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.

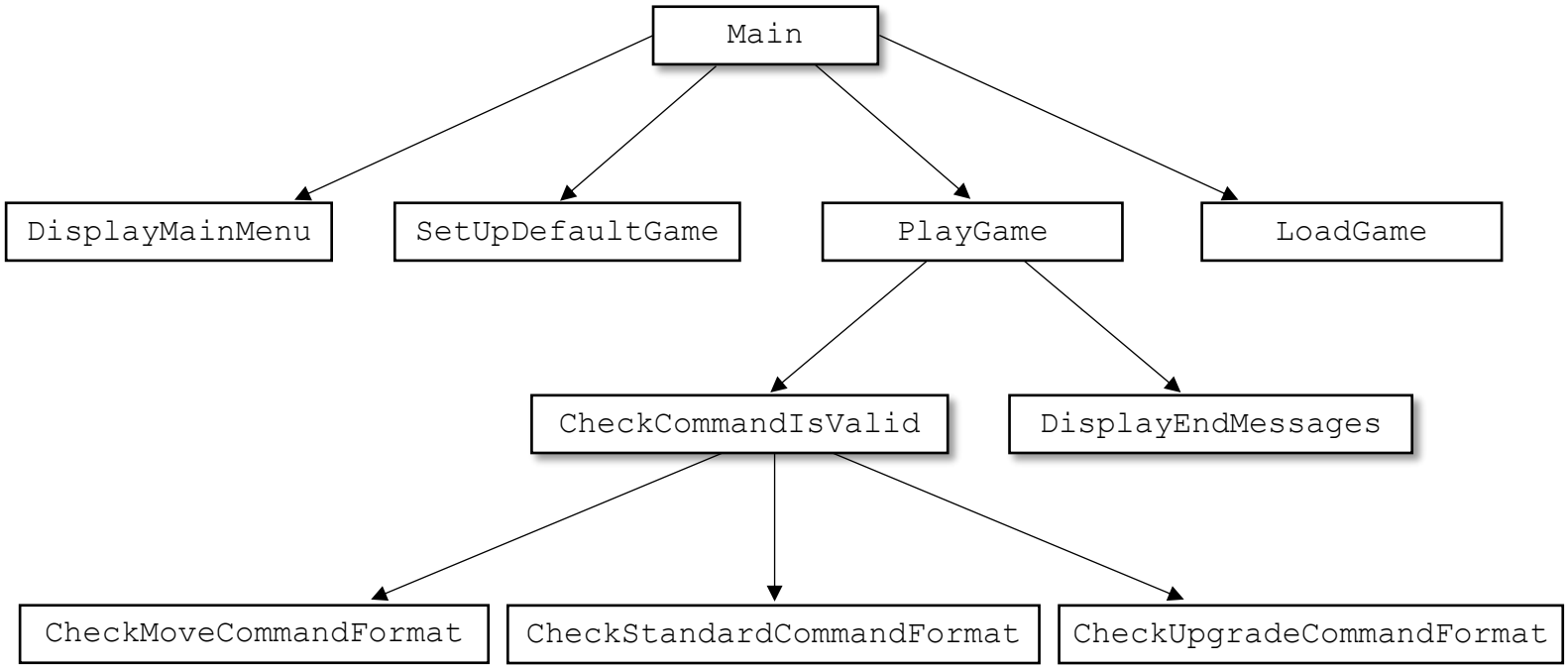
In the game you have to enter three moves before any of them are validated, so if you make a mistake with the first move you could end up losing your entire turn of all three moves as it might affect your second and third move if you thought you'd achieved something that had in fact failed.

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

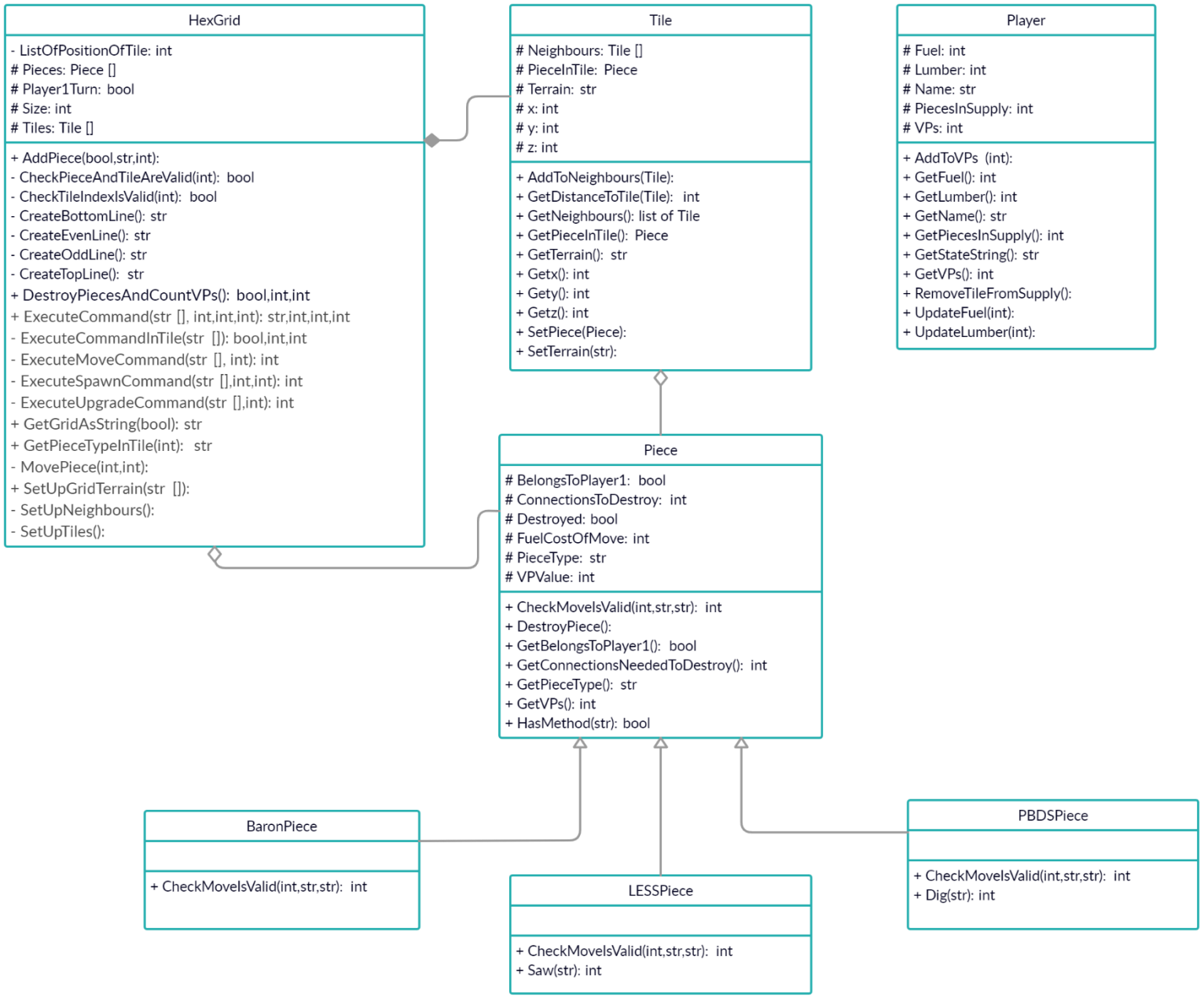


Subroutines (All)

Name	Data	Description
`CheckCommandIsValid`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Depending on the first string in the list `Items`, this calls one of the other `Check...CommandFormat` subroutines.
`CheckMoveCommandFormat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are three elements in the `Items` list 2) The second and third items are strings containing integers Otherwise it returns False.
`CheckStandardCommandFormat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are two elements in the `Items` list 2) The second item is a string containing an integer Otherwise it returns False.
`CheckUpgradeCommandFormat`	***Parameters:*** `Items` (list of strings) ***Return value(s):*** Boolean	Returns True if the following conditions are met: 1) There are three elements in the `Items` list 2) The third item is a string containing an integer 3) The second item is either less or pbds (any case) Otherwise it returns False.
`DisplayEndMessages`	***Parameters:*** `Player1`, `Player2` (Player objects) ***Return value(s):*** -	Displays the following for both players: 1) Their remaining resources 2) Their VPs Then, displays the winner.
`DisplayMainMenu`	***Parameters:*** - ***Return value(s):*** -	Prints out the main menu for the game.
`LoadGame`	***Parameters:*** - ***Return value(s):*** Boolean, `Player1` (`Player` object), `Player2` (`Player` object), `Grid` (`HexGrid` object)	Asks for the name of a .csv text file in the format documented by AQA which will load a saved game into memory and then allow it to be played by returning the `HexGrid` and `Player` objects, which can be passed into `PlayGame`.
`Main`	***Parameters:*** - ***Return value(s):*** -	Calls `DisplayMainMenu` and processes the result to either quit, load a game by calling `LoadGame` or play the default

Name	Data	Description
		game by calling `SetUpDefaultGame` and then calling `PlayGame` to play the game.
`PlayGame`	***Parameters:*** `Player1` (`Player` object), `Player2` (`Player` object), `Grid` (`HexGrid` object) ***Return value(s):*** -	This alternates between Player 1 and Player 2 and always ensures that Player 2 has an equal number of turns even if Player 1 has just killed their Baron. For each player's turn, the board is printed out and then three commands are read in and validated (by calling `CheckCommandIsValid`) and then executed (by calling `ExecuteCommand` on the `HexGrid` object for the game – stored in the variable `Grid`). After each command, the `UpdateLumber`, `UpdateFuel` and `RemoveTileFromSupply` (if necessary) methods are called on the `Player` object for that player's turn. Once the commands have been executed, pieces are destroyed, VPs are assigned and a check is made to see whether the game is over. Before the game moves on to the next player's turn, the status of both players (resources and VPs) is displayed. If the game is over and Player 2 has played, then `DisplayEndMessages` is called to display the final scores and the winner.
`SetUpDefaultGame`	***Parameters:*** - ***Return value(s):*** `Player1` (`Player` object), `Player2` (`Player` object), `Grid` (`HexGrid` object)	Initialises the game with the default values and board size as determined by AQA.

Classes



Hex Baron: Class Diagram

`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` to 0 and then calls the private methods `SetUpTiles` and `SetUpNeighbours`.
`AddPiece` (public)	***Parameters:*** `BelongsToPlayer1` (Boolean), `TypeOfPiece` (string), `Location` (int) ***Return value(s):*** -	Calls the appropriate constructor to create a new `Piece` of the correct type according to `TypeOfPiece` and passes `BelongsToPlayer1` as an argument to the constructor. Once created, it appends the `Piece` to the protected attribute `Pieces` and then adds the `Piece` to the `Tile` using `Location` as an index into the `Tiles` list and calling the `SetPiece` method on the `Tile`, passing the newly created `Piece` as an argument.

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):*** Boolean	Returns True if the parameter `TileToCheck` is a valid index of the protected attribute `Tiles`.
`CreateBottomLine` (private)	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the bottom line of the `HexGrid` (as displayed at the start of each turn when playing the game).
`CreateEvenLine` (private)	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the correct string of an even line of the `HexGrid` (as displayed at the start of each turn when playing the game).
`CreateOddLine` (private)	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the correct string of an odd line of the `HexGrid` (as displayed at the start of each turn when playing the game).
`CreateTopLine` (private)	***Parameters:*** - ***Return value(s):*** `Line` (string)	Returns a string containing the top line of the `HexGrid` (as displayed at the start of each turn when playing the game).
`DestroyPiecesAndCountVPs` (public)	***Parameters:*** - ***Return value(s):*** `BaronDestroyed` (Boolean), `Player1VPs` (int), `Player2VPs` (int)	Loops through every `Tile` in the `HexGrid` and checks for any `Pieces` that need to be destroyed by counting the number of connections for each `Piece` and comparing it to the number of connections needed to destroy the `Piece` in question. 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.
`ExecuteCommand` (public)	***Parameters:*** `Items` (list of strings), `FuelAvailable` (int), `LumberAvailable` (int), `PiecesInSupply` (int) ***Return value(s):*** `Status` (string), `FuelChange` (int), `LumberChange` (int), `SupplyChange` (int)	Depending on the first element of `Items`, it either calls `ExecuteMoveCommand`, `ExecuteSpawnCommand`, `ExecuteUpgradeCommand` or `ExecuteCommandInTile`. Each of these private methods' return values is used to create a suitable status message and determine the figures for the return values of `FuelChange`, `LumberChange` and `SupplyChange` (which were initialised to 0).

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), `PiecesInSupply` (int) ***Return value(s):*** `LumberCost` (int)	Checks to see whether the player has at least 1 `PieceInSupply` and 3 `Lumber` and that the `Tile` specified as the second string in the `Items` list is empty, otherwise it returns -1. The method then checks to see that the player's own Baron is a neighbour and then spawns a new Serf in the `Tile`, appends it to the attribute `Pieces` and returns 3, otherwise it returns -1.
`ExecuteUpgradeCommand` (private)	***Parameters:*** `Items` (list of strings), `LumberAvailable` (int) ***Return value(s):*** `LumberCost` (int)	If the tile specified as the third item of the `Items` list is available and contains a Serf belonging to the player whose turn it is and that player has at least 5 lumber available then it is upgraded to a `LESSPiece` or `PBDSPiece` according to the second string in `Items` and 5 is returned as the cost, otherwise -1 is returned.

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.
GetPieceTypeInTile (public)	***Parameters:*** ID (int) ***Return value(s):*** PieceType (string)	If there is no Piece in the Tile specified by ID then this returns " " (string with a single space), otherwise it returns the result of the GetPieceType method which is called on the Piece in the Tile.
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.
SetUpTiles (private)	***Parameters:*** - ***Return value(s):*** -	Loops through and creates all of the tiles needed for the HexGrid according to the protected attribute Size and adds them to the protected attribute Tiles.

`LESSPiece` (inherits from `Piece`)

Name	Data	Description
`__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 • `PieceType` to "P" • `VPValue` to 2
`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 peat bog then it returns the value of the protected attribute `FuelCostOfMove` otherwise it returns -1.
`Dig` (public)	***Parameters:*** `Terrain` (string) ***Return value(s):*** `Fuel` (int)	If `Terrain` is a peat bog then it has a 90% chance of returning 1 and a 10% chance of returning 5, otherwise it returns 0.

Piece

Name	Data	Description
`__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:*** - ***Return value(s):*** `PieceType` (string)	If the attribute `BelongsToPlayer1` is True then it returns the value of the attribute `PieceType` otherwise it returns the lower case value.
`GetVPs` (public)	***Parameters:*** - ***Return value(s):*** `VPValue` (int)	Returns the value of the protected attribute `VPValue`.
`HasMethod` (public)	***Parameters:*** `MethodName` (string) ***Return value(s):*** `callable` (Boolean)	If the method name specified as a parameter exists in the object then it returns True, otherwise it returns False.

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 `V` • `Fuel` from parameter F • `Lumber` from parameter `L` • `PiecesInSupply` from parameter `T`
`AddToVPs` (public)	***Parameters:*** `n` (int) ***Return value(s):*** -	Increments the attribute `VPs` by `n`.
`GetFuel` (public)	***Parameters:*** - ***Return value(s):*** `Fuel` (int)	Returns the value of the attribute `Fuel`.
`GetLumber` (public)	***Parameters:*** - ***Return value(s):*** `Lumber` (int)	Returns the value of the attribute `Lumber`.
`GetName` (public)	***Parameters:*** - ***Return value(s):*** `Name` (string)	Returns the value of the attribute `Name`.
`GetPiecesInSupply` (public)	***Parameters:*** - ***Return value(s):*** `PiecesInSupply` (int)	Returns the value of the attribute `PiecesInSupply`.
`GetStateString` (public)	***Parameters:*** - ***Return value(s):*** 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` (public)	***Parameters:*** - ***Return value(s):*** -	Decrements the `PiecesInSupply` attribute by 1.
`UpdateFuel` (public)	***Parameters:*** `n` ***Return value(s):*** -	Increments the attribute `Fuel` by `n`.
`UpdateLumber` (public)	***Parameters:*** `n` ***Return value(s):*** -	Increments the attribute `Lumber` by `n`.

Tile

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

