Objective
Create a version of the Battleship game that is entirely database-backed. This means that the state of the
game will be stored entirely in the database at all times. We will use two databases: boards.mdb and
games.mdb.
The databases will have the following schema:
Games
   tblUser
       Username (String), UserId (Integer)
   tblGame
       GameId (Integer), UserId (Integer), GameDate (Date), UserBoardId (Integer),
       ComputerBoardId (Integer), Status (Integer)
Boards
   tblBoard
       BoardId (Integer), X (Integer), Y (Integer), Hit (Boolean), ShipName (String)
   tblShip
       BoardId (Integer), ShipName (String), Size (Integer), HitsRemaining (Integer)

Use the VisData program to build these databases from within Visual Basic.

When the program starts, it should ask the user to either select his username from those already in the
system, or add a new name. If an existing name is chosen, the user should have the option of selecting a
saved game or starting a new one.

The Boards database will handled entirely by the BattleshipControl ActiveX controls. You will need to add
a “BoardId” property to this control so that the game can tell the control what board to display, and a
“NewBoard” function that generates a new board id, sets it up and returns the value to the caller.

The BattleshipControl will be implemented completely differently in this assignment. The control will no
longer use the Board or Ship objects, since everything is stored in the database. Since ADO does not
support Access QueryDefs correctly, we will use parameterized queries stored in ADO “Command”
objects.

You will make use of the ADO “Command” object. Because VB does not support QueryDefs in Access
properly, you will have to use parameterized SQL and your functions that encapsulate it. For example,
consider the query that returns the record in tblBoard that corresponds to a particular coordinate on a
particular board.

Instead of the Hit function describe in Assignment #4, your BattleshipControl should have the following
function:
CellStatus (x, y) – Returns cell status (equivalent to what the screen displays).
Use the following enum

```vbnet
Public Enum btlCellStatus
    btlUnknown = 0
    btlEmpty = 1
    btlShip = 2
    btlSunk = 3
End Enum
```

**Activity**

Develop this project as you did Assignment #4.

**Submission**

Students should

Turn in at the start of class a hardcopy of the code of the control and the application with a cover page clearly indicating the number and name of the assignment and the student’s name and ID #.

Before class time, submit a folder containing two sub-folders – one with the ActiveX control project, and one with the game application project. The folder should be named with your last name and the assignment number. For example, student Smith would submit a folder titled “Smith_A5” containing two folders: Control and Game, each containing a VB project and associated files.

**Assessment**

This assignment will be assessed on the completeness of the solution to the problem and on the clarity of your program code. Use readable variable and method names. Break large functions and subroutines into small, cohesive ones. Document your SQL.

**Hints and Notes**

1. Implement the login form using two ADO DataControl objects, one that returns the contents of tblUsers and one that calculates the MaxId for the purpose of creating new users. The list of usernames should be a ComboBox control with a separate button for adding new users.

2. Implement the saved game selection form by using the DataGrid control bound to a DataControl that shows the contents of tblGames.

3. Write a series of functions that encapsulate your SQL commands. If we were using a more full-featured database, this would be done with stored procedures. For each command, you will need: a global declaration, a setup function, a BoardId setting function, and an execute function:

   ```vbnet
   Private CmdBoardRecord As ADODB.Command
   Private Sub SetupBoardRecord ()
       Set CmdBoardRecord = New ADODB.Command
       CmdText = "SELECT BoardId,X,Y,Hit,ShipName FROM tblBoard WHERE "
       CmdText = CmdText & "BoardId=paramBoardId AND "
       CmdText = CmdText & "X=paramX AND Y=paramY "
       CmdBoardRecord.CommandText = CmdText
       CmdBoardRecord.CommandType = adCmdText
       CmdBoardRecord.ActiveConnection = ConnBattleDB
   End Sub
   Private Sub SetBoardRecordBoardId (BoardId As Integer)
       CmdBoardRecord.Parameters("paramBoardId").Value = BoardId
   End Sub
   ```
Private Function ExecBoardRecord (X As Integer, Y As Integer, _
ReadOnly As Boolean) As ADODB.Recordset
    Dim BoardRecord As New ADODB.Recordset
    If (ReadOnly) Then
        ' set appropriate parameters
    Else
        ' set parameters
    End If
    CmdBoardRecord.Parameters("paramX").Value = X
    CmdBoardRecord.Parameters("paramY").Value = Y
    Set BoardRecord = CmdBoardRecord.Execute
End Function

You will need at least the following commands:
CmdMaxBoardId: Calculates the maximum board id
CmdPlaceX: Returns a set of records corresponds to cells on the board beginning with some location X, Y and continuing for Size cells in the X direction. Returns only those cells that do not contain a ship.
CmdPlaceY: Same as PlaceX but in the Y direction.
CmdBoardRecord: Described above.
CmdShipRecord: Returns the record for a ship of a given name on a particular board.
CmdBoardCells: Returns all the cells on a given board ordered by X, Y.
CmdBoardShips: Returns all the ships on a given board.
CmdFloatingShips: Returns all the ship on a given board that are not sunk.