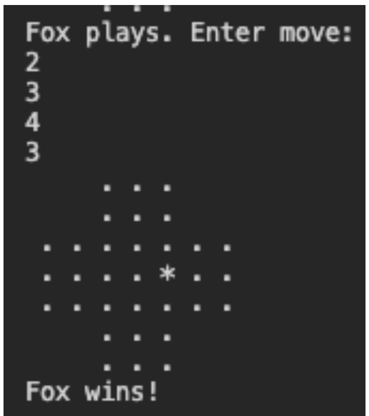
Overview
In this practical we were asked to create a version of the board game 'fox and geese'. We were asked to ensure the validity of moves and check whether one side has won. We were then asked to allow for arbitrary board sizes in our program and to ensure that the number of geese scaled with that change. We were also asked to allow for an increase in the number of foxes and to force foxes to make captures, and to allow for them to 'chain' jumps when taking pieces. I completed all parts of this assignment except the very final section (I have not implemented functionality that enables part 4: Chaining jumps).

Design
I created two classes for the first two parts of the assignment and two more for the latter parts. The first class: 'Board' contains a constructor and a single method. The constructor creates the board and places the required seventeen geese on it, in their required positions. It then does the same for the fox. I chose to first fill the board in with 'INVALID' characters so that there would never be a situation where if the shape of the board was changed some locations in the array might end up with a null value. The same is true for covering the playable part of the board with 'FREE' characters; if the starting positions of geese (or the fox) were ever changed there will not be a case where some part of the board becomes accidentally unplayable. The method is unchanged from the starter code as instructed. In addition, I changed many fields from their original private modifier to public so as to allow them to be easily accessed from other classes as needed.
The second class 'Game' contains a constructor and a single method. The constructor is very simple only creating board and scanner objects. I chose to use only one method to perform the functionality of the game (checking move validity, making moves, and testing win conditions) because I found it easier to test the validity and impact of moves in the same method as the moves are inputted and made. I used a flag variable to control the turns, and whether the game was over, as in both cases there are two situations which are both mutually exclusive and exhaustive, therefore, when one is no longer true the other must be true. I used if statements to ensure that the moves are valid, one for normal movement to check that the piece isn't being moved to an invalid position, and an extra one for the fox to check that the move is valid in the event of a take. To check for fox win states, I used a for each loop which has not been explicitly taught in lectures and that I learned how to use from: https://stackoverflow.com/questions/13383692/for-each-loop-using-2d-array (Accessed November 2020). This allows me to easily check every location in the array for a goose. For the goose win states I used nested for loops instead because I was checking a much smaller and more specific area, which also had to change depending on the position of the fox so that no index errors occurred.
For the 'BoardEx' class I made changes to the constructor in order to facilitate different board sizes. I created three different scenarios based on the size of the board: if the board size can be divided into three equal pieces then construct it so the width of each sector is the same; if the board size modulo three is one then make the central sector one larger than the sides and two larger if it is two. The reason I chose to make the central sector wider in the final case, rather than increasing the width of each outer sector by one, was because I thought it made more sense to have the playable area always increase with increasing board size which is not achieved with the alternative method. I also allowed for the scaling of the number of geese based on the increase in size of the board. I was not exactly sure what the ratio between number of geese and number of playable squares was intended to be, so I decided that it was most likely meant to be roughly half and accommodated for that. Finally, I allowed the increasing of the number of foxes. No changes were made to the method.

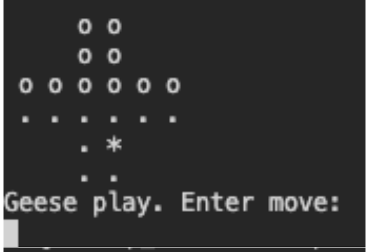
Testing

My program passed all of the tests from the autochecker. In addition, I constructed tests to check the remaining functions of the game and the extension code.



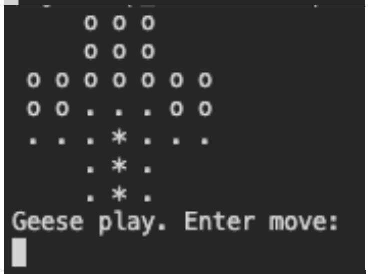
```
Fox plays. Enter move:
2
3
4
3
      . . .
      . . .
  . . . . . . .
  . . . . * . .
  . . . . . . .
      . . .
      . . .
Fox wins!
```

Figure 1: Fox wins by eliminating all geese



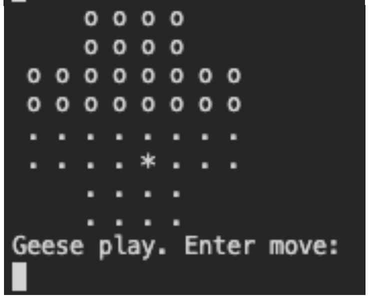
```
   o o
   o o
o o o o o o
. . . . . .
   . *
   . .
Geese play. Enter move:
```

Figure 2: Initialised board size of 6



```
    o o o
    o o o
o o o o o o o
o o . . . o o
. . . * . . .
    . * .
    . * .
Geese play. Enter move:
```

Figure 3: Initialised board size of 7 with multiple foxes



```
    o o o o
    o o o o
o o o o o o o o
o o o o o o o o
. . . . . . . .
. . . . * . . .
    . . . .
    . . . .
Geese play. Enter move:
```

Figure 4: Initialised board size of 8

Figure 1 shows a completed game that has been won by the fox to demonstrate that the code that checks for a fox's victory is functioning correctly. Figures 2, 3, and 4 show the three different states for board initialisation in the 'BoardEx' class and that they are working as intended. They also show the two different states for geese layout and that they are working as intended, at least for these values. Figure 3 also shows the adding of multiple foxes to demonstrate that it functions. In conjunction with the autochecker tests these demonstrate the functionality of all the parts of my code required by the specification.

Evaluation

My code allows for the playing of a version of the boardgame fox and geese with detection for illegal moves and winners. It also provides functionality for playing on different board sizes, scales up the number of geese accordingly and gives the option to play with additional foxes. However, it does not have the ability to force foxes to take geese if they have the option and it does not allow for foxes 'chaining jumps'. In addition, there are certain limitations to the addition of more foxes. Foxes cannot exceed a certain number (dependant on the board size) as I have not included functionality for preventing foxes being placed out of index range and causing an error.

Conclusion

While my code does fulfil the majority of the specification, I would really like to have finished it off given more time. Furthermore, the placement of geese when increasing board size has certain irregularities I would have liked to have corrected. For example, when the board size is set to 9 the layout of geese is not symmetrical, although the number remains accurate. The most difficult part of this assignment was appropriately using methods to create code that is concise and easy to parse. The use of only a single method in the 'Game' class was not ideal and I also would have liked to split it up into multiple methods had I had time to work out the best way to implement that. Similarly, the if statements used to check for illegal moves are somewhat too dense and I would have liked to reformat them to make them easier to parse or perhaps even use some other method for catching errors, such as try catch statements.