HEX BARON – Programming Tasks

Task 1 (4 marks)

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

The commands input by the player are already converted to lower case, which makes them case-insensitive. However, sometimes a player may additionally or accidentally type a space at the beginning or end of their command – currently this often causes an invalid command.

- Create a new subroutine called **`strip`** to remove any leading and trailing spaces from the command
- Change the existing line of code that converts the command to lower case, so that it calls the new **`strip`** subroutine to *also* remove the unneccesary spaces

Test that the change you have made works:

a) Choose menu option 1 to run the default game and then enter the following commands (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 from the computer up to the 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 subroutine **`playGame`** and the new **`strip`** subroutine.

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

Task 2 (3 marks)

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

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

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter "Tom" as the name for Player One and 'Vicky' as the name for Player Two.

b) Show a screen copy of entering the commands and the responses from the computer up to and 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 **`setUpDefaultGame`**.

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

Task 3 (4 marks)

This question refers to the **playGame** and **setUpDefaultGame** subroutines in the main program, to the **Player** class and to the **destroyPiecesAndCountVPs** method in the **HexGrid** class.

Normally when a piece is destroyed it depletes your supply chain and moves you one step closer to losing the game as the number of pieces that you can make is limited. This rule has been changed so that if a piece is destroyed then you gain one piece in your supply chain.

- Add a method to the **Player** class that will allow the **piecesInSupply** protected attribute to be modified.

- Modify the **destroyPiecesAndCountVPs** method to take `player1` and `player2` as parameters and pass them in from **playGame** in both places that it is called.

- Further modify the **destroyPiecesAndCountVPs** method so that it calls the **addPiecesInSupply** method that you created for the appropriate player to give them an additional piece in their supply every time one of their pieces is destroyed.

- Modify the **setUpDefaultGame** subroutine so that the existing four **AddPiece** commands are replaced with the following six:

```
grid.addPiece(true, "Baron", 0);
grid.addPiece(true, "Serf", 7);
grid.addPiece(true, "Serf", 13);
grid.addPiece(false, "Baron", 31);
grid.addPiece(false, "Serf", 15);
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 following commands:

- move 13 10
- move 10 14
- move 14 19

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

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine **playGame**, the **destroyPiecesAndCountVPs** method of the **HexGrid** class and the new code for the **Player** class.

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

Task 4 (4 marks)

This question refers to the **setUpDefaultGame** subroutine in the main program and to the **executeSpawnCommand** method in the **HexGrid** class.

Change the rules of the game so that each player can have a maximum of six pieces in play at any given time. There should be an additional output message generated saying that the 'Spawn attempted to exceed max pieces.'

- Modify the private method **executeSpawnCommand** of the **HexGrid** class.

- Modify the **setUpDefaultGame** subroutine so that the existing four **addPiece** commands 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 following commands:

- spawn 4
- move 4 9
- move 9 17

b) Show a screen copy of entering the commands and the responses from the computer up to the 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 **executeSpawnCommand** method of the **HexGrid** class.

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

Task 5 (9 marks)

This question refers to the **playGame** subroutine in the main program, and to a new method **getGridAsIndices** in the **HexGrid** class.

Add an additional command to the game which will print out the hex grid and show the index for each hex; these should be in the top half of each hex and the board should be the same size as the normal display. The new command, "hexes" should not count as one of the three commands that you enter and the results should be displayed immediately upon entry, i.e. the grid should be printed out with the numbers displayed as soon as Enter is pressed. The player should be able to enter hexes for any one of their commands and still have three normal commands.

- Create a new method for the **HexGrid** class called **getGridAsIndices** which will return a string containing the grid with all of the index numbers in the correct places; this could be based on the current **getGridAsString** method. You may wish to duplicate and then modify the **createOddLine** and **createEvenLine** methods.

- Modify the **playGame** subroutine to implement a call to the new method whenever a player enters the command "hexes" and then request the normal command.

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the following commands:
   - move 8 16
   - hexes
   - move 16 24
   - hexes
   - hexes
   - upgrade pbds 24

b) Show a screen copy of entering the commands and the responses from the computer up to and including the prompt for the second player to enter their commands, including the printout of the hex grid with the numbers and the printout of the board after Player One has completed their move.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine **playGame**, the **getGridAsIndices** method of the **HexGrid** class and any new methods such as modified versions of **createOddLine** and **createEvenLine** added to the **HexGrid** class.

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

Task 6 (4 marks)

This question refers to the **playGame** and **setUpDefaultGame** subroutines in the main program.

Change the move command so that if you move the same piece three times in a row, you get a reduction in cost of 1 fuel (in total). For example, if you moved a Serf to a field (from a field), then to a peat bog and then to a field, it would only cost 4 fuel instead of 5. If you moved a Baron three squares, it would only cost 2 fuel instead of 3. They should be able to do the triple move even if it causes their fuel to end up on 0.

- Modify the **playGame** subroutine to check whether three valid move commands have been executed by the player and refund them one fuel.

- Modify the **setUpDefaultGame** subroutine so that Player One only starts with 2 fuel:

```
player1.setUpPlayer("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 following commands:

- move 0 4
- move 4 9
- move 9 17

b) Show a screen copy of entering the commands and the responses from the computer up to the 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 subroutine **playGame**.

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

Task 7 (7 marks)

This question refers to the **playGame** subroutine in the main program and to the **executeCommandInTile** method and a new method, **makeField,** in the **HexGrid** class.

Change the saw and dig commands so that if you saw or dig the same location three times in a row in the same turn then you get 5 of that resource and the terrain reverts to a field. Remove the random chance of getting 5 fuel when you dig so that you always get only 1.

- Modify the **executeCommandInTile** to remove the possibility of a dig command generating a field.
- Modify the **dig** method of the **PBDSPiece** to remove the possibility of a dig command generating 5 fuel.
- Add a method to the **HexGrid** class called **makeField** which has one parameter, the index of the tile which is to be made into a field.
- Modify the **playGame** subroutine to check whether three dig or saw commands have been executed in a single turn by either player and give them the extra resources and turn the tile into a field by calling the newly added **makeField** method in the **HexGrid** class.

Test that the changes you have made work:

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

Player One	• move 8 16 • move 16 24 • upgrade pbds 24
Player Two	• move 23 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 have been executed and include the Player One and Player Two current states that are shown immediately before the final board print.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine **playGame**, the **executeCommandInTile** and **makeField** methods of the **HexGrid** class, and the modified **dig** method of the **PBDSPiece** class.

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

Task 8 (7 marks)

This question refers to the **checkCommandIsValid** and **setUpDefaultGame** subroutines in the main program, the creation of a new subroutine **checkDowngradeCommandFormat** and to a new method, **executeDowngradeCommand** as well as the **executeCommand** method in the **HexGrid** class.

Introduce a new downgrade command so that a PDBS or LESS can be downgraded to a Serf for a cost of 1 lumber. The syntax for the command is "downgrade NUM" where NUM is the location of the piece to be downgraded.

- Modify the **checkCommandIsValid** subroutine to allow for and check the format of the new downgrade command.
- Create a new subroutine **checkDowngradeCommandFormat** which is called by the modified **checkCommandIsValid** subroutine.
- Modify the **executeCommand** method of **HexGrid** to call a new private method **executeDowngradeCommand**.
- Create a new private method in **HexGrid** called **executeDowngradeCommand**.
- Modify the **setUpDefaultGame** subroutine so that **player1** Serf piece starts at location 16:

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

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the following commands:
   - move 16 24
   - upgrade pbds 24
   - downgrade 24

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

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine **checkCommandIsValid**, the new subroutine **checkDowngradeCommandFormat**, the **executeCommand** method of the **HexGrid** class and the new code for the **executeDowngradeCommand** method of **HexGrid**.

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

Task 9 (3 marks)

This question refers to the **`setUpDefaultGame`** subroutine in the main program and to the **`BaronPiece`** class.

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

- Modify the **`BaronPiece`** class to make it so that it can only be destroyed by three connections.
- Modify the **`setUpDefaultGame`** subroutine so that the four starting pieces are replaced with the following 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 following commands:

Player One	• move 25 21 • move 21 18 • move 18 14
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`** class.

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

Task 10 (4 marks)

This question refers to the **checkCommandIsValid** subroutine in the main program and to the **executeCommand** method and a new **executeSalvageCommand** method in the **HexGrid** class.

Develop the salvage command. Any of your own Serf, PBDS or LESS pieces can be salvaged. Salvaging a piece will add 1 to your supply chain, give you 5 lumber and remove the piece from the board. The format of the command is salvage NUM, where NUM is the location on the board of the piece to salvage.

- Modify the **checkCommandIsValid** subroutine to allow for and check the format of the new salvage command.
- Modify the **executeCommand** method of **HexGrid** to call a new private method **executeSalvageCommand**.
- Create a new private method in **HexGrid** called **executeSalvageCommand**.

Test that the changes you have made work:

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

b) Show a screen copy of entering the commands and the response from the computer up to the point at which it prints out the updated board.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine **checkCommandIsValid** and the **executeCommand** and **executeSalvageCommand** methods of the **HexGrid** class.

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

Task 11 (10 marks)

This question refers to the **playGame** subroutine in the main program and to the **Player** class.

Introduce three chances for each player so that if they enter an invalid move or a move that could not be executed for some reason, they can correct it, but only three times per game. The system should prompt them to re-enter the invalid move and deduct 1 from their chances. The remaining chances should be printed out each time a player uses up a chance and at the start of each player's turn.

Also, as the commands and replacements could now be confusing, make sure that each time a player is asked for a command or a replacement command they are told which command number they are entering/re-entering.

- Modify the **Player** class to create an attribute called **chances** for their three chances.
- Modify the **playGame** subroutine so that it allows the player to enter a replacement command if their command is invalid or could not be executed for some reason.
- Create three new methods for the **Player** class to control access to the **chances** attribute: **deductChance**, **getChances** and **resetChances**.
- By way of example and clarity, the testing output for Player One's first turn should start as follows:

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

Test that the changes you have made work:

a) Choose menu option 1 to run the default game and then enter the following input, waiting for a prompt between each entry:

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 copy of entering the commands and all the responses from the computer.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended **playGame** subroutine and **Player** class.

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

Task 12 (13 marks)

This question refers to the **checkCommandIsValid** and **setUpDefaultGame** subroutines and a new **checkSpawnCommandFormat** subroutine in the main program, to a new **BombPiece** class, to the **Piece** class and to the **executeSpawnCommand**, **destroyPiecesAndCountVPs**, **getPieceTypeInTile** and **executeMoveCommand** methods in the **HexGrid** class.

Develop a Bomb piece. The bomb can only be spawned by the Baron but should look like a Serf piece. It costs 10 lumber to spawn and 2 fuel for each space that it moves regardless of the terrain.

When the bomb explodes it will destroy all pieces in the hexes adjacent to the hex containing the bomb. It will explode if two pieces are adjacent to it (as per normal attacks), but the bomb has a further ability.

Once the bomb has moved five spaces it is immobilised and becomes 'primed': any piece newly entering (even in the middle of a move) one of the hexes surrounding the bomb will cause it to explode. The bomb will not explode if it is primed next to an existing piece (but of course it will if primed next to two existing pieces), only if another piece enters a hex next to it, kind of like a motion detector.

The command is a modification of the spawn command, so now spawn can optionally take a second argument of bomb, e.g. spawn bomb NUM would spawn a bomb at NUM if NUM is adjacent to the Baron, and spawn NUM would operate as normal. The bomb has a VP value of 5, meaning that when it explodes, the opposing player receives 5 VPs.

- Modify the **checkCommandIsValid** subroutine to make the new spawn command function correctly.
- Create a new subroutine called **checkSpawnCommandFormat** that is called from the modified **checkCommandIsValid** to make sure that the new spawn bomb command is valid.
- Modify the private method **executeSpawnCommand** so that it will spawn a bomb and charge the player 10 lumber or report an error if they don't have enough lumber.
- Create a new **BombPiece** class for the bomb.
- Modify the method **destroyPiecesAndCountVPs** so that if a bomb is destroyed then it explodes and destroys the surrounding pieces too.
- Modify the method **getPieceTypeInTile** for **HexGrid** so that the bomb piece appears as a Serf when the board is displayed.
- Modify the private method **executeMoveCommand** so that it checks whether the player moved 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 be destroyed at the end of the turn in case a bomb was set off during the turn.
- Modify the **setUpDefaultGame** subroutine so that both players start with 30 of each resource:

```
player1.setUpPlayer("Player One", 0, 30, 30, 5);
player2.setUpPlayer("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 following commands:

Player One	• spawn bomb 4 • move 4 9 • move 9 13
Player Two	• move 23 19 • move 19 11 • move 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 from the computer up to the 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 following commands:

Player One	• spawn bomb 4 • move 4 9 • move 9 13
Player Two	• move 23 19 • upgrade less 19 • saw 19
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 from the computer up to the point at which it prints out that Player One is the winner.

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the modified **`checkCommandIsValid`** and the new **`checkSpawnCommandFormat`** subroutines, the new **`BombPiece`** class, the modified **`Piece`** class and the modified methods **`executeSpawnCommand`**, **`destroyPiecesAndCountVPs`**, **`getPieceTypeInTile`** and **`executeMoveCommand`** of the **`HexGrid`** class.

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

Task 13 (10 marks)

This question refers to the **playGame**, **checkUpgradeCommandFormat** and **setUpDefaultGame** subroutines in the main program, to the new **WizardPiece** class and to the **executeUpgradeCommand** and new **resetWizards** methods in the **HexGrid** class.

Introduce a new piece, a Wizard which can teleport a distance of three spaces but only once per turn; this costs 5 fuel. It can also move one square for a cost of 1 fuel. The piece can be created by upgrading a Serf using the new command upgrade wiz NUM, where NUM is the location of a hex containing a Serf. To move the wizard, you simply use the move command as normal and if the distance is two or three squares then it teleports, if it is one square then it moves normally and if it is more than three squares then it is an invalid move. The wizard should be displayed with W for Player One and w for Player Two and should have a VP value of 3.

- Create a new **WizardPiece** class.
- Modify the **checkUpgradeCommandFormat** subroutine to allow for the new wiz upgrade command.
- Modify the method **executeUpgradeCommand** to create the new wizard piece if spawned.
- Add a new method to the **HexGrid** class called **resetWizards** to reset the teleport commands at the end of each turn.
- Modify the **playGame** subroutine to call the new **resetWizards** method (only show the lines either side and the new line).
- Modify the **setUpDefaultGame** subroutine so that Player One starts with 20 fuel:

```
player1.setUpPlayer("Player One", 0, 20, 10, 5);
```

Test that the changes you have made work:

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

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 commands and the responses from the computer up to the 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 subroutines **playGame** and **checkUpgradeCommandFormat**, the **executeUpgradeCommand** and new method **resetWizards** methods of the **HexGrid** class and the new **WizardPiece** class.

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

Task 14 (10 marks)

This question refers to the **playGame**, **checkCommandIsValid**, **checkUpgradeCommandFormat** and **setUpDefaultGame** subroutines in the main program, to the new **SCFWPiece** class, to the **Player** class and to the **executeUpgradeCommand**, **executeCommand** and new **executeSupplyCommand** methods in the **HexGrid** class.

Introduce a new piece, a supply chain factory worker, for the same cost of 5 lumber of any upgrade. The syntax for the upgrade is "upgrade scfw NUM" where NUM is the location of the Serf to be upgraded. Once created, the piece is displayed as F for Player One and f for Player Two. A new command will be available "supply NUM", which will take all three of your moves (so it can only be the first command), will cost 5 lumber and 5 fuel and will add 1 to your supply chain; NUM is the location of the factory. The VP value of the factory is 3. The factory cannot be moved once created.

- Create the new **SCFWPiece** class.
- Modify the **checkCommandIsValid** subroutine to validate the supply command.
- Modify the **CheckUpgradeCommandFormat** subroutine so that it can validate the new upgrade command.
- Modify the private method **ExecuteUpgradeCommand** of the **HexGrid** class to allow the new **SCFWPiece** to be created.
- Modify the method **ExecuteCommand** of the **HexGrid** class to allow for the supply command to work by calling a new private method, **ExecuteSupplyCommand**.
- Create the new private method **ExecuteSupplyCommand** in the **HexGrid** class to implement the new supply command.
- Modify the **PlayGame** subroutine to only allow the supply command to be the first command of the three and terminate input for the other two commands if received as the first command. Also modify it to call the **AddPieceToSupplyChain** method.
- Create a new **AddPieceToSupplyChain** method in the **Player** class that will add 1 piece to the supply chain.
- Modify the **SetUpDefaultGame** subroutine so that Player One starts with 30 of both resources:

```
player1.setUpPlayer("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 following commands:

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

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

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutines **playGame**, **checkCommandIsValid** and **checkUpgradeCommandFormat**, the **executeUpgradeCommand**, **executeCommand** and new **executeSupplyCommand** methods of the **HexGrid** class, the new **SCFWPiece** class and the **addPieceToSupplyChain** method in the **Player** class.

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

Task 15 (10 marks)

This question refers to the **`displayMainMenu`**, **`loadGame`** and new **`createGame`** subroutines in the main program. The question also relies on using the **`getGridAsIndices`** method from Task 5, including any methods called by that method. Therefore it is best to begin your solution with the codebase that you ended Task 5 with.

Introduce a new command to the main menu that will allow a custom game to be created. The user should be able to enter the grid size as a choice of 6, 8 (the current size) or 10 (and the board will be displayed for them to check – this is just a reuse of the code from Task 5 and hence the final code from Task 5 will be the starting point for this task), the location of a Baron for each player as well as the location of any additional pieces that they wish to add. They should be able to define which hexes contain lumber and peat bogs and define the starting resources (including supply chain) for each player. There is no need to perform any validation on the data entry except for the grid size (which must be 6, 8 or 10).

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 with no spaces: "
- "Please enter all the locations of forests separated by commands with no spaces: "
- "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 VPs for Player 1: "
- "Please enter the starting amount of VPs 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: "
- "Enter piece to add for Player One (S=Serf, L=LESS, P=PBDS) or D for Done: "
- "Enter piece to add for Player Two (S=Serf, L=LESS, P=PBDS) or D for Done: "
- "What name would you like to save the file as (please include the extension): "
- "Would you like to play the game that you've created and saved? "

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

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

The game should automatically save in a file (it should prompt for the name) and then, once saved, the players should have the opportunity to play it immediately if they wish without having to type in the file name. For reference, see the **game1.txt** file supplied with the pre-release code; the format of the file is as follows (it is all comma-separated values):

Line 1:	text for Player One's name, Player One's VPs, Player One's fuel, Player One's lumber, Player One's supply chain
Line 2:	text for Player Two's name, Player Two's VPs, Player Two's fuel, Player Two's lumber, Player Two's supply chain
Line 3:	board size
Line 4:	terrain 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 indicate the player number, piece name (case sensitive), index (board location)

TASK CONTINUES ON THE NEXT PAGE

- Modify the **`displayMainMenu`** subroutine to add option 3. Create game.
- Modify the main subroutine so that it calls a new subroutine **`createGame`** and then asks the player if they would like to load the game and then loads the game. You will also need to modify the call **`loadGame`** so that it accepts the file name as a parameter.
- Create a new subroutine called **`createGame`** which will create the file containing the game save and return two values, **`success`** (as a Boolean) and **`fileName`** (the text name of the file created).
- Modify the **`loadGame`** subroutine to correct the bug as described above.

Test that the changes you have made work:

a) Choose menu option 3 to create a game and then enter the following data at each consecutive prompt:

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

b) Show a screen copy of entering the commands and the responses from the computer up to the 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. Your PROGRAM SOURCE CODE for the amended subroutines **`loadGame`** and **`displayMainMenu`** and the new subroutine **`createGame`** in the main subroutine. Also include any import statements that you added to the main program.

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

