Treasure Hunt CA Write-up

1. User needs 2. Game rules 3. Write up general 4. Diagram 5. Pseudocode 6. Code 7. Testing 8. Explanation

1 – User needs

* This program will be a game where the user hunts for treasure in a grid
* The user will need to be able to see a menu interface which has the options of starting a game and also quitting the game
* If the user decides to quit, the program must exit successfully
* If the user decides to start a game, the program must generate a board with the correct amount of treasure chests and bandits in it
* The user will need to be able to move around the board, and choose if they wish to search that grid, or simply move on
* If the user decides to search the square, the program will need to recall if there is a treasure chest or a bandit there, or neither
* If the user finds a treasure chest, 10 gold coins are added to their total
* The user must be able to see the number of gold coins they currently possess
* If the user finds a bandit, the bandit will steal all of the coins that the player currently possesses
* If the user finds nothing, a dialogue will appear that informs them of this
* The user will also need to see how many treasure chests and bandits are present on the screen
* The user can visit the same treasure chest three times before it changes into a bandit
* The user needs to be able to easily see when they win or lose, and have the option to start a new game or quit

2 – Game rules

* There is an eight by eight grid
* There are ten treasure chests and five bandits
* The player starts in the bottom left corner of the grid
* The goal is to find the treasure chests and avoid to the bandits
* The player chooses how many squares they want to move up or down then how many squares they want to move left or right
* If the player lands on a square that contains a treasure chest collect 10 gold coins
* If the player lands on a square that contains a bandit steals all the coins
* The player cannot move beyond the limits of the grid
* Treasure chest can be visited three times then replaced by a bandit in the same square
* With each move, the grid showing the new position of the player is displayed, the current number of gold coins collected and the number of bandits and treasure chests in the grid
* The player wins if they have collected 100 gold coins
* The player loses the game if all the treasure chests have been changed into bandits and player has not collected 100 gold coins

3 – Write up General

Pt 1: Programming techniques.

Programming techniques are small portions of code that allow your program to look and function far better than if you were to simply hardcode it. These are often used to repeat or display items, without the need for other more complicated methods of doing the same thing.

The following will be displayed in the following format:

-Name:

Description of use

Alternatives available

Example

Some simple yet useful programming techniques include:

-Print:

Can be used to display information or results to a user, but also allows the user to input data via the ‘input’ code.

There are no real alternatives to print, aside from using a drawing program such as pygame to draw the results, which is far more inefficient.

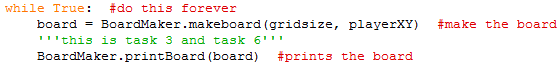


Here the print statement is used to tell the user information.

-While:

Tells the program to repeat a set section of code loop runs until a condition is met.

A for loop could be used in its place, however a for loop only runs a set number of times, but a while loop run until a condition is met.

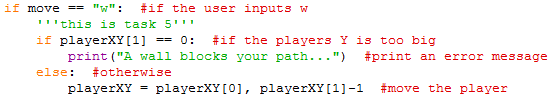


Here the while loop runs the code forever to run the game.yh 6

-If:

If a condition has occurred, the program will run the code, however if the condition has not been met, it will run any elif’s or else’s that are attached or just skip the if statement.

A while loop could be used in its place, but the while loop would need to end after on run.

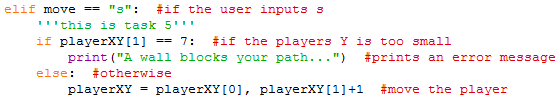


This if loop is used to check if the player pressed w.

-Elif:

If the condition of an if loop has not occurred, this will check if a different condition has been met, and if it has not it will skip this and end the if loop if nothing more is present.

A separate if loop could be used, however this may cause more problems.



This elif is used to check if the move was s instead.

-Else:

If none of the conditions in an if loop are met, and none of the elif conditions are met either, the else loop will run, and this is what will run in every situation that the if loop fails.

You could use an if loop with the != operation, so in any other case that is not the first if, it will run.



This else is used to tell the user that their input is not valid.

-Function:

Allows the program to have a repeatedly callable segment of code that can be ran from anywhere in the code.

Copying a section of code multiple time could be used, however it is very inflexible and inefficient.



This function is to search the square that the player is in.

Pt 2: Data structures are pieces of code that store data for the program.

-Variable:

Stores basic information, in the format of an integer, float or string. An integer is a whole number, a float is any number with a decimal and a string is a line of text.

Variables can be replaced by lists, but a variable is more compatible with other processes.

|  |  |  |
| --- | --- | --- |
| gridsize | Pre-game determining var | Determines the size of the grid |
| playerXY | Used throughout the program | Determines the player’s location |
| board | Used when making the board | Stores the board for use |
| pastPos | Used to erase previous char | Stores the player’s last position |
| notreasure | Used in making the board | Determines the number of treasures |
| nobandit | Used in making the boars | Determines the number of bandits |
| coins | Used as winning criteria | Stores the number of coins the player owns |
| square | Used to fill in board | Stores key pattern for a blank space |
| player | Used to fill in board | Stores key pattern for a character |
| row | Used to make board | Stores each row |
| boardbase | Used to make board | Stores where chests and bandits go |
| rowbase | Used to make board | Stores where chests and bandits go |
| blank | Used to fill pseudoboard | Stores what blank spaces look like on the pseudoboard |
| treasure | Used to fill pseudoboard | Stores what treasure spaces look like on the pseudoboard |
| bandit | Used to fill pseudoboard | Stores what bandit spaces look like on the pseudoboard |
| treasureplace | Used to fill pseudoboard | Stores where treasures go |
| banditplace | Used to fill pseudoboard | Stores where bandits go |
| dier | Used when moving | Repeats asking what move you want to make until it gets a valid answer |

-List:

Stores multiple variables in one structure, used mainly for storing items to be recalled all at once.

Dictionaries are an alternative, and specific items can be selected and be changed.

Playermoves = [“w”, “w”, “a”, “ d”, “s”, “s”, “d”]

Pt 3: How the program is robust

These are techniques that mean the program will not break.

-Using external libraries:

By using libraries that come with python that were made by experienced coder, I can assure that my code is of a high quality, as these external libraries are refined from a lot of use and abuse. An example of an external library I used would be the ‘random’ library, used to generate a random number between two numbers chosen by the coder.

-I commented consistently throughout my code:

By commenting almost every line, I can easily see what my program is doing at any given point, without any knowledge of python or any other code. This means any other users who are altering or viewing my code can easily see what the code means without an in-depth knowledge of code.

-Checking the option that the user chooses on the menu is correct:

When the user is selecting an option in the main menu, this program checks each option and if it does not fall into

One of the valid options, it inform the user that answer is not valid and asks again

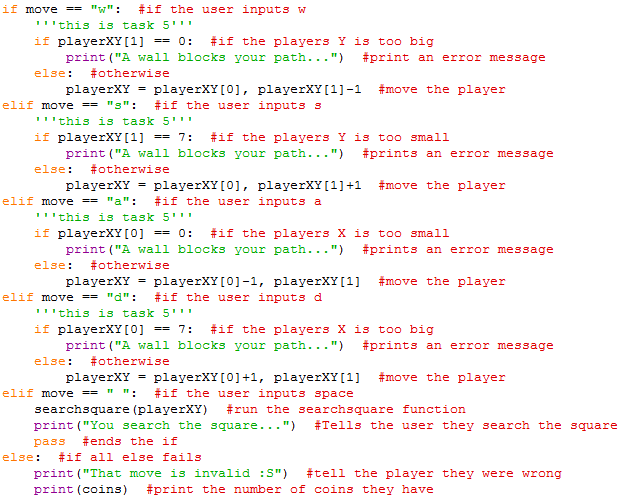


Pt 4: How the program is efficient

How efficient my program is how well my code can run quickly, putting any code that is repeated into functions so it can be faster, and also, I check the answer that the user inputs before it gets to a later part of the code and crashes the program. Here are a few things I did in my program to make it more efficient:

-Checking answers:

I used a simple ‘if’ loop to check if the answer the user inputted was within the acceptable boundaries,



-Everything is in functions:

Most of the code is in functions, allowing for maximum flexibility to call other functions from anywhere at and any time, also this means that all the code is sorted into specific sections, increasing readability and error checking ease.

-Python shortcuts:

Instead of using ‘variable = +1’ I used ‘variable += 1 for simplicity and looking good in the code.

-Comments:

Throughout my code, I used comments to describe what the code does, meaning if I come back to the code later I can easily pick up where I left off, without an in-depth knowledge of python. The comments were meant to be detailed enough to allow someone who doesn’t know any python should be able to get a vague idea of what the code is doing at that given point.

-Modules:

I used an in-built python module, randint. This module allows me to select a random number between two other numbers. I used this module not only to make the code shorter, but also because this module was created by coders far more skilled than me, meaning it is a strong link in the code.

Pt 5:

This is a full basic run-through of what the program should do:

-Run program

Dialogue appears asking if you want to start the game

User inputs ‘N’

-Program closes

-Run program

Dialogue appears asking if you want to start the game

User inputs ‘Y’

Grid displays

Program asks user what direction they want to move

User inputs ‘w’

Program checks if input is valid

Input is valid

Program moves player up

Program asks user what direction they want to move

User inputs ‘d’

Program checks if input is valid

Input is valid

Program moves the player right

Program asks user what direction they want to move

User inputs ‘s’

Program checks if input is valid

Input is valid

Program moves the player down

Program asks user what direction they want to move

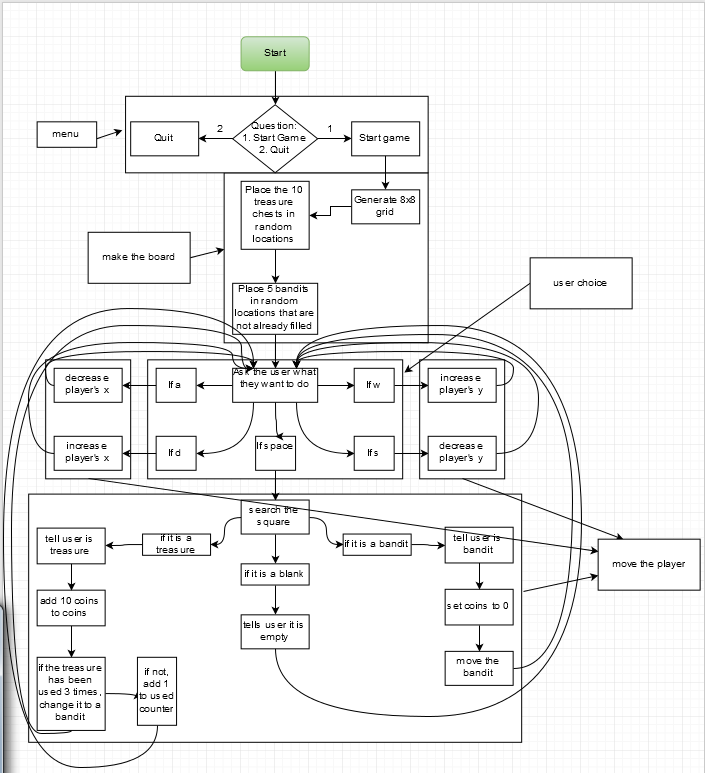
User inputs ‘a’

Program checks if input is valid

Input is valid

Program moves the player left

4 – Diagram



If coins is equal to or bigger than 100

6

3

5

Display board

2

4

1

5 – Pseudocode

from time import sleep

from random import randint

define function makeboard(gridsize, playerXY) #function to make the gameboard

board is [] #a list to store the board

square is "[ ]" #a variable to store the blank square character

player is "[X]"

row is [] #a list to store the row

for c in range(0, gridsize) #a for loop to make the row characters

row.append(square) #adds the blank character to row

for r in range(0, gridsize) #a for loop to output the row characters

row is list(row) #set row to the new value of row

board.append(row) #adds the row character to board

board[playerXY[1]][playerXY[0]] is player

return board #sends board out

define function printBoard(board) #function to output the board

for r in board #for every character in board

output (r) #output the character

define function checkBlank(board, playerXY)

if board[playerXY[0]][playerXY[1]] is "{\*}"

return "blank"

elif board[playerXY[0]][playerXY[1]] is "{^}"

return "treasure"

elif board[playerXY[0]][playerXY[1]] is "{#}"

return "bandit"

else

return "error"

define function makeblankboard(gridsize, nobandit, notreasure)

'''this is task 2'''

boardbase is [] #a list to store the treasure chests and bandits

rowbase is [] #a list to store the rows

blank is "{\*}" #blank square

treasure is "{^}" #treasure square

bandit is "{#}" #bandit square

for x in range(0,gridsize)

rowbase.append(blank)

for v in range(0,gridsize)

roww is list(rowbase)

boardbase.append(roww)

for i in range(0, notreasure)

treasureplace is randint(0, gridsize), randint(0, gridsize)

treasure is treasureplace[0], treasureplace[1]

for i in range(0, nobandit)

banditplace is randint(0, gridsize), randint(0, gridsize)

bandit is banditplace[0], banditplace[1]

return boardbase

define function searchsquare(board, playerXY) #define function a function

boaard is checkBlank(board, playerXY) #use an outside function to do something

return(boaard)

define function gamefunc(playerXY, board, coins, gridsize) #do this forever

while True

if coins >is 100

output ("You win!")

sleep(1)

quit()

board is makeboard(gridsize, playerXY) #make the board

'''this is task 3 and task 6'''

printBoard(board) #prints the board

'''this is task 1'''

move is input("Which direction do you wish to go? (wasd) ") #asks the user how they would like to move

'''this is task 4'''

if move is "w" #if the user inputs w

'''this is task 5'''

if playerXY[1] is 0 #if the players Y is too big

output ("A wall blocks your path...") #output an error message

else #otherwise

playerXY is playerXY[0], playerXY[1]-1 #move the player

elif move is "s" #if the user inputs s

'''this is task 5'''

if playerXY[1] is 7 #if the players Y is too small

output ("A wall blocks your path...") #prints an error message

else #otherwise

playerXY is playerXY[0], playerXY[1]+1 #move the player

elif move is "a" #if the user inputs a

'''this is task 5'''

if playerXY[0] is 0 #if the players X is too small

output ("A wall blocks your path...") #prints an error message

else #otherwise

playerXY is playerXY[0]-1, playerXY[1] #move the player

elif move is "d" #if the user inputs d

'''this is task 5'''

if playerXY[0] is 7 #if the players X is too big

output ("A wall blocks your path...") #prints an error message

else #otherwise

playerXY is playerXY[0]+1, playerXY[1] #move the player

elif move is " " #if the user inputs space

result is "\\#~ERR.EXE IS NOT RESPONDING~#//"

result is searchsquare(board, playerXY) #run the searchsquare function

output ("You search the square...") #Tells the user they search the square

output ("You find a", result)

else #if all else fails

output ("That move is invalid S") #tell the player they were wrong

output (coins) #output the number of coins they have

#haven't done task 8 yet

define function yetanotherstarter()

gridsize is 8 #sets the size of the grid

playerXY is 0, 7 #sets the player's original X and Y

board is makeboard(gridsize, playerXY) #set a variable to a function

pastPos is 0,0 #used for removing the previous X

notreasure is 10 #sets the number of treasures

nobandit is 5 #sets the number of bandits

coins is 0

ye is str(input("Do you want to start the game? "))

if ye is "Y" or "y" or "Yes" or "yes"

makeblankboard(gridsize, nobandit, notreasure)

gamefunc(playerXY, board, coins, gridsize)

else

output (";-;")

sleep(1)

quit()

yetanotherstarter()

6 – Code

1 2 3 4 5

1 1 2 3 4 5

2 6 7 8 9 10

3 11 12 13 14 15

4 16 17 18 19 20

5 21 22 23 24 25