Hidden Treasures

by
Bronwyn Conroy, James Leahy, Declan Moore & Eanna O'Donnachadha

Introduction: Problem Description

The desired outcome for this project was to create a working game using Python code that runs in the Linux terminal for a normal user. The game is called Hidden Treasures. The aim of the game is to guess the coordinates of the treasure on the map presented in the terminal.

The game may be accessed via the terminal by entering *python hidden_treasures.py*. The user will receive a message explaining the game and asked select a difficulty ('Easy', 'Medium', or 'Hard'). A 'gameboard' will appear consisting of many '-'symbols as placeholders. The user is asked to enter a row number and then a column number, to guess the coordinates of the treasure. The user gets 5 guesses and if they have not guessed correctly, the game is over. There will be messages for winning, losing, incorrect selection, already used coordinates, and out of bounds selection.

The code behind this program was written by the participants of this project, using information learned in CA278 module, complemented by some individual research. We have included lists, dictionaries, user input, functions, while-loops, for-loops and 'if', 'elif', and 'else' statements.

Program Design

Below the algorithm has been divided into 3 parts:

Part 1: Choosing difficulty to create the game board

An empty list is made as the base of the board. This list is then filled with the '-'symbol. The size of the board is dependent on the difficulty chosen by the user.

In order to ask the user what level they would like, their input is stored in a variable called 'diff' and the raw_input() method is used to select 'Easy, Medium or Hard'. The user input is changed to lowercase to make it case-insensitive. If the user does not enter one of the words "Easy", "Medium" or "Hard" – they will be asked to choose again until they select a valid difficulty level.

The user's selection is checked by a dictionary, which will have the corresponding board size. The board is then created by appending the symbol '-' as many times as listed in the dictionary holding the difficulty and board sizes.

Part 2: Generating a random coordinate

We used 'randrange' from the 'random' module to generate a random row and random coordinate as the coordinate of the treasure.

Part 3: Introducing and playing the game

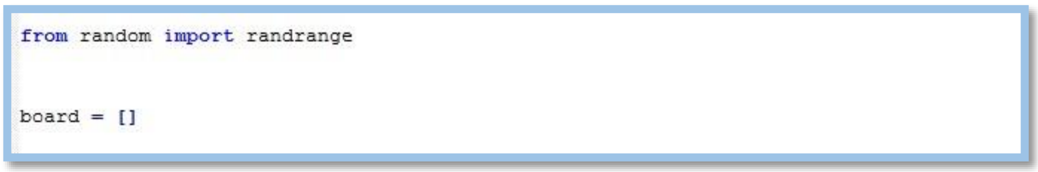
The maximum number of attempts a user can make is saved into a variable called 'turn'. A for-loop is then used to allow the user to make 5 attempts, the loop is completed one additional time to output a game over message where applicable.

The game rules are then applied. Each time the user makes an attempt by entering coordinates, the programme runs through the ‘if’, ‘elif’, and ‘else’ statements in the following order:

1. The user’s inputted coordinates are compared to the randomly generated coordinates. If correct, the guessed coordinate on the ‘gameboard’ is changed from a ‘-‘ to an ‘X’ to indicate a correct guess. A break is applied to the end the for-loop (and therefore the game). If incorrect, the for-loop then cycles on to the ‘else’ statement and the guessed coordinate on the ‘gameboard’ changes from a ‘-‘ to an ‘O’ to indicate an incorrect attempt. An incorrect attempt will count towards 1 of the 5 available attempts.
2. The programme checks if the user has made their 5 attempts and still failed. If so, a message will be printed indicating that the game is over, notifying the user of the correct coordinates. If the user has not used up their 5 attempts, they are invited to guess again.
3. The guess is then checked if it is within the size of the board – if it’s not, the user will be notified, asked to guess again, and will have used 1 of their 5 available turns.
4. The guess is checked if it has already been guessed by seeing if that coordinate in our list has been changed form a ‘-‘ symbol to an ‘O’, the user will be notified, asked to guess again, and will have used 1 of their 5 available turns.

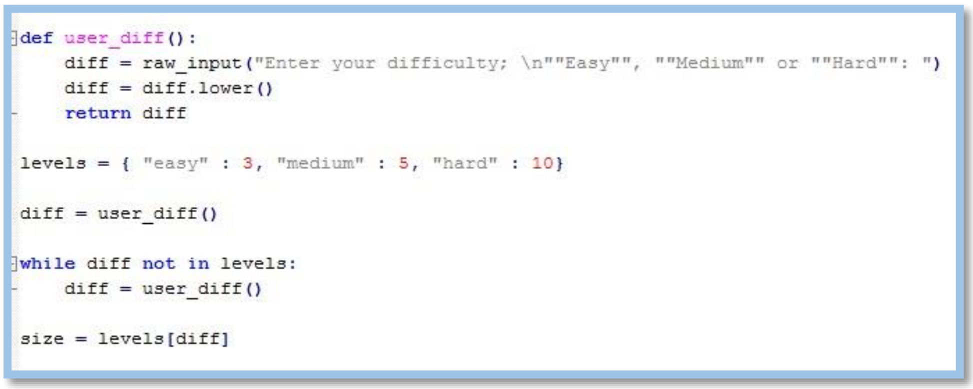
Algorithm

Part 1: Choosing difficulty to create the game board



```
from random import randrange


board = []
```

| *Here, we import the object ‘randrange’ from the ‘random’ module (see later, in part 2). An empty list is created to enter items into to make the ‘gameboard’.*



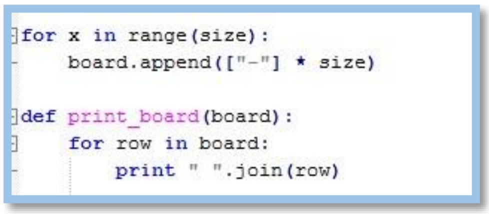
```
def user_diff():
    diff = raw_input("Enter your difficulty; \n""Easy"", ""Medium"" or ""Hard"": ")
    diff = diff.lower()
    return diff

levels = { "easy" : 3, "medium" : 5, "hard" : 10}

diff = user_diff()

while diff not in levels:
    diff = user_diff()

size = levels[diff]
```

| *Here, a function is defined to ask the user to enter a difficulty, and to change the entry to lowercase.*

| *A dictionary is created which holds the difficulty and corresponding size this is stored in the variable size by **size = levels[diff]***

| *The function ‘diff’ calls the **user_diff()** function to ask the user to enter a difficulty level.*

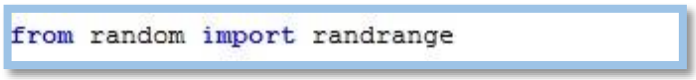
- *A while-loop checks if the user has entered one of the options held in the dictionary, if not, it calls the function **user_diff()** again to ask the enter to enter a word again.*



```
for x in range(size):
    board.append(["-"] * size)

def print_board(board):
    for row in board:
        print " ".join(row)
```

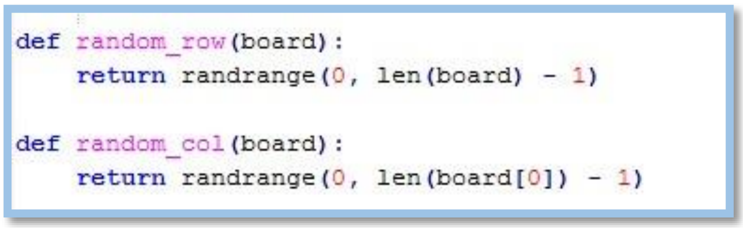
- *The for-loop creates a row according to the size listed in the dictionary above, and does this x times for the columns.*
- *The **print_board(board)** function is defined to join the row by a space.*

Part 2: Generating a random coordinate



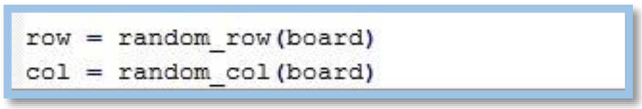
```
from random import randrange
```

- *From above, the 'randrange' object from the 'random' module was imported.*



```
def random_row(board):
    return randrange(0, len(board) - 1)

def random_col(board):
    return randrange(0, len(board[0]) - 1)
```

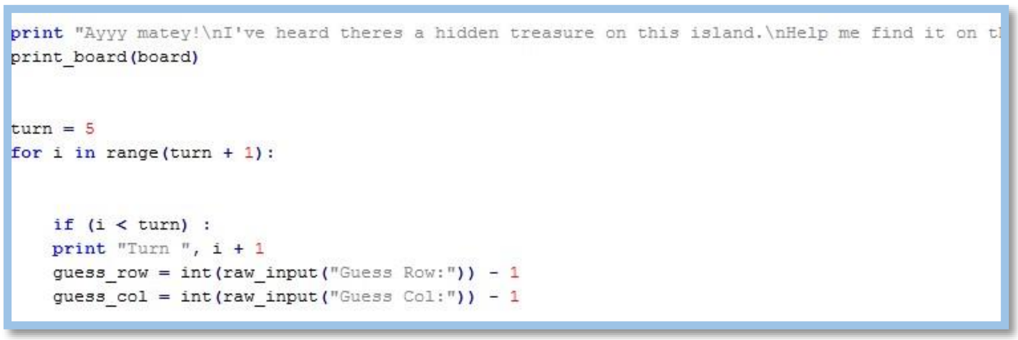
- *The function 'random_row' chooses a random number from the range of the size of the board.*
- *The function 'random_col' chooses a random number from the range of the size of the board.*



```
row = random_row(board)
col = random_col(board)
```

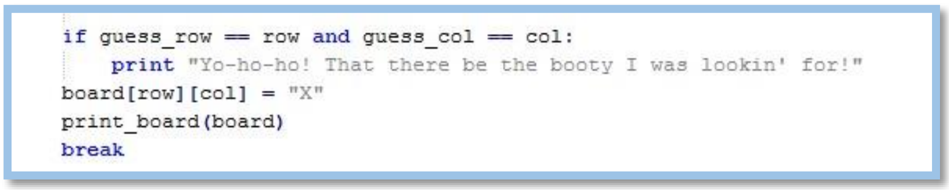
- *These functions were called and stored in variables 'row' and 'col'.*

Part 3: Introducing and playing the game



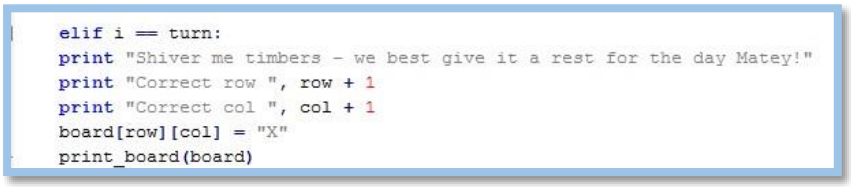
```
print "Ayyy matey!\nI've heard theres a hidden treasure on this island.\nHelp me find it on t
print_board(board)


turn = 5
for i in range(turn + 1):


    if (i < turn) :
    print "Turn ", i + 1
    guess_row = int(raw_input("Guess Row:")) - 1
    guess_col = int(raw_input("Guess Col:")) - 1
```

- *The board is introduced and the maximum tries a user is allowed is stored in variable 'turn'.*
- *A for-loop runs 6 times – allowing for 5 guesses and a final run through the loop to print a game over message (if applicable).*
- *While the user has not used all their turns, they will be asked to enter a guess each turn.*



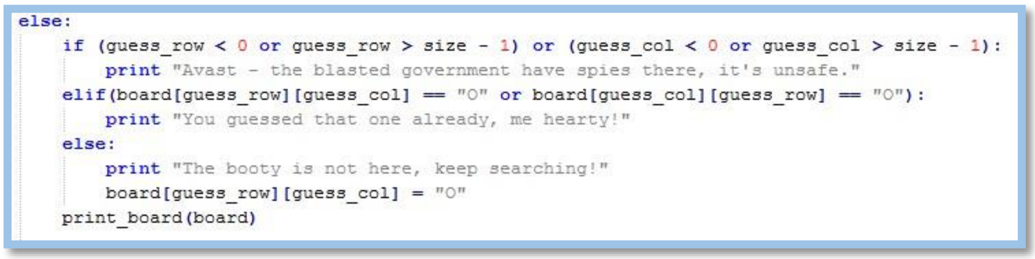
```
    if guess_row == row and guess_col == col:
        print "Yo-ho-ho! That there be the booty I was lookin' for!"
    board[row][col] = "X"
    print_board(board)
    break
```

- *The user's 'guess_row' and 'guess_col' is compared to the randomly generated 'row' and 'col' to see if the answer is correct, if so – the coordinate in the 'gameboard' is changed to an 'X'.*
- *The break makes the for-loop stop and ends the game.*



```
    elif i == turn:
    print "Shiver me timbers - we best give it a rest for the day Matey!"
    print "Correct row ", row + 1
    print "Correct col ", col + 1
    board[row][col] = "X"
    print_board(board)
```

- *The amount of times the for-loop has run is compared with the maximum attempts allowed. If they are equal – the user's attempts are up – they are shown the correct answer on the board and given a game over message as above.*



```
else:
    if (guess_row < 0 or guess_row > size - 1) or (guess_col < 0 or guess_col > size - 1):
        print "Avast - the blasted government have spies there, it's unsafe."
    elif(board[guess_row][guess_col] == "O" or board[guess_col][guess_row] == "O"):
        print "You guessed that one already, me hearty!"
    else:
        print "The booty is not here, keep searching!"
        board[guess_row][guess_col] = "O"
    print_board(board)
```

- *Finally, if the user is not correct or out of attempts – the system checks the coordinates:*
    - *If the guess is (less than or greater than the size of the board's rows) or is (less than or greater than the size of the board's columns) it is an invalid input and the user is notified with an output message. This then goes through the for-loop again for the user's next turn.*
    - *If the guessed coordinate has already been guessed and changed to an 'O' in the list board, then it has already been guessed and the user is notified with a print message. This then goes through the for-loop again for the users next turn.*
    - *If the guessed coordinate does not match the random coordinates generated – it is incorrect and changed to an 'O' in the list board, then it has already been guessed and the user is notified with a print message. This then goes through the for-loop again for the user's next turn.*

Individual Contribution

Bronwyn Conroy – 15357066

- Concept design.
- Functions – ‘user_diff’, ‘random_row’ and ‘random_col’.
- Dictionary and creating the board size.
- Debugging in the if-statements.

James Leahy – 15464928

- Concept design.
- Functions – ‘Print_board’.
- While loop to allow user to re-enter level.
- Debugging in the if-statements.

Declan Moore - 15375071

- Functions and user input – ‘user_diff’.
- If statements – Checking if attempt is invalid.
- Debugging in the if-statements.

Eanna O Donnchadha – 15482802

- Concept design.
- Changing guessed coordinates to an ‘O’ in list.
- Debugging in the if-statements.