HEX BARON – Commentary

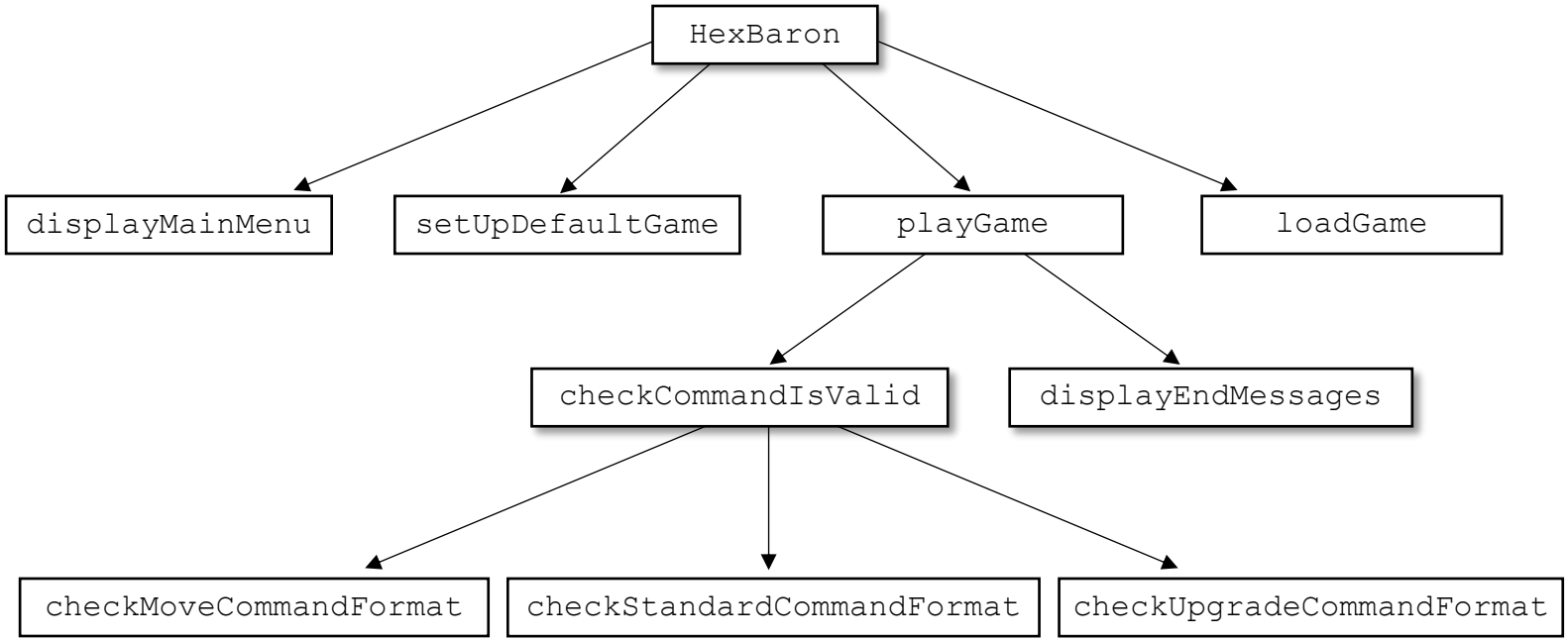
Hex Baron is a two-player game in which the main objective is to gain as many Victory Points (VPs) as possible, normally by killing your opponent's Baron. All adjacent hexes count as a distance of 1 and are legal moves, although for some pieces this will cost 2 fuel. It is best to start by creating a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

Important: note that although the whole solution is class-based, the `HexBaron` class is considered to be the main program. It is referred to as such throughout this resource and methods in the `HexBaron` class are hence referred to as subroutines. This is for two reasons, firstly to keep it in line with the way that the resource is presented in the other languages (which is therefore more likely to match any exam content) and secondly because this is the logical function of the HexBaron class given that all Java programs only contain classes.

Subroutines

Subroutines (Main Program): Hierarchy Diagram



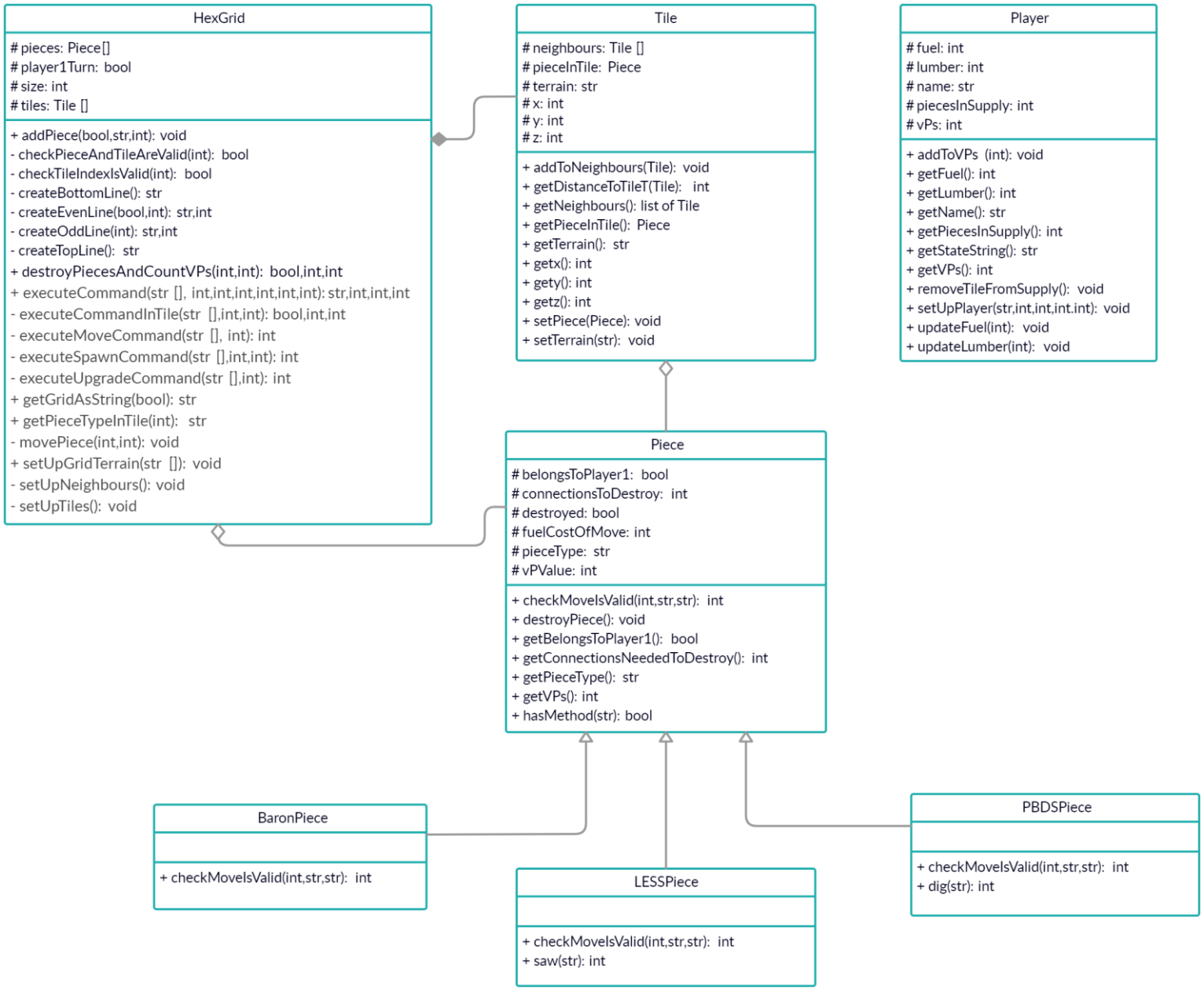
Subroutines (All)

* indicates parameters passed by reference, all other parameters are passed by value

Name	Data	Description
`checkCommandIsValid`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Boolean	Depending on the first string in the list `items`, this calls one of the other `check...CommandFormat` subroutines.
`checkMoveCommandFormat`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Boolean	Returns true if the following conditions are met: 1) There are three elements in the `items` list 2) The second and third items are strings containing integers Otherwise it returns false.
`checkStandardCommandFormat`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Boolean	Returns true if the following conditions are met: 1) There are two elements in the `items` list 2) The second item is a string containing an integer Otherwise it returns false.
`checkUpgradeCommandFormat`	***Parameters:*** `items` (list of strings) ***Return value(s):*** Boolean	Returns true if the following conditions are met: 1) There are three elements in the `items` list 2) The third item is a string containing an integer 3) The second item is either less or pbds (any case) Otherwise it returns false.
`displayEndMessages`	***Parameters:*** `player1`, `player2` (Player objects) ***Return value(s):*** -	Displays the following for both players: 1) Their remaining resources 2) Their VPs Then, displays the winner.
`displayMainMenu`	***Parameters:*** - ***Return value(s):*** -	Prints out the main menu for the game.
`loadGame`	***Parameters:*** `*player1` (`Player` object), `*player2` (`Player` object) ***Return value(s):*** `grid` (`HexGrid` object), Boolean	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 setting the `Player` objects and returning the `HexGrid` object, which can be passed into `playGame`.

Name	Data	Description
`HexBaron`	***Parameters:*** - ***Return value(s):*** -	Calls `displayMainMenu` and processes the result to either quit, load a game by calling `loadGame` or play the default game by calling `setUpDefaultGame` and then calling `playGame` to play the game.
`playGame`	***Parameters:*** `player1` (`Player` object), `player2` (`Player` object), `grid` (`HexGrid` object) ***Return value(s):*** -	This alternates between Player 1 and Player 2 and always ensures that Player 2 has an equal number of turns even if Player 1 has just killed their Baron. For each player's turn, the board is printed out and then three commands are read in and validated (by calling `checkCommandIsValid`) and then executed (by calling `executeCommand` on the `HexGrid` object for the game – stored in the variable `grid`). After each command, the `updateLumber`, `updateFuel` and `removeTileFromSupply` (if necessary) methods are called on the `Player` object for that player's turn. Once the commands have been executed, 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:*** `*player1` (`Player` object), `*player2` (`Player` object) ***Return value(s):*** `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
`BaronPiece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Calls the super constructor (`Piece`) and passes `player1` as an argument. Initialises the following protected attributes: • `pieceType` to "B" • `vPValue` to 10
`checkMoveIsValid` (public)	***Parameters:*** `distanceBetweenTiles` (int), `startTerrain` (string), `endTerrain` (string) ***Return value(s):*** `fuelCostOfMove` (int)	If the value of the parameter `distanceBetweenTiles` is 1 then it returns the value of the protected attribute `fuelCostOfMove` otherwise it returns -1.

`HexGrid`

Name	Data	Description
`HexGrid` (constructor)	***Parameters:*** `n` (int) ***Return value(s):*** -	Initialises the following protected attributes: • `size` from parameter `n` • `player1Turn` to true It 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.
`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.

Name	Data	Description
`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 grid (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 grid (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 grid (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 grid (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 grid 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 grid 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 grid.
`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 grid, it loops through all the tiles of the grid 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 grid according to the protected attribute `size` and adds them to the protected attribute `tiles`.

`LESSPiece` (inherits from `Piece`)

Name	Data	Description
`LESSPiece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Calls the super constructor (`Piece`) and passes `player1` as an argument. Initialises the following protected attributes: • `pieceType` to "L" • `vPValue` to 3
`checkMoveIsValid` (public)	***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
`PBDSPiece` (constructor)	***Parameters:*** `player1` (Boolean) ***Return value(s):*** -	Calls the super constructor (`Piece`) and passes `player1` as an argument. Initialises the following protected attributes: • `fuelCostOfMove` to 2 • `pieceType` to "P" • `vPValue` to 2
`checkMoveIsValid` (public)	***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
`Piece` (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
`Player` (constructor)	***Parameters:*** - ***Return value(s):*** -	This is an empty constructor that simply creates an empty object, later on it will be populated using `setUpPlayer`().
`addToVPs` (public)	***Parameters:*** `n` (int) ***Return value(s):*** -	Increments the attribute `vPs` by `n`.
`getFuel` (public)	***Parameters:*** - ***Return value(s):*** `fuel` (int)	Returns the value of the attribute `fuel`.
`getLumber` (public)	***Parameters:*** - ***Return value(s):*** `lumber` (int)	Returns the value of the attribute `lumber`.
`getName` (public)	***Parameters:*** - ***Return value(s):*** `name` (string)	Returns the value of the attribute `name`.
`getPiecesInSupply` (public)	***Parameters:*** - ***Return value(s):*** `piecesInSupply` (int)	Returns the value of the attribute `piecesInSupply`.
`getStateString` (public)	***Parameters:*** - ***Return value(s):*** `stateString` (string)	Returns a string containing the `vPs`, the `piecesInSupply`, the `lumber` and the `fuel`.
`getVPs` (public)	***Parameters:*** - ***Return value(s):*** `vPs` (int)	Returns the value of the attribute `vPs`.
`removeTileFromSupply` (public)	***Parameters:*** - ***Return value(s):*** -	Decrements the `piecesInSupply` attribute by 1.
`setUpPlayer` (public)	***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`
`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
`Tile` (constructor)	***Parameters:*** `xCoord` (int), `yCoord` (int), `zCoord` (int) ***Return value(s):*** -	Initialises the following protected attributes: • x from parameter `xCoord` • y from parameter `yCoord` • z from parameter `zCoord` • `terrain` to " " (a string containing a single space) • `pieceInTile` to None
`addToNeighbours` (public)	***Parameters:*** `N` (`Tile` object) ***Return value(s):*** -	Adds the tile `N` to the end of the list stored in the protected attribute `neighbours`.
`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`.

