This is my Development Report on the development of my First year Introduction to Programming project, specifically the development of a program which is a text based Knights tour run game.

Bare necessities

To begin, I started by creating a basic board, this I represented with a 2d array and through the output block shown below, making it possible for the user to see a representation of the board.

```
TextIO.putln("|   | A | B | C | D |");
TextIO.putln("| 1 | " + Board[0][0] + " | " + Board[0][1] + " | " + Board [0][2] + " | " + Board[0][3] + " |" );
TextIO.putln("| 2 | " + Board[1][0] + " | " + Board[1][1] + " | " + Board [1][2] + " | " + Board[1][3] + " |" );
TextIO.putln("| 3 | " + Board[2][0] + " | " + Board[2][1] + " | " + Board [2][2] + " | " + Board[2][3] + " |" );
TextIO.putln("");
TextIO.putln("");
```

The first version of my board representation

Once the basic board had been implemented I set out to work on the most basic and necessary features of the program; that being the input detection and movement of the knight, the ability to reset the board, a feature to check that the location the user is trying to move to hasn't been used already, a check that the game has been completed and a check that the intended move is legal for the knight piece to move.

Of these features the simplest was the input and movement of the knight, for this, starting on a 3x4 board I took in a string which I then separated into two separate string with the use of the substring feature, and finally I used a String switch statement for both the row and column to see that the move is directed somewhere within the boards parameters, and if it is not it will automatically default to the last case which sets them to 0 (the final case shown is the control case mentioned, and in the switch statement for the Column there is also a case for q this is for when the user wishes to end this attempt and reset the board.) and to get the knight to move the next location, I accessed the Board Variable using the two variables declared and set here. I checked if either of them where equal to 0 to see that the move entered is not valid. If it was valid however I altered the board location to equal the current move set in the board. In the final implementation of these two features the switch statements go up to 5 possible points and the turn is only altered when the board itself is also changed.

3x4 version of the input and move feature

```
Column = Move.substring(0, 1);
Row = Move.substring(1, 2);
```

```
switch(Column){
    case "a":
    case "A":
        File = 1;
        break;
    case "b":
    case "B":
        File = 2;
        break;
    case "c":
    case "C":
        File = 3;
        break;
    case "d":
    case "D":
        File = 4;
        break;
    case "q":
    case "Q":
        File = 5;
        temp1 = 1;
        break;
    case "e":
        File = 0;
        break;
```

```
switch(Row){
    case "1":
        Rank = 1;
        break;
    case "2":
        Rank = 2;
        break;
    case "3":
        Rank = 3;
        break;
    case "u":
    case "U":
        Rank = 5;
        break;
    case "4":
        Rank = 0;
        break;
}
```

```
if(File <= 0 || Rank <= 0){
    TextIO.putln("The Location entered is not valid please try again.");
    turn--;
}else if((File > 0 && Rank > 0) && (File <= 4 && Rank <= 3)){
    Board[Rank-1][File-1] = turn;
}
```

```
do{
    do{
        Board[one][two] = 0;
        two = two + 1;
    }while(two <= 3);
    two = 0;
    one = one + 1;
}while(one <= 2);
```

```
for(one = 0 ; one <= 4 ; one++){
    for(two = 0 ; two <= 4 ; two++){
        Board[one][two] = 0;
    }
}
```

Next of the most basic parts of the program was board reset, my first version of this used two Do, While loops that set the currently selected point on the board to 0, however I later realised that a for loop would be much simpler and require less code while doing exactly what the do, while loops do.

Next I set out to develop the feature that will check the location the character is trying to go to has not already been travelled to this also was quite simple as all I had to do was check if the location was equal to 0 or not, if it was not I would skip over the code that changes the Board location and output text saying the location requested has already been visited, and that this turn would not be counted. – This also shows the final code I had for the turn – And if it was equal to 0 then it would alter the board and change the turn.

```
if(Board[Rank-1][File-1] != 0 ){

    TextIO.putln("The location you are trying to move to has already been used");
    TextIO.putln("the selected move is invalid and therefore will not be counted");

}else if(Board[Rank-1][File-1] == 0){

    Board[Rank-1][File-1] = turn;
    turn++;
```

Next up I began working on making sure the program would not allow you to move move to a position on the board that was not a move that the knight could make according to the rules of chess. This was achieved in a separate class, I used took in the current File and Rank as well as the last File and Rank and through an extensively long If statement determined whether or not the file and rank are a legal move for the Knight. This implementation however was not the first thing I attempted to do this, however this was the first thing I found which worked correctly and in the most concise manner.

```
public static int LegalMove (int File, int LastFile, int Rank, int LastRank){

    /**
     * Taking the current move file, previous move file current Rank and previous move rank this procedure
     * will check to make sure the current move is a legal move.
     */

    int legal = 0;

    if(File != 7){      // this external if statement is to ensure the legality of the move isn't put into questi
        if(((File == LastFile + 1 || File == LastFile - 1) && (Rank == LastRank + 2 || Rank == LastRank - 2)) ||
            ((File == LastFile + 2 || File == LastFile - 2) && (Rank == LastRank + 1 || Rank == LastRank - 1)) ){
                                                    // Legal move checker code
            legal = 1;                              // The extremely long If statement above simply checks th
                                                    // a legal move and then changes the Legal Variable to 1
        }else{                                      // it to 0
            legal= 0;
        }
}
    return legal;
```

And now for the last of the basic and necessary features, which is detecting that game has been complete, for this I kept the code quite simple by implementing an if statement which was activated when the turn counter is greater than the number of positions in the Board array. When this if statement is activated the program outputs its congratulatory statement and then asks if the player wishes to reset the game, as well as asking them if the want to save their board movement to a file.

Extra additions

After I had finished writing the necessary parts of the program, I began to adapt it and make additions to the code. Of the eight suggested additions to the code on the specification I decide only to implement five of them; these where firstly to upgrade the board to a 5x5 board, to allow the user to choose their starting location, detect a dead end in the code, undo the last move and to save the program to a file.

The first of these that I did was implement the 5x5 board, as it would be the most simple of these to implement, seeing as all I had to do was increase the size of the Board array and the switch statements. This process began the end of me being able to easily check the full game for errors due to the fact that I had no clear route through the program to completion.

```
public static int[][] Board =   {   {0,0,0,0,0},
                                    {0,0,0,0,0},
                                    {0,0,0,0,0},
                                    {0,0,0,0,0},
                                    {0,0,0,0,0}
```

Next I made it possible for the user to choose their starting location in the game, this was achieved easily also by implementing a second loop, which contains the main loop. This secondary loop calls the procedure which checks for the users input and detects the location entered, it then sets the initial location to the board location that the user has selected. It also includes all of the necessary validation and checks to ensure that the initial location is on the board, ensureing no errors.

I then implemented the procedure which detects a dead end in the program, this however was not too simple, and also the implementation that has been done may not be the most efficient possible, however it is all I could think of and it is capable of doing what is necessary. I implemented a series of If statements in a procedure, each If statement works independently of the others, and preceding each statement is a set of two variable declarations through the PosFile and PosRank Variables. In all there is eight if statements and eight declarations of the possible File and Rank Variables. These each act to check if the locations which are all possible moves for the knight have been used, so each if statement checks if the location defined using the possible file and rank variables has been used before and if it has it adds one to a variable, then after this procedure is complete, the program checks to see if this variable is equal to eight, and if it is, will tell the user they have reached a dead end and prompt them to either undo their last move or reset the board.

```
        PosRank = Rank + 1;
/*1*/   PosFile = File + 2;
        if((PosRank <= 5 && PosFile <= 5) && (PosRank >=1 && PosFile >=1)){
            if(knight.Board[PosFile-1][PosRank-1] != 0){
                used = used + 1;
            }
        }
```

First part of the dead end check procedure

Next I implemented the possibility to undo the last move that was done. This was quite simple also as it requiredd little work. It is quite simply asking the user if the want to undo their last move and then taking the last file and rank and checking their location on the board and setting that location to 0, then taking the turn and decreasing it by one and returning the value. However if the user decides not to undo their move it will skip the alteration of the board and decreasing the turn and instead just return the turn value that it was given by the main procedure.

```
public static int UndoLast(int LastFile, int LastRank, int turn,boolean Dead){
    /**
     * Taking the last file and last rank from the main class this procedure resets the lo
     * the user decides instead to not undo their last move in which case it simply return
     */

    boolean undo = false;

    if(Dead = false){

    TextIO.putln("");
    TextIO.putln("Are you sure you would like to undo your last move?(enter 'y' if yes)");
    undo = !TextIO.getlnBoolean();

    }

    if(undo != true){

        knight.Board[LastRank-1][LastFile-1] = 0;
        turn--;
        return turn;

    }else{

        return turn;
```

Final implementation of Undo procedure.

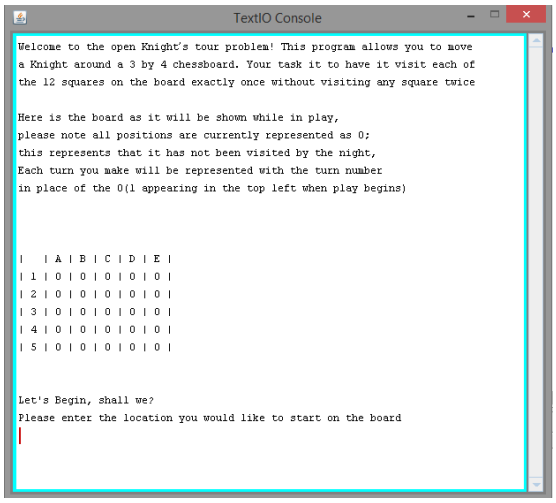
The last part to be implemented to the game was the feature which allowed the user to save the path that they have used in their run through of the program. This feature was relatively simple as it only required me to set the program to print to a different location than the console, and then print out the board to that location instead of the standardoutput. The final implementation of the procedure is more concise and calls on the procedure which prints out the details of the board instead of storing its own version of the same code.

```
String FileName;

TextIO.putln("");
TextIO.putln("Please enter the name you would like to save your file under");
TextIO.putln("WARNING: If the name entered is the same as a current file it will be overwritten");
FileName = TextIO.getln();

TextIO.writeFile(FileName + ".txt");    // this designates the name of the file to be written to, c
                                        // and overwriting it if it does.
TextIO.putln("|   | A | B | C | D | E |");
TextIO.putln("| 1 | " + Board[0][0] + " | " + Board[0][1] + " | " + Board [0][2] + " | " + Board[0]
TextIO.putln("| 2 | " + Board[1][0] + " | " + Board[1][1] + " | " + Board [1][2] + " | " + Board[1]
TextIO.putln("| 3 | " + Board[2][0] + " | " + Board[2][1] + " | " + Board [2][2] + " | " + Board[2]
TextIO.putln("| 3 | " + Board[3][0] + " | " + Board[3][1] + " | " + Board [3][2] + " | " + Board[3]
TextIO.putln("| 3 | " + Board[4][0] + " | " + Board[4][1] + " | " + Board [4][2] + " | " + Board[4]

TextIO.writeStandardOutput();       //this sets the program to now set all output to the standard o
```

First implementation of the save feature

Testing

(All images of Testing have been taken in the final version of the program.)



The program, was tested several times throughout its development, most of the in-development testing was used to test a specific feature of the program, namely whichever feature had just been implemented; to ensure that both that feature worked, and that it did not cause any other features to fail – which happened a few times – and require a lot of the code to be altered to accommodate. This image shows the final version of the program at its launch, showing everything up until the user inputs their starting location, at the top there is the rules of the game for the user to learn, below that is a representation of the board with no moves having been made. Below is a set of test variables that I wil attempt to put into the program as well as what I expect the output to be and what the output actually comes out as.

Input	**Expected output**	**Actual Output**
A1	A1 equals 1	A1 equals 1
0q	Not valid move try again	Not valid move try again
B30fret	B3 equals 2	B3 equals 2
000000000	Not valid move try again	Not valid move try again
D2	D2 equals 3	D2 equals 3
F1	Not valid move try again	Not valid move try again
Q	Ask if want to save route	Ask if want to save route
N	No save, ask if want to reset	Board reset(no choice)

Next I intend to follow through a route to complete the knights tour on the 5x5 board ba1elow is the path

Input: A5, C4, E5, D3, C5, E4, D2, B1, A3, B5, D4, E2, C3, A4, B2, D1, E3, D5, B4, A2, C1, B3, A1, C2, E1, <End>

Expected output: Congratulations you have completed the knights tour, would you like to save you route?

Actual output: Congratulations you have completed the knights tour, would you like to save you route?

As can be seen from this image, the program works exactly as it is intened. The only feature which has not been shown to be tested in this report is the feature to undo your last move, which I have tested many times and am sure works perfectly.

```
Congratulations you have completed the Knights Tour on a 3x4 board
Here is you final board:


|   | A | B | C | D | E |
| 1 | 23 | 8 | 21 | 16 | 25 |
| 2 | 20 | 15 | 24 | 7 | 12 |
| 3 | 9 | 22 | 13 | 4 | 17 |
| 4 | 14 | 19 | 2 | 11 | 6 |
| 5 | 1 | 10 | 5 | 18 | 3 |


Would you like to save your route  to a text file for analysis?(Enter 'y' if so
```

Critical Evaluation

This project involved a lot of work, involving iteration, as I mentioned earlier sometimes when I tested a newly implemented feature another feature would break and need to be altered to fit around the new feature, that or the new feature would have to be rewritten to fit around the current code, which was done less often as I preferred making the code at the core of the program more adaptable than the lesser lying parts of the program. Also the problems in the main code tended to be caused due to the way I had linked the new feature into the main code so the actual problem was infact in the main code.

I would have preferred to spend more time working on this and make the additions of more in-depth features such as being able to undo as many turns as you want or replaying the users moves to a point where they wish to deviate from and try a new path from there. However in the end I decided against implementing these features in order to give myself time to focus on the report and work for other modules, including revision for tests.

Finally I believe that there are obvious improvements that could be made to this program, however most of them would involve the inclusion of images into the program. However from a standpoint which maintains the program as a text game, then the most important improvement that I could imagine would be to make it more readable in the sense that the board tends to go all over the place when you begin to get into a move counter over nine.

Final Evaluation

The development of this program, has allowed me to learn how to program in java further, as I decided to work on this project myself it was required of me to figure out my way around every problem on my own, which is how I tend to prefer how to work, only asking for help when I believe that I cannot find any way around it myself. The complete project has managed to give me the first glimpse of the feeling that I knew I would find from going down this path in life, and that is that of excitement as I got to see something that I developed and created take shape and become a working piece of software. This piece of software was developed in a matter of 2 and a half weeks, and this report created at the end of its development in three days of actual work time. The program as whole was created with care and a

great attention to detail, the program was developed for my first year intro to programming course, also this programs development required a lot of iteration over the pieces of code, including periods of utter hatred towards computers because I could not find why a piece of code was not working, when the answer to that question was right infront of my eyes the entire time, or so simple that not noticing it was and incompetent move to do. Finally the programs development allowed me to experience the development of an application which would allows someone to attempt to solve a problem with their input.