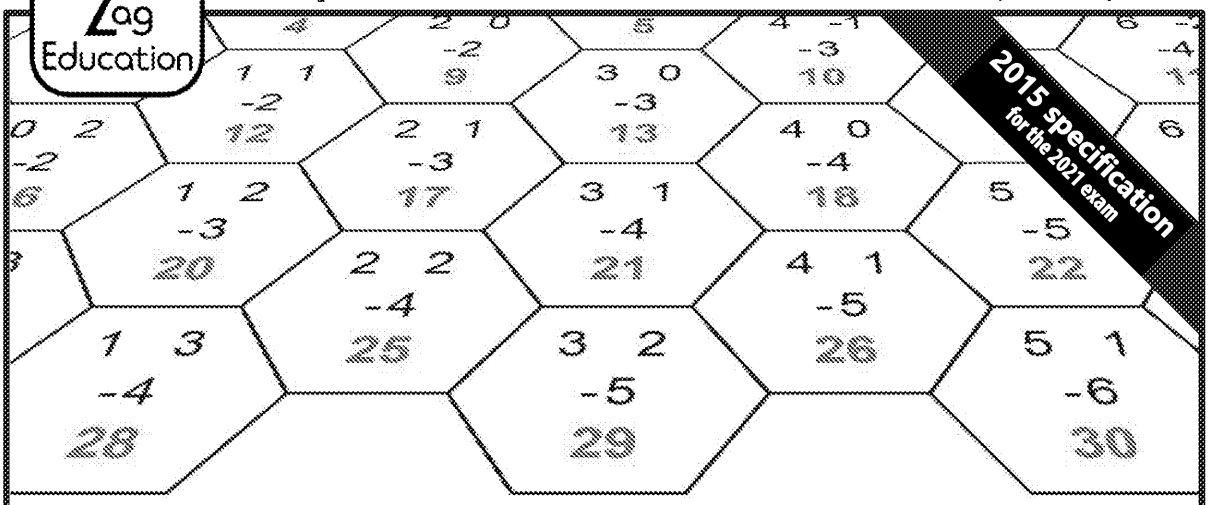
Zig Zag Education

Computer Science

A Level | AQA | 7517

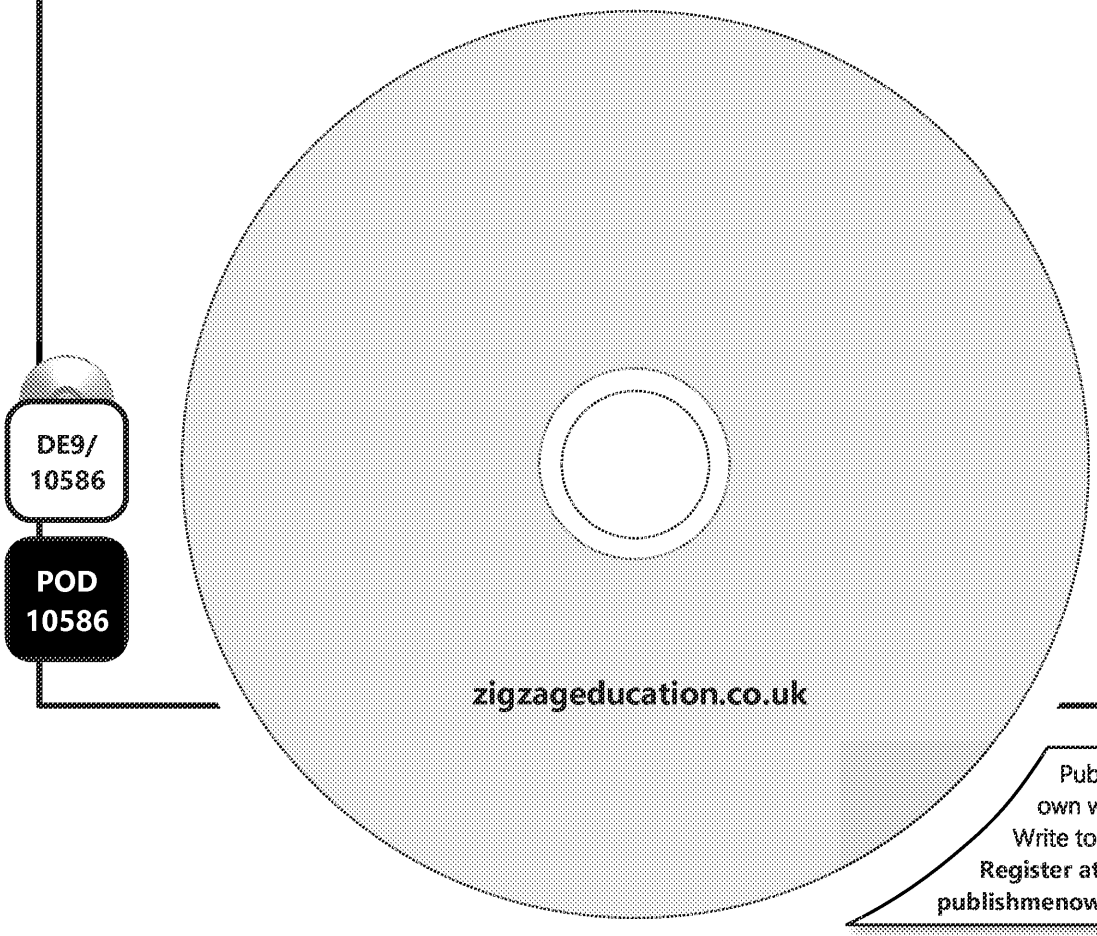


PAPER 1 EXAM RESOURCE PACK 2021

HEX BARON

for A Level AQA Computer Science

PYTHON³ EDITION



DE9/
10586

POD
10586

zigzageducation.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 (Python[3]) – 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:

- `p06a-ClassDiagramTasksMS.pdf`
- `p06b-ClassDiagramFull.pdf`
- `p07-TheoryQuestionsMS.pdf`
- `p08-ProgrammingTasksMS.pdf`

The resource pack consists of the following:

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

> ***Note:** although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves.*

② **Class Diagram Tasks**

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

③ **Written Questions**

Theory questions testing students' understanding of the skeleton program, like Section C in the exam. These questions require access to the program, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on version are available format. Suggested answers are provided via the *Solutions* web page as a password-protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test, like Section D in the exam. Example solutions with suggested mark schemes are provided via the *Solutions* web page as a password-protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

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

HEX BARON – Commentary

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

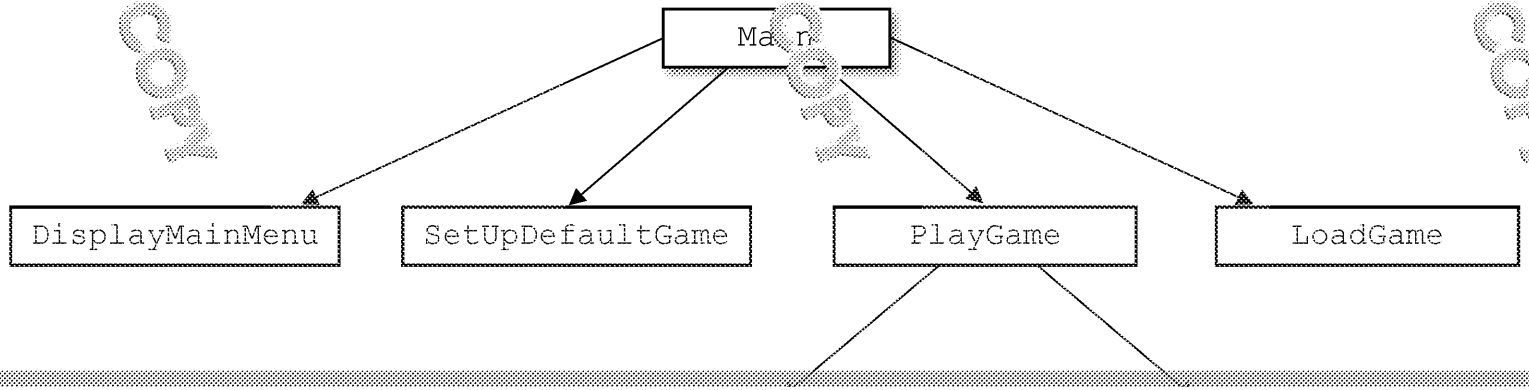
In the game you have to enter three moves before any of them are validated, so if you make a mistake with the first move you could end up losing your entire turn of all three moves as it might affect your second and third move if you thought you'd achieved something that had in fact failed.

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

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

Subroutines

Subroutines (Main Program): Hierarchy Diagram

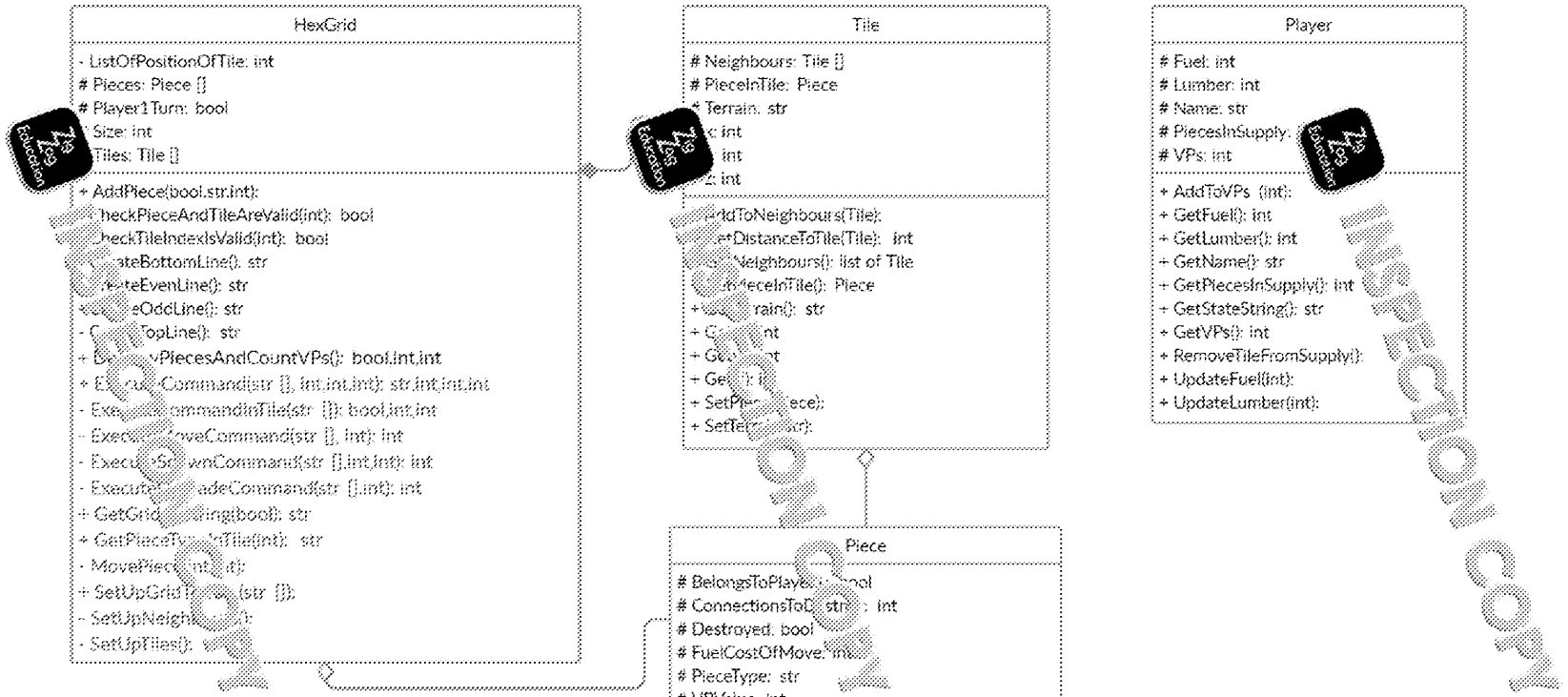


Subroutines (All)

Name	Data	Description
`CheckCommandIsValid`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Bo	Depending on the first string in the list `Items`, this calls one of the other `Check.mandFormat` subroutines.
`CheckMove mandFormat`	***Parameters:*** `Items` (li strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are three eleme in the `Items` list 2) The second and third ite are strings containing integers Otherwise it returns False.
`CheckStandardC mandFormat`	***Parameters:*** `Items` (list of str) ***Return value(s):*** Boolean	Returns True if the following condition are met: 1) There are two elements in th `tems` list 2) The second item is a string co ning an integer Otherwise it returns False.
`CheckUpgradeComman rmat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions a met: 1) There are three elements in the `I s` list 2) The third item is a string containing a integer 3) The second item is either less or pb (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
		game by calling `SetUpDefaultGame` and then calling `PlayGame` to play the game.
`PlayGame`	***Parameters:*** `Player1` (`Player` object), `Player2` (`Player` o… `Grid` (`HexGrid` objec… ***Return value(s):*** -	This alternates between Player 1 and Player 2 and always ensures that Player 2 has …al number of turns even if Player 1 has just killed thei… n. For each player's turn, the boa… s printed out and then three commands are read in and valid…d (by calling `CheckCommandIsValid`) an… en executed (by calling `ExecuteCommand` on the `HexG…d` object for the game – stored in the variable `Grid`). After each command, the `UpdateL…ber`, `UpdateFuel` and `RemoveTileFromSupply` (if …essary) methods are called on the `Player` object for that p…r's turn. Once the commands have been execute… ieces are destroyed, VPs are assigned and a check is made to see whether the game is over. Before the gam… oves on to the next player's turn, the status of both players (…sources and VPs) is displayed. If the game is over and Player 2 has played, then `DisplayEndMessages` is called to display the final scores and the winner.
`SetUpDefaultGame`	***Parameters:*** -	Initialises the game with the default values and board size as

Classes



`BaronPiece` (inherits from `Piece`)

Name	Data	Description
`__init__` (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`

Name	Data	Description
`__init__` (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 initialises the private attribute `ListPositionOfTile`

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:*** - ***Return value(s):*** `Line` (string)	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).
`CreateOddLine` (private)	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the corre ring of an odd line of the `HexGrid` (as displayed at the start o ch turn when playing the game).
`CreateTopLine` (priva	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the top line o e `HexGrid` (as displayed at the start of each turn when pla ing the game).
`DestroyPiecesAndCo VPs` (public)	***Parameters:*** - ***Return value(s):*** `BaronDestroyed` (Boolean), `Player1VPs` (int), `Player2VPs` (int)	Loops through every `Tile` in the `HexGri` nd checks for any `Pieces` that need to be destroyed by coun the number of connections for each `Piece` and comparing it the number of connections needed to destroy the `Piece` in question. For each piece that is destroyed, track the VPs for the relevant player and also whether their Baron was destroyed or not. Finally, it then removes any `Pieces` from the `HexGrid` that were destroyed.

Name	Data	Description
`ExecuteCommandInTile` (private)	***Parameters:*** `Items` (list of strings) ***Return value(s):*** `Status` (Boolean), `Fuel` (int), `Lumber` (int)	Checks whether there is a `Piece` belonging to the player in the `Tile` specified as the second string in `Items`, if not then False, 0 and 0 are returned. It then uses `HasMethod` to determine whether the `Piece` in the `Tile` has a `saw` or `dig` command as specified by the first string in the `Items` list and calls that method. If it was a dig and more than 5 fuel was returned then it sets the terrain of the `Tile` to a field using the `SetTerrain` method. The method then returns True and the `Fuel` or `Lumber` gained from digging or sawing or it returns False and 0,0 if the command could not be executed.
`ExecuteMoveCommand` (private)	***Parameters:*** `Items` (list of strings), `FuelAvailable` (int) ***Return value(s):*** `FuelCost` (int)	Checks whether there is a `Piece` belonging to the player in the `Tile` specified as the second string in `Items` and that the `Tile` specified in the third string of `Items` is empty, if not then -1 is returned straight away. Otherwise, it then checks that the move is allowed by calling the `CheckMoveIsValid` method on the `Piece` in 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), `LumberAvailable` (int),	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:*** `P1Turn` (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
`__init__` (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
`__init__` (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
`__init__` (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
`__init__` (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 • `Fuel` from parameter • `Lumber` from parameter • `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):*** 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

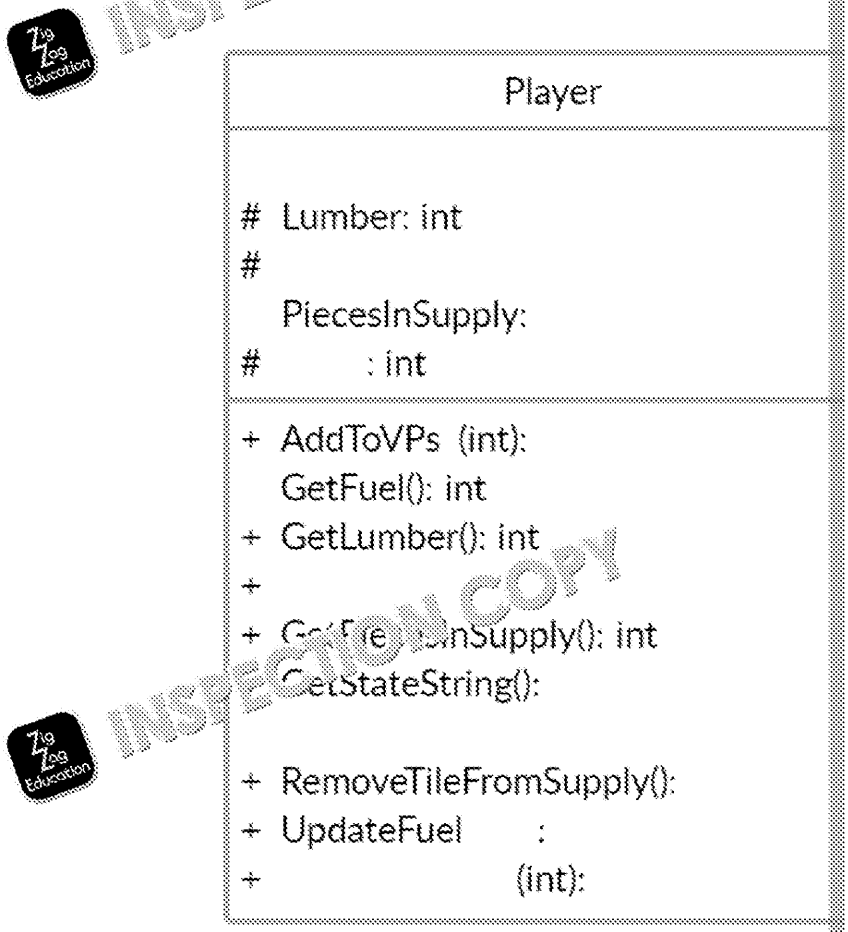
Name	Data	Description
`__init__` (constructor)	***Parameters:*** `xcoord` (int), `ycoord` (int), `zcoord` (int) ***Return value(s):*** -	Initialises the following protected attributes: • x from parameter `xc` • y from parameter `yc` • z from parameter `zcoor` • `Terrain` to " " (a string containing a single space) • `PieceInTile` to None • `Neighbours` to an empty li
`AddToNeighbour` (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`.

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.

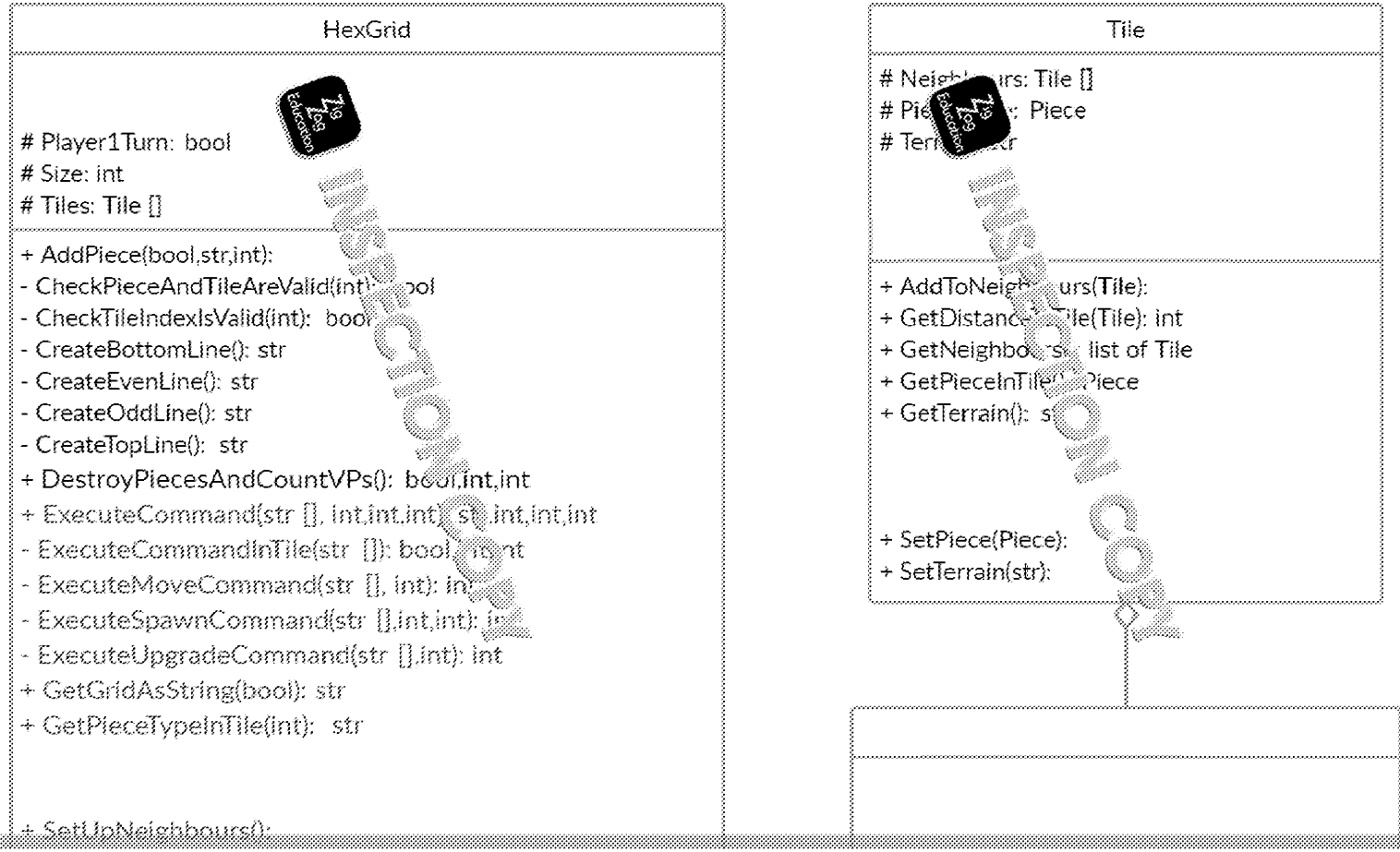


Task 2

(5 marks)

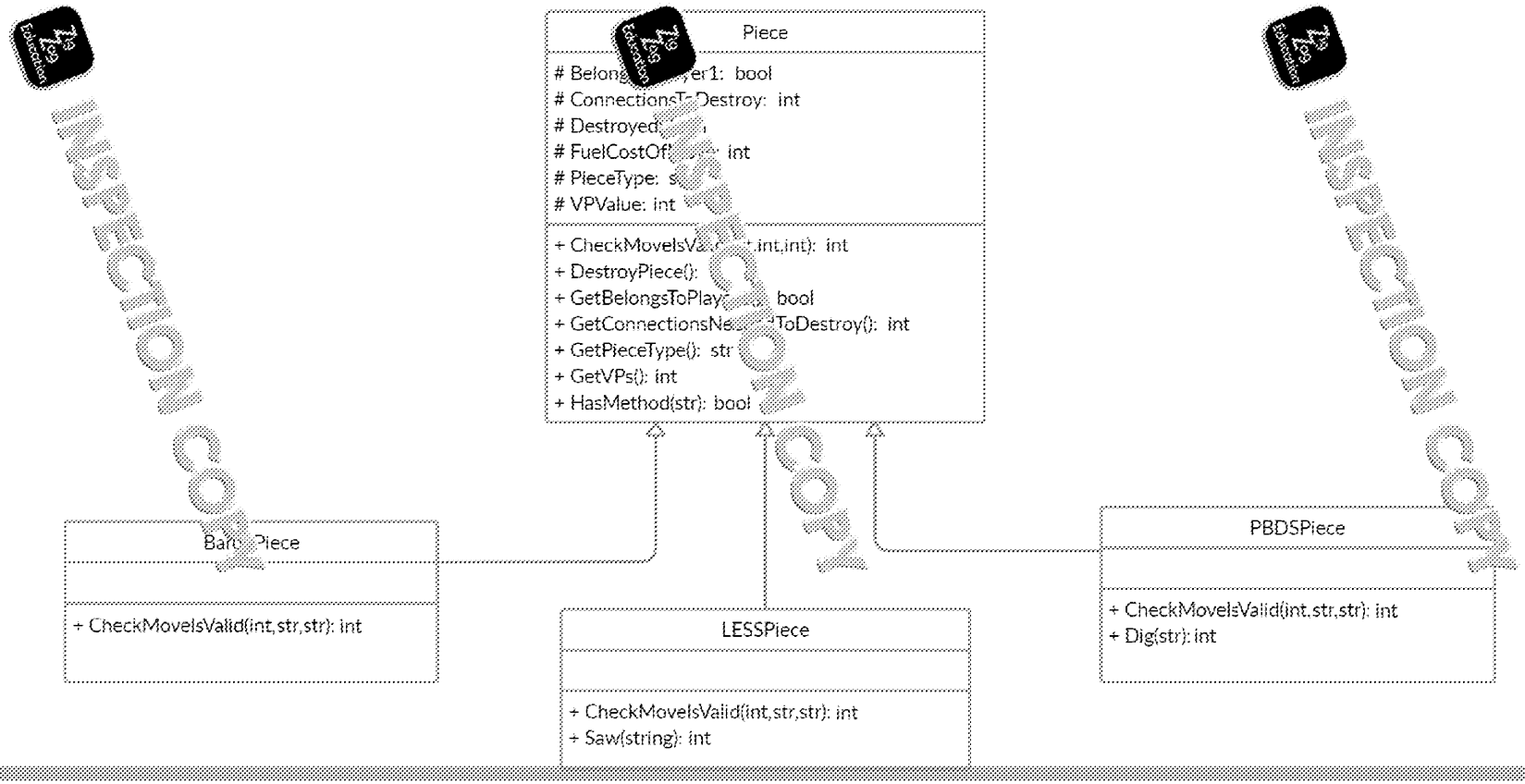
A partially-complete UML class diagram is shown.

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



Task 3 (4 marks)

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



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] the actual player names wh
at the start of each turn; it just pri[nts] 'Player One' and 'Player Two'

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

.....

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

.....

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

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

.....

.....

.....

3 This q[uestion] r[efe]rs to the `HasMethod` method within the `Piece` cla
`Execu[te]CommandInTile` method in the `HexGrid` class.

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

.....

.....

.....

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

a) Explain how this calculation is 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... Piece, LESSPiece and PBDSPiece classes all inherit

a) Explain what is meant by 'inheritance'.

.....

.....

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

.....

.....

6 This question refers to the BaronPiece class a... the Piece class.

The BaronPiece class overrides the public method CheckMoveIs

a) Explain what is mea... '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 S... Neighbours method in the HexGrid class

.....

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

.....

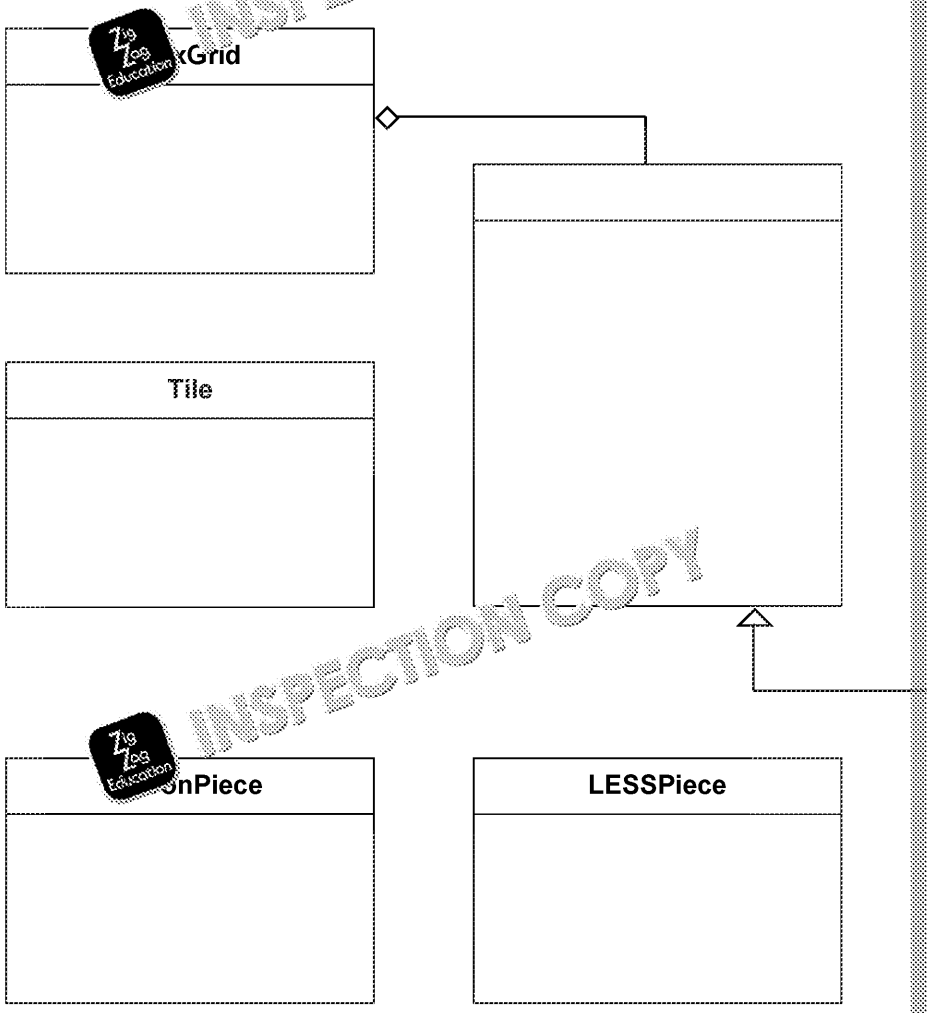
8 The skeleton program uses a Comma Separated Values (CSV) file as
saved game. Explain why a CSV file is suitable for cross-platform appl

.....

.....

9 Class diagrams can be represented using Unified Modeling Language

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



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

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

.....

.....

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

.....

.....

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

When a 'dig' command is issued by a player, it is possible that the tile 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

.....

.....

.....

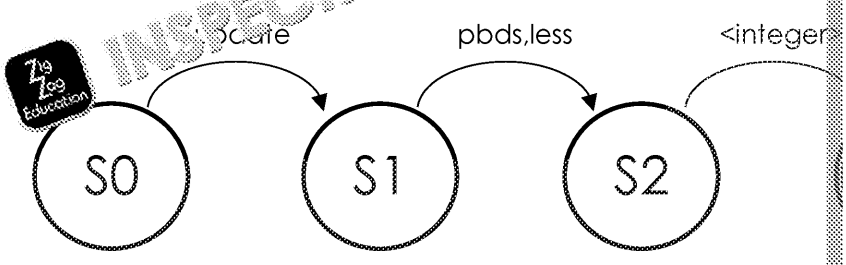
b) Why is polymorphism not possible for private attributes?

.....

.....

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 `HexG[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 `BaronPiece` 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.

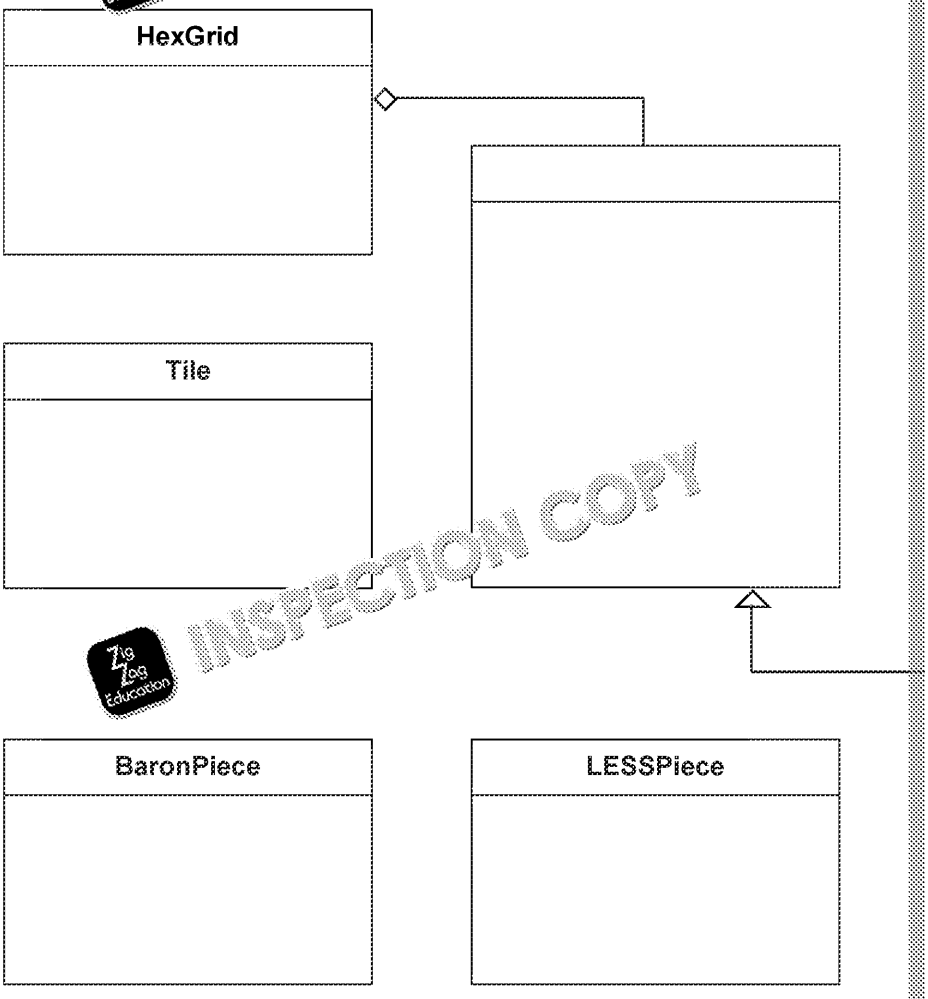
a) Examine the `SetUpNeighbours` method in the `HexGrid` class

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 refer[illegible] method of the `PBDSPiece` class and `Execu[illegible]omainTile` 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.

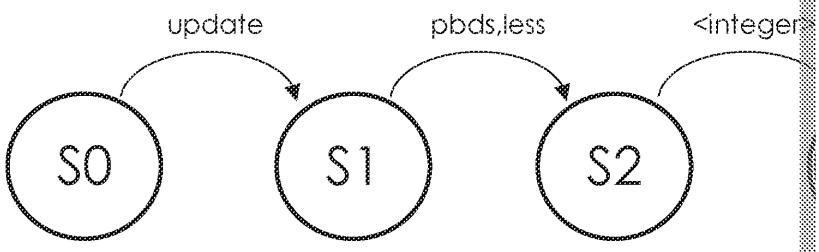
For a piece to be destroyed in the game, it needs to have two connect neighbours (i.e. pieces in two of the tiles that share a side with it).

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 reference to how it meets the spawning a new Serf (it must be on an empty square adjacent to that p

HEX BARON – Programming Tasks

Task 1

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

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

Test that the change you have made works:

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

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

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

Evidence that you need to provide:

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

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

Task 2

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

At the moment, the game sets the default names of the players to 'Player
`SetUpDefaultGame` subroutine. Change this subroutine to prompt for
and then use the values entered to construct the 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[illegible]ly chain and mov
losing the game as the number of pieces that you can make is limited. Th
that if a piece is destroyed then you gain a piece in your supply chain.

- Add a method to the `Player` class that will allow the `PiecesIn`
be [illegible]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", [illegible])
Grid.AddPiece(False, "[illegible]", [illegible]31)
Grid.AddPiece(Fal[illegible]"[illegible]rf", 15)
Grid.AddPie[illegible]"Serf", 23)
```

Test that t[illegible]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 [illegible]r the amended subroutine `Pl`
`DestroyPiece[illegible]CountVPs` method of the `HexGrid` class
`Play[illegible]` c[illegible]

b. SCR[illegible]N CAPTURE(S) showing the required test.

Task 4

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

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

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

```
Grid.AddPiece(True, "Baron", 0)
Grid.AddPiece(True, "Serf", 8)
Grid.AddPiece(True, "Serf", 24)
Grid.AddPiece(True, "Serf", 5)
Grid.AddPiece(True, "Serf", 2)
Grid.AddPiece(True, "Serf", 3)
Grid.AddPiece(False, "Baron", 31)
Grid.AddPiece(False, "Serf", 23)
```

Test that the changes you have made work:

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

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

Evidence that you need to provide:

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

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

Task 5

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

Add an additional command to the game which will pri ut the hex grid a hex; these should be in the top half of each he and the board should be t display. The new command, "hexes" sh l not count as one of the three the results should be displayed im ately upon entry, i.e. the grid shoul numbers displayed as s n 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 ge 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 Pl[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 `Execute...InTile` 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 ...
Player Two	• ... 27 • 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]ov[illegible] 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] the required test.

Task 9

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

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

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

```
Grid.AddPiece(True, "Baron", 0)
Grid.AddPiece(True, "Serf", 15)
Grid.AddPiece(True, "Serf", 27)
Grid.AddPiece(True, "Serf", 25)
Grid.AddPiece(False, "Baron", 19)
Grid.AddPiece(False, "Serf", 8)
Grid.AddPiece(False, "Serf", 2)
```

Test that the changes you have made work:

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

Player One	• move 25 21 • move 21 18 • move 18
Player Two	• move 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 for their three chances.
- 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

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

b) Show 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 **`Chec sp a CommandFormat`** **`CheckCommandIsVali t `** sure that the new spawn bom
- Modify the priv e **`ExecuteSpawnCommand`** so that it wi the r number or report an error if they don't have enough lu
- Cre 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 t n.
- Modify the **`SetUpDefaultGam`** su ine so that both players

```
Player1 = Play ("Player One", 0, 30, 30, 5)
Player2 = Play ("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]`xt`[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 to 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 current 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 choice of 6, 8 (the current size) o displayed for them to check - this is just a reuse of the code from task 5 ar task 5 will be the starting point for this task), the location of a Baron for eac of any additional pieces that they wish to add. They should be able to defin and peat bogs and define the starting resources (including supply chain) fo 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 a piece for Player One (S=Serf, L=LESS, P=PBDS) or D
- Enter a piece to add for Player One (S=Serf,L=LESS,P=PBDS) or D fo
- What name would you like to save the file as (please include the ext
- Would you like to play the game that you've created and saved?

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

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

Line 1:	text for Player One's name, Player One's VPs, Player One's fue Player One's supply chain
Line 2:	text for Player Two's name, Player Two's VPs, Player Two's fue Player Two's supply chain
Line 3:	board size
Line 4:	peat bog 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] follow player
3. Mak[illegible]e g[illegible]ore user-friendly so that each time any command
des[illegible]n is give, e.g. 5 lumber deducted for upgrading PBDS in
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 don
6. Introduce an assassin piece that can use a kill command once per
serf, the kill command will kill the Baron if it is next to it.
7. Introduce a poison piece command so that each player can poison
poisoned piece is killed the victim gains VPs instead of the killer.
8. Introduce a random game command from the main menu where th
resources are randomised (but fairly with suitable rules for distribut
9. Introduce a new blocking/wall piece that requ[illegible] 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]n[illegible] the coordinates system is removed and
by th[illegible]un[illegible]exes that you need to move through. The IDs sh
he[illegible]on[illegible]ng out / maintaining which tiles are neighbours.
19. Change the game rules so that the game is terminated immediatel
At the moment if Player 1 destroys Player 2's Baron, they still get t
20. Change the victory rules at the end of the game so that once the g
VP for all of their remaining pieces according to the VP value for th

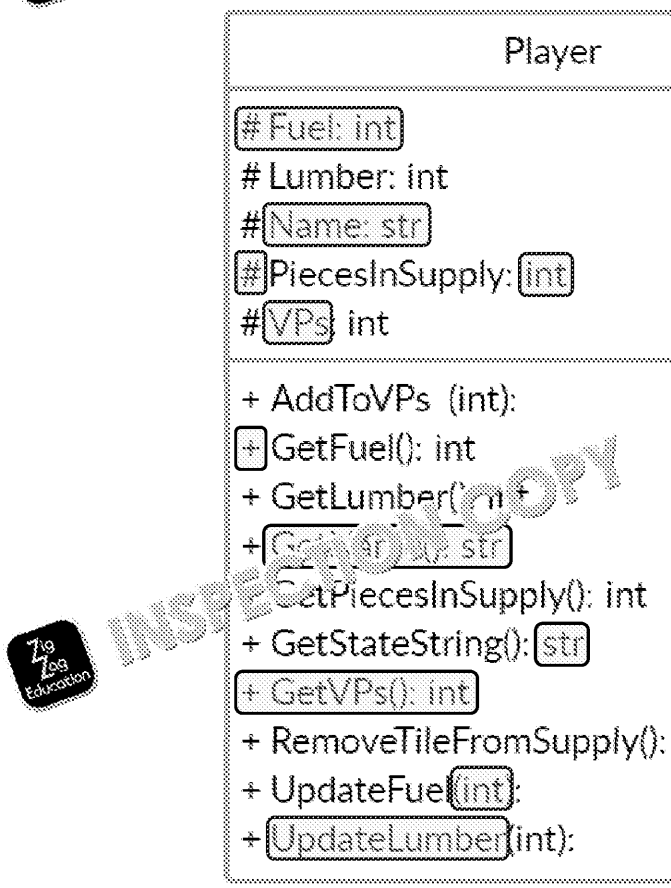
HEX BARON – Class Diagram Tasks (MS)

Task 1

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

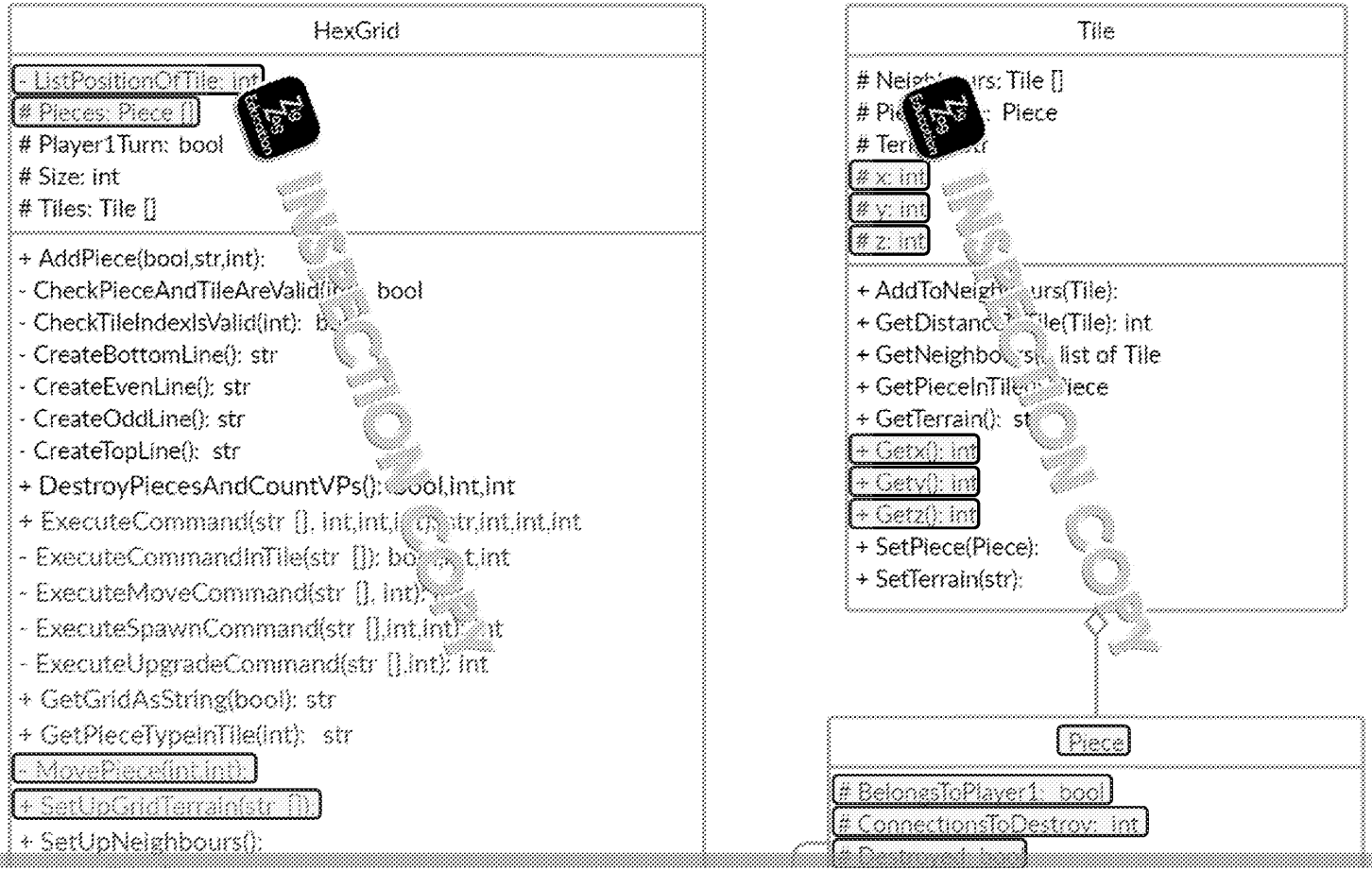
A. Use of void where there is no return value or there are no parameters

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 means that it has overridden the meth
  (although some UML diagram tools use a <<override>> stereotype o

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

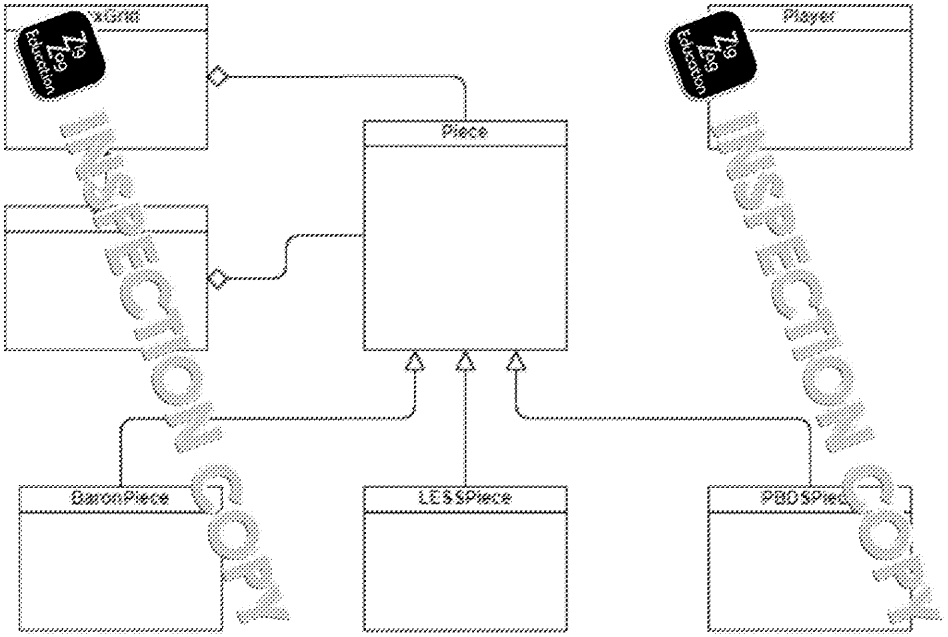
HEX BARON

Class Diagram

HEX BARON – Theory Questions (MS)

(Total Marks: 50)

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

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

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

HEX BARON – Programming Tasks (MS)

Task 1 (2 marks)

Coding:

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

For example:

```
Commands.append(input("Enter command: ").lower().strip())
```

A. Solutions which manually strip off as many spaces as there are, but no 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 passing through the names entered correctly to the constructor for the relevant Player objects.

For example:

```
Player1 = Player(input("Player One, please enter your name: "), 0, 10, 10, 5)
Player2 = Player(input("Player Two, please enter your name: "), 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 state 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 calling the AddPiecesInSupply method for the correct player so that it adds one piece to their supply.

For example:

AddPiecesInSupply method added to the Player class:

```
def AddPiecesInSupply(self, n):
    self._PiecesInSupply += n
```

Both calls to DestroyPiecesAndCountVPs:

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

Adding parameters to the DestroyPiecesAndCountVPs method:

```
def DestroyPiecesAndCountVPs(self,Player1,Player2):
```

Modifications to the DestroyPiecesAndCountVPs method:

```
ListOfFilesContainingDestroyedPieces.append(T)
if ThePiece.GetBelongsToPlayer1():
    Player1.AddPiecesInSupply(1)
```

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:

```
Player One state your three commands, pressing enter after each one.
Enter command: move 13 10
Enter comman ove 10 14
Enter comma ve 14 19
Command exec
Command execute
Command execute
Player One curr state - VPs: 2   Pieces in supply: 6   L er: 10   Fuel: 7
Player Two curre state - VPs: 2   Pieces in supply: 7   Li r: 10   Fuel: 10
Press Enter to co nue...

 __   __   __
/ B\__/# \__/# \__/ \__
\__/~ \__/~ \__/   \_/ S\
/   \__/~ \__/   \__/#   _/
\__/# \__/   \__/   \__/ \
/   \__/   \__/# \__/# \__/
\__/# \__/# \__/~ \__/   \
/~ \__/~ \__/~ \__/   \_/
\__/# \__/   \__/# \__/ B
   \__/   \__/   \__/   \__/

Player Two state your three commands, pressing enter after each one.
```

Task 4 (4 marks)

Coding:

- 1 mark for counting the number of pieces on the board for the current player correctly.
- 1 mark for printing out the message 'Spawn attempted to exceed max pieces' when the number of pieces for the player is 6 or more (award even if the first mark calculates 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 NewPiece = Piece(self._Player1Turn) line of code (or anywhere before it in the method).

For example:

```
PlayerPieces = sum(1 for PlayerPiece in self._Pieces if
PlayerPiece.GetBelongsToPlayer1() == self._Player1Turn])
if PlayerPieces >=
  print("Spawn attempted to exceed max pieces.")
  return -1
```

A. Solutions which use a loop instead of a list comprehension.

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 and that the 'hexes' command did not count as one of the three commands (no matter how many times they enter 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 (only if the format is correct).
- 1 mark for creating the string with correct index numbers for the even lines (even if the format is off).
- 1 mark for creating the string with correct index numbers for the odd lines (even if the format is off).

For example:

Modifications to the PlayGame subroutine:

```
for Count in range (0,4):
    Command = input("Enter command: ").lower()
    while Command == "hexes":
        print(Grid.GetGridAsIndices())
        Command = input("Enter command: ").lower()
    Commands.append(Command)
```

Code for new GetGridAsIndices method:

```
Index = 0
self.__ListPositionOfTile = 0
GridString = self._CreateTopLine()
GridLine,Index = self.__CreateEvenLineIndices(Index,True)
GridString += GridLine
```

```
def __CreateOddLineIndices(self, Index):
  Line = ""
  for count in range (1, self._Size // 2 + 1):
    if count > 1 and count < self._Size // 2:
      Line += "\\__/"
      self._ListPositionOfTile += 1
      Line += "{:<2}".format(Index)
      Index += 1
    elif count != 1:
      Line += "\__/" + "{:<2}".format(Index)
      Index += 1
  Line += "\\__/"
  self._ListPositionOfTile += 1
  if self._ListPositionOfTile < len(self._Tiles):
    Line += "{:<2}".format(Index) + "\\" + os.linesep
    Index += 1
  else:
    Line += "\\" + os.linesep
  return Line, Index
```

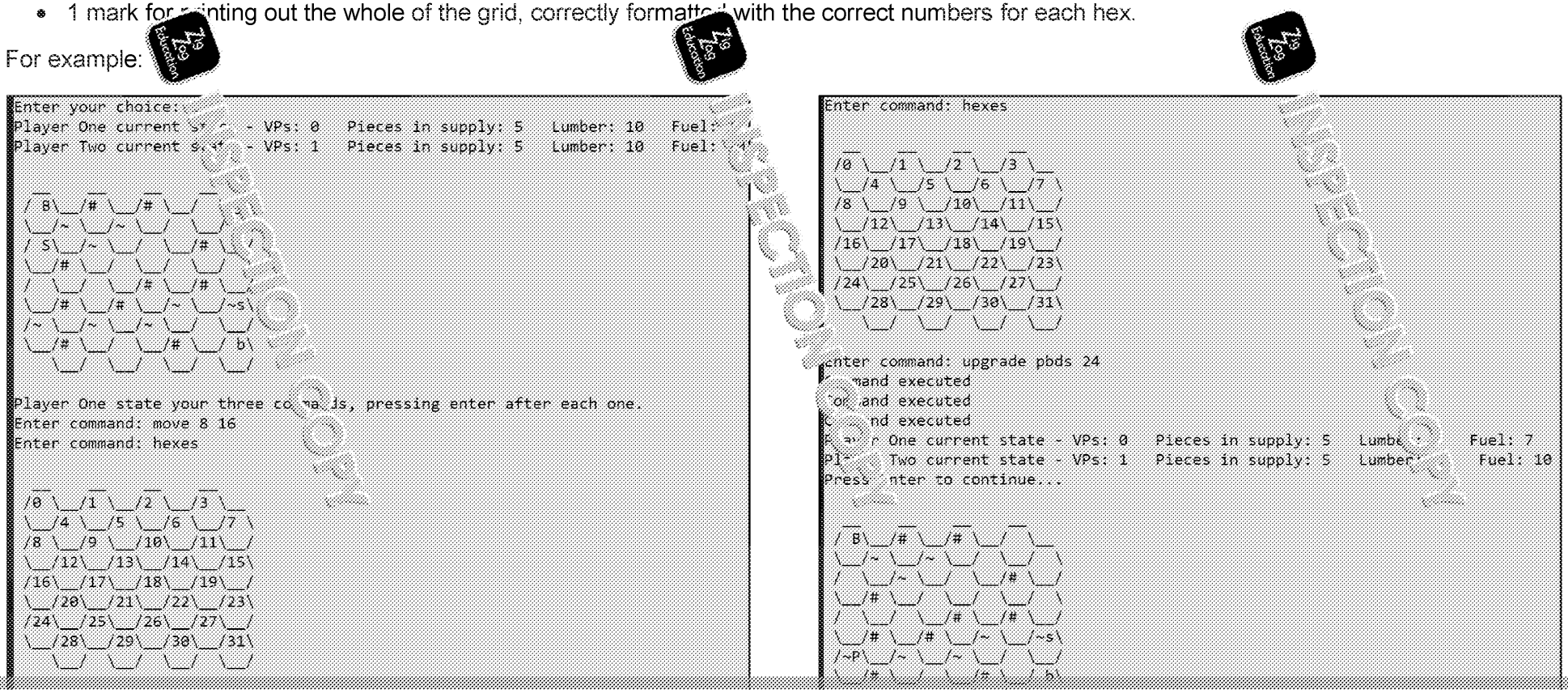
Code for the new CreateEvenLineIndices method:

```
def __CreateEvenLineIndices(self, Index, FirstEvenLine):
  Line = " /" + "{:<2}".format(Index)
  Index += 1
  for count in range (1, self._Size // 2):
    self._ListPositionOfTile += 1
    Line += "\\__/" + "{:<2}".format(Index)
    Index += 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 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:

```
ValidMoveCommands = 0
for C in Commands:
    Items = C.split(",")
    ValidCommand = CheckCommandIsValid(Items)
    if not ValidCommand:
        print("Invalid command")
    else:
        if Items[0] == "move":
            ValidMoveCommands = 1
            if ValidMoveCommands == 3:
                if Player1Turn:
                    Player1.UpdateFuel(1)
                else:
                    Player2.UpdateFuel(1)
    FuelChange = 0
```

A. Solutions which iterate over the Commands list and count the number of move commands as long as they take account of whether or not they are valid.

Task 7 (7 marks)

Coding:

- 1 mark for correctly removing the code from ExecuteCommandInTile.
- 1 mark for initialising variables to track the valid dig and saw commands.
- 1 mark for 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:

```
    elif Items[0] == "S":
        Fuel += Method(self.Tiles[TileToUse].GetTerrain())
    return True, Fuel, Number
```

Modifications to Dig:

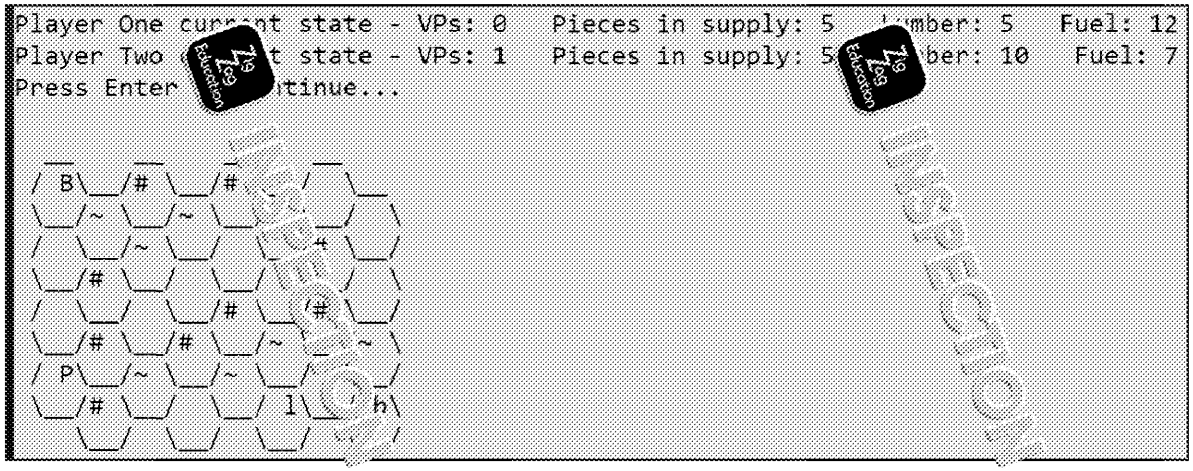
```
def Dig(self, Terrain):
    if Terrain == "~":
        return 1
    else:
        return 0
```

Code for the modified PlayGame subroutine:

```
DigSawLocation = Items[1]
if Items[0] == 'saw' and Items[1] == DigSawLocation:
    ValidSawCommands += 1
elif Items[0] == 'dig' and Items[1] == DigSawLocation:
    ValidDigCommands += 1
FuelChange = 0
LumberChange = 0
SupplyChange = 0
if Player1Turn:
    SummaryOfResult = Grid.ExecuteCommand
    (Items, Player1.GetFuel(), Player1.GetLumber(), Player1.GetPiecesInSupply())
    Player1.UpdateLumber(LumberChange)
    Player1.UpdateFuel(FuelChange)
    if SupplyChange == 1:
        Player1.RemoveTileFromSupply()
else:
    SummaryOfResult = Grid.ExecuteCommand
    (Items, Player2.GetFuel(), Player2.GetLumber(), Player2.GetPiecesInSupply())
    Player2.UpdateLumber(LumberChange)
    Player2.UpdateFuel(FuelChange)
    if SupplyChange == 1:
        Player2.RemoveTileFromSupply()
if ValidSawCommands == 3:
    Player1Turn =
    Player1.UpdateLumber(2)
else:
    Player2.UpdateLumber(2)
Grid.MakeField(int(DigSawLocation))
```

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 PBDS or a LESS.
- 1 mark for correctly converting the piece to a Serf and updating the pieces list.

For example:

Code for modified CheckCommandIsValid subroutine:

```
elif Items[0] == "downgrade":
    return CheckDowngradeCommandFormat(Items)
```

Code for new CheckDowngradeCommandFormat subroutine:

```
def CheckDowngradeCommandFormat(Items):
    if len(Items) == 2:
        try:
            Result = int(Items[1])
        except:
            return False
```

Code for new ExecuteDowngradeCommand method:

```
def __ExecuteDowngradeCommand(self, Items):
  TileToUse = int(Items[1])
  if not self.__CheckPieceAndTileAreValid(TileToUse) :
    return -1
  else:
    ThePiece = self._Tiles[TileToUse].GetPieceInTile()
    if ThePiece.GetPieceType().upper() not in ["P","L"]:
      return -1
    ThePiece.DestroyPiece()
    ThePiece = Piece(self._Player1Turn)
    self._Pieces.append(ThePiece)
    self._Tiles[TileToUse].SetPiece(ThePiece)
    return -1
```

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 system 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:

```
super(BaronPiece, self).__init__(Player1)
self._ConnectionsToDestroy = 3
self._PieceType = "B"
```

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 outputting 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 pieces.
- 1 mark for correctly removing the piece from the board when it is salvaged.

For example:

Code for modified CheckCommandIsValid subroutine:

```
elif Items[0] in ["big", "saw", "spawn", "salvage"]:
    return CheckStandardCommandFormat(Items)
```

Code for modified ExecuteCommand method:

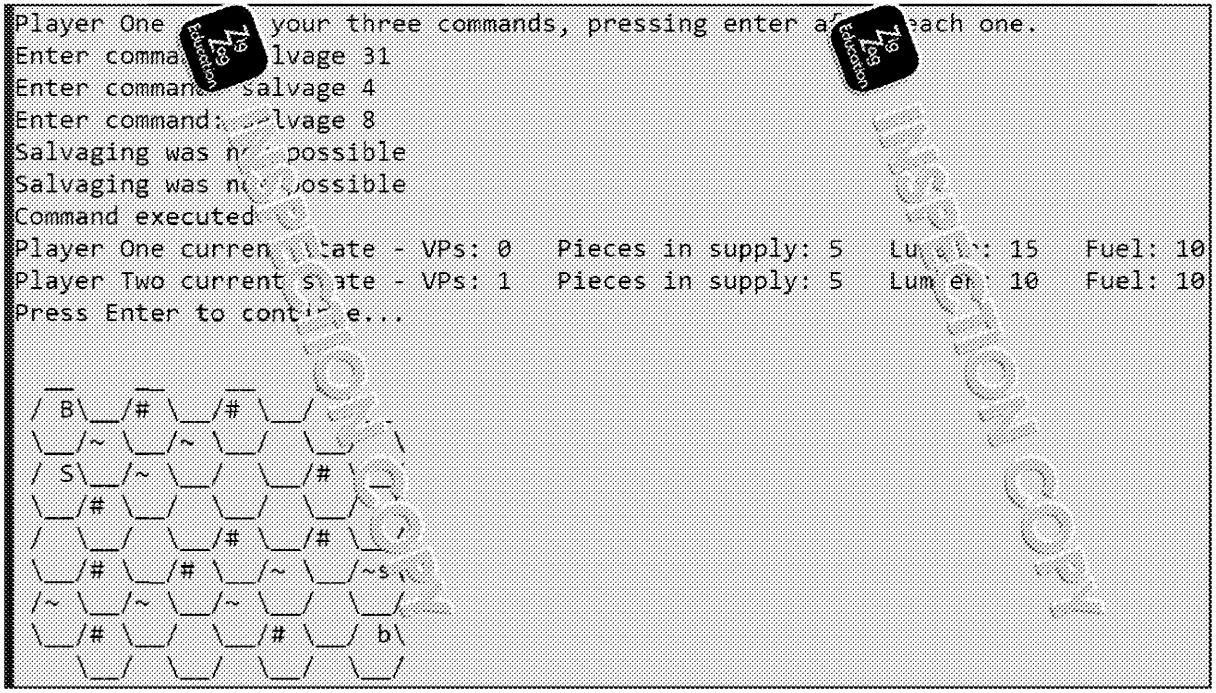
```
elif Items[0] == "salvage":
    LumberCost = self.__ExecuteSalvageCommand(Items)
    if LumberCost == 0:
        return "Salvaging was not possible", FuelChange, LumberChange, SupplyChange
    LumberChange = -LumberCost
    SupplyChange = -1
```

Code for new ExecuteSalvageCommand method:

```
def __ExecuteSalvageCommand(self, Items):
    TileToUse = int(Items[1])
    if not self.__CheckPieceAndTileAreValid(TileToUse):
        return 0
```

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 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 resetting 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:

```
def __init__(self, N, V, F, L, T):
    self._Name = N
    self._VPs = V
    self._Fuel = F
    self._Lumber = L
    self._PiecesInSupply = T
    self.ResetChances()
```

```
def PlayGame(Player1, Player2, Grid):
    GameOver = False
    PlayerTurn = False
    Commands = []
    Player1.ResetChances()
    Player2.ResetChances()
    print("Player 1 starts")
    print("Two current state ... " + Player1.GetStateString())
    print("Player 2 starts ... " + Player2.GetStateString())
    while not GameOver and PlayerTurn:
        print(GridAsString(PlayerTurn))
        if PlayerTurn:
            Player = Player1
        else:
            Player = Player2
        print(Player.GetName(), "you have ", Player.GetChances(), "chances remaining.")
        print(Player.GetName() + " state your three commands, press enter after each one.")
        for Count in range(1, 4):
            Commands.append(input("Enter command " + str(Count)+": ").lower())
        for CommandNo in range(0, len(Commands)):
            Command = Commands[CommandNo]
            Items = Command.split(" ")
            ValidCommand = CheckCommandIsValid(Items)
            while not ValidCommand and Player.GetChances() > 0:
                print("Command number ",CommandNo+1," is an invalid command")
                Player.GetChances() > 0:
                    Player.DeductChance()
```


Zig Zag Education

```
    print("Player One current state - " + Player1.GetStateString())
    print("Player Two current state - " + Player2.GetStateString())
    input("Press Enter to continue...")
print(Grid.GetGridAsString(PlayerITurn))
DisplayEndMessages(Player1, Player2)
```

A. Solution which retain the use of Player1 and Player2 throughout.

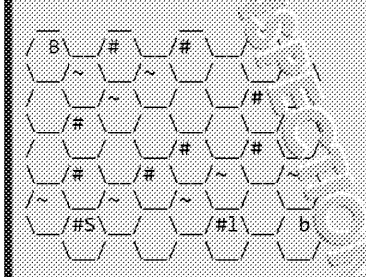
Testing:

- 1 mark for showing the commands entered correctly and the commands and error messages ... 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 2 chances remaining).
- 1 mark for displaying the final error message correctly and not letting Player One have a fourth chance.

For example:

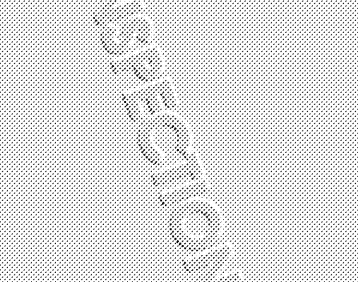
```
Player One you have 3 chances remaining.
Player One state your three commands, pressing enter after each one.
Enter command 1: move
Enter command 2: move 8 8
Enter command 3: upgrade less 28
Command number 1 is an invalid command
You have 2 chances remaining.
Enter new command 1: move 8 16
Command 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
You have 0 chances remaining.
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: 10   Fuel: 10
Press Enter to continue...

 __   __   __   __
/ B\__/# \__/# \__/  \__
\__/~ \__/~ \__/  \__/  \
/  \__/~ \__/  \__/# \__/
```

```
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...
```



```
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
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 6   Fuel: 7
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 5   Fuel: 7
Press Enter to continue...
```

```
Player Two you have 3 chances 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...
```



```
Player One you have 0 chances 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
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 6   Fuel: 7
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 5   Fuel: 7
Press Enter to continue...
```

Task 12 (13 marks)

Coding:

- 1 mark for modifying the CheckCommandIsValid method to process the new format for the spawn bomb variant of the spawn command.
- 1 mark for correctly checking the format of the new spawn command (ideally using the CheckSpawnCommandFormat method they wrote).
- 1 mark for modifying the spawn command in ExecuteSpawnCommand so that it will correctly spawn a bomb and charge 10 lumber (do not award if they don't check for the availability of the lumber).
- 1 mark for having a new BombPiece class and setting the FuelAmountOfMove 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 prior to two 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 (awarded 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:

```
elif Items[0] in ["dig", "saw"]:
    return CheckStandardCommandFormat(Items)
elif Items[0] == "spawn":
    return CheckSpawnCommandFormat(Items)
```

Code for new CheckSpawnCommandFormat subroutine:

```
def __ExecuteSpawnCommand(self, Items, LumberAvailable, PiecesInSupply):
    if len(Items) == 2:
        TileToUse = int(Items[1])
        SpawnCost = 3
    else:
        TileToUse = int(Items[2])
        SpawnCost = 10
    if PiecesInSupply < 1 or LumberAvailable < SpawnCost or
    self.__CheckTileIndexIsValid(TileToUse):
        return -1
    ThePiece = self._Tiles[TileToUse].GetPieceInTile()
    if ThePiece is not None:
        return -1
    OwnBaronIsNeighbour = False
    ListOfNeighbours = self._Tiles[TileToUse].GetNeighbours()
    for N in ListOfNeighbours:
        ThePiece = N.GetPieceInTile()
        if ThePiece is not None and ThePiece.GetPieceType() == "B" or
        self._PlayerITurn and ThePiece.GetPieceType() == "b":
            OwnBaronIsNeighbour = True
            break
    if not OwnBaronIsNeighbour:
        return -1
    Items[1] == "Rook"
    NewPiece = BasePiece(self._PlayerITurn)
```

Code for new BombPiece class:

```
class BombPiece(Piece):
    def __init__(self, Player1):
        super(BombPiece, self).__init__(Player1)
        self._FuelCostOfMove = 2
        self._MovesBeforePrimed = 5
        self._PieceType = "X"
        self._VPValue = 5

    def CheckMoveIsValid(self, DistanceBetweenTiles, StartTerrain, EndTerrain):
        if DistanceBetweenTiles == 1:
            if self._MovesBeforePrimed > 0:
                self._MovesBeforePrimed -= 1
                return self._FuelCostOfMove
        return -1

    def DestroyPiece(self, TileLocation):
        if not self._Destroyed:
            self._ConnectionsToDestroy = 0
            self._MovesBeforePrimed = -1 # so the bomb is no longer primed
            self._Destroyed = True
            Neighbours = TileLocation.GetNeighbours()
            for Neighbour in Neighbours:
                TilePiece = Neighbour.GetPieceInTile()
                if TilePiece is not None:
```

```
            if TilePiece.GetBelongsToPlayer1():
                Player2VPs += TilePiece.GetVPs()
            else:
                Player1VPs += TilePiece.GetVPs()
            if TilePiece.GetPieceType().upper() == "B":
                BaronDestroyed = True
            TilePiece.DestroyPiece()
            Neighbour.SetPiece(None)
        self._Destroyed = True
    return BaronDestroyed, Player1VPs, Player2VPs

def Primed(self):
    return self._MovesBeforePrimed == 0
```

Code for modified `DestroyPiecesAndCountVPs` method:

```
def DestroyPiecesAndCountVPs(self):
    BaronDestroyed = False
    Player1VPs = 0
    Player2VPs = 0
    ListOfTilesContainingDestroyedPieces = []
    for T in self._Tiles:
        if T.GetPieceInTile() is not None:
            ListOfNeighbours = T.GetNeighbours()
            NoOfConnections = 0
            for N in ListOfNeighbours:
                if N.GetPieceInTile() is not None:
                    NoOfConnections += 1
```

```
        ListOfTilesContainingDestroyedPieces.append(T)
        if ThePiece.GetBelongsToPlayer1():
          Player2VPs += ThePiece.GetVPs()
        else:
          Player1VPs += ThePiece.GetVPs()
    for T in ListOfTilesContainingDestroyedPieces:
      T.SetPiece(None)
    return BarsDestroyed, Player1VPs, Player2VPs
```

Code for modified GetPieceTypeInTile method:

```
  def GetPieceTypeInTile(self, ID):
    ThePiece = self.Tiles[ID].GetPieceInTile()
    if ThePiece is None:
      return " "
    else:
      PieceType = ThePiece.GetPieceType()
      if PieceType == "S":
        PieceType = "S"
      elif PieceType == "s":
        PieceType = "s"
      return PieceType
```

Code for modified ExecuteMoveCommand method:

```
  def __ExecuteMoveCommand(self, Items, FuelAvailable):
    StartID = int(Items[1])
```

```
        self.__MovePiece(EndID, StartID)
    Neighbours = self._Tiles[EndID].GetNeighbours()
    for Neighbour in Neighbours:
      PieceInTile = Neighbour.GetPieceInTile()
      if PieceInTile is not None:
        if PieceInTile.GetPieceType().upper() == "X" and PieceInTile.Primed():
          PieceInTile.DestroyPiece(Neighbour)
    return FuelUsed
```

Code for modified Piece class:

```
  def Explode(self):
    self._ConnectionsToDestroy = 0
```

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

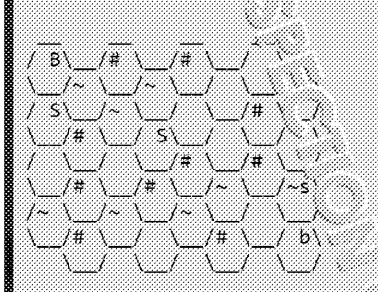
R. Solutions that break OOP by accessing private and protected attributes in illegal or unreasonable ways, or solutions that unnecessarily provide direct access to things when they could be solved in a more suitable way using the OOP hierarchy, encapsulation and polymorphism.

Testing:

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

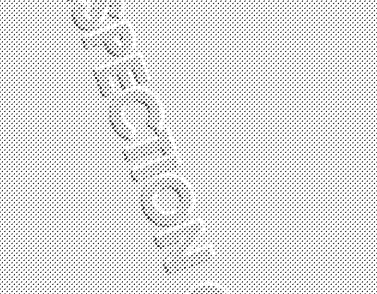
For example – Game One:

```
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: mo   9 13
Command execut
Command execute
Command executed
Player One current   te - VPs: 0   Pieces in supply: 4   Lumber: 20   Fu   26
Player Two current   re - VPs: 1   Pieces in supply: 5   Lumber: 30   Fu   30
Press Enter to cont   ...

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

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...

 __   __   __   __
/ B\__/# \__/# \__/ \__
```

```
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   pply: 4   Lumber: 20   Fuel: 20
Player Two current state - VPs: 6   Pieces in   ply: 5   Lumber: 26   Fuel: 28
Press Enter to continue...

 __   __   __   __
/ B\__/# \__/# \__/ \__
\__/~ \__/~ \__/ \__/ \
/ S\__/~ \__/ \__/# \__/
\__/# \__/ \__/ \__/ \
/ \__/ \__/# \__/# \__/
\__/# \__/# \__/~ \__/~ \
/~ \__/~ \__/~ \__/ \__/
\__/# \__/ \__/# \__/ \
   \__/ \__/ \__/ \__/

Player Two state your three commands, pressing enter aft   each one.
Enter command: saw 19
   r command: saw 19
   r command: saw 19
Co   n'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...

 __   __   __   __
/ B\__/# \__/# \__/ \__
```

Task 13 (10 marks)

Coding:

- 1 mark for creating the WizardPiece class with a VP value of 3
- 1 mark for overriding the CheckMoveIsValid method to implement the teleport.
- 1 mark for having 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 move of 4 or more.

For example:

Code for the new WizardPiece class:

```
class WizardPiece(Piece):
    def __init__(self, Player1):
        super(WizardPiece, self).__init__(Player1)
        self._PieceType = "W"
        self._VPValue = 3
        self.ResetTeleport()

    def CheckMoveIsValid(self, DistanceBetweenTiles, StartTerrain, EndTerrain):
```

Code for modified CheckUpgradeCommandFormat subroutine:

```
def CheckUpgradeCommandFormat(Items):
  if len(Items) == 3:
    if Items[1].upper() not in ["LESS","PBDS","WIZ"]:
      return False
    try:
      Result = int(Items[2])
    except:
      return False
    return True
  return False
```

Code for modified Execute-UpgradeCommand method:

```
def __ExecuteUpgradeCommand(self, Items, LumberAvailable):
  TileToUse = int(Items[2])
  if not self.__CheckTileAndTileAreValid(TileToUse) or LumberAvailable < 5 or not (Items[1] in ["pbds","less","wiz"]):
    return -1
  else:
    ThePiece = self._Tile[TileToUse].GetPieceInTile()
    if ThePiece.GetPieceType().upper() != "S":
      return -1
    ThePiece.DestroyPiece()
    if Items[1] == "pbds":
      ThePiece = PBDSPiece(self._PlayerTurn)
    elif Items[1] == "less":
```

Code for new method ResetWizards in the HexGrid class:

```
def ResetWizards(self):
    for T in self._Tiles:
        PieceInTile = T.GetPieceInTile()
        if PieceInTile is not None:
            if PieceInTile.GetPieceType().upper() == "W":
                PieceInTile.ResetTeleport()
```

Code for new call to ResetWizards in the PlayGame subroutine:

```
    print(SummaryResult)
Grid.ResetWizards()
Commands.clear()
```

A. Solutions which use a loop instead of a list comprehension.

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:  3 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 another 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 updating 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 checking 3 lumber for the upgrade (and making sure it's available).
- 1 mark for checking whether the first command entered is a supply command and terminating the loop early if it is.

For example:

Code for new SCFWPiece class:

```
class SCFWPiece(Piece):
    def __init__(self, Player1):
        super(SCFWPiece, self).__init__(Player1)
        self._PieceType = "F"
        self._VPValue = 3

    def CheckMoveIsValid(self, DistanceBetweenTiles, StartTerrain, EndTerrain):
        return -1

    def Supply(self, FuelAvailable, LumberAvailable):
        if FuelAvailable >= 5 and LumberAvailable >= 5:
            return -1, -5, -5
```

```
        return False
    else:
        return ValidCommand
elif Items[0] == "upgrade":
    return self.__CheckUpgradeCommandFormat(Items)
return False
```

Code for the modified CheckUpgradeCommandFormat subroutine:

```
def CheckUpgradeCommandFormat(Items):
    if len(Items) == 3:
        if Items[1].lower() not in ["LESS","PBDS","SCFW"]:
            return False
        try:
            Result = int(Items[2])
        except:
            return False
        return True
    return False
```

Code for modified method ExecuteUpgradeCommand of the HexGrid class:

```
if not self.__CheckPieceAndTileAreValid(TileToUse) or LumberAvailable < 5 or Items[1] not in ["pbds","less","scfw"]:
    return -1
else:
    ...
    elif Items[1] == "less":
        ThePiece = LESSPiece(self._Player1Turn)
```

Code for new ExecuteSupplyCommand method:

```
def __ExecuteSupplyCommand(self, Items, FuelAvailable, LumberAvailable):
  TileToUse = int(Items[1])
  Fuel = 0
  Lumber = 0
  SupplyChange = 0
  if self.__CheckPieceAndTileAreValid(TileToUse) == False:
    return False, SupplyChange, Fuel, Lumber
  ThePiece = self.__Tiles[TileToUse].GetPieceInTile()
  Items[0] = Items[0][0].upper() + Items[0][1:]
  if ThePiece.HasMethod(Items[0]):
    Method = getattr(ThePiece, Items[0], None)
    if Items[0] == "Supply":
      SupplyChange, Fuel, Lumber = Method(FuelAvailable,LumberAvailable)
      if SupplyChange != -1:
        return True, SupplyChange, Fuel, Lumber
  return False, SupplyChange, Fuel, Lumber
```

Code for modified PlayGame subroutine:

```
def PlayGame(Player1, Player2, Grid):
  GameOver = False
  Player1Turn = True
  Commands = []
  SupplyFirst = False
  print("Player One current state - " + Player1.GetStateString())
  print("Player Two current state - " + Player2.GetStateString())
  while not GameOver and Player1Turn:
```

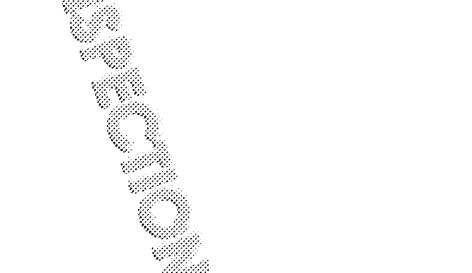
```
        break
    for C in Commands:
        Items = C.split(" ")
        ValidCommand = CheckCommandIsValid(Items, SupplyFirst)
        Supp    rst = False
        if Val    idCommand:
            pri    "Invalid command")
        else:
            Fuel    e = 0
            Lumber    ge = 0
            Supply    e = 0
            if Play    urn:
                Summary    esult, FuelChange, LumberChange, Supply    =
                Grid.Exe    eCommand(Items, Player1.GetFuel(), Player1.GetLumber(), Player1.GetPiecesInSupply())
                Player1.    teLumber(LumberChange)
                Player1.U    eFuel(FuelChange)
                if SupplyChange == 1:
                    Player1.R    veTileFromSupply()
                elif Supply    e == -1:
                    Player1.Add    eToSupplyChain()
            else:
                SummaryOfResult, FuelChange, LumberChange, SupplyChange =
                Grid.ExecuteCommand(Items, Player2.GetFuel(), Player2.GetLumber(), Player2.GetPiecesInSupply())
                Player2.UpdateLumber(LumberChange)
                Player2.UpdateFuel(FuelChange)
                if SupplyChange == 1:
                    Player2.RemoveTileFromSupply()
```

Testing:

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

For example:

```
Player One state your three commands, pressing enter after each one.
Enter command: move 8 16
Enter command: upgrade scfw 16
Enter command: supply 16
Command executed
Command executed
Invalid command
Player One current state - VPs: 0   Pieces in supply: 5   Lumber: 25   Fuel: 29
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: 25   Fuel: 29
Player Two current state - VPs: 1   Pieces in supply: 5   Lumber: 5   Fuel: 7
Press Enter to continue...
```

Task 15 (10 marks)

Coding:

- 1 mark for modifying the main menu to contain a new option 3 for creating a game.
- 1 mark for changing the LoadGame subroutine so that it accepts the FileName as a parameter correctly and does not ask for it to be input inside the subroutine, but asks in the Main 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 8, 9 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:

```
def DisplayMainMenu():
    print("1. Default game")
    print("2. Load game")
    print("3. Create game")
    print("0. Quit")
    print()
    print("Enter your choice: ", end="")
```

```
    if answer in ["y","yes","yep","ok","yeah","k"]:
        FileLoaded, Player1, Player2, Grid = LoadGame(FileName)
        if FileLoaded:
            PlayGame(Player1, Player2, Grid)
    else:
        print("Okay but the game could not be created.")
```

LoadGame subroutine:

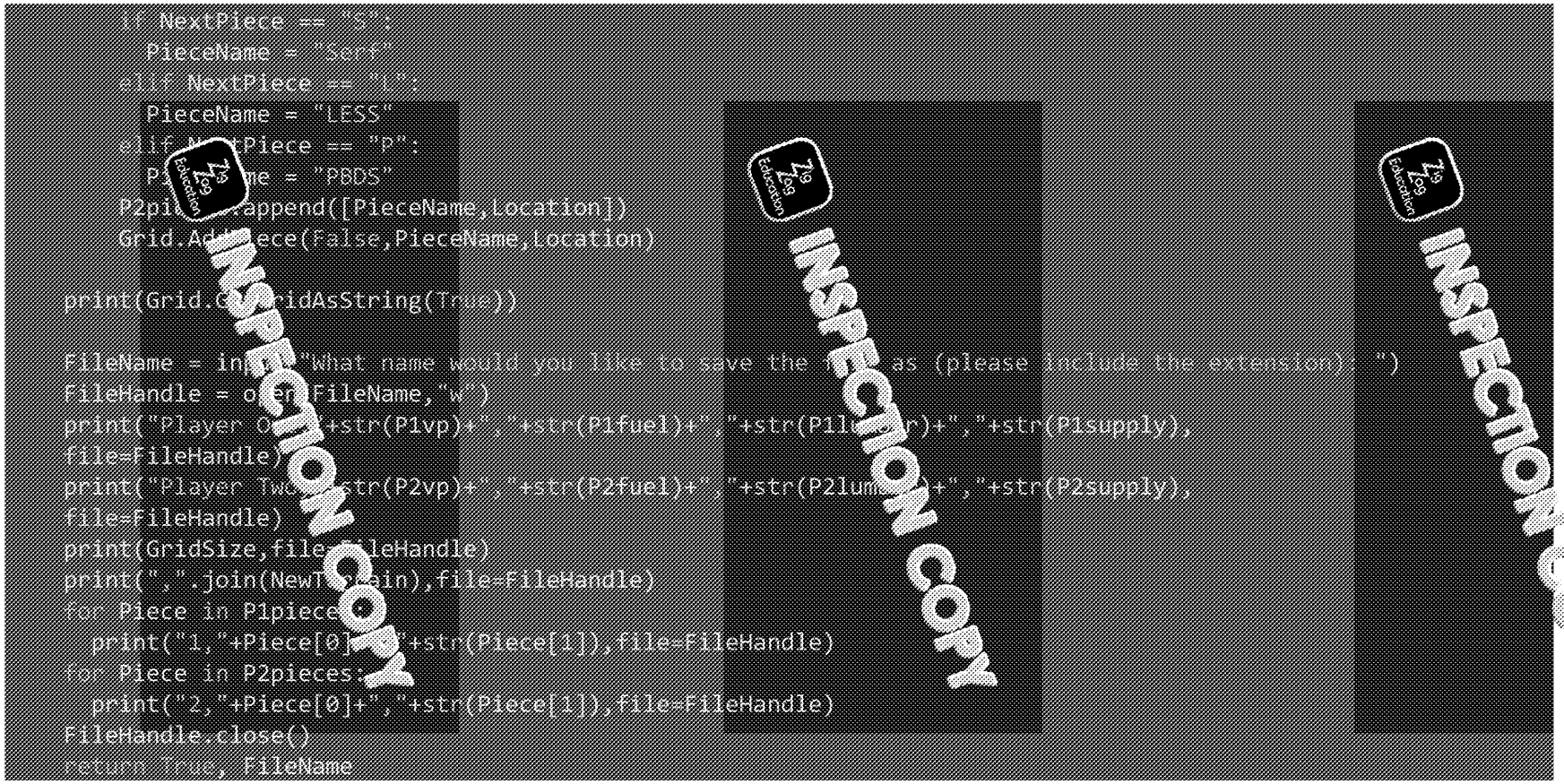
```
def LoadGame(FileName):
    Items = []
...
    T = f.readline().strip().split(",")
    if T[-1] == '':
        T[-1] = " "
    Grid.SetUpGridTerrain(T)
```

CreateGame subroutine:

```
def CreateGame():
    print("Welcome to the Baron Game Creator")
    print("-------------------------------")
    print()
    GridSize = input("Please enter the grid size (6, 8 or 10): ")
    while GridSize not in ["6","8","10"]:
        GridSize = input("Not valid, please enter 6, 8 or 10 for the grid size: ")
    GridSize = int(GridSize)
    TerrainList = [" " for i in range(GridSize*GridSize//2)]
```


INSPECTION COPY

```
else:
    NewTerrain.append( )
Grid.SetUpGridTerrain(NewTerrain)
P1baron = int(input("Please enter the location of the Baron for Player 1: "))
P2baron = int(input("Please enter the location of the Baron for Player 2: "))
Grid.AddPiece(True,"Baron",P1baron)
Grid.AddPiece(False,"Baron",P2baron)
print(Grid.GridAsString(True))
P1fuel = int(input("Please enter the starting amount of fuel for Player 1: "))
P2fuel = int(input("Please enter the starting amount of fuel for Player 2: "))
P1lumber = int(input("Please enter the starting amount of lumber for Player 1: "))
P2lumber = int(input("Please enter the starting amount of lumber for Player 2: "))
P1supply = int(input("Please enter the starting amount of pieces in the supply
for Player 1: "))
P2supply = int(input("Please enter the starting amount of pieces in the supply
for Player 2: "))
P1vp = int(input("Please enter the starting amount of VPs for Player 1: "))
P2vp = int(input("Please enter the starting amount of VPs for Player 2: "))
P1pieces = [["Baron",P1baron]]
P2pieces = [["Baron",P2baron]]
NextPiece = ""
while NextPiece != "0":
    NextPiece = input("Enter pieces to add for Player One
(Scarf, Press, Barge, or 0 for done): ").upper()
    NextPiece += ","
    if NextPiece != "0,":
        Location = int(input("Please enter the index of the location you placed the
piece: "))
        if NextPiece == "S,":
            PieceName = "Scarf"
```

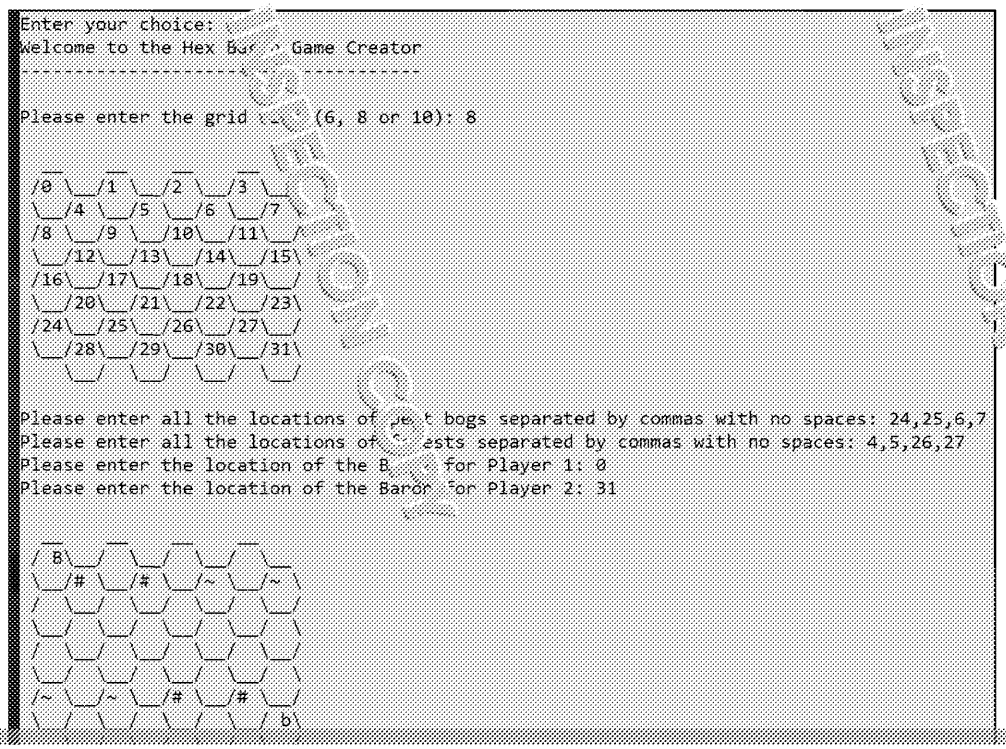


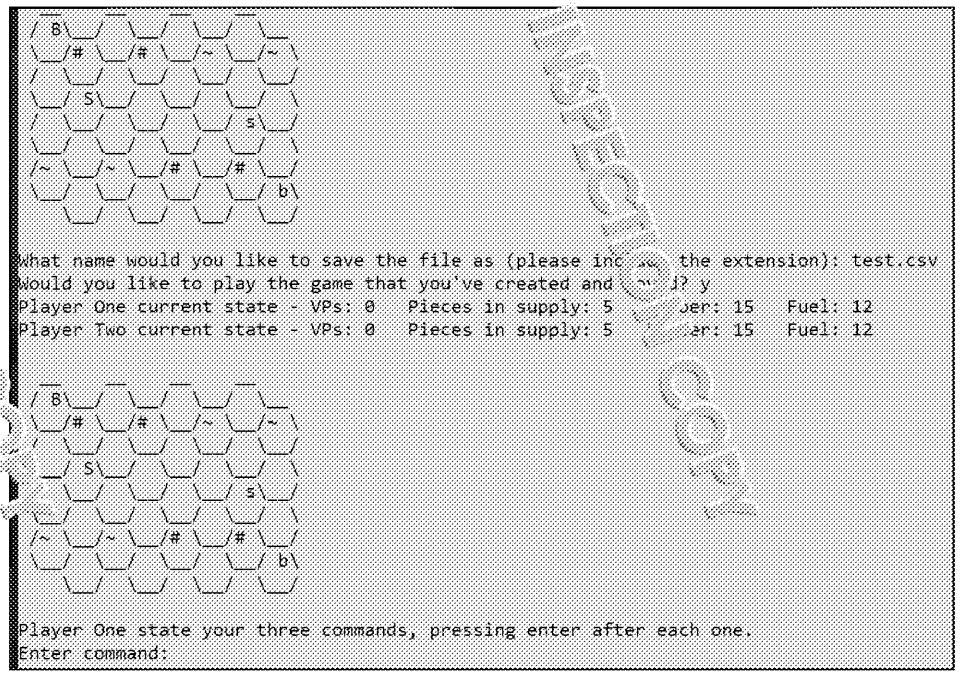
A. Solutions which have alternative input mechanisms or file printing mechanisms/structures as long as the code works.

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:





test.csv file:

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

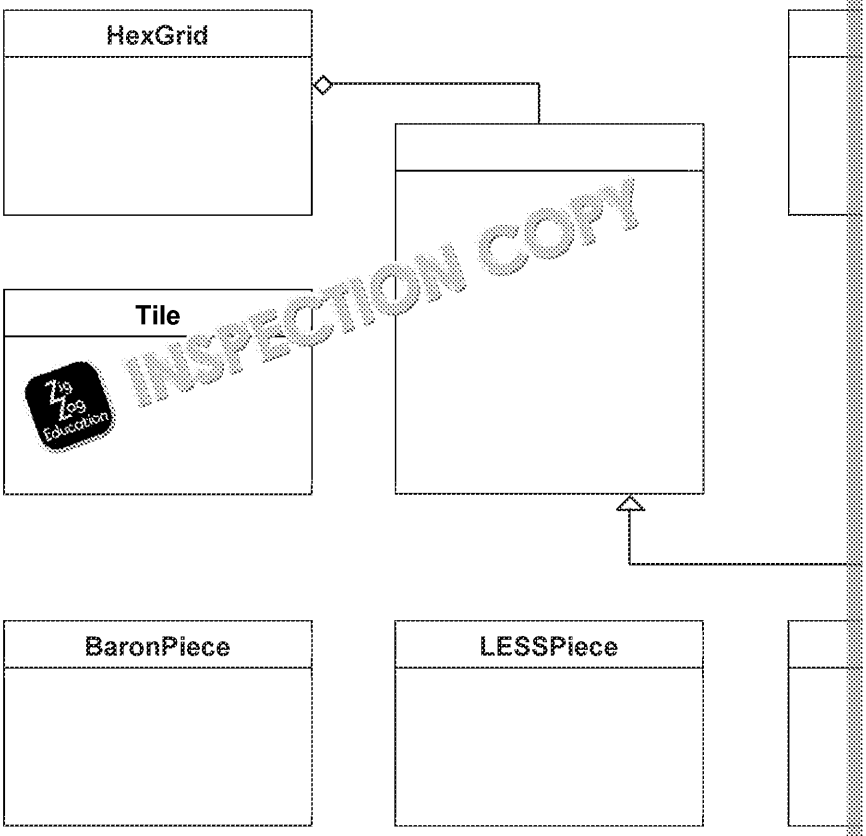
- You will need:
  - access to a computer
  - access to a printer
  - access to appropriate software
  - electronic copies of the required skeleton code
  - EAD (Electronic Answer Document)

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	