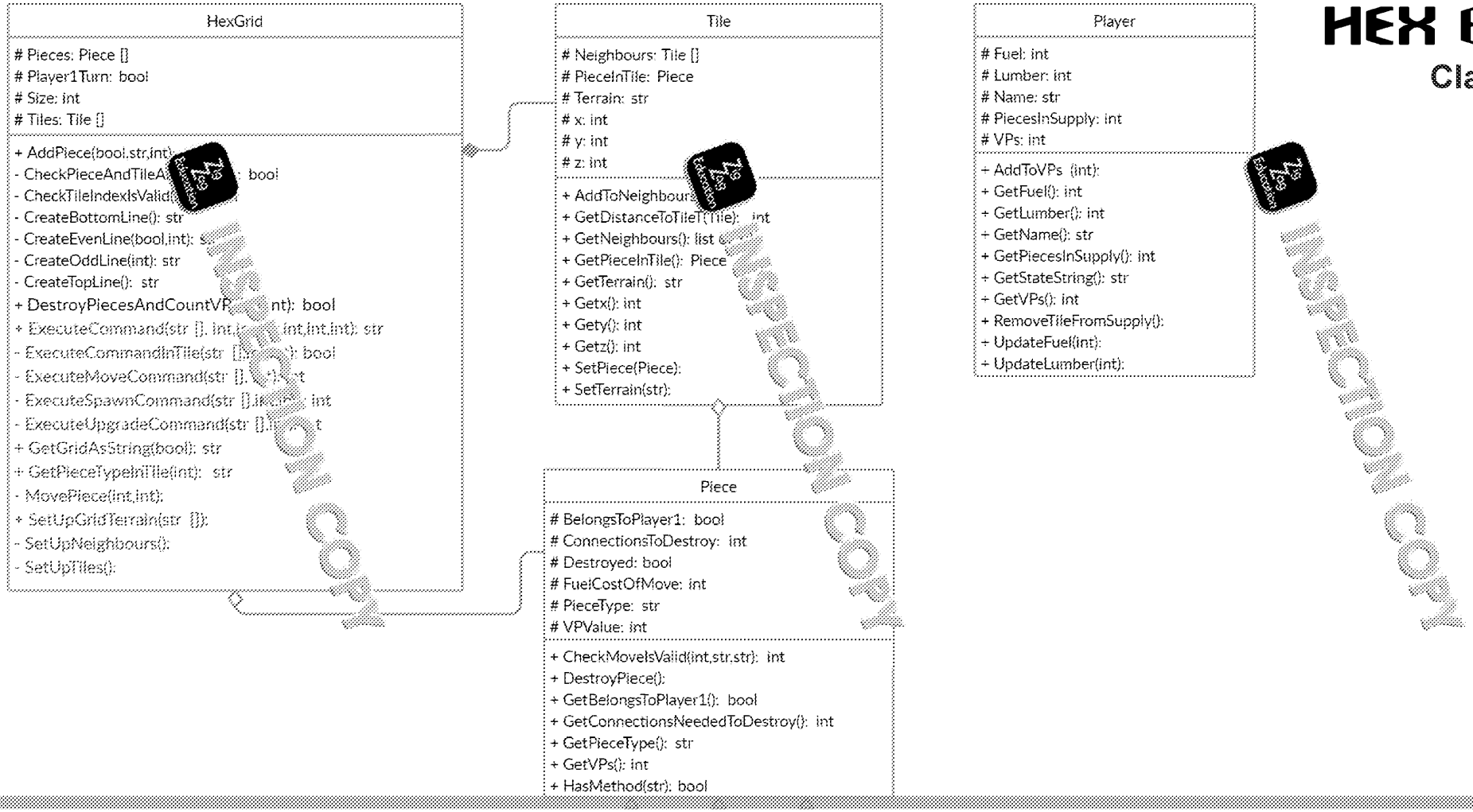
HEX BARON

Class Diagram



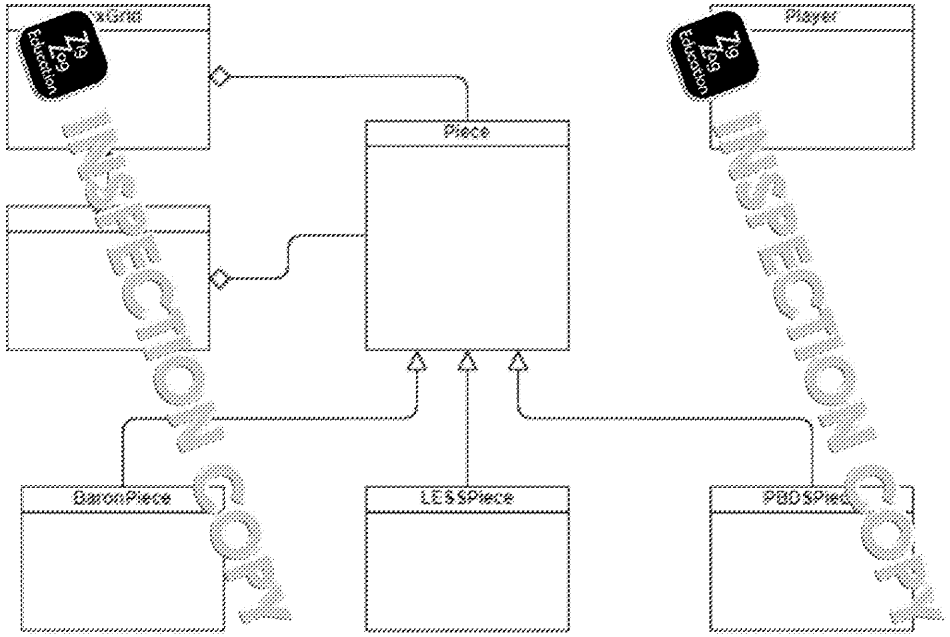
HEX BARON

Class Diagram



HEX BARON – Theory Questions (MS) (Total Marks: 50)

Question		Suggested Solution	Marks	Guidance
1	a	Bo[illegible]g `Player1.GetName()` and `Player2.[illegible]ame()`	1 mark	• A. any sens[illegible]ention of `GetName()`
	b	To [illegible]ovide an interface to private or protected attribu[illegible]	1 mark	• A. either priv[illegible] or protected • A. access or an[illegible]ther similar word for interface
2		The va[illegible]ble `Player1Turn` is toggled using `Not Pla[illegible]1Turn;` Player [illegible] always gets their turn if Player One ends the g[illegible]e as the game can only e[illegible]d when `Player1Turn` is True	2 marks	• 1 mark for each p[illegible]t
3		`HasMeth[illegible]` 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 me[illegible]hod 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 tile `T` and itself; The absolute difference between the y coordinate of tile `T` and itself; The absolute difference between the z coordinate of tile `T` and itself	3 marks	• 1 mark for the first poi[illegible] • 1 mark for any of the oth[illegible] 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(-4),abs(-3-(-4)))` which is `max(2,1)` = 2 then `max(2,abs(-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 `PBDSPiece`) • 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 for only strictly positive integers inc 0
	b	Upg (pbds|less) (0|[1-9][0-9]?)	1 mark	• 1 mark for any ression that only allows integers from 0– • DPT problem wit egers starting with 0, e.g. 01 from part a
	c	One that can be represented by a FSM (with no outputs)	1 mark	• A. one that can be p esented by a regular expression

14	a	`GridAsSt ng`
		`ListPosi nOfTile`
		`Count`
			1 mark	• MAX 1 of the three ch ces

	b	Their scope is m ed to the current subroutine; So that means th you can test the subroutine in isolation; And, therefore, a people to use the subroutine through its interfa only so they do not need o concern themselves with the implementation	3 marks	• A. any valid point about s o e being the subroutine or a related adv ge (e.g. no need to worry that there might be anges elsewhere) • A. equivalent ideas • A. method or function for subroutine
	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	6 marks	MAX 3 from Group 1 Marks

HEX BARON – Programming Tasks (MS)

Task 1 (2 marks)

Coding:

- 1 mark for stripping the leading and trailing spaces from all three input commands.

For example:

```
Commands.Add(Trim(Console.ReadLine().ToLower()))
```

A. Solutions which manually strip off as many spaces as there are, but not solutions that only check for a single space and then remove it.

Testing: 1 mark for showing that the move command is now valid the first time and that the upgrade can happen successfully, despite the leading space.

For example:

```
Player One state your three commands, pressing enter after each one.
Enter command: move 8 16
Enter command: move 8 16
Enter command:  upgrade pbds 16
Command executed
That move can't be done
Command executed
```

Task 2 (3 marks)

Coding:

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

For example:

```
        Console.Write("Player One, please enter your name: ");
        Player1 = new Player(Console.ReadLine(), 0, 10, 10, 5);
        Console.Write("Player Two, please enter your name: ");
        Player2 = new Player(Console.ReadLine(), 1, 10, 10, 5);
```

A. Solutions which store the inputs in two variables and then pass those through to the Player when constructed.

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:

```
Enter your choice: 1
Player One, please enter your name: Tom
Player Two, please enter your name: Vicky
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


 ___  ___  ___  ___
/ B\__/# \__/# \__/  \__
\__/~ \__/~ \__/  \__/  \
/ S\__/~ \__/  \__/# \__/
\__/# \__/  \__/  \__/  \
```

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 updating 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 Overrides Sub AddPiecesInSupply(ByVal n As Integer)
    PiecesInSupply = n
End Sub
```

Both calls to DestroyPiecesAndCountVPs:

```
Grid.DestroyPiecesAndCountVPs(Player1VPsGained, Player2VPsGained, Player1, Player2)
```

Adding parameters to the DestroyPiecesAndCountVPs method:

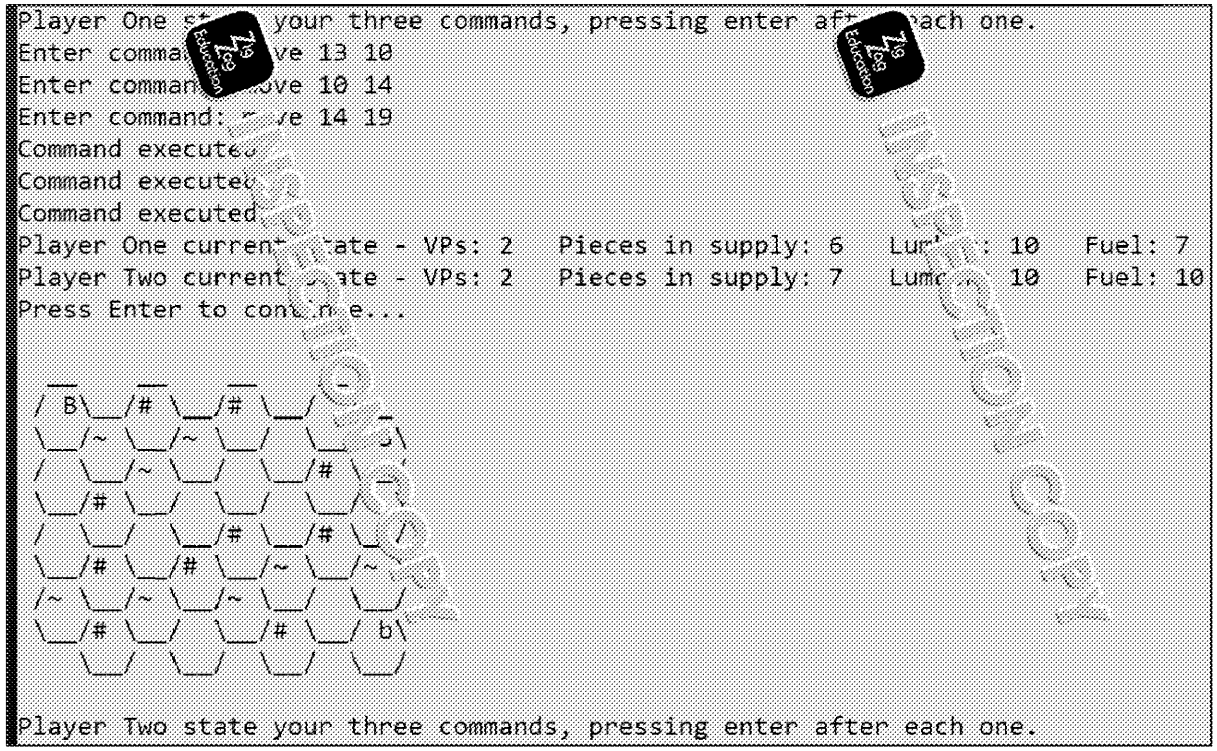
```
Public Function DestroyPiecesAndCountVPs(ByRef Player1VPs As Integer, ByRef
Player2VPs As Integer, ByVal Player1 As Player, ByVal Player2 As Player) As Boolean
```

Modifications to the DestroyPiecesAndCountVPs method:

```
If ThePiece.GetBelongsToPlayer1() Then
    Player1.AddPiecesInSupply(1)
    Player2VPs += ThePiece.GetVPs()
Else
```

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 outputting 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 `Dim NewPiece As New Piece(Player1Turn)` line of code (or anywhere before it in the method).

For example:

```
        Dim PlayerPieces As Integer = 0
        For Each PlayerPiece In Pieces
            If PlayerPiece.GetBelongsToPlayer1() = Player1Turn Then
                PlayerPieces += 1
            End If
        Next
        If PlayerPieces >= 6 Then
            Console.WriteLine("Spawn attempted to exceed max pieces.")
            Return -1
        End If
```

A. Solutions which calculate the number of PlayerPieces differently (but validly).

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:

```
Player One state your three commands, pressing enter after each one.
Enter command: spawn 4
Enter command: move 4 9
```

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 after the 'hexes' command did not count as one of the three commands (no matter how many times the ... it).
- 1 mark for having 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 (even 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:

```
    For Count = 1 To 2
        Dim Command As String
        Console.Write("Enter command: ")
        Command = Console.ReadLine().ToLower()
        While Command = "hexes"
            Console.WriteLine(Grid.GetGridAsIndices(Player1Turn))
            Console.Write("Enter command: ")
            Command = Console.ReadLine().ToLower()
        End While
        Commands.Add(Command)
    Next
```

Code for new GetGridAsIndices method:

```
    Public Function GetGridAsIndices(ByVal P1Turn As Boolean) As String
        Dim ListPositionOfTile As Integer = 0
```

Code for the new CreatOddLineIndices method:

```
Private Function CreateOddLineIndices(ByRef ListPositionOfTile As Integer, ByRef Index As Integer) As String
    Dim Line As String = ""
    For count = 1 To Size / 2
        If count > 1 And count < Size / 2 Then
            Line &= "\__/"
            ListPositionOfTile += 1
            Line &= CStr(Index).PadLeft(2, " ")
            Index += 1
        ElseIf count = 1 Then
            Line &= " \__/" & CStr(Index).PadLeft(2, " ")
            Index += 1
        End If
    Next
    Line &= "\"
    ListPositionOfTile += 1
    If ListPositionOfTile < Tiles.Count() Then
        Line &= CStr(Index).PadLeft(2, " ") & "\" & Environment.NewLine
        Index += 1
    Else
        Line &= "\" & Environment.NewLine
    End If
    Return Line
End Function
```

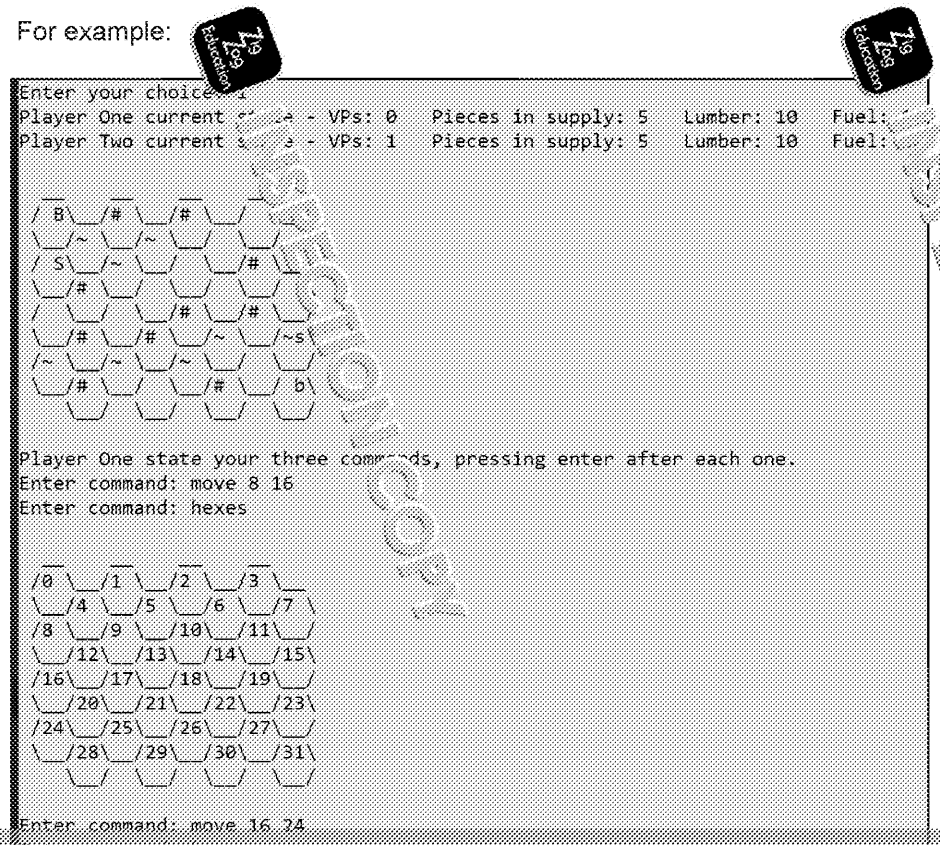
Code for the new CreateEvenLineIndices method:

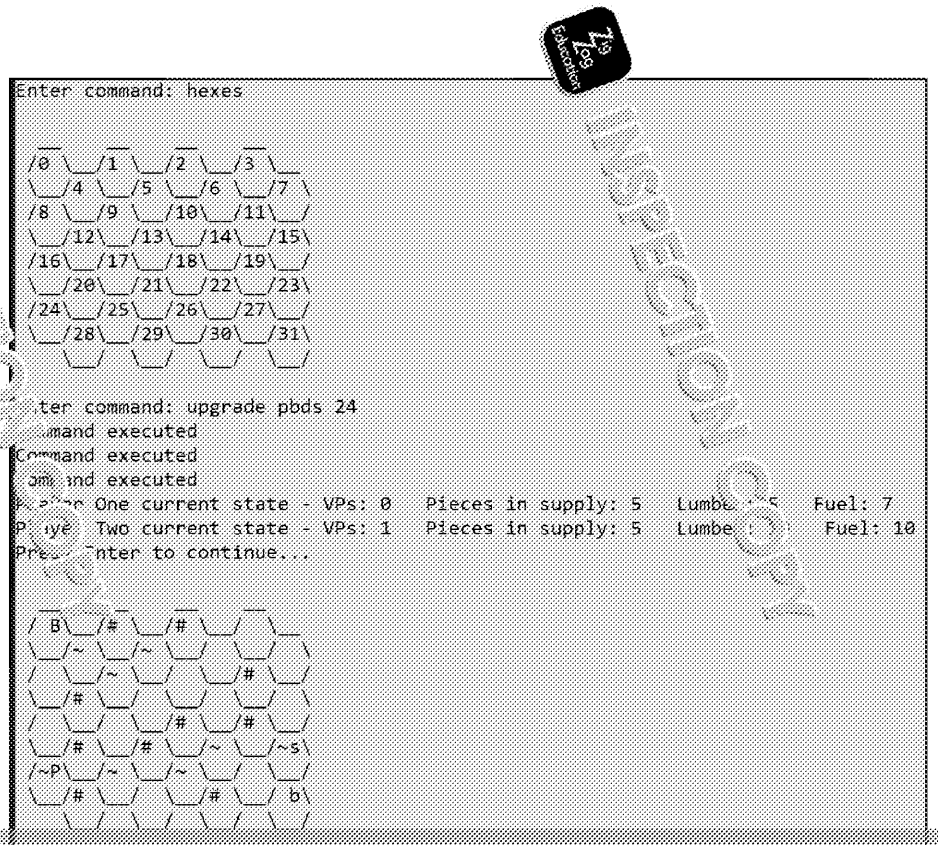
```
Private Function CreateEvenLineIndices(ByVal FirstEvenLine As Boolean, ByRef ListPositionOfTile As Integer,
ByRef Index As Integer) As String
    Dim Line As String = " /" & CStr(Index).PadLeft(2, " ")
    Index += 1
    For count = 1 To Size / 2 - 1
        ListPositionOfTile += 1
```

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 Each 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:

```
Do
    Dim ValidMoveCommands As Integer = 0
    Console.WriteLine(Grid.GetGridAsString(Player1Turn))
    If Player1Turn Then
        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.")
    End If
    For Count = 1 To 3
        Console.Write("Enter command: ")
        Commands.Add(Console.ReadLine().ToLower())
    Next
    For Each C In Commands
        Dim Items As List(Of String) = New List(Of String)(C.Split(" "))
        ValidCommand = CheckCommandIsValid(Items)
        If Not ValidCommand Then
            Console.WriteLine("Invalid command")
        Else
            Dim FuelChange As Integer = 0
            Dim LumberChange As Integer = 0
            Dim SupplyChange As Integer = 0
            Dim SummaryOfResult As String
```

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:

```
Player One state your three commands, pressing enter after each one.
Enter command: move 0 4
Enter command: move 4 9
Enter command: move 9 17
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 10   Fuel: 0
```

Task 7 (7 marks)

Coding:

- 1 mark for correctly removing the code from ExecuteCommandInTile and updating the Dig method to return 1.
- 1 mark for initialising variables to track the valid dig and saw commands.
- 1 mark for ensuring 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 MyItems(0) = "Dig" Then
            Func = Method.Invoke(ThePiece, Parameters)
        End If
```

Modifications to Dig:

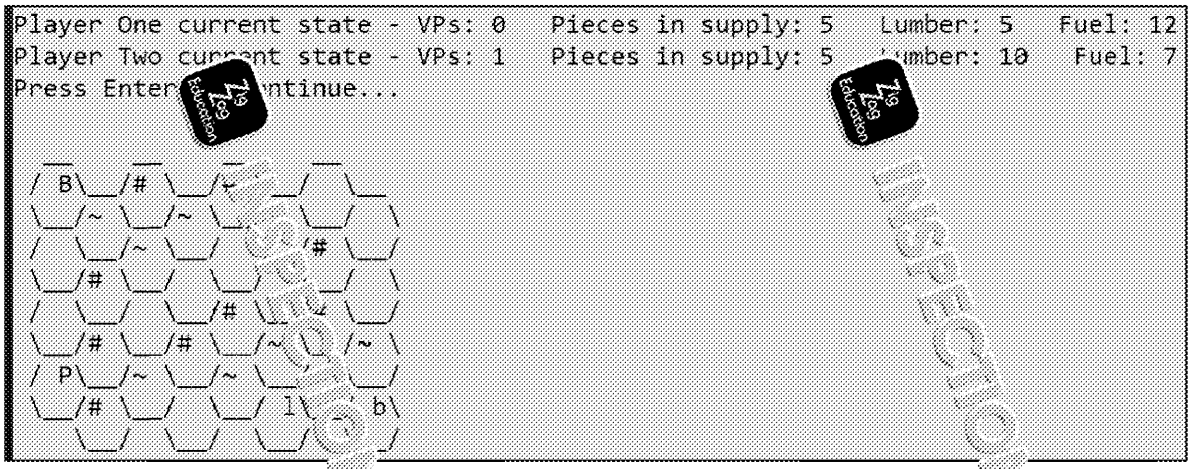
```
    If Terrain = "" Then
        Return 1
    Else
        Return 0
    End If
```

Code for the modified PlayGame subroutine:

```
Do
Dim ValidSawCommands As Integer = 0
Dim ValidDigCommands As Integer = 0
Dim DigSawLocation As String = ""
Console.WriteLine(Grid.GetGridAsString(Player1Turn))
If Player1Turn Then
    Console.WriteLine(Player1.GetName() & " state your three commands, pressing enter after each one.")
Else
```

```
Dim SupplyChange As Integer = 0
Dim SummaryOfResult As String
If DigSawLocation = "" Then
    DigSawLocation = Items(1)
End If
If Items(0) = "Saw" And Items(1) = DigSawLocation Then
    ValidSawCommands += 1
ElseIf Items(0) = "Dig" And Items(1) = DigSawLocation Then
    ValidDigCommands += 1
End If
If PlayerlTurn Then
    SummaryOfResult = Grid.ExecuteCommand(Items, FuelChange, LumberChange, SupplyChange, Player1.GetFuel(),
Player1.GetLumber(), Player1.GetPiecesInSupply())
    Player1.UpdateLumber(LumberChange)
    Player1.UpdateFuel(FuelChange)
    If SupplyChange = 1 Then
        Player1.RemoveTileFromSupply()
    End If
Else
    SummaryOfResult = Grid.ExecuteCommand(Items, FuelChange, LumberChange, SupplyChange, Player2.GetFuel(),
Player2.GetLumber(), Player2.GetPiecesInSupply())
    Player2.UpdateLumber(LumberChange)
    Player2.UpdateFuel(FuelChange)
    If SupplyChange = 1 Then
        Player2.RemoveTileFromSupply()
    End If
End If
If ValidSawCommands = 3 Then
    If PlayerlTurn Then
        Player1.UpdateLumber(2)
    Else
        Player2.UpdateLumber(2)
    End If
    Grid.MakeField(DigSawLocation)
    ValidSawCommands = 2
End If
```

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 PEASANT 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 "downgrade"
        Return CheckDowngradeCommandFormat(Items)
```

Code for new CheckDowngradeCommandFormat subroutine:

```
Function CheckDowngradeCommandFormat(ByVal Items As List(Of String)) As Boolean
    Dim Result As Integer
    If Items.Count = 2 Then
        Try
            Result = Items(1)
        Catch
            Return False
        End Try
        Return True
    End If
    Return False
End Function
```

Code for new ExecuteDowngradeCommand method:

```
Private Function ExecuteDowngradeCommand(ByVal Items As List(Of String)) As Integer
    Dim TileToUse As Integer = ToInt32(Items(1))
    If Not CheckPieceAndTileAreValid(TileToUse) Then
        Return 0
    Else
        Dim ThePiece As Piece = Tiles(TileToUse).GetPieceInTile()
        If ThePiece.GetPieceType().ToUpper <> "P" And ThePiece.GetPieceType().ToUpper <> "L" Then
            Return 0
        End If
        TheBoard.DestroyPiece()
        ThePiece = New Piece(Player1Turn)
        Pieces.Add(ThePiece)
        Tiles(TileToUse).SetPiece(ThePiece)
        Return -1
    End If
End Function
```

A. Solutions which refund the 1 lumber in a different way and hence would have alternative logic/return values from those shown above. Also accept any solution that just adds the downgrade command to the list of standard commands and uses the existing CheckStandardCommandFormat -- give marks the same as per developing your own method for this.

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

For example:

```
Player One: state your three commands, pressing enter after each one.
Enter command: move 16 24
Enter command: upgrade pbds 24
Enter command: downgrade 24
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 6   Fuel: 8
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 10   Fuel: 10
Press Enter to continue...
```

Task 9 (3 marks)

Coding:

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

For example:

```
MyBaseMethod(Player1)
PieceType = "B"
ConnectionsToDestroy = 3
VPValue = 6
```

Testing:

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

For example:

```
Player One state your three commands, pressing enter after each one.
Enter command: move 25 21
Enter command: move 21 18
Enter command: move 18 14
Command executed
Command executed
Command executed
Player One current state - VPs: 10   Pieces in supply: 5   Lumber: 10   Fuel: 6
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 10   Fuel: 10
Press Enter to continue...

 ___  ___  ___  ___
/ B\__/# \__/#s\__/  \__
\__/~ \__/~ \__/  \__/  \
/ s\__/~ \__/  \__/# \__/
\__/# \__/  \__/ S\__/ S\
/  \__/  \__/# \__/# \__/
\__/# \__/# \__/~ \__/~ \
/~ \__/~ \__/~ \__/ S\__/
\__/# \__/  \__/# \__/  \
   \__/  \__/  \__/  \__/

Player Two state your three commands, pressing enter after each one.
Enter command: move 2 5
Enter command: move 5 1
Enter command: move 1 4
Command executed
Command executed
Command executed
Player One current state - VPs: 12   Pieces in supply: 5   Lumber: 10   Fuel: 6
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 10   Fuel: 4
Press Enter to continue...
```

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 p
- 1 mark fo ctly removing the piece from the board when it is ed.

For example:

Code for modified Cr kCommandIsValid subroutine:

```
        "dig", "saw", "spawn", "salvage"
    Return CheckStandardCommandFormat(Items)
```

Code for modified Exec Command method:

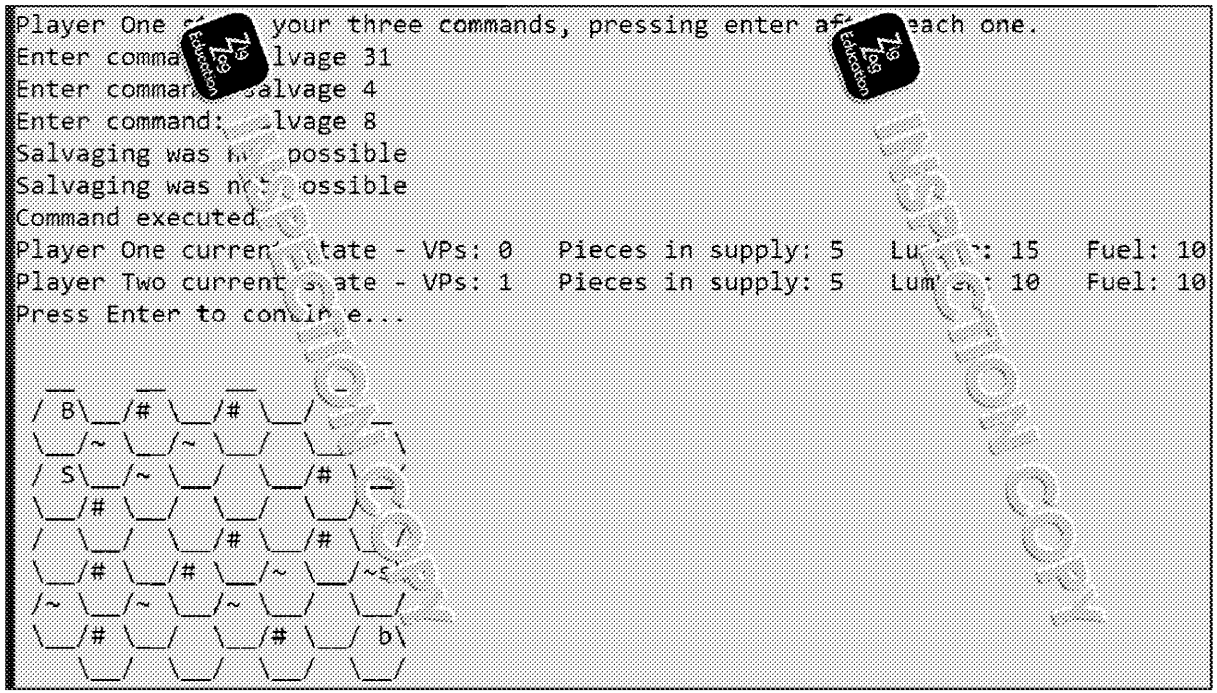
```
        Case "Salvage"
            Dim LumberCost As Integer = ExecuteSalvageComma Items)
            If LumberCost = 0 Then
                Print "Salvaging was not possible"
            End If
            LumberCh ge = -LumberCost
            SupplyC rge = -1
```

Code for new ExecuteSalvageC mand method:

```
Private Function ExecuteSalvageCommand(ByVal Items As List(Of String)) As Integer
    Dim TileToUse As Integer = ToInt32(Items(1))
    If Not CheckPieceAndTileAreValid(TileToUse) Then
        Return 0
    Else
        Dim ThePiece As Piece = Tiles(TileToUse).GetPieceInTile()
        Select Case ThePiece.GetPieceType()
            Case "S", "P", "L"
                If Player1Turn Then
```

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 PiecesInSupply, Fuel, VPs, Lumber, Chances As Integer
    Protected Name As String

    Public Sub New(ByVal N As String, ByVal V As Integer, ByVal F As Integer, ByVal L As Integer, ByVal T As Integer)
        Name = N
        VPs = V
        Fuel = F
        Lumber = L
        PiecesInSupply = T
        ResetChances()
    End Sub

    Public Overridable Sub DeductChance()
        Chances = Chances - 1
```

Code for modified PlayGame subroutine:

```
Sub PlayGame(ByVal Player1 As Player, ByVal Player2 As Player, ByVal Grid As HexGrid)
    Dim GameOver As Boolean = False
    Dim Player1Turn As Boolean = True
    Dim ValidCommand As Boolean
    Dim Commands As New List(Of String)
    Dim CurrentPlayer As Player
    Player1.ResetChances()
    Player2.ResetChances()
    Console.WriteLine("Player one current state - " & Player1.GetStateString())
    Console.WriteLine("Player Two current state - " & Player2.GetStateString())
    Do
        Console.WriteLine(Grid.GetGridAsString(Player1Turn))
        If Player1Turn Then
            CurrentPlayer = Player1
        Else
            CurrentPlayer = Player2
        End If
        Console.WriteLine(CurrentPlayer.GetName() & " you have " & CStr(CurrentPlayer.GetChances()) & " chances remaining.")
        Console.WriteLine(CurrentPlayer.GetName() & " state your three commands, pressing enter after each one.")
        For Count = 1 To 3
            Console.Write("Enter command " & CStr(Count) & ": ")
            Commands.Add(Console.ReadLine().ToLower())
        Next
        For CommandNo = 1 To 3
            Dim C As String = Commands(CommandNo - 1)
            Dim Items As List(Of String) = New List(Of String)(C.Split(" "))
            Dim CommandExecuted As Boolean = False
            ValidCommand = CheckCommandIsValid(Items)
            While Not ValidCommand And CurrentPlayer.GetChances() > 0
                Console.WriteLine("Command number " & CStr(CommandNo) & " is an invalid command.")
                If CurrentPlayer.GetChances() > 0 Then
                    CurrentPlayer.DeductChance()
                    Console.Write("Enter new command " & CStr(CommandNo) & ": ")
```

```
        CurrentPlayer.GetLumber(), CurrentPlayer.GetPiecesInSupply())
        CurrentPlayer.UpdateLumber(LumberChange)
        CurrentPlayer.UpdateFuel(FuelChange)
        If SupplyChange = 1 Then
            CurrentPlayer.RemoveTileFromSupply()
        End If
        Console.WriteLine(SummaryOfResult)
        If SummaryOfResult = "Command executed" Then
            CommandExecuted = True
        ElseIf CurrentPlayer.GetChances() > 0 Then
            CurrentPlayer.DeductChance()
            Console.Write("Enter new command " & Str(CommandNo) & ": ")
            C = Console.ReadLine().ToLower()
            Items = New List(Of String)(C.Split(" "))
            ValidCommand = CheckCommandIsValid(Items)
            If Not ValidCommand And CurrentPlayer.GetChances() = 0 Then
                Console.WriteLine("Command number " & Str(CommandNo) & " is an invalid command.")
            Else
                While Not ValidCommand And CurrentPlayer.GetChances() > 0
                    Console.WriteLine("Command number " & Str(CommandNo) & " is an invalid command.")
                    If CurrentPlayer.GetChances() > 0 Then
                        CurrentPlayer.DeductChance()
                        Console.Write("Enter new command " & Str(CommandNo) & ": ")
                        C = Console.ReadLine().ToLower()
                        Items = New List(Of String)(C.Split(" "))
                        ValidCommand = CheckCommandIsValid(Items)
                        If Not ValidCommand And CurrentPlayer.GetChances() = 0 Then
                            Console.WriteLine("Command number " & Str(CommandNo) & " is an invalid command.")
                        End If
                    Else
                        CommandExecuted = True
                        ValidCommand = True
                    End If
                End While
            End If
```

```
        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()
    Loop Until GameOver And Player1Turn
    Console.WriteLine(Grid.GetGridAsString(Player1
    DisplayEndMessages(Player1, Player2)
End Sub
```

A. Solutions will retain the use of Player1 and Player2 throughout.

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:

```
Player One you have 3 chances remaining.
Player One state your three commands, pressing enter after each one.
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
You have 1 chances remaining.
Command 2 : Command executed
Command 3 : Upgrade not possible
Enter new command 3: move 20 28
```

```
Player Two you have 3 remaining.
Player Two state your three commands, pressing enter after each one.
Enter command 1: move 23 27
Enter command 2: move 27 30
Enter command 3: upgrade less 30
Command 1 : Command executed
Command 2 : Command executed
Command 3 : Command executed
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 10   Fuel: 7
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 5   Fuel: 7
Press Enter to continue...

 / 8\__/#\__/#\__/ \__/ \
 \__/~ \__/~ \__/ \__/ \
 /  \__/~ \__/ \__/#\__/
 \__/# \__/  \__/  \__/ \
 /  \__/  \__/# \__/# \__/
 \__/# \__/# \__/~ \__/~ \
 /~ \__/~ \__/~ \__/  \__/
 \__/#5\__/  \__/#1\__/ b\
    \__/  \__/  \__/  \__/

Player One you have 0 remaining.
Player One state your three commands, pressing enter after each one.
Enter command 1: upgrade less 28
Enter command 2: saw 28
Enter command 3: saw 29
Command 1 : Command executed
Command 2 : Command executed
Command 3 : Couldn't do that
```

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 VPs (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 FuelCostToMove 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 "order", "save"
                Return CheckStandardCommandFormat(Items)
            Case "spawn"
                Return CheckSpawnCommandFormat(Items)
```

Code for new CheckSpawnCommandFormat subroutine:

```
    Function CheckSpawnCommandFormat(ByVal Items As List(Of String)) As Boolean
        Dim Result As Integer
        If Items.Count = 2 Then
            Return CheckStandardCommandFormat(Items)
        Else
```

Code for modified private ExecuteSpawnCommand method:

```
Dim SpawnCost, TileToUse As Integer
If Items.Count = 2 Then
    TileToUse = ToInt32(Items(1))
    SpawnCost = 3
Else
    TileToUse = ToInt32(Items(2))
    SpawnCost = 10
End If
If PieceSpawnSupply < 1 Or LumberAvailable < SpawnCost Or Not CheckTileIndexIsValid(TileToUse) Then
    Return -1
End If
Dim ThePiece As Piece = Tiles(TileToUse).GetPieceInTile()
If ThePiece IsNot Nothing Then
    Return -1
End If
Dim OwnBaronIsNeighbour As Boolean = False
Dim ListOfNeighbours As List(Of Tile) = New List(Of Tile)(Tiles(TileToUse).GetNeighbours())
For Each N In ListOfNeighbours
    ThePiece = N.GetPieceInTile()
    If ThePiece IsNot Nothing Then
        If PlayerlTurn And ThePiece.GetPieceType() = "B" Or Not PlayerlTurn And ThePiece.GetPieceType() = "b" Then
            OwnBaronIsNeighbour = True
            Exit For
        End If
    End If
Next
If Not OwnBaronIsNeighbour Then
    Return -1
End If
If Items(1) = "bomb" Then
    Dim NewPiece As New BombPiece(PlayerlTurn)
    Pieces.Add(NewPiece)
    Tiles(TileToUse).SetPiece(NewPiece)
```

Code for new BombPiece class:

```
Class BombPiece
    Inherits Piece
    Protected MovesBeforePrimed As Integer
    Public Sub New(ByVal Player1 As Boolean)
        MyBase.New(Player1)
        MovesBeforePrimed = 5
        PieceType = "Y"
        FuelCostOfMove = 2
        VPValue = 5
    End Sub

    Public Overrides Function CheckMoveIsValid(ByVal DistanceBetweenTiles As Integer, ByVal StartTerrain As String, ByVal EndTerrain
    As String) As Integer
        If DistanceBetweenTiles = 1 And MovesBeforePrimed > 0 Then
            MovesBeforePrimed -= 1
            Return FuelCostOfMove
        End If
        Return -1
    End Function

    Public Overrides Sub Explode(ByVal TileLocation As Tile, ByRef Player1VPs As Integer, ByRef Player2VPs As Integer, ByRef
    BaronDestroyed As Boolean)
        If Not Destroyed Then
            Dim Neighbours As New List(Of Tile)(TileLocation.GetNeighbours())
            For Each Neighbour In Neighbours
                Dim TilePiece As Piece = Neighbour.GetPieceInTile()
                If TilePiece IsNot Nothing Then
                    If TilePiece.GetBelongsToPlayer1() Then
                        Player2VPs += TilePiece.GetVPs()
                    Else
                        Player1VPs += TilePiece.GetVPs()
                    End If
                    If TilePiece.GetPieceType().ToUpper() = "B" Then
```

```
    Public Overloads Sub DestroyPiece(TileLocation As Tile)
        If Not Destroyed Then
            Dim Neighbours As New List(Of Tile)(TileLocation.GetNeighbours())
            ConnectionsToDestroy = 0
            MovesBeforePrimed = -1
            Destroyed = True
            For Each Neighbour In Neighbours
                Dim TilePiece As Piece = Neighbour.GetPieceInTile()
                If TilePiece IsNot Nothing Then
                    TilePiece.Explode()
                End If
            Next
        End If
    End Sub
End Class
```

Code for modified DestroyPiecesAndCountVPs method:

```
    Dim BaronDestroyed As Boolean = False
    Dim ListOfTilesContainingDestroyedPieces As New List(Of Tile)
    For Each T In Tiles
        If T.GetPieceInTile() IsNot Nothing Then
            Dim ListOfNeighbours As List(Of Tile) = New List(Of Tile)(T.GetNeighbours())
            Dim NoOfConnections As Integer = 0
            For Each N In ListOfNeighbours
                If N.GetPieceInTile() IsNot Nothing Then
                    NoOfConnections += 1
                End If
            Next
            Dim ThePiece As Piece = T.GetPieceInTile()
            If NoOfConnections >= ThePiece.GetConnectionsNeededToDestroy() Then
                If ThePiece.GetPieceType().ToUpper() = "A" Then
                    ThePiece.Explode(T, Player1VPs, Player2VPs, BaronDestroyed)
                Else
                    ThePiece.DestroyPiece()
                    If ThePiece.GetPieceType().ToUpper() = "B" Then
                        BaronDestroyed = True
```

Code for modified GetPieceTypeInTile method:

```
Public Function GetPieceTypeInTile(ByVal ID As Integer) As String
    Dim ThePiece As Piece = Tiles(ID).GetPieceInTile()
    If ThePiece Is Nothing Then
        Return " "
    Else
        Dim PieceType As String = ThePiece.GetPiece
        If PieceType = "X" Then
            PieceType = "S"
        ElseIf PieceType = "x" Then
            PieceType = "s"
        End If
        Return PieceType
    End If
End Function
```

Code for modified ExecuteMoveCommand method:

```
    Dim StartID As Integer = ToInt32(Items(1))
    Dim EndID As Integer = ToInt32(Items(2))
    If Not CheckPieceAndTileAreValid(StartID) Or Not CheckTileIndexIsValid(EndID) Then
        Return -1
    End If
    Dim ThePiece As Piece = Tiles(StartID).GetPieceInTile()
    If Tiles(EndID).GetPieceInTile() IsNot Nothing Then
        Return -1
    End If
    Dim Distance As Integer = Tiles(StartID).GetDistanceToTileT(Tiles(EndID))
    Dim FuelCost As Integer = ThePiece.CheckMoveIsValid(Distance, Tiles(StartID).GetTerrain(), Tiles(EndID).GetTerrain())
    Dim Neighbours As List(Of Tile) = New List(Of Tile)(Tiles(EndID).GetNeighbours())
    If FuelCost = -1 Or FuelAvailable < FuelCost Then
        Return -1
    End If
    MovePiece(EndID, StartID)
```

Code for modified Piece class:

```
Public Overridable Sub Explode()
    ConnectionsToDestroy = 0
End Sub
```

A. Solutions w[illegible]lve the problem using a different approach and [illegible]of the criteria on the mark scheme, award full marks. [illegible] awarding partial marks the criteria can b[illegible]nterpreted slightly loosely if that seems reasonable g[illegible]n the nature of the alternative solution.

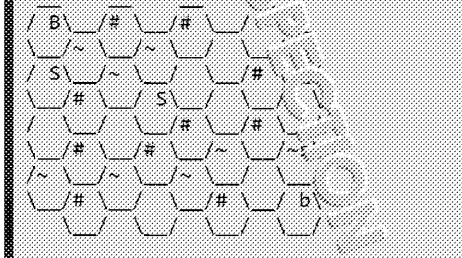
R. Solutions that b[illegible]k OOP by accessing private and protected attributes [illegible]ugly or unreasonable ways, or solutions that unnecessar[illegible]rovide direct access to things when they could [illegible]olved in a more suitable way using the OOP hierar[illegible]encapsulation and polymorphism.

Testing:

- 1 mark for showing t[illegible]e commands entered correctly and all of the output fr[illegible]e computer (for both games).
- 1 mark for Player One [illegible]nning the first game with the correct VP totals.
- 1 mark for Player One winning the second game with the correct VP totals.

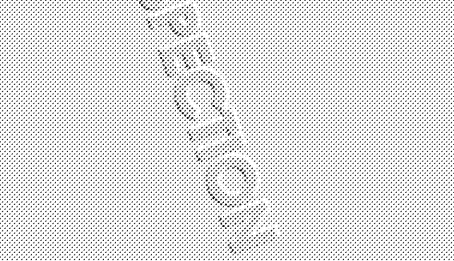
For example – Game One:

```
Player One state your three commands, pressing enter after each one.
Enter command: spawn bomb 4
Enter command: move 4 9
Enter command: move 9 13
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 4   Lumber: 20   Fuel: 26
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 30   Fuel: 30
Press Enter to continue...
```



```
Player Two state your three commands, pressing enter after each one.
Enter command: move 23 19
Enter command: move 19 11
Enter command: move 11 14
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 4   Lumber: 20   Fuel: 26
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 30   Fuel: 26
Press Enter to continue...
```

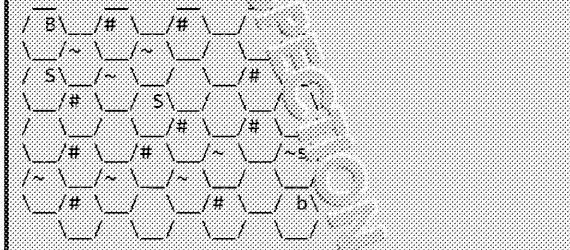
```
Player One state your three commands, pressing enter after each one.
Enter command: move 13 18
Enter command: move 18 22
Enter command: move 22 27
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 4   Lumber: 20   Fuel: 20
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 30   Fuel: 26
Press Enter to continue...
```



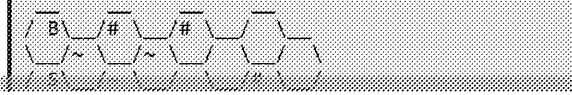
```
Player Two state your three commands, pressing enter after each one.
Enter command: move 31 23
Enter command: move 14 18
Enter command: move 18 22
Command executed
Command executed
Command executed
Player One current state - VPs: 10   Pieces in supply: 4   Lumber: 20   Fuel: 20
Player Two current state - VPs: 6   Pieces in supply: 5   Lumber: 30   Fuel: 22
Press Enter to continue...
```

For example – Game Two:

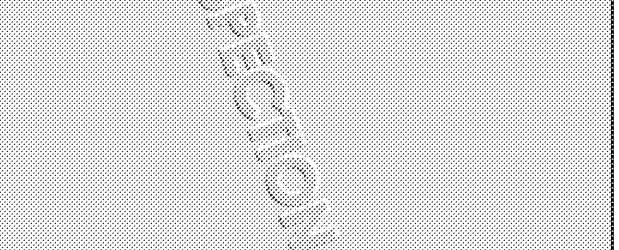
```
Player One state your three commands, pressing enter after each one.
Enter command: spawn bomb 4
Enter command: move 4 9
Enter command: move 9 13
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 4   Lumber: 20   Fuel: 26
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 30   Fuel: 30
Press Enter to continue...
```



```
Player Two state your three commands, pressing enter after each one.
Enter command: move 23 19
Enter command: upgrade less 19
Enter command: saw 19
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 4   Lumber: 20   Fuel: 26
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 26   Fuel: 28
Press Enter to continue...
```



```
Player One state your three commands, pressing enter after each one.
Enter command: move 13 18
Enter command: move 18 22
Enter command: move 22 27
Command executed
Command executed
Command executed
Player One current state - VPs: 13   Pieces in supply: 4   Lumber: 20   Fuel: 20
Player Two current state - VPs: 6   Pieces in supply: 5   Lumber: 26   Fuel: 28
Press Enter to continue...
```



```
Player Two state your three commands, pressing enter after each one.
Enter command: saw 19
Enter command: saw 19
Enter command: saw 19
Couldn't do that
Couldn't do that
Couldn't do that
Player One current state - VPs: 13   Pieces in supply: 4   Lumber: 20   Fuel: 20
Player Two current state - VPs: 6   Pieces in supply: 5   Lumber: 26   Fuel: 28
Press Enter to continue...
```

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 teleport.
- 1 mark for using 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 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
    Inherits Piece
    Private Teleported As Boolean
    Public Sub New(ByVal Player1 As Boolean)
        MyBase.New(Player1)
        PieceType = "W"
        VPValue = 3
        ResetTeleport()
    End Sub

    Public Overrides Function CheckMoveIsValid(ByVal DistanceBetweenTiles As Integer, ByVal StartTerrain As String, ByVal EndTerrain
    As String) As Integer
        If DistanceBetweenTiles = 1 Then
```

Code for modified CheckUpgradeCommandFormat subroutine:

```
Dim Result As Integer
If Items.Count = 3 Then
    If Items(1).ToUpper() <> "LESS" And Items(1).ToUpper() <> "PBDS" And Items(1).ToUpper() <> "WIZ" Then
        Return False
    End If
    Result = Items(2)
    Catch
        Return False
    End Try
    Return True
End If
Return False
```

Code for modified ExecuteUpgradeCommand method:

```
Dim TileToUse As Integer = ToInt32(Items(2))
If Not CheckPieceAndTileAreValid(TileToUse) Or LumberAvailable < 5 Or Not (Items(1) = "pbds" Or Items(1) = "less" Or
Items(1) = "wiz") Then
    Return -1
Else
    Dim ThePiece As Piece = Tiles(TileToUse).GetPieceInTile()
    If ThePiece.GetPieceType().ToUpper <> "S" Then
        Return -1
    End If
    ThePiece.DestroyPiece()
    If Items(1) = "pbds" Then
        ThePiece = New PBDSPiece(Player1Turn)
    ElseIf Items(1) = "less" Then
        ThePiece = New LESSPiece(Player1Turn)
    Else
        ThePiece = New WizardPiece(Player1Turn)
```

Code for new method ResetWizards in the HexGrid class:

```
    Private Sub ResetWizards()
        For Each T In Tiles
            Dim PieceInTile As Piece = T.GetPieceInTile()
            If PieceInTile IsNot Nothing Then
                If PieceInTile.GetPieceType().ToUpper() = "W" Then
                    Dim Wizard As WizardPiece = T.GetPieceInTile()
                    Wizard.ResetTeleport()
                End If
            End If
        Next
    End Sub
```

Code for new call to ResetWizards in the PlayGame subroutine:

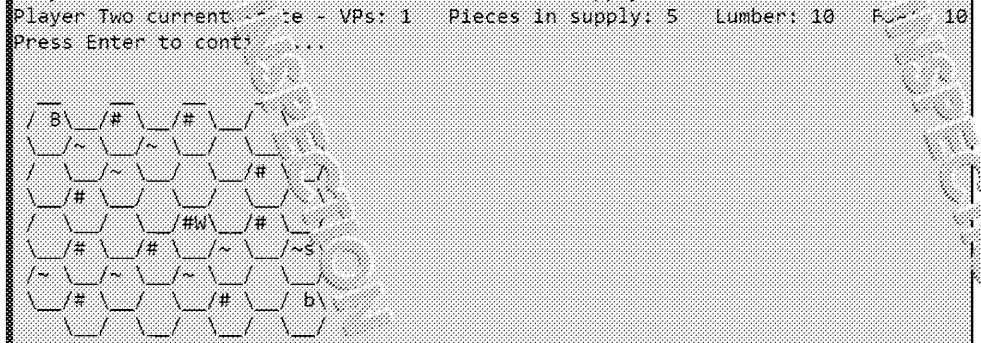
```
            Next
            Grid.ResetWizards()
            Commands.Clear()
```

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:

```
Player One state your three commands, pressing enter after each one.
Enter command: upgrade wiz 8
Enter command: move 8 13
Enter command: move 13 18
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 5   Fuel: 14
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 10   Fuel: 10
Press Enter to continue...
```



```
Player Two state your three commands, pressing enter after each one.
Enter command: move 23 19
Enter command: move 19 11
Enter command: upgrade less 11
Command executed
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 5   Fuel: 14
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 5   Fuel: 7
Press Enter to continue...
```

```
Player One state your three commands, pressing enter after each one.
Enter command: move 18 8
Enter command: move 18 13
Enter command: move 13 8
That move can't be done
Command executed
Command executed
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 5   Fuel: 8
```


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
    Inherits Piece
    Public Sub New(ByRef Player1 As Boolean)
        MyBase.New(Player1)
        PieceType = "F"
        VPValue = 3
    End Sub

    Public Overrides Function CheckMoveIsValid(ByVal DistanceBetweenTiles As Integer, ByVal StartTerrain As String, ByVal EndTerrain
As String) As Integer
        Return -1
    End Function

    Public Overridable Function Supply(ByVal FuelAvailable As Integer, ByVal LumberAvailable As Integer, ByRef Fuel As Integer,
ByRef Lumber As Integer) As Integer
        If FuelAvailable >= 5 And LumberAvailable >= 5 Then
            Fuel -= 5
```

Code for the modified CheckCommandIsValid subroutine:

```
Function CheckCommandIsValid(ByVal Items As List(Of String), SupplyFirst As Boolean) As Boolean
    If Items.Count > 0 Then
        Select Case Items(0)
            Case "move"
                Return CheckMoveCommandFormat(Items)
            Case "dig", "saw", "spawn", "supply"
                Dim ValidCommand As Boolean = CheckStandardCommandFormat(Items)
                If Items(0) = "supply" And Not SupplyFirst Then
                    Return False
                Else
                    Return ValidCommand
                End If
            Case "upgrade"
                Return CheckUpgradeCommandFormat(Items)
        End Select
    End If
    Return False
End Function
```

Code for the modified CheckUpgradeCommandFormat subroutine:

```
Function CheckUpgradeCommandFormat(ByVal Items As List(Of String)) As Boolean
    Dim Result As Integer
    If Items.Count = 3 Then
        If Items(1).ToUpper() <> "LESS" And Items(1).ToUpper() <> "MORE" And Items(1).ToUpper() <> "SAME" Then
            Return False
        End If
        Try
            Result = Items(2)
        Catch
            Return False
        End Try
        Return True
    End If
```

Code for modified ExecuteCommand method:

```
    Case "supply"
        Dim Supply As Integer = ExecuteSupplyCommand(Items, FuelAvailable, LumberAvailable, FuelChange, LumberChange)
        If Supply = 0 Then
            Return "Couldn't do that"
        Else
            LumberChange = Supply
        End If
```

Code for new ExecuteSupplyCommand method:

```
    Private Function ExecuteSupplyCommand(ByVal Items As List(Of String), ByVal FuelAvailable As Integer, ByVal LumberAvailable As
    Integer, ByRef Fuel As Integer, ByRef Lumber As Integer) As Integer
        Dim TileToUse As Integer = ToInt32(Items(1))
        If CheckPlayerAndTileAreValid(TileToUse) = False Then
            Return 0
        End If
        Dim ThePiece As Piece = Tiles(TileToUse).GetPieceInTile()
        Items(0) = Items(0)(0).ToString().ToUpper() & Items(0).ToString().Substring(1)
        If ThePiece.GetPieceType().ToUpper() = "F" Then
            Dim Factory As SCFWPiece = ThePiece
            Return Factory.Supply(FuelAvailable, LumberAvailable, Fuel, Lumber)
        End If
        Return 0
    End Function
```

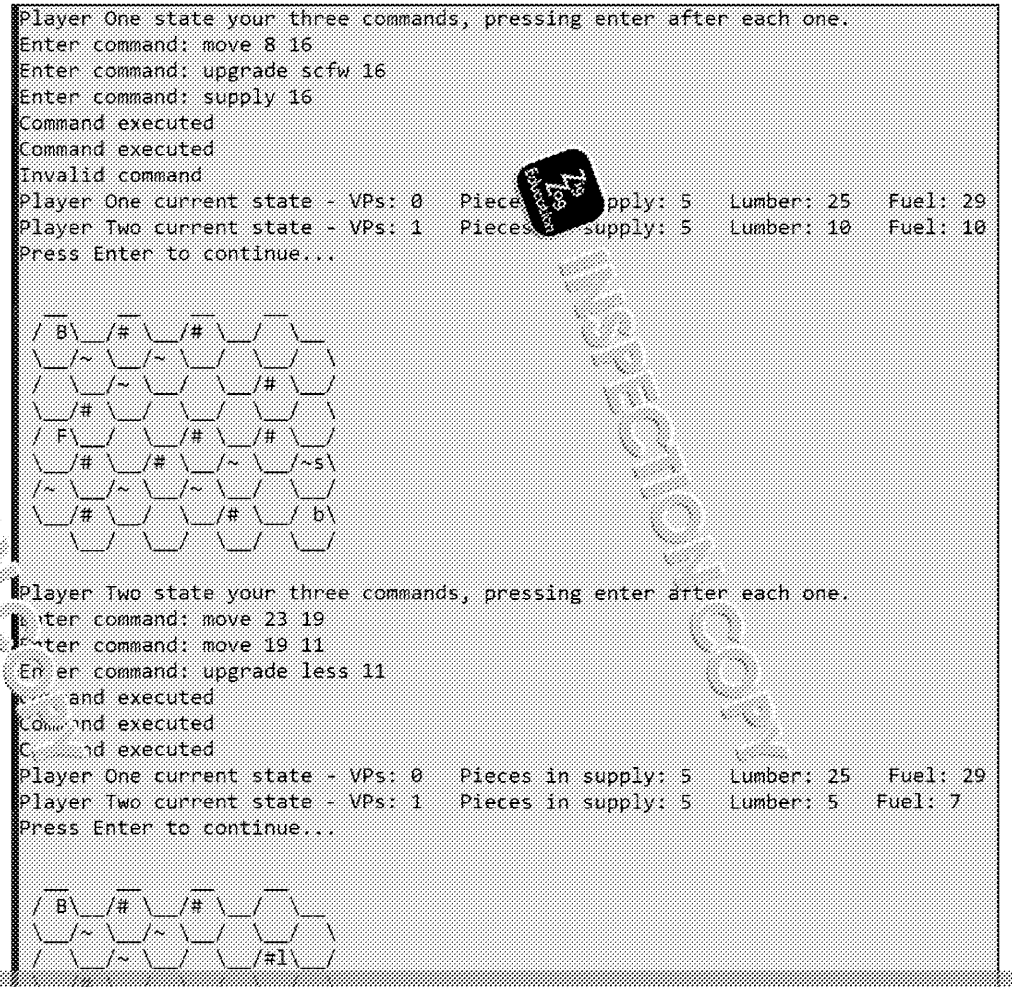
Code for modified PlayGame subroutine:

```
    Sub PlayGame(ByVal Player1 As Player, ByVal Player2 As Player, ByVal Grid As HexGrid)
        Dim GameOver As Boolean = False
        Dim Player1Turn As Boolean = True
        Dim ValidCommand As Boolean
        Dim Commands As New List(Of String)
        Dim SupplyFirst As Boolean = False
        Console.WriteLine("Player One current state - " & Player1.GetStateString())
```

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 it 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 FileName as a parameter correctly and doesn't ask for 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:

```
Sub DisplayMainMenu()
    Console.WriteLine("1. Default game")
    Console.WriteLine("2. Load game")
    Console.WriteLine("3. Create game")
    Console.WriteLine("4. Quit")
    Console.WriteLine()
    Console.Write("Enter your choice: ")
End Sub
```

Main subroutine:

```
        Answer = Console.ReadLine().ToLower()
        If Answer = "yes" Or Answer = "y" Then
            FileLoaded = LoadGame(FileName, Player1, Player2, Grid)
            If FileLoaded Then
                PlayGame(Player1, Player2, Grid)
            End If
        End
    End If
```

LoadGame subroutine:

```
    Function LoadGame(ByVal FileName As String, ByRef Player1 As Player, ByRef Player2 As Player, ByRef Grid As HexGrid) As Boolean
        Dim Items As List(Of String)
...
    Dim T As List(Of String) = New List(Of String)(MyStream.ReadLine().Split(","))
    Dim LastT As Integer = T.Count - 1
    If T(LastT) = "" Then
        T(LastT) = "Empty"
    End If
    Grid.SetUpGridTerrain(T)
```

CreateGame subroutine:

```
    Function CreateGame(ByRef FileName As String) As Boolean
        Dim Grid As HexGrid
        Dim GridSize As Integer
        Dim Forests As List(Of String)
        Dim PeatBogs As List(Of String)
        Dim TerrainList As List(Of String) = New List(Of String)
        Dim NewTerrain As List(Of String) = New List(Of String)
        Dim P1stats As List(Of Integer) = New List(Of Integer)
        Dim P2stats As List(Of Integer) = New List(Of Integer)
```

```
End While
Grid = New HexGrid(GridSize)
For TerrainIndex = 0 To (GridSize * GridSize / 2) - 1
    TerrainList.Add(" ")
Next
Grid.SetUpGridTerrain(TerrainList)
Console.WriteLine(Grid.GetGridAsIndices(True))
Console.Write("Please enter all the locations of forests separated by commas with no spaces: ")
Forests = New List(Of String)(Console.ReadLine().Split(","))
Console.Write("Please enter all the locations of peaks separated by commas with no spaces: ")
Peaks = New List(Of String)(Console.ReadLine().Split(","))
For TerrainIndex = 0 To TerrainList.Count - 1
    Dim Hex As String = CStr(TerrainIndex)
    If Peaks.Contains(Hex) Then
        NewTerrain.Add("^")
    ElseIf Forests.Contains(Hex) Then
        NewTerrain.Add("#")
    Else
        NewTerrain.Add(" ")
    End If
Next

Grid.SetUpGridTerrain(NewTerrain)
Console.Write("Please enter the location of the Baron for player 1: ")
P1locations.Add(Int32(Console.ReadLine()))
P1pieces.Add("Baron")
Grid.AddPiece(True, "Baron", P1locations(0))
Console.Write("Please enter the location of the Baron for player 2: ")
P2locations.Add(ToInt32(Console.ReadLine()))
P2pieces.Add("Baron")
Grid.AddPiece(False, "Baron", P2locations(0))
Console.WriteLine(Grid.GetGridAsString(True))

Console.Write("Please enter the starting amount of Vis for player 1: ")
Player1.AddVis(Int32(Console.ReadLine))
```

```
While NextPiece <> "n"
    Console.Write("Enter piece to add for Player One (S-Serif,I-Less,P-PBS) or N for None: ")
    NextPiece = Console.ReadLine().ToUpper()
    If NextPiece <> "N" Then
        Console.Write("Enter the index of the box that piece should be placed: ")
        Plocations.Add(ToInt32(Console.ReadLine()))
        If NextPiece = "S" Then
            Pieces.Add("Serif")
        ElseIf NextPiece = "P" Then
            Pieces.Add("PBS")
        ElseIf NextPiece = "I" Then
            Pieces.Add("Less")
        End If
        Grid.AddPiece(True, Pieces(Pieces.Count - 1), Plocations(Plocations.Count - 1))
    End If
End While
NextPiece = "n"
While NextPiece <> "n"
    Console.Write("Enter piece to add for Player Two (S-Serif,I-Less,P-PBS) or N for None: ")
    NextPiece = Console.ReadLine().ToUpper()
    If NextPiece <> "N" Then
        Console.Write("Enter the index of the box where that piece should be placed: ")
        P2locations.Add(ToInt32(Console.ReadLine()))
        If NextPiece = "S" Then
            P2pieces.Add("Serif")
        ElseIf NextPiece = "P" Then
            P2pieces.Add("PBS")
        ElseIf NextPiece = "I" Then
            P2pieces.Add("Less")
        End If
        Grid.AddPiece(False, P2pieces(P2pieces.Count - 1), P2locations(P2locations.Count - 1))
    End If
End While
```

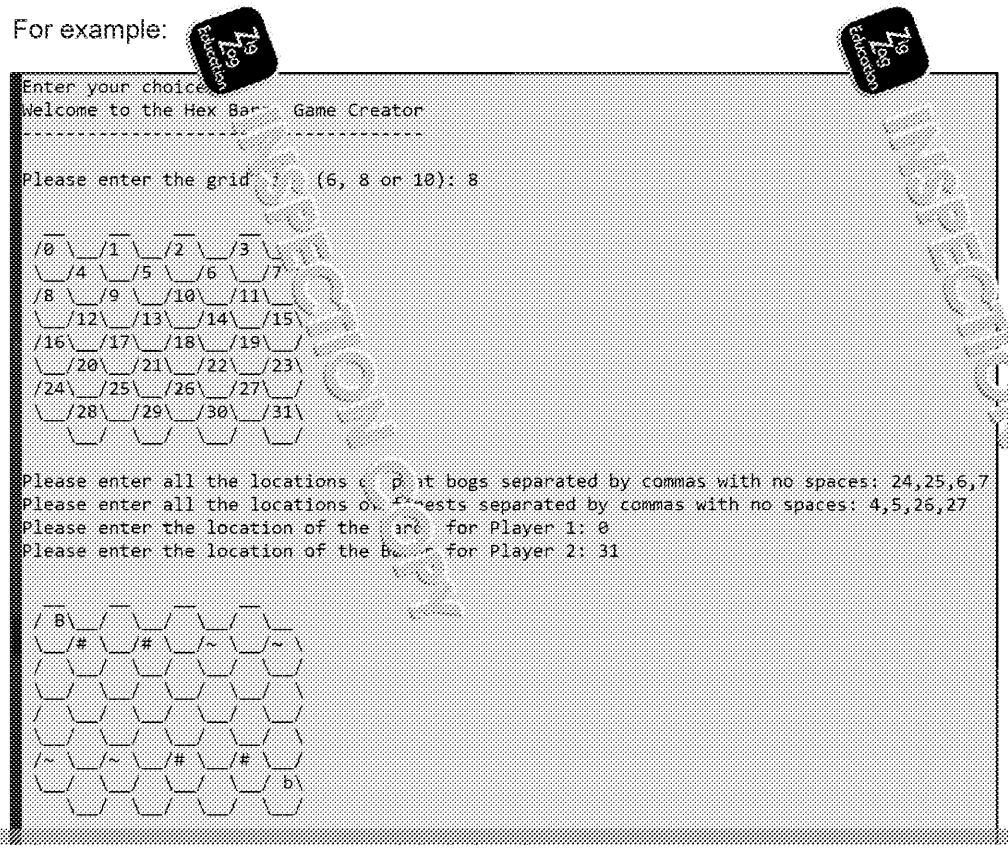
```
            P2string &= "," & CStr(value)
        Next
        File.WriteLine(P1string)
        File.WriteLine(P2string)
        File.WriteLine(CStr(GridSize))
        For TerrainIndex = 1 To NewTerrain.Count - 1
            GridString &= "," & NewTerrain(TerrainIndex)
        Next
        File.WriteLine(GridString)
        For Index = 0 To P1pieces.Count - 1
            File.WriteLine("1," & P1pieces(Index) & "," & CStr(P1locations(Index)))
        Next
        For Index = 0 To P2pieces.Count - 1
            File.WriteLine("2," & P2pieces(Index) & "," & CStr(P2locations(Index)))
        Next
        End Using
        Return True
    Catch
        Console.WriteLine("File could not be saved.")
    End Try
    Return False
End Function
```

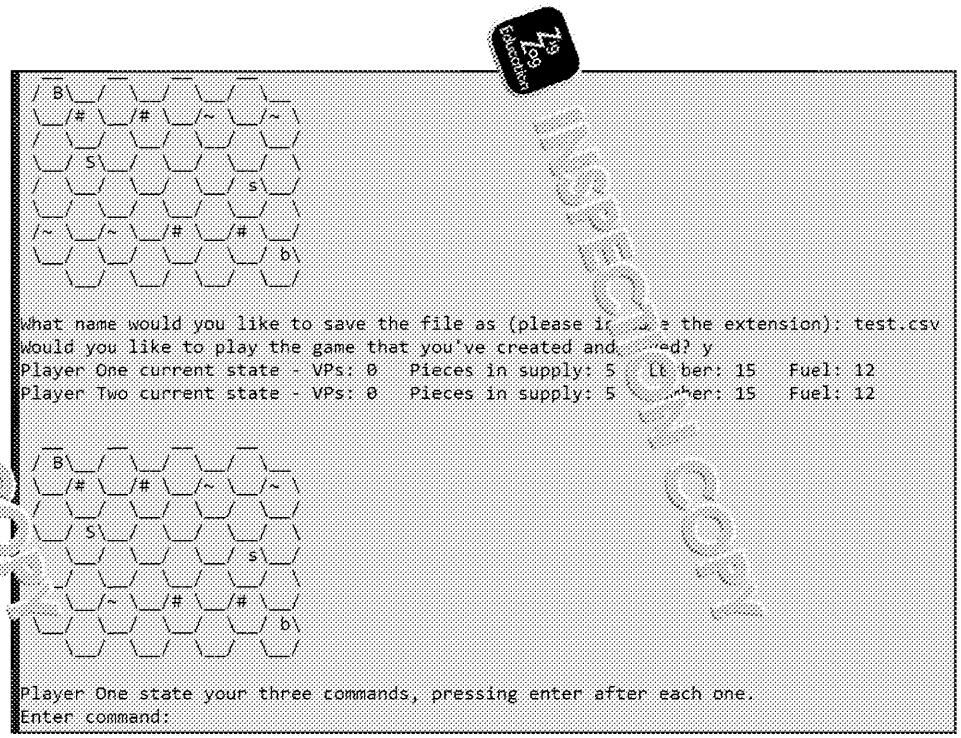
A. Solutions which have alternative input mechanisms or file printing mechanisms/structures as long as the code works.

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

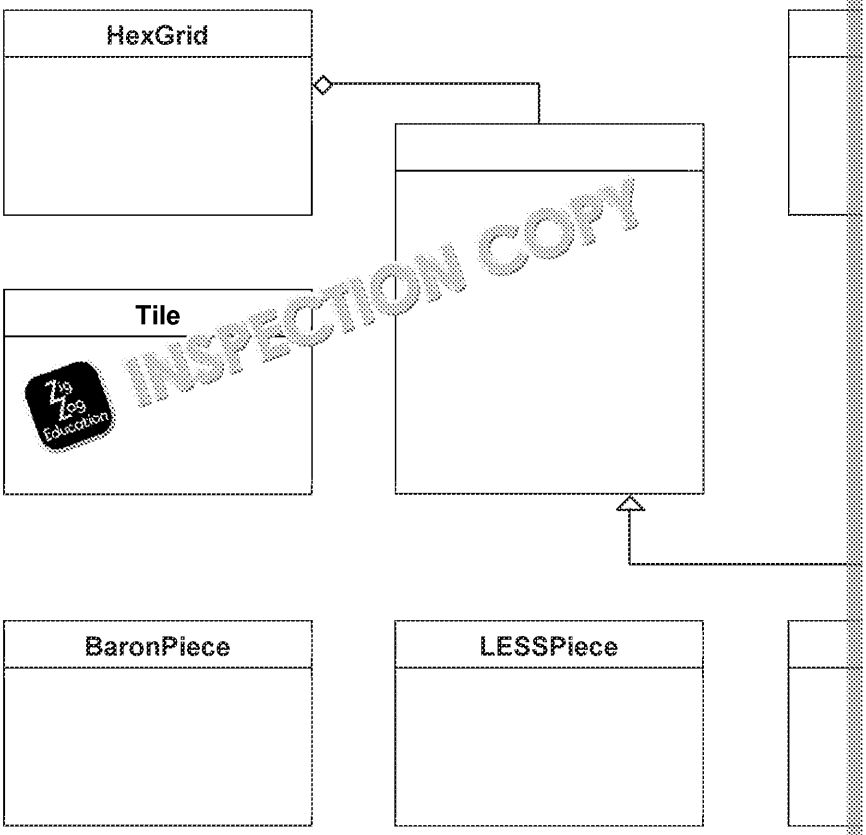
- You will need:
  - access to a computer
  - access to a printer
  - access to appropriate software
  - electronic copies of the required skeleton code
  - EAD (Electronic Answer Document)

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	