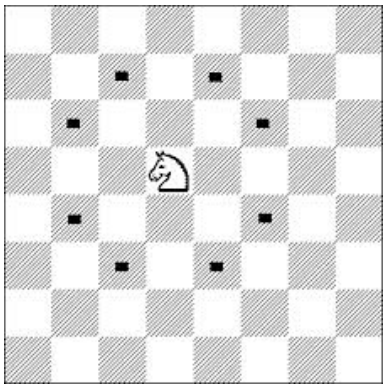
FIT1008 – Intro to Computer Science
Workshop Week 6
Semester 1, 2017

Objectives of this practical session

- To be able to write in and use classes and objects in Python.
- To be able to use a try/except clause and raise exceptions.
- To be able to use assertions to handle pre-conditions.
- To be able to use unit testing appropriately.



The Knight in Chess moves in a way that can be considered to be a combination of two simpler moves, namely either move two squares either horizontally followed by one square vertically, or move two squares vertically followed by one square horizontally.
The purpose of the Tasks in this week's prac is develop a Python program that assists a user to find a sequence of moves that will take a Knight in a circuit around all the squares of the chessboard visiting each square exactly once, before ending at the square where the circuit began.
[1]

Task 1

(i) Write a Python class, *Tour*, that contains a representation of the chess board and where the knight has moved, and the current position of the knight.[2]

(ii) Write a Python program *knight_tour.py* that uses the class *Tour*, and has Menu that with two options:

[1] Provide documentation and testing for each piece of functionality. Your documentation needs to include pre and post conditions, and information on any parameters used. It should also specify which exceptions you are handling. Pre-conditions should be checked using assertions. Remember to keep things separate so that there is evidence that you have done all the questions in the prac. Don't just put everything on a single file/folder but organise your files according to Task numbers.

[2] When designing the class Tour, apply the principles of Encapsulation. For example, visualising a Tour – as a string that can be printed – is the responsability of the class Tour and this functionality should be implemented as a method of the class.

position: which read in a position on the chess board, and displays the chess board with the current position displayed by a *K*, and all the previous positions displayed by *.

quit: which quits the program.

Task 2

(i) Write a method, *next_moves()*, for the class *Tour*, that returns a list of possible next positions for the knight. *Note: this list must not contain any position previously visited by the knight*.

(ii) Modify your Python program, *knight_tour*, so it has the option *start*, which clears the board and reads in the starting position for the Knight, and instead of the option *position* in the Menu, it has options for each possible next position.

Task 3

Using the try/except clause *catch and handle* all the Exceptions that could be raised when reading in the user's input. *Note: You may need to create your own Exceptions.*

Task 4

1. For each file, create a set of tests in a different file that uses unit tests to check that your class works as required. Note that unit tests often focus on the different methods that make up your program. For each unit consider a few test cases. You should aim to have good coverage, which means that a large portion of your code is tested.
2. Use assertion to handle any pre-conditions in the methods of your class.

Task 5

Modify the Python program, *knight_tour*, so that the Menu now contains, as well, the options *solution* and *undo*.

solution: When the user chooses this option the program should determine whether or not there exists a solution from that point on in the tour. If there does exist a solution, the program should print out the board and the board should show, in number order, all the remaining moves. Otherwise if there is no solution the program should print an error message.

undo: When the user chooses this option the program should undo the last move.

Optional Challenge

You can extend your program to implement a game, *Combating Knights*. In this game there are two players, *white* and *black*. Each player has a single knight, and at each turn each player must move their knight to a square which has not been visited by either knight during the game. If a player is unable to move their knight they lose the game. The game starts with *white* first choosing a square to start on, then *black* choosing a square.