

GCSE Computer Science

Gaming Scenario
Student 2 – Annie

Bot Mod User needs

The following is a description regarding the desired outcome of the programme and its features.

- A.N. Other

Candidate number: 0000

Centre number: 00000

User needs

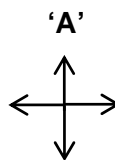
Introduction:

I have been instructed to program a game under the name BotMod. BotMod is a game in which you control a robot as he traverses landscape, in search of people to carry to the base camp.

Basic Movement and Passengers:

BotMod traverses a grid in which he can only move one square per-turn and only in the directions shown in 'A'. BotMod has specific movement capabilities. He cannot make diagonal moves or make a move that will take him out of the confines of the grid.

Additionally, BotMod has more specific rules concerning people and passengers, If BotMod has space within its passenger bay, any people encountered will automatically be picked up therefore if no space is available in the passenger bay, the person remains in their position. If and when BotMod reaches base camp, all stored passengers will be automatically unloaded, emptying the loading bay.

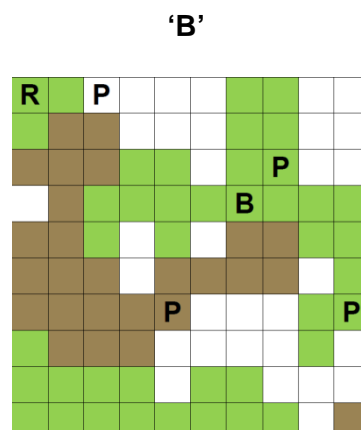


Power and Modified Movement:

BotMod starts with 150 power units. Before the player starts the game they have the opportunity to modify their BotMod, changing traction and passenger bay size, these modifications will have explained advantages regarding movement and space but will have balanced disadvantages in power consumption, making the user think more in depth about their ideal and efficient BotMod.

Display and recognition:

The Game will have a start menu with the options for the user to modify their robot, play the game and exit the game. Before each turn the user will be shown the landscape (see 'B'), which includes people, base camp and the BotMod's position, as well as the quantity of power units remaining in the robot and the number of passengers being carried, alongside the max number of passengers the user's BotMod is able to carry. This will influence and guide the user's decision and dictate their course of action. The game will also display a custom message for the user when the game has finished, telling the user of their game's outcome.



Bot Mod Flowchart

The following is the flow chart I created for the Bot Mod programme (see 'BotModt.py print window')

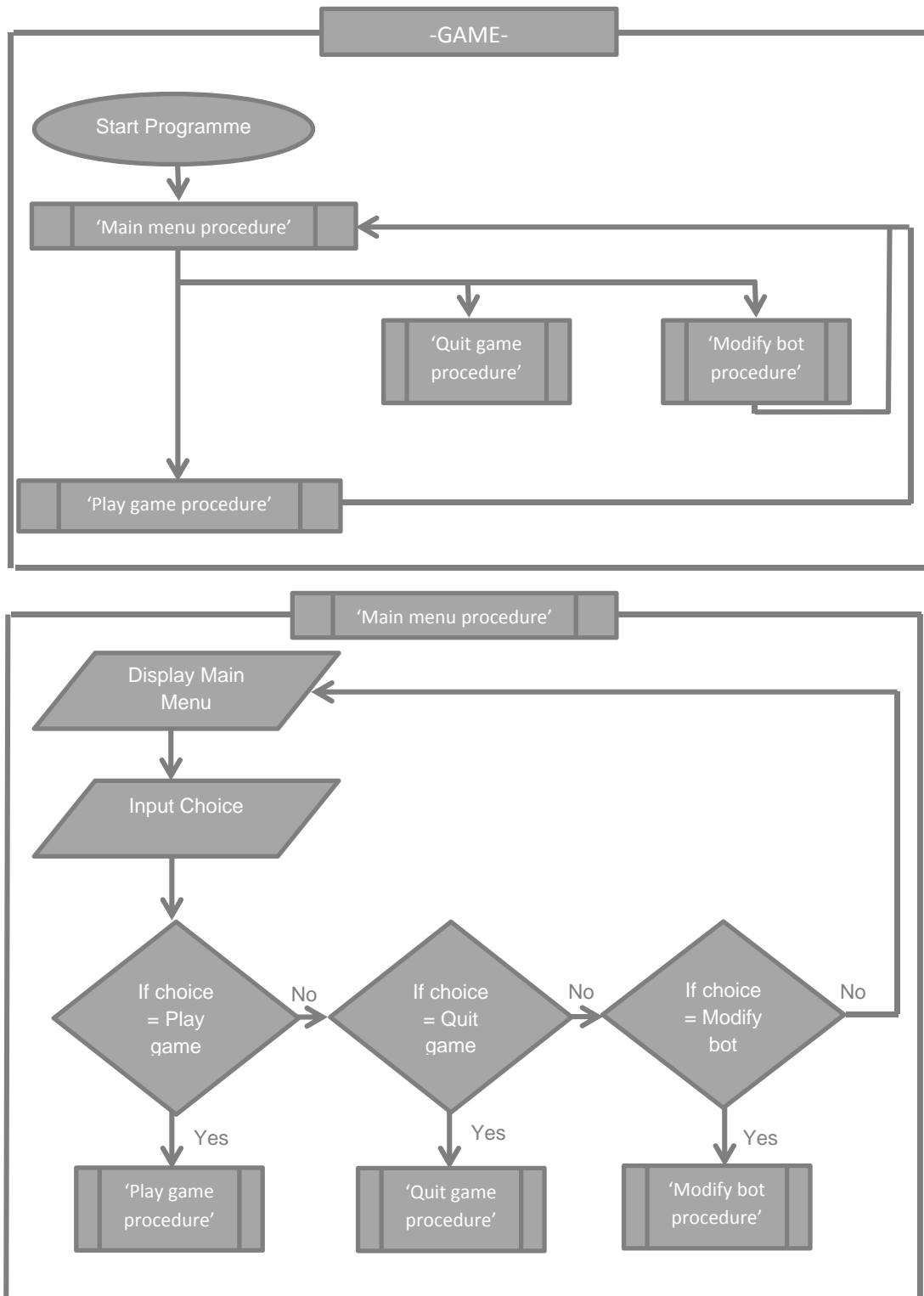
- A.N. Other

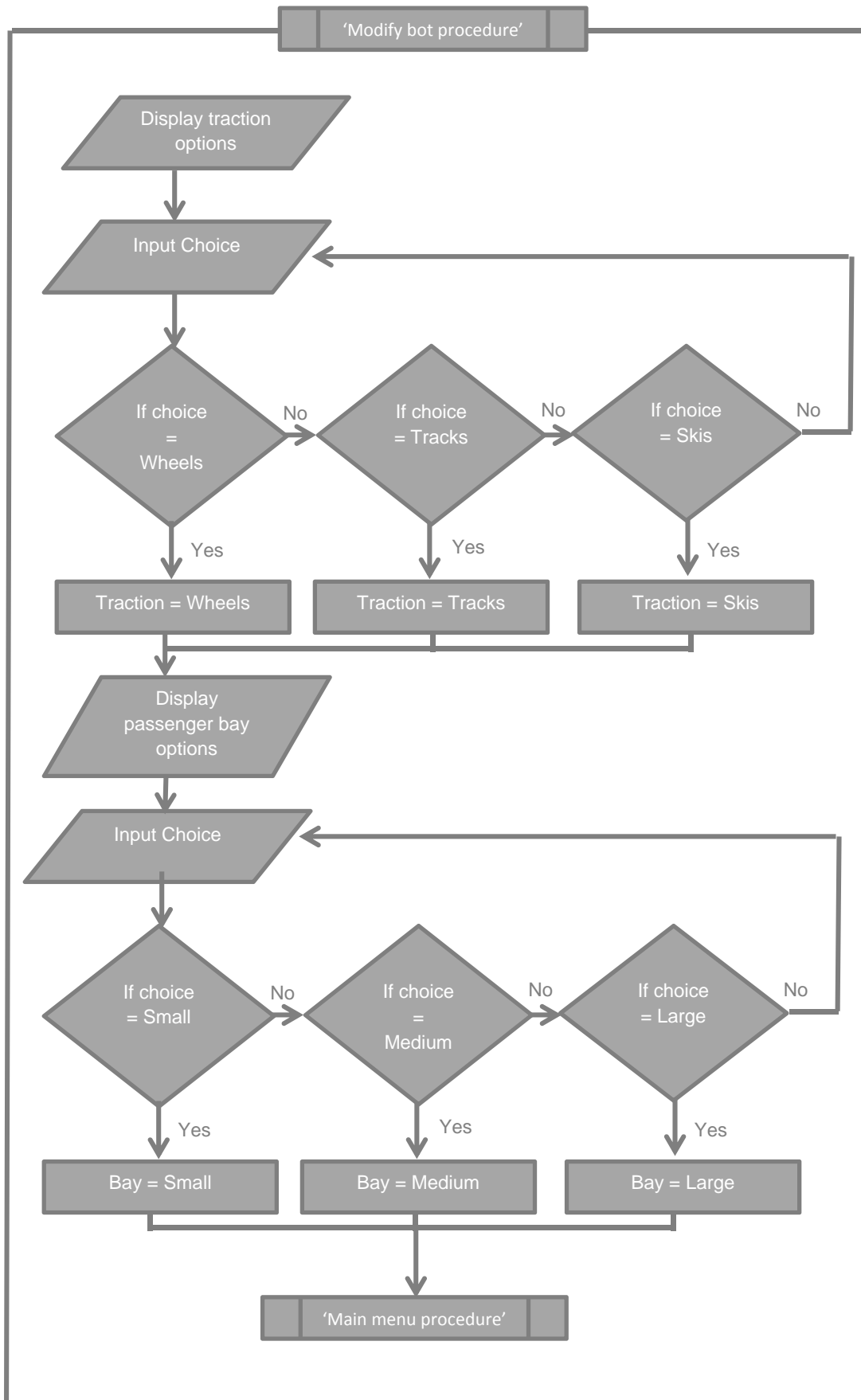
Candidate number: 0000

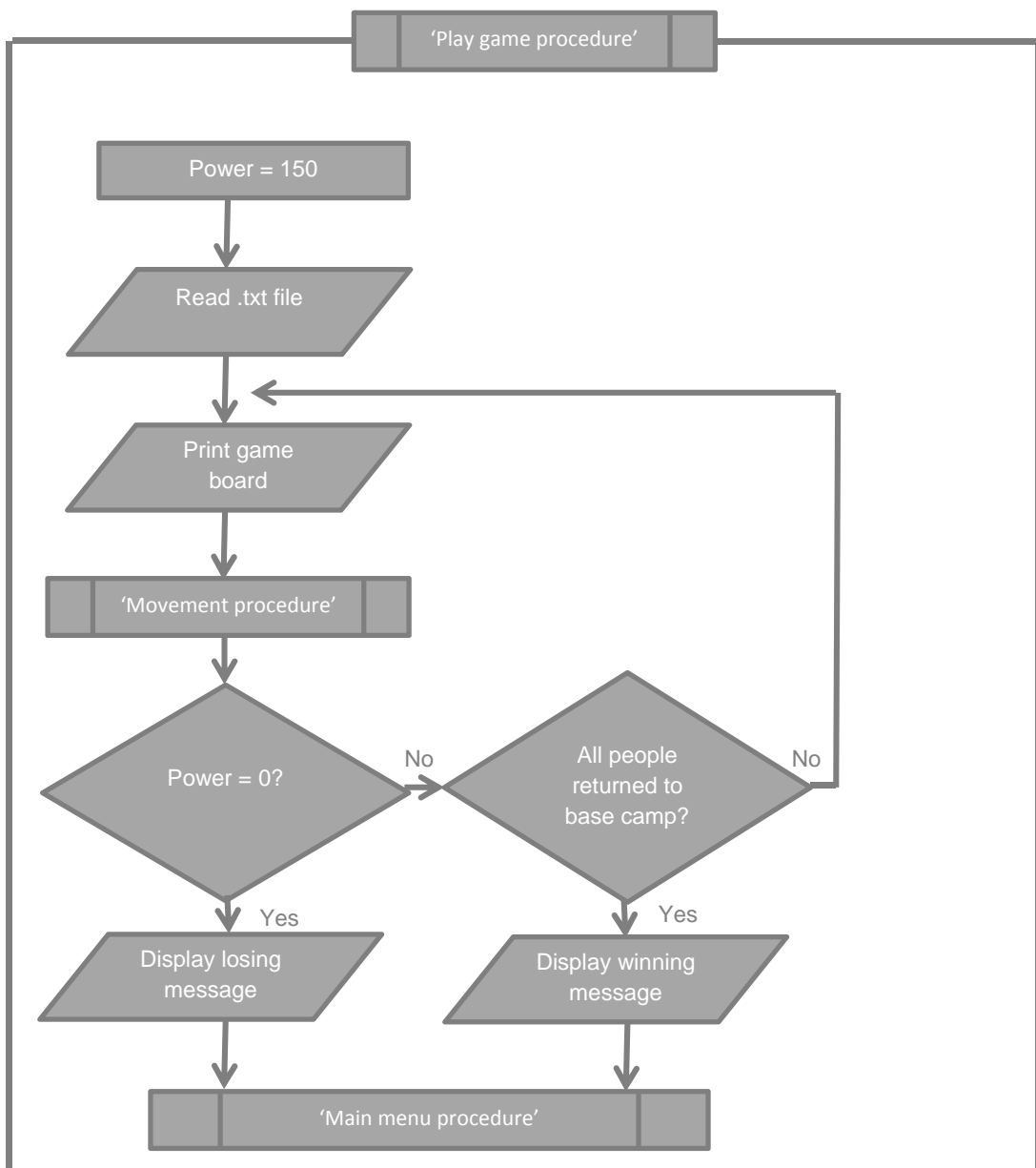
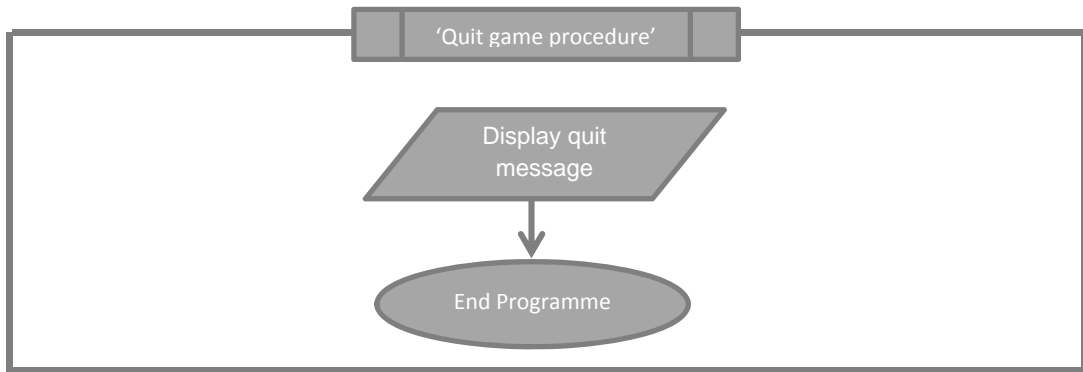
Centre number: 00000

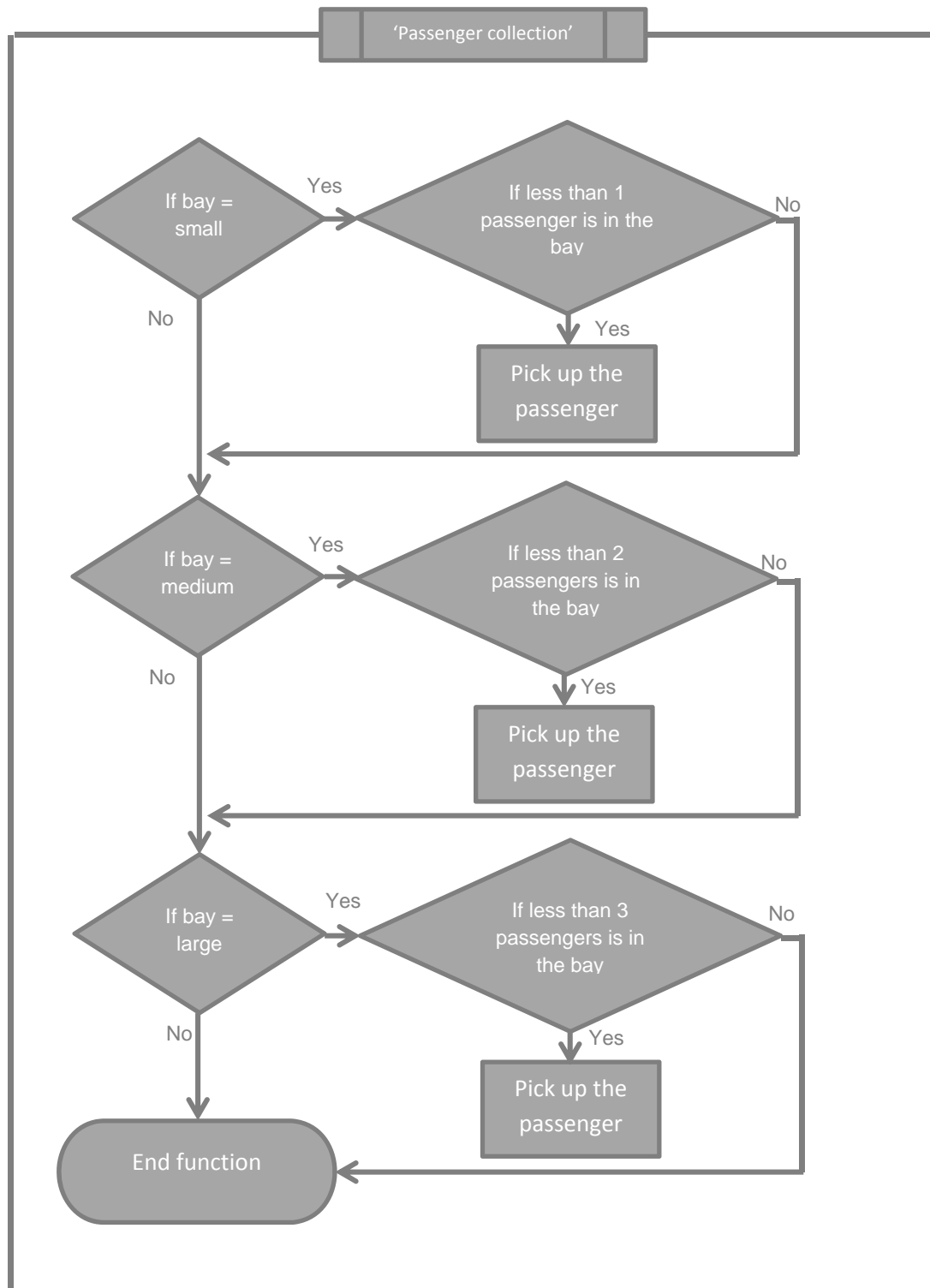
Bot Mod Flow chart

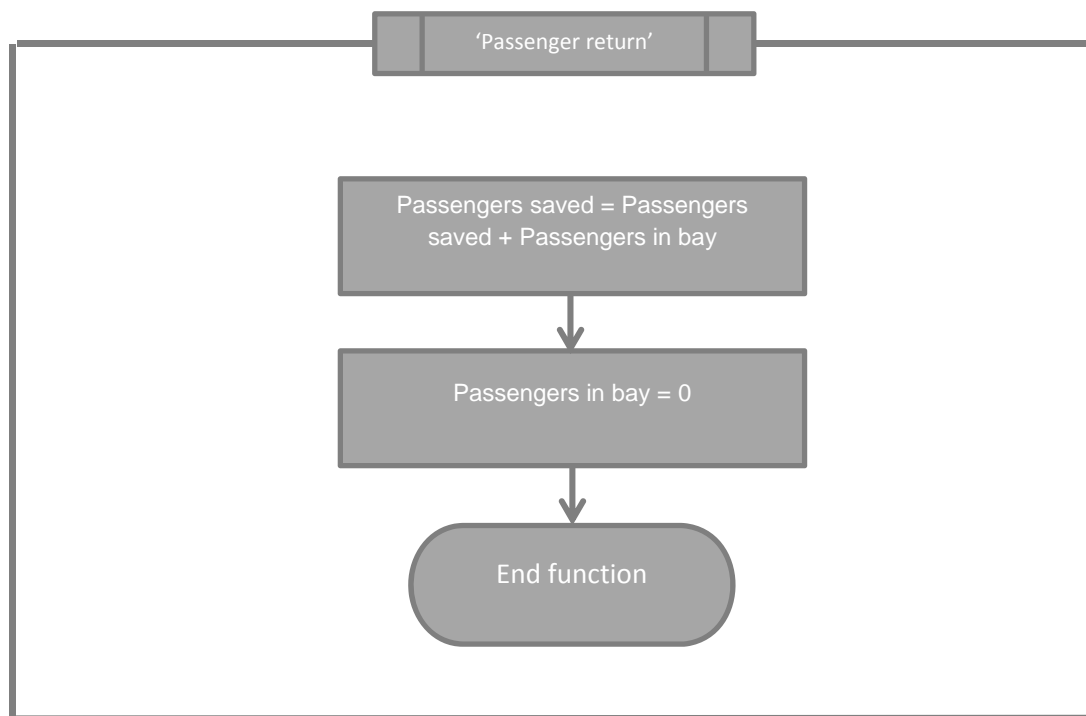
A.N. Other

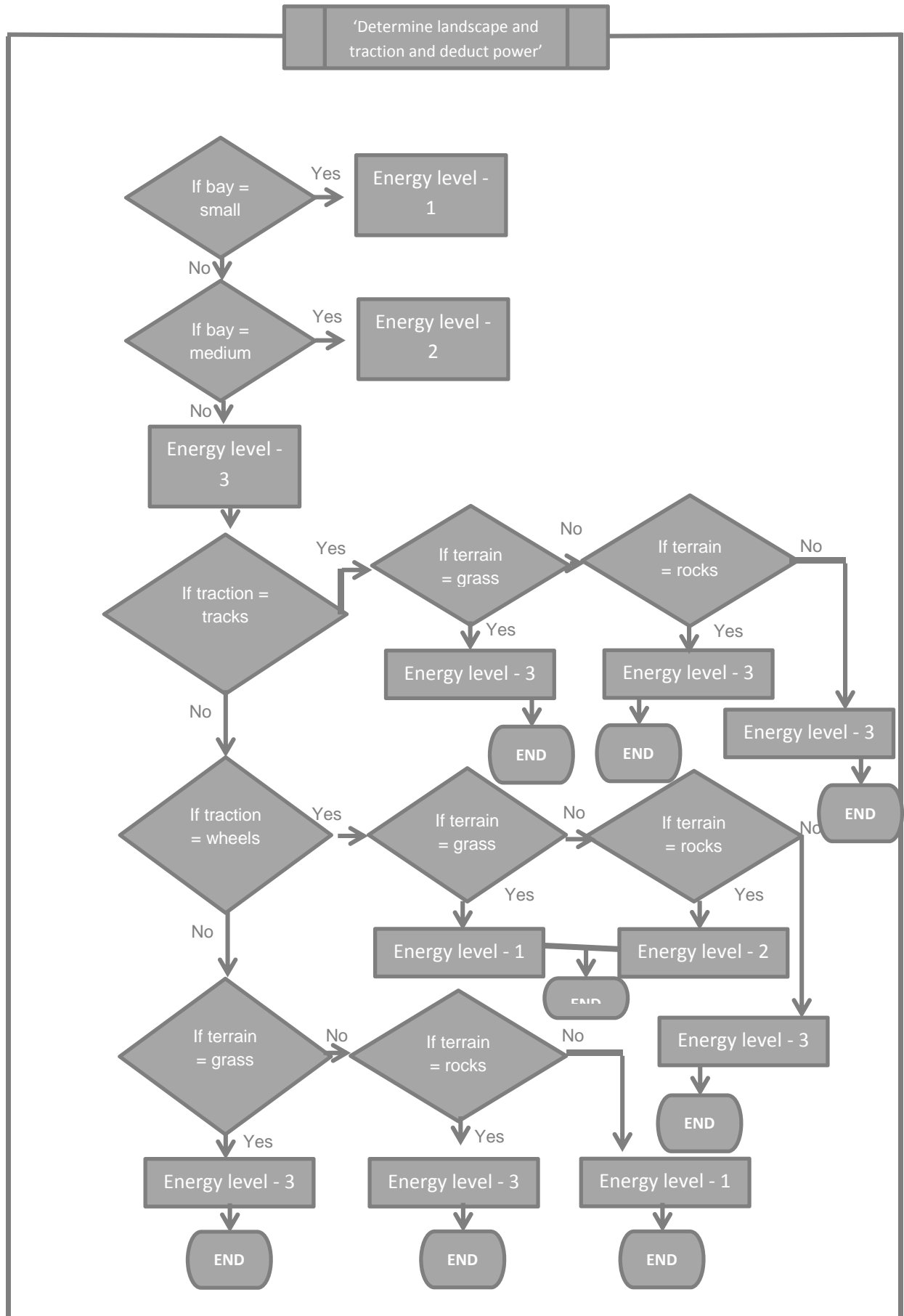


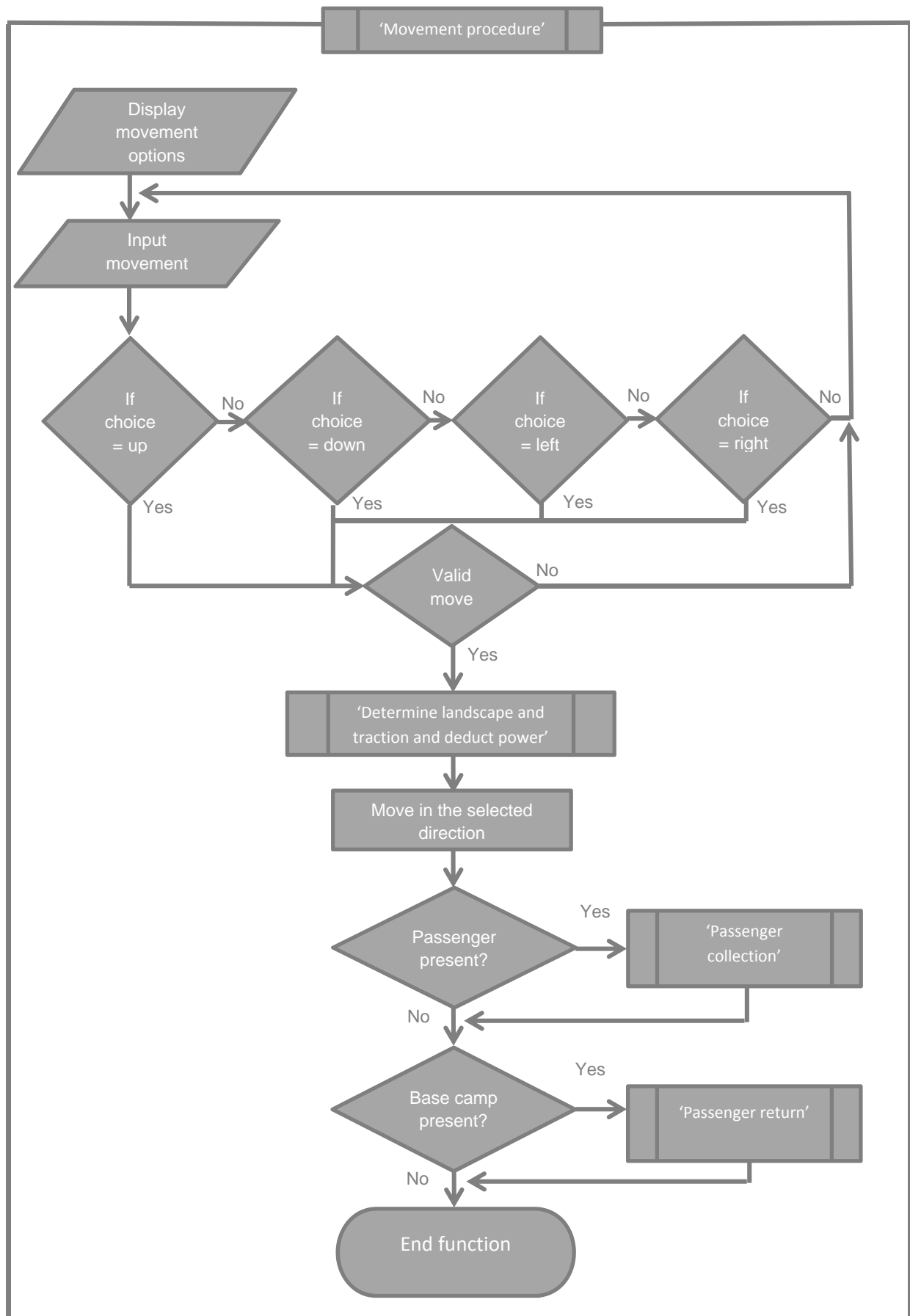












BotMod.py print window

The following is the printed window of my entire programme. This programme is split into labelled sections, for section by section comments see the 'Bot Mod Comments' page.

- A.N. Other

Candidate number: 0000

Centre number: 00000

#Created by A.N. Other
 #For commenting and explanation see Microsoft Word Document 'A.N. Other, BotMod'

#-----see section : 1-----#

from numpy import *

#-----see section : 2-----#

def resetoverlay ():

 overlay = zeros((10,10), dtype=str)

 for i in range(10):
 for j in range(10):
 overlay[i][j] = "-"

 overlay [0][2] = "P"
 overlay [2][7] = "P"
 overlay [6][4] = "P"
 overlay [6][9] = "P"
 overlay [3][6] = "B"

 return overlay

overlay = resetoverlay ()

#-----see section : 3-----#

#Passengers in the bay
 pasinbay = 0
 #Passengers returned to base camp
 passaved = 0
 #player's x co-ordinate
 px = 0
 #player's y co-ordinate
 py = 0
 #Traction choice
 traction = 0

#Bay size choice
 bay = 0
 #energy level
 elevel = 150
 #finished game?
 finished = False
 #current row
 crow = 0
 #current column
 ccol = 0
 #game on ?
 gameon = True
 #Players name
 playername = "Player"
 #Additional games played
 repeats = 0

#-----see section : 4-----#

print """"\nWelcome to BotMod, you must first
 modify your bot""""

def startmenu():
 gameon = True
 print "\n1) Start Game"
 print "2) Modify Bot"
 print "3) Quit\n"

```

chosen = False
while not chosen:
    choice = str(raw_input("What would you like to do? "))
    if choice == "1" or choice == "2" or choice == "3" :
        chosen = True
return int(choice)

```

```
#-----see section : 5-----#
```

```

def setupplayer():
    print "\nEnter desired name"
    print "-----\n"
    player = str(raw_input("Name : "))

    return player

```

```
#-----see section : 6-----#
```

```

def setupmod():
    print "\nTraction types : "
    print "\n1) Wheels"
    print "\n2) Tracks"
    print "\n3) Skis"
    chosen = False
    while not chosen:
        traction = str(raw_input("\nEnter traction type : "))
        if traction == "1" or traction == "2" or traction == "3" :
            chosen = True

    print "\n-----"
    print "\n"
    print "Passenger bay size : "
    print "\n1) Small"
    print "\n2) Medium"
    print "\n3) Large"
    chosen = False
    while not chosen:
        bay = str(raw_input("\nEnter passenger bay size : "))
        if bay == "1" or bay == "2" or bay == "3" :
            chosen = True
            print "\n"
            print "\n-----"

    return(int(traction),int(bay))

```

```
#-----see section : 7-----#
```

```
board = zeros( (10,10), dtype=str )
```

```

def printboard():
    print "-----",
    for i in range(10):
        print "\n"
        for j in range(10):
            if px == i and py == j:
                print "| X",
            elif overlay[i][j] <> "-":
                print "|", overlay[i][j],
            else:
                print "|", board[i][j],
        print "|",
    print "\n-----"

```

```
#-----see section : 8-----#
```

```

def loadboard():
    fin = open('landscape.txt')

```

```

                                IDLE_tmp_v5uzyt
for i in range(10):
    for j in range(10):
        board[i][j] = fin.readline()
fin.close()

#-----see section : 9-----#

def moveplayer(cr,cc):
    validmove = False

    while not validmove:
        direction = str(raw_input("\nDirection : "))

        if direction == "w" and cr <> 0:
            validmove = True
            newrow = cr - 1
            newcolumn = cc

        if direction == "a" and cc <> 0:
            validmove = True
            newrow = cr
            newcolumn = cc - 1

        if direction == "d" and cc <> 9:
            validmove = True
            newrow = cr
            newcolumn = cc + 1

        if direction == "s" and cr <> 9:
            validmove = True
            newrow = cr + 1
            newcolumn = cc

    return (newrow, newcolumn)

#-----see section : 10-----#

loadboard()

while finished == False:
    choice = startmenu()
    if choice == 1 and traction == 0 :
        print "\n First modify your Bot"
    elif choice == 1 and traction > 0:
        playername = setupplayer()
        gameon = True
        while gameon:
            printboard()
            print "\nThe current energy level is : ", elevel
            print "\nThere are", pasinbay, "passengers in the bay"
            print "\nYou have saved", passaved, "passengers"
            print "\nThe player is marked as 'X'"
            print """\nDirections :
'w' = up
'a' = left
's' = down
'd' =right""""
            print "\nGood Luck", playername

            crow,ccol = moveplayer(crow,ccol)

#-----see section : 11-----#

    if px == 0 and py == 2 :

```

I DLE_tmp_v5uzyt

```
if bay == 1 and pasi nbay < 1 :  
    overlay [0][2] = "-"  
    pasi nbay = pasi nbay + 1
```

```
if bay == 2 and pasi nbay < 2 :  
    overlay [0][2] = "-"  
    pasi nbay = pasi nbay + 1
```

```
if bay == 3 and pasi nbay < 3 :  
    overlay [0][2] = "-"  
    pasi nbay = pasi nbay + 1
```

```
if px == 2 and py == 7 :
```

```
    if bay == 1 and pasi nbay < 1 :  
        overlay [2][7] = "-"  
        pasi nbay = pasi nbay + 1
```

```
    if bay == 2 and pasi nbay < 2 :  
        overlay [2][7] = "-"  
        pasi nbay = pasi nbay + 1
```

```
    if bay == 3 and pasi nbay < 3 :  
        overlay [2][7] = "-"  
        pasi nbay = pasi nbay + 1
```

```
if px == 6 and py == 4 :
```

```
    if bay == 1 and pasi nbay < 1 :  
        overlay [6][4] = "-"  
        pasi nbay = pasi nbay + 1
```

```
    if bay == 2 and pasi nbay < 2 :  
        overlay [6][4] = "-"  
        pasi nbay = pasi nbay + 1
```

```
    if bay == 3 and pasi nbay < 3 :  
        overlay [6][4] = "-"  
        pasi nbay = pasi nbay + 1
```

```
if px == 6 and py == 9 :
```

```
    if bay == 1 and pasi nbay < 1 :  
        overlay [6][9] = "-"  
        pasi nbay = pasi nbay + 1
```

```
    if bay == 2 and pasi nbay < 2 :  
        overlay [6][9] = "-"  
        pasi nbay = pasi nbay + 1
```

```
    if bay == 3 and pasi nbay < 3 :  
        overlay [6][9] = "-"  
        pasi nbay = pasi nbay + 1
```

#-----see section : 12-----#

```
if px == 3 and py == 6 :  
    passaved = passaved + pasi nbay  
  
    pasi nbay = 0
```

#-----see section : 13-----#


```

                                I DLE_tmp_v5uzyt
if bay == 1 :
    elevel = elevel
if bay == 2 :
    elevel = elevel - 1
if bay == 3 :
    elevel = elevel - 2

#-----see section : 14-----#

if (board[crow][ccol] == "g") or (board[crow][ccol] == "grass"):
    if traction == 1 :
        elevel = elevel - 1
    elif traction == 2 :
        elevel = elevel - 3
    elif traction == 3 :
        elevel = elevel - 3
elif (board[crow][ccol] == "r") or (board[crow][ccol] == "rock"):
    if traction == 1 :
        elevel = elevel - 2
    elif traction == 2 :
        elevel = elevel - 3
    elif traction == 3 :
        elevel = elevel - 3
elif (board[crow][ccol] == "i") or (board[crow][ccol] == "ice"):
    if traction == 1 :
        elevel = elevel - 3
    elif traction == 2 :
        elevel = elevel - 3
    elif traction == 3 :
        elevel = elevel - 1

#-----see section : 15-----#

px = crow
py = ccol

#-----see section : 16-----#

if elevel < 1:
    print "\nSorry", playername, "you have run out of energy!"
    print "\n"
    gameon = False

#-----see section : 17-----#

if passaved == 4:
    print "Congratulations you beat the level!"

    if repeats == 2:
        print "You beat the game! Congratulations!"
        print "\n"
        print "\nCome back again !"
        gameon = False
        repeats = 0
        px = 0
        py = 0
        elevel = 150
        crow = 0
        ccol = 0
        pasinbay = 0
        passaved = 0
        traction = 0

```

```

                                IDLE_tmp_v5uzyt
bay = 0
overlay = resetoverlay ()

    if repeats == 1 :
        repeat = str(raw_input("\nWould you like to play the next
level ? (yes/no) : "))
        if repeat in ("Yes", "YES", "yEs", "YeS", "yES", "YEs",
"yes") :
            overlay = resetoverlay ()
            px = 0
            py = 0
            elevel = 130
            crow = 0
            ccol = 0
            pasinbay = 0
            passaved = 0
            repeats = repeats + 1
        else :
            gameon = False
            repeats = 0
            px = 0
            py = 0
            elevel = 150
            crow = 0
            ccol = 0
            pasinbay = 0
            passaved = 0
            traction = 0
            bay = 0
            overlay = resetoverlay ()

    if repeats == 0 :
        repeat = str(raw_input("\nWould you like to play the next
level ? (yes/no) : "))
        if repeat in ("Yes", "YES", "yEs", "YeS", "yES", "YEs",
"yes") :
            overlay = resetoverlay ()
            px = 0
            py = 0
            elevel = 140
            crow = 0
            ccol = 0
            pasinbay = 0
            passaved = 0
            repeats = repeats + 1
        else :
            gameon = False
            repeats = 0
            px = 0
            py = 0
            elevel = 150
            crow = 0
            ccol = 0
            pasinbay = 0
            passaved = 0
            traction = 0
            bay = 0
            overlay = resetoverlay ()

#-----see section : 18-----#

    if choice == 2:
        print "\n-----"
        traction, bay = setupmod()
        print "\n"

```

I DLE_tmp_v5uzyt

```
#-----see section : 19-----#  
    if choice == 3:  
        finished = True  
print "\nGoodbye and thank you for playing"  
print "\n-----"
```

Bot Mod Comments

The following is the commenting of my Bot Mod programme combined with brief discussions of each section. The sections are as stated and separated in the Programme itself (see BotMod.py print window).

- A.N. Other

Candidate number: 0000

Centre number: 00000

A.N. Other, GCSE computer science, Bot Mod.

Modbot Commenting

Section 1:

```
from numpy import *
```

This line imports all modules of numpy allowing me to use their features within the programme. Numpy is used for handling arrays and therefore is a paramount feature within my programme.

Section 2:

This section is centred around the board overlay and is a function that allows it to be created, the board overlay is a second game board that originally consists completely of hyphens but later also holds the players and base camp at specific points of the board. The 2nd line creates the overlay.

```
def resetoverlay ():
```

```
    overlay = zeros( (10,10), dtype=str )
```

```
    for i in range(10):
        for j in range(10):
            overlay[i][j] = "-"
```

```
    overlay [0][2] = "P"
    overlay [2][7] = "P"
    overlay [6][4] = "P"
    overlay [6][9] = "P"
    overlay [3][6] = "B"
```

```
    return overlay
```

```
overlay = resetoverlay ()
```

The next section replaces every variable within the array to a hyphen so later if commands can search for hyphens or values that or not hyphens. The last segment of the function consists of applying the player and base camp locations upon the overlay, replacing the appropriate hyphen. Finally the now created overlay is returned. The line 'overlay = resetoverlay ()' uses the function to create an overlay and save it to the variable overlay so it can be used later in the programme.

Section 3:

This section simply defines variables that will be later used in the programme,

'pasinbay = 0' – This variable holds the amount of passengers in the robot's bay.
'passaved = 0' – This variable is the amount of passengers that have been returned to the basecamp.
'px = 0' – The Bot's x co-ordinate
'py = 0' – The Bot's y co-ordinate
'traction = 0' – The decided traction type.
'bay = 0' – The decided bay size.
'elevel = 150' – The level of energy in the Bot.
'finished = False' – The true false statement to dictate whether the game has finished or not.
'crow = 0' – A variable to hold the player's row.
'ccol = 0' – A variable to hold the player's column.
'gameon = True' – The true false as to whether the game itself is running.
'playername = "Player"' – The variable that holds the name of the player, standardised as player in case the player does not enter their name.
'repeats = 0' – This variable counts every time the game is played again so the difficulty can be increased accordingly.

Section 4:

This section includes a welcome message and the function that displays the menu and allows choices. The first two lines are simply a welcome message that will only display once, when you first open the game.

```
print """\nWelcome to BotMod, you must first  
modify your bot""
```

The next section is the function that prints the start menu, allows input, and handles the decision.

```
def startmenu():  
    gameon = True  
    print "\n1) Start Game"  
    print "2) Modify Bot"  
    print "3) Quit\n"  
    chosen = False  
    while not chosen:  
        choice = str(raw_input("What would you like to do? "))  
        if choice == "1" or choice == "2" or choice == "3" :  
            chosen = True  
    return int(choice)
```

The function here is named 'startmenu' the following 3 lines simply print the options the player has and the number allocated to each, the gameon line just ensures the game is allowed to start if a player has returned from game and wishes to play again.. The next lines are concerning the variable 'chosen' it is defined as False and then used in the line 'while not chosen' this means that the following appropriately indented lines will loop until chosen becomes True. Within this loop the question 'What would you like to do?' is printed and the player now inputs their decision, the following if statements ensure that the input is either 1 2 or 3 as these are the only options, if they

are one of these then the loop ends and the choice is returned (ending the function), if the input is not 1, 2 or 3 then the user will be asked to input an option again.

Section 5:

This section solely consists of a function to record the players name for later use in the game. It asks the player to enter their desired name then leaves a section for their input, this input is saved in a variable named player and returned to the programme, consequentially ending the function.

```
def setupplayer():
    print "\nEnter desired name"
    print "-----\n"
    player = str(raw_input("Name : "))

    return player
```

Section 6:

Section 6, the function titled 'setupmod', is the function to allow customization of the player's robot's bay and traction type.

```
def setupmod():
    print "\nTraction types :"
    print "\n1) Wheels"
    print "\n2) Tracks"
    print "\n3) Skis"
    chosen = False
    while not chosen:
        traction = str(raw_input("\nEnter traction type : "))
        if traction == "1" or traction == "2" or traction == "3" :
            chosen = True

    print "\n-----"
    print "\n"
    print "Passenger bay size :"
    print "\n1) Small"
    print "\n2) Medium"
    print "\n3) Large"
    chosen = False
    while not chosen:
        bay = str(raw_input("\nEnter passenger bay size : "))
        if bay == "1" or bay == "2" or bay == "3" :
            chosen = True
        print "\n"
        print "\n-----"

    return(int(traction),int(bay))
```

The first lines within the function simply tell the user what they are about to choose from ('Traction Types') and the options they have, a 'Chosen' variable is then created and set as False and

subsequently used to create a loop around the line 'while not chosen:'. The user then inputs their decision and if it is a valid input (1,2 or 3) then 'Chosen' becomes true and they can continue within the function, if not they are asked again to input a decision. This process is then repeated but now for 'Passenger bay size' instead of 'Traction type' when they choose a valid input for this choice the function will return the functions holding the chosen passenger bay size and also convert them to integers.

Section 7:

This section includes the 'printboard' function, as well as the creation of the board its self.

```
board = zeros( (10,10), dtype=str )
```

```
def printboard():
    print "-----",
    for i in range(10):
        print "\n"
        for j in range(10):
            if px == i and py == j:
                print "| X",
            elif overlay[i][j] <> "-":
                print "|",overlay[i][j],
            else:
                print "|",board[i][j],
        print "|",
    print "\n-----"
```

In this function the 'i' is the row or rows within the board and the 'j' is the column or columns. The 'in range(10)' following each section simply means *in the range of up to 10* which entails numbers 0 to 9. These two loops allow every variable within the array to be *checked*. The line 'if px ==i and py==j' checks the players coordinates against the coordinates of the board array and when or if they find them to be equal then the player's symbol 'X' is printed. This allows the player's position to be visible upon the board but without removing anything from the .txt file and board itself and instead acts as an overlay. The next line concerns the array 'overlay' and printing its contents, the line 'elif overlay[i][j] <> "-:' followed by print '|,overlay[i][j],' essentially scans every position within the array 'overlay' and if it is not a hyphen and is alternatively a passenger or the base camp it will print it correspondingly within the board. The final else statement will only take place after the previous If and elif statements and therefore will not overwrite their input but will simply fill the empty coordinates with the terrain from the .txt file.

Section 8:

This section includes a function to read from the .txt file.

```
def loadboard():
    fin = open('landscape.txt')
    for i in range(10):
        for j in range(10):
            board[i][j] = fin.readline()
    fin.close()
```


In this function opens the text file 'landscape.txt' and then inputs all of its contents in order on the 10 by 10 'board' array by using 'in range 10' statements, after it has been implemented into the board array the .txt file is closed as it is no longer required due to it's contents now being transferred to an in-programme array.

Section 9:

The following function results in the ability to take a move as the player but will reject any incorrect or unavailable moves attempted.

```
def moveplayer(cr,cc):
    validmove = False

    while not validmove:
        direction = str(raw_input("\nDirection : "))

        if direction == "w" and cr <> 0:
            validmove = True
            newrow = cr - 1
            newcolumn = cc

        if direction == "a" and cc <> 0:
            validmove = True
            newrow = cr
            newcolumn = cc - 1

        if direction == "d" and cc <> 9:
            validmove = True
            newrow = cr
            newcolumn = cc + 1

        if direction == "s" and cr <> 9:
            validmove = True
            newrow = cr + 1
            newcolumn = cc

    return (newrow,newcolumn)
```

I first of all add two variables to the brackets of the function ('cr' current row and 'cc' current column) that I need to import from the programme. I then create a variable to centre my loop around, 'validmove' which I default as false, I use the while not command to make a loop with this variable. The first line of code in the loop is the one that asks and receives the user input for the direction. The available direction are up ('w'), down ('s'), left ('a') and right ('d'). The two sections regarding "s" and "w" movement concern the player's y position, the if commands are followed with an and command that checks that the move won't let them exceed the locations within the array, e.g. if the player is in the upper left corner they are unable to go, the "d" and "a" sections do the same but with the x axis. If the player does not select a direction that is available the loop will repeat, starting with the prompt for a direction. However if the player chooses a valid direction the next line of code 'validmove = True' ensures that the loop will now end, preventing additional movement it then formulates two variables newcolumn and newrow, these are a result of the imported currentrow('cr') and currentcolumn ('cc') variables combined with the direction of their

move. As an example if the player chose "d" (which is right) then newrow is the same as the currentrow but newcolumn is the currentcolumn + 1 as the player is moving a column right.

Section 10:

This function displays to the user their stats, and their possible moves it then uses the function 'moveplayer' to calculate the player's new location.

loadboard()

while finished == False:

```
choice = startmenu()
if choice == 1 and traction == 0 :
    print "\n First modify your Bot"
elif choice == 1 and traction > 0:
    playername = setupplayer()
    gameon = True
    while gameon:
        printboard()
        print "\nThe current energy level is : ", elevel
        print "\nThere are", pasinbay, "passengers in the bay"
        print "\nYou have saved", passaved, "passengers"
        print "\nThe player is marked as 'X'"
        print """\nDirections :
'w' = up
'a' = left
's' = down
'd' =right"""\nGood luck",playername

    crow,ccol = moveplayer(crow,ccol)
```

The first line in the function starts a loop around the variable 'False'. It then defines the variable 'choice' using the startmenu function mentioned previously. The programme then checks that if you selected to start the game that you have set up the Bot, it does this by using the variable 'traction', this variable is a reflection on the player's choice when setting up their Bot and will be 1, 2 or 3 however it is defaulted at 0. Therefore if the traction = 0 then the Bot has not been set up, the line of code checks this and if true tells the user to set up their mod first and presents the start menu. The next line starting 'elif' checks again that they have set up their mod and if they have continues the function. The function then uses the setupplayer function (described previously) to retrieve a playername and then starts the game. It will then print the board and then print multiple pieces of information. (in order) The player's energy level, the number of passengers in the bay, the number of passengers saved and it then tells you that the player is marked by X. Next it prints the possible moves and wishes the player good luck using the previously retrieved playername, the programme then

Section 11:

This is an excerpt of section as 11 as the entire section is unnecessary and excessive to include, I will therefore explain the other section in comparison to this one as they are simply iterations of each other that are only slightly modified. This section is allocated the role of collecting passengers in bay but also ensuring it is a legal and legitimate circumstance.

```
if px == 0 and py == 2 :  
  
    if bay == 1 and pasinbay < 1 :  
        overlay [0][2] = "-"  
        pasinbay = pasinbay + 1  
  
    if bay == 2 and pasinbay < 2 :  
        overlay [0][2] = "-"  
        pasinbay = pasinbay + 1  
  
    if bay == 3 and pasinbay < 3 :  
        overlay [0][2] = "-"  
        pasinbay = pasinbay + 1
```

The purpose of the first line is to compare the player's x (px) and y (py) to that of a passenger's coordinates in the passenger base camp overlay. The subsequent if bay == lines check the size of the bay in comparison to the passengers in the bay, so that you can determine whether there is space for the passenger. The next line happens if the previous query was true and there *is* space it then changes what was the passenger to a hyphen so it will no longer be printed on the board, this is because my print function only printed the contents of the overlay that weren't hyphens. The final line is pasinbay = pasinbay + 1 this simply adds 1 passenger to your bay. As mentioned this is only a part of this section, the other parts are iterations of the just-described part except they apply for different coordinates and therefore different passengers.

Section 12:

This section handles the base camp and its reaction to the player and it's bay.

```
if px == 3 and py == 6 :  
    passaved = passaved + pasinbay  
  
    pasinbay = 0
```

the first line is checking the player's x (px) and y (py) so that they are the same as the coordinates of the Basecamp. The next line defines the passenger saved variable, it takes the current value meaning the amount saved already and adds to it the value of the passengers in bay variable, the next line then sets the passengers in bay to 0. This means this section adds your collected passengers to those saved and then empties your bay.

Section 13:

This section makes an energy level reduction per turn based on the chosen bay size.

```
if bay == 1 :  
    elevel = elevel  
if bay == 2 :  
    elevel = elevel - 1  
if bay == 3 :  
    elevel = elevel - 2
```

The if command(s) separate the energy reductions based on base size so that if the bay is x size then the correct amount of energy level will be removed as a result of the player's customization choice(s).

Section 14:

This section determines the landscape type the player is on and then makes the appropriate energy level reductions based on the information given regarding terrain and traction.

```
if (board[crow][ccol] == "g") or (board[crow][ccol] == "grass"):  
    if traction == 1 :  
        elevel = elevel - 1  
    elif traction == 2 :  
        elevel = elevel - 3  
    elif traction == 3 :  
        elevel = elevel - 3  
  
elif (board[crow][ccol] == "r") or (board[crow][ccol] == "rock"):  
    if traction == 1 :  
        elevel = elevel - 2  
    elif traction == 2 :  
        elevel = elevel - 3  
    elif traction == 3 :  
        elevel = elevel - 3  
  
elif (board[crow][ccol] == "i") or (board[crow][ccol] == "ice"):  
    if traction == 1 :  
        elevel = elevel - 3  
    elif traction == 2 :  
        elevel = elevel - 3  
    elif traction == 3 :  
        elevel = elevel - 1
```

These sections initially determine the terrain the player is on by comparing the player current row (crow) and column (ccol) to the board, each part does this but with the 3 varying terrains. The subsequent if and elif commands then remove the appropriate volume of energy based on the information supplied in the candidate booklet.

Section 15:

This section solely converts the established player's current row (crow) and column (ccol) from the move function and it re-assigns it as player x (px) and y (py) this is so that px and py can then be re-used in a function to establish crow and ccol and this is repeated.

```
px = crow  
py = ccol
```

Section 16:

This section ends the game if the player's energy level is 0 or less.

```
if elevel < 1:  
    print "\nSorry",playername, "you have run out of energy!"  
    print "\n"  
    gameon = False
```

The first line checks if elevel is less than one, this check will be done after every turn to stop cheating in-game. It then prints a message letting the player know it has lost and then set gameon to false, ending the game and returning the player to the main menu.

Section 17:

This section regards the completion of a game and the appropriate treatment.

```
if passaved == 4:  
    print "Congratulations you beat the level!"  
  
if repeats == 1 :  
    repeat = str(raw_input("\nWould you like to play the next level ? (yes/no) : "))  
    if repeat in ("Yes", "YEs", "YES", "yeS", "yES", "YEs", "yes") :  
        overlay = resetoverlay ()  
        px = 0  
        py = 0  
        elevel = 130  
        crow = 0  
        ccol = 0  
        pasinbay = 0  
        passaved = 0  
        repeats = repeats + 1  
    else :  
        gameon = False  
        repeats = 0  
        px = 0  
        py = 0  
        elevel = 150  
        crow = 0  
        ccol = 0  
        pasinbay = 0  
        passaved = 0
```

```

traction = 0
bay = 0
overlay = resetoverlay ()

```

```

elif repeats == 2:
    print "You beat the game! Congratulations!"
    print "\n"
    print "\nCome back again !"
    gameon = False
    repeats = 0
    px = 0
    py = 0
    elevel = 150
    crow = 0
    ccol = 0
    pasinbay = 0
    passaved = 0
    traction = 0
    bay = 0
    overlay = resetoverlay ()

```

Section 17:

This section checks if the player has saved all the passengers and therefore won the game and also allows for the game to be continued to the next level.

```

if passaved == 4:
    print "Congratulations you beat the level!"

if repeats == 1 :
    repeat = str(raw_input("\nWould you like to play the next level ? (yes/no) : "))
    if repeat in ("Yes", "YEs", "YES", "yeS", "yES", "YEs", "yes") :
        overlay = resetoverlay ()
        px = 0
        py = 0
        elevel = 130
        crow = 0
        ccol = 0
        pasinbay = 0
        passaved = 0
        repeats = repeats + 1
    else :
        gameon = False
        repeats = 0
        px = 0
        py = 0
        elevel = 150
        crow = 0
        ccol = 0

```

```

    pasinbay = 0
    passaved = 0
    traction = 0
    bay = 0
    overlay = resetoverlay ()

elif repeats == 2:
    print "You beat the game! Congratulations!"
    print "\n"
    print "\nCome back again !"
    gameon = False
    repeats = 0
    px = 0
    py = 0
    elevel = 150
    crow = 0
    ccol = 0
    pasinbay = 0
    passaved = 0
    traction = 0
    bay = 0
    overlay = resetoverlay ()

```

The first part of this programme regarding 'passaved' checks after every move whether 4 passengers have been saved, if this statement it is true the programme will continue within this section, the first action is the congratulating message. There are then 3 checks around the variable 'repeats' however I haven't included the second check, which checks 0 as it is unnecessary, it can alternatively be found in the file. The first check is for 1 repeat, the reason the check is for 1 first is because if 0 is checked first and is true then part of the result is adding 1 to repeat, which would then activate the repeats = 1 skipping a level. If the players repeats = 1 then a question will be posed asking whether the player wants to continue if they type yes (which will be recognised regardless of the way it is written due to the inclusion of multiple spelling possibilities) then the appropriate variables are reset to enable a fair game and the elevel is reduced accordingly, also 1 is added to the repeat variable so the programme knows the game has been replayed once. If the player does not type yes, they will return to the menu and once again all the appropriate variables are reset to ensure they can play the game once again and it will work fully as it did the first time. This as mentioned is repeated but with the conditional of repeats == 0 to distinguish progression. The section checks if repeats = 2, if it does it means the player has finished all the levels and will then display congratulating message and return them to the main menu, once again I reset the essential variables so my game is instantly repayable which is an extremely important aspect of a game.

Section 18:

This section regards the choice to modify bot in the start menu and it retrieves two variables from the setupmod function and assigns them to traction and bay size.

```

if choice == 2:
    print "\n-----"
    traction, bay = setupmod()
    print "\n"

```

Section 19:

This section is a simple section of code that will end the ModBot game. It starts with a conditional that relates to a menu choice, the selection 3 corresponds with the option 'quit'. The programme then (if the conditional is met) redefines finished, the variable upon which the game is looped to True, ending the loop. A line is then printed saying goodbye to the player.

```
if choice == 3:  
    finished = True  
  
print "\nGoodbye and thank you for playing"  
print "\n-----"
```


Bot Mod Discussion

The following is an in-depth discussion regarding the programming techniques I've used and an analysis of robustness, efficiency and data types. (This document may reference 'Bot Mod Comments').

- A.N. Other

Candidate number: 0000

Centre number: 00000

Handling Data

I have used multiple personal programming techniques which I believe make my programme more efficient and robust. An example of a technique that improves my programme is the way I handle user input. All of my in game choices are designated to numbers so the user will type '1', '2' or '3' (as seen below) to select their desired option.

```
print "\n1) Start Game"
print "2) Modify Bot"
print "3) Quit\n"
```

However the user has no constraint in what they type so could theoretically type a letter, symbol or an irrelevant number which without precaution would crash the programme. The way I have handled this issue is by surrounding the programme regarding a choice in a loop (an example is the loop below which is centred around the chosen variable),

```
chosen = False
while not chosen:
```

using conditionals and editing the data type. Inside of my choice loop I would first have the choice which was saved as a string , the next line would then check that the string was either 1, 2 or 3 (the valid choices)

```
choice = str(raw_input("What would you like to do? "))
if choice == "1" or choice == "2" or choice == "3" :
```

and if it was then the loop would be terminated and the programme would continue (as seen below this conflicts with the loops conditionals and therefore ends the loop)

```
chosen = True
```

(also the string would be changed to an integer when returned to aid later programme).

```
return int(choice)
```

If the choice was not a valid one the question would be asked again and repeated until a valid entry was made as while chosen = False the loop will run. This method is advantageous in many ways as it solves multiple problems, it removes the possibility of a crash and also allows the player to re-enter their information indefinitely. This is an example of a **PROGRAMMING TECHNIQUE** used, **ROBUSTNESS** and control of **DATA STRUCTURES** (for an example of handling data see 'Bot Mod commenting' section 4).

Arrays

Another personal programming technique I used was the use of an array in my programme. An array is a variable which in itself holds multiple variables, much alike a grid (in my case 10x10) with every section holding a different variable. There are two examples of arrays in my programme and they are my board and my overlay. My board array was a 10x10 array which meant it had 100 different variables within it (as seen below),

```
board = zeros( (10,10), dtype=str )
```

these were all filled with the appropriate landscape as stated in the candidate booklet. The use of this technique and the way in which I would fill my arrays greatly improves the efficiency of my programme. As an example I used a single line of code to fill my overlay array

```
overlay = zeros( (10,10), dtype=str )
```

this makes an enormous difference to my programme as the alternative to filling the array in this way is to individually fill every single one of the 100 variables which would be far less efficient than the technique in place in my programme. This is an example of a **PROGRAMMING TECHNIQUE** used, **EFFICIENCY** as well as an example of the use of **DATA STRUCTURES** (for an example of Arrays see 'Bot Mod commenting' section 2).

Functions and Procedures

Functions and procedures were also used in my programme as they make my programme more efficient and robust. Functions are essentially segments of code that return specific variables, one of the most useful aspects of a function is its ability to save space as rather than repeating a section of code you used previously you instead re-call the function and the code will be executed and the variables retrieved.

```
return overlay
```

(Above is the line used at the end of the function to return a variable to the programme) I tried to use this technique on all the sections of programme I may need to repeat as the use of the function allows repeatability of programme sections as well as efficiency in the form of fewer lines in my programme. This is an example of a **PROGRAMMING TECHNIQUE** used and of programme **EFFICIENCY** (for an example of a function see 'Bot Mod commenting' section 2).

Handling external files

Another technique I used in my programme was the importing of an external file instead of alternatively typing every variable that I wished to retrieve from it and setting them as variables manually. I used an external file provided to me with my candidate booklet that was a .txt file containing 100 lines, each of which had the name of a type of terrain. I opened the file via its directory within the programme

```
fin = open('landscape.txt')
```

and then it read it line by line saving the first letter of each line and assigning it to the array in a linear order. This resulted in a 10x10 array which now has the contents of the .txt file that I wished to retrieve. By using this method I simplified my programme dramatically and also improved its efficiency, I also used specific data types (an array) to improve my programme further by storing the desired information in a single array rather than 100 individual variables. This **PROGRAMMING TECHNIQUE** used improves my programmes **EFFICIENCY** and utilizes **DATA STRUCTURES** (for an example of handling external files see 'Bot Mod commenting' section 8).

Conditionals

Conditionals are statements that perform different actions depending on what a programmer-specified *condition* evaluates to. I've used these frequently in my program as it is

essential to analyse and segregate user input to ensure the correct response and action to their input. An example of conditionals is the following:

```
if px == 3 and py == 6 :  
    passaved = passaved + pasinbay
```

This is an example of a conditional used in my program this conditional requires an answer to comply with two requirements and if it does then the following indented code will be executed, in this case the amount of passengers in the bay is added to the amount of passengers saved. I have used conditionals in my program as there other method to analyse input or variables and execute specific code based on this information. This **PROGRAMMING TECHNIQUE** increases the **EFFICIENCY** of my program (for an example conditionals see 'Bot Mod commenting' section 14).

Looping statements

Looping statements are conditionals that allow for the repetition of a section of code until a specified condition is fulfilled. For example the below extract of code features a loop centred around the condition while and the variable validmove. The variable is first defined as false and then included in the conditional 'while not' this means that until validmove = True then the following code will be repeated indefinitely.

```
validmove = False
```

```
while not validmove:
```

Variables

A variable is simply a value that has the ability to vary, hence variable. Variables are used in my programme to hold information for use in functions and my programme. If variables were not used and the ability to edit a value was not present then programming would be significantly less efficient. Values would have to be created constantly instead of being edited which would greatly lengthen a program. I've used variables to hold text, numbers, symbols and sometimes other variables (arrays). Variables are an essential **PROGRAMMING TECHNIQUE** and improve my programmes **EFFICIENCY** (for an example of a function see 'Bot Mod commenting' section 3).

Efficiency

Functions

A function within Python is a segment of predefined code that is contained under a name (like a variable), however as this section of code is a function it is re-callable and re-usable. What this potentially means is that an infinite number of objects can be defined by a finite statement. The ability to be called upon provided by a function allows for the re-application of code without the re-insertion, same outcome, less lines (Improving efficiency). Without functions within my programme it would be considerably less efficient and a great deal larger.

- One example of a function in my programme is the 'startmenu' function. This function displays the main menu, and accepts and returns a valid navigation choice. By containing this code as a function it allows it to be recalled potentially infinitely whilst it is only present in the programme code once.
- The 'returnplayer' function in my programme prints messages telling the user to input their name and then defines a variable ('player') as their inputted name, it finally returns the name. This segment of code is only 4 lines long however is able to function indefinitely, meaning it will always work.
- A vital function to my programme is the 'printboard' function. This is because the board is printed so frequently in the programme, to finish the game the player will have to make at least 75 moves meaning the board will be printed at least 75 times. The 'printboard' segment of code is 12 lines long; the equivalent length of code without functions is around 900 lines, making it over 90% larger. Another reason why the use of a function is infinitely better is because a function can be called when it is needed, meaning if the player makes 82 moves the board will have been displayed 82 times, were as if the code was typed repeatedly the programmer would have to guess how many moves would be made.

Loops

Loops consist of conditionals that allow for the repetition of a section of code until a specified condition is fulfilled. Loops are commonly used when involved with user input, if user input is required the most efficient way to code a question is within a loop so that the loop will only end when (correct) user input is received. Another example of a loop is simply if you want an operation completed a specific number of times, rather than typing 30 lines you instead type 1 and contain it within a loop that will only end when the 1 line has been executed 30 times.

- An example of a loop when concerning user input can be seen in my 'startmenu' function. The loop is centred around the variable chosen which is defined as False and the loop will run until chosen becomes True. This loop holds a request for user input in the form of navigating the menu and in this instance the only valid choices are 1, 2 or 3. Following the user input there is a conditional which checks whether the user input is in fact 1,2 or 3 and if it chosen becomes True and the programme can proceed, if the input is not 1,2 or 3 then of course chosen remains False and the loop continues.
- Examples of loops that simply repeat code are seen throughout my programme. An example of this can be seen in the function 'resetoverlay'. The loop in this case is a for loop and with a range of 10 meaning it will be executed ten times, these loops are centred around variables 'i' and 'j' which I then insert into the brackets concerning the overlay array

followed by the command to print the symbol '-'. What this section of code does is for every variable within 10 columns and 10 rows, it inserts a hyphen, resulting in a 10x10 array filled with hyphens which is essential in my programme later.

Arrays

An array is a collection of variables stored in a rows and columns form. An array allows for potential thousands of variables stored in one (an array). Therefore it is invaluable when it comes to efficiency within my programme as it lowers the amount of defined variables and therefore lines. Simply put, rather than defining the variable 'one' as 1 and '2' as 2 etc. I can instead insert these numbers into an array meaning that rather than having 100 variables (and 100 lines) I can have one array which stores 100 variables. Making it thousands of times more efficient.

- Arrays in my programme are mostly used when concerning the game board as itself is a 10x10 shape. I decided that my 'board' would be a 10x10 array rather than what would be 100 variables as it reduces both programme size and lines but also makes the board more manageable.
- I did the same with my overlay which held additional board features such as passengers and the base camp.

Bot Mod Testing and Evaluation

The following is a table containing the name of each test I conducted as well as a description, expected result, actual result and any action taken. Includes referenced screenshots from the Python 2.7 IDLE as evidence of testing.

- A.N. Other

Candidate number: 0000

Centre number: 00000

Testing Plan

Test ID	Description	User Input	Expected Outcome	Outcome	Action Taken
1:1	Select the start game option in the main menu.	1	If the bot has been modified the game will ask for your name and then start if not it will say 'Modify your bot first' and re-print the main menu.	As expected.	None necessary.
1:2	Select the Modify bot option in the main menu.	2	Display the modify bot page.	As expected.	None necessary.
1:3	Select the quit option in the main menu.	3	End the game.	As expected.	None necessary.
1:4	Input an incorrect number in the main menu to test for crash.	4	The programme will re-issue the question 'What would you like to do?'	As expected.	None necessary.
1:5	Input a letter in the main menu to test for crash.	h	The programme will re-issue the question 'What would you like to do?'	As expected.	None necessary.
1:6	Input a symbol in the main menu to test for crash.	*	The programme will re-issue the question 'What would you like to do?'	As expected.	None necessary.
2:1	Select the wheels option in the traction type menu	1	The programme will advance to the Passenger bay type menu.	As expected.	None necessary.
2:2	Select the tracks option in the traction type menu	2	The programme will advance to the Passenger bay type menu.	As expected.	None necessary.
2:3	Select the skis option in the traction type	3	The programme	As expected.	None necessary.

	menu		will advance to the Passenger bay type menu.		
2:4	Input an incorrect number in the main menu to test for crash.	4	The programme will re-issue the prompt 'Enter traction type :'	As expected.	None necessary.
2:5	Input a letter in the main menu to test for crash.	h	The programme will re-issue the prompt 'Enter traction type :'	As expected.	None necessary.
2:6	Input a symbol in the main menu to test for crash.	*	The programme will re-issue the prompt 'Enter traction type :'	As expected.	None necessary.
3:1	Select the small option in the Passenger bay size menu	1	The programme will return the user to the main menu.	As expected.	None necessary.
3:2	Select the medium option in the Passenger bay size menu	2	The programme will return the user to the main menu.	As expected.	None necessary.
3:3	Select the large option in the Passenger bay size menu	3	The programme will return the user to the main menu.	As expected.	None necessary.
3:4	Input an incorrect number in the main menu to test for crash.	4	The programme will re-issue the prompt 'Enter passenger bay size :'	As expected.	None necessary.
3:5	Input a letter in the main menu to test for crash.	h	The programme will re-issue the prompt 'Enter passenger bay size :'	As expected.	None necessary.
3:6	Input a symbol in the	*	The	As expected.	None

	main menu to test for crash.		programme will re-issue the prompt 'Enter passenger bay size :'		necessary.
4:1	Attempt to move the bot up or left when in the most north-westerly point on the board.	W A	The game will re-issue the prompt 'Direction :'	As expected.	None necessary.
4:2	Attempt to move the bot up or right when in the most north-easterly point on the board.	W D	The game will re-issue the prompt 'Direction :'	As expected.	None necessary.
4:3	Attempt to move the bot down or left when in the most south-westerly point on the board.	S A	The game will re-issue the prompt 'Direction :'	As expected.	None necessary.
4:4	Attempt to move the bot down or right when in the most south-easterly point on the board.	S D	The game will re-issue the prompt 'Direction :'	As expected.	None necessary.
5:X	Check that the 9 possible traction-bay combinations all remove the correct value of energy on each type of terrain	-	Energy will be removed as stated to happen in the candidate booklet.	As expected	None necessary.
6:1	Check that the passenger at co-ordinate (3, 10) is able to be 'picked up' and added to the passengers in bay.	-	The position where the passenger was will return to 'i' and passengers in bay will increase by 1.	As expected.	None necessary.
6:2	Check that the passenger at co-ordinate (5, 4) is able to be 'picked up' and added to the passengers in bay.	-	The position where the passenger was will return to 'r' and passengers in bay will increase by 1.	As expected.	None necessary.
6:3	Check that the passenger at co-ordinate (8, 8) is able to be 'picked up' and	-	The position where the passenger was will return to	As expected.	None necessary.

	added to the passengers in bay.		'g' and passengers in bay will increase by 1.		
6:4	Check that the passenger at co-ordinate (10, 4) is able to be 'picked up' and added to the passengers in bay.	-	The position where the passenger was will return to – and passengers in bay will increase by 1.	As expected.	None necessary.
7:1	Check that after completing the game once that the ability to continue is available and operational.	yes	The game will replay but with an energy level of 140.	As expected.	None necessary.
7:2	Check that after completing the game once that the ability to stop is available and operational.	no	The game will end and the user will be returned to the main menu.	As expected.	None necessary.
8:1	Check that after completing the game twice that the ability to continue is available and operational.	yes	The game will replay but with an energy level of 130.	As expected.	None necessary.
8:2	Check that after completing the game twice that the ability to stop is available and operational.	no	The game will end and the user will be returned to the main menu.	As expected.	None necessary.
9:1	Check that the player can move up on the board.	w	The game will move the player up once on the board.	As expected.	None necessary.
9:2	Check that the player can move right on the board.	d	The game will move the player right once on the board.	As expected.	None necessary.
9:3	Check that the player can move left on the board.	a	The game will move the player left once on the board.	As expected.	None necessary.
9:4	Check that the player can move down on the board.	s	The game will move the player down once on the board.	As expected.	None necessary.

Testing Evidence

1:1

```
Welcome to BotMod, you must first  
modify your bot
```

- 1) Start Game
- 2) Modify Bot
- 3) Quit

```
What would you like to do? 1
```

```
    First modify your Bot
```

- 1) Start Game
- 2) Modify Bot
- 3) Quit

```
What would you like to do?
```

or

- ```

1) Start Game
2) Modify Bot
3) Quit
```

```
What would you like to do? 1
```

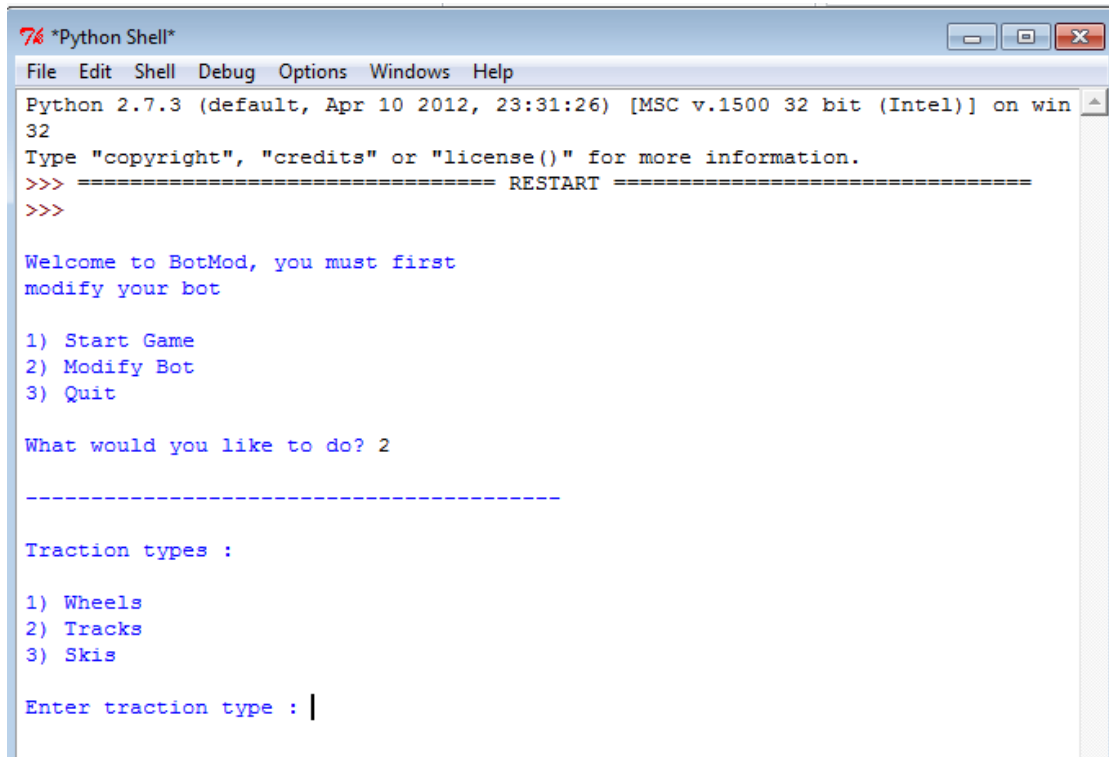
```
Enter desired name

```

```
Name : |
```

Ln: 59 Col: 7

1:2



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>

Welcome to BotMod, you must first
modify your bot

1) Start Game
2) Modify Bot
3) Quit

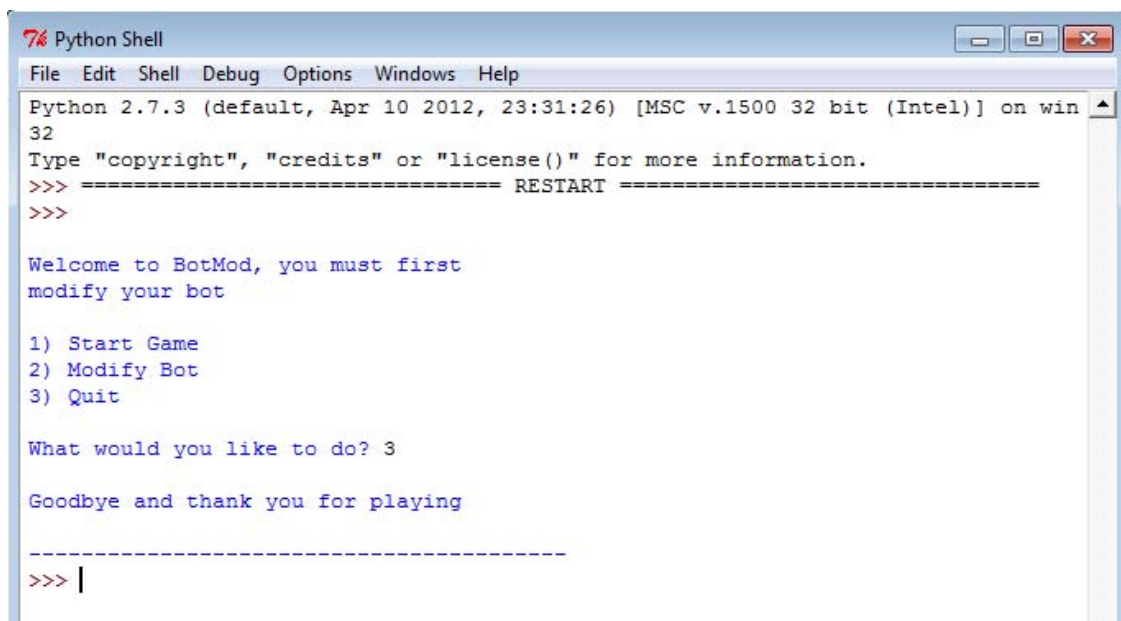
What would you like to do? 2

Traction types :

1) Wheels
2) Tracks
3) Skis

Enter traction type : |
```

1:3



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>

Welcome to BotMod, you must first
modify your bot

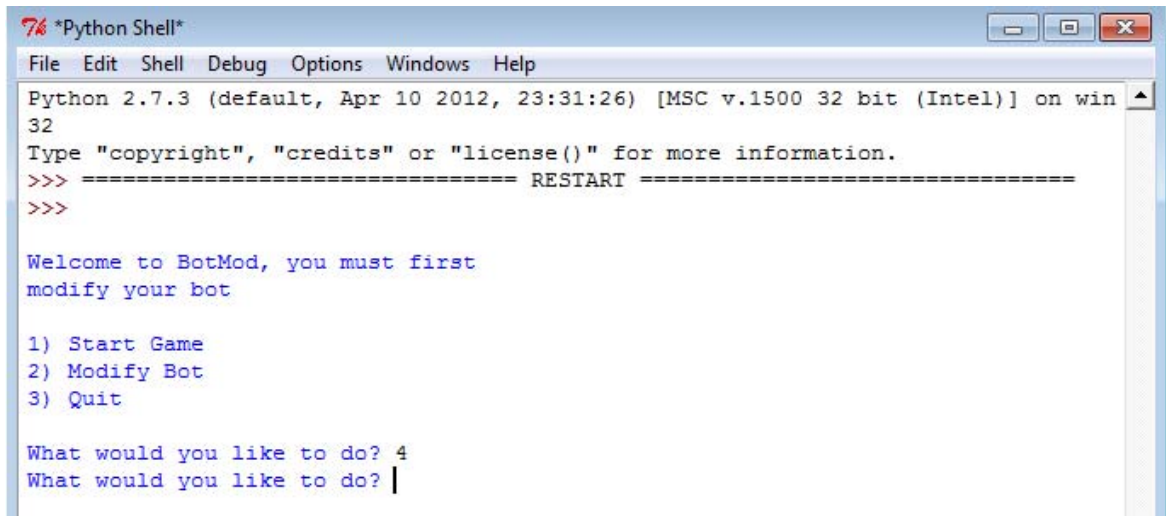
1) Start Game
2) Modify Bot
3) Quit

What would you like to do? 3

Goodbye and thank you for playing

>>> |
```

1:4



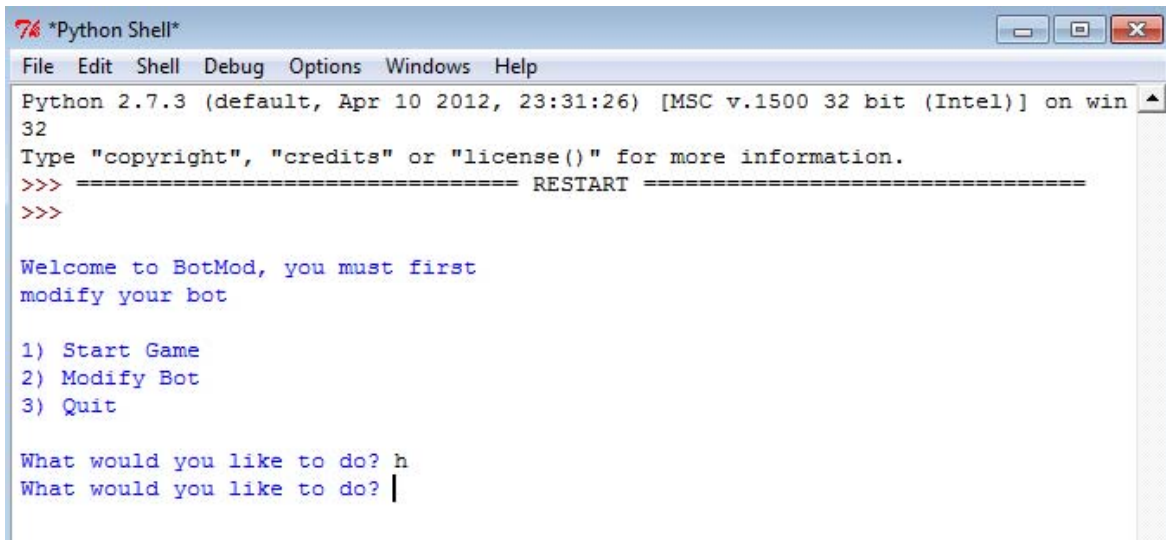
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>

Welcome to BotMod, you must first
modify your bot

1) Start Game
2) Modify Bot
3) Quit

What would you like to do? 4
What would you like to do? |
```

1:5



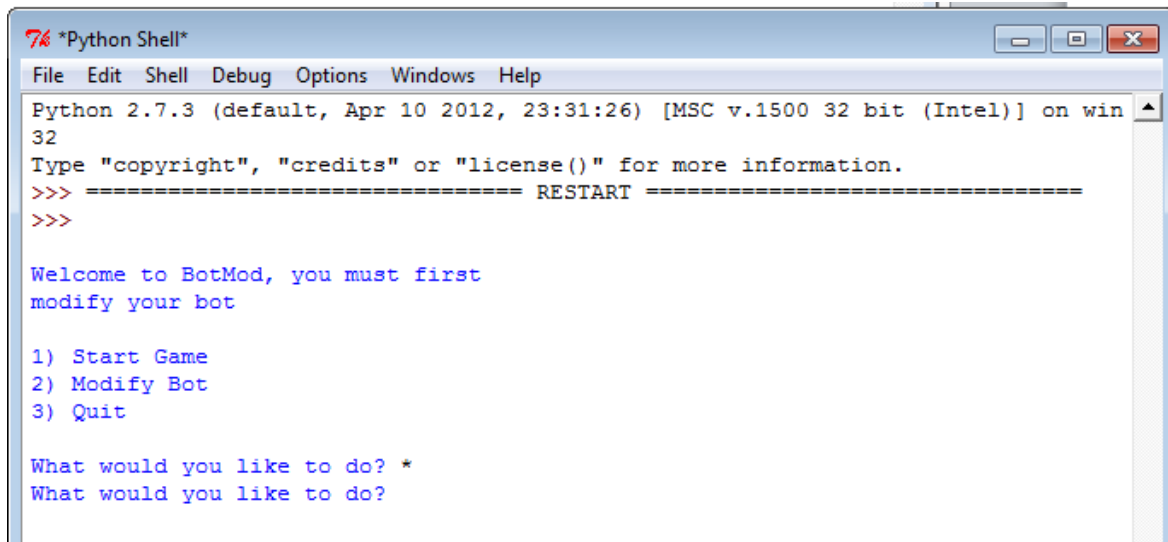
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>

Welcome to BotMod, you must first
modify your bot

1) Start Game
2) Modify Bot
3) Quit

What would you like to do? h
What would you like to do? |
```

1:6



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The text inside the shell shows the Python 2.7.3 interpreter running a script. The script displays a welcome message and a menu with three options: "1) Start Game", "2) Modify Bot", and "3) Quit". It then prompts the user with "What would you like to do? \*".

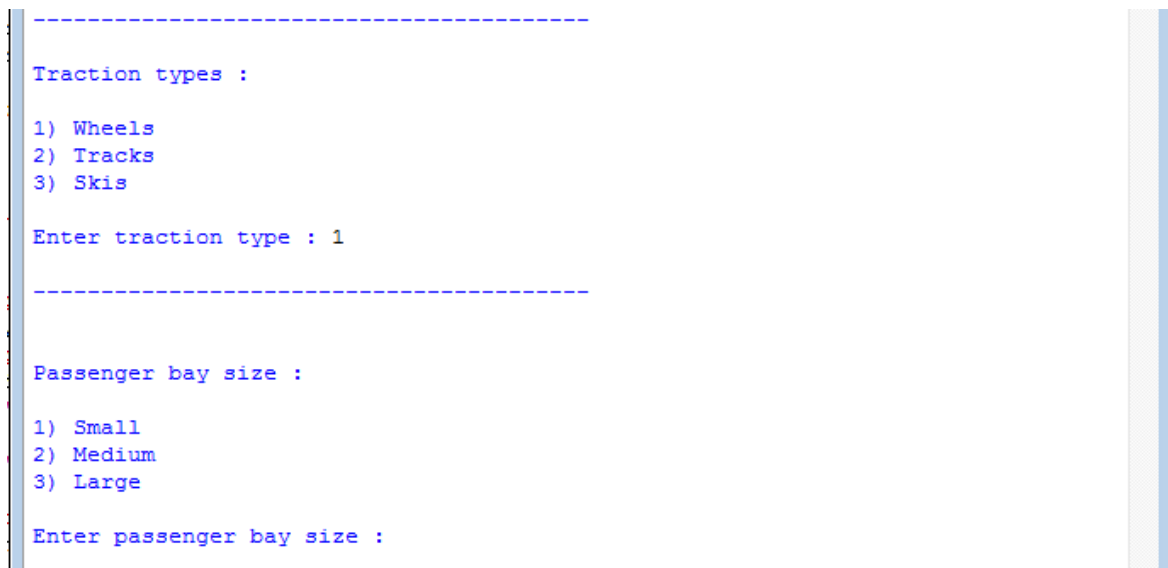
```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>

Welcome to BotMod, you must first
modify your bot

1) Start Game
2) Modify Bot
3) Quit

What would you like to do? *
```

2:1



A screenshot of a text-based menu for the BotMod program. The menu is displayed in a monospaced font. It starts with a separator line of dashes, followed by the heading "Traction types :". Below this, there is a list of three options: "1) Wheels", "2) Tracks", and "3) Skis". The user has entered "1" for the traction type. Another separator line of dashes follows. Then, the heading "Passenger bay size :" is shown, followed by a list of three options: "1) Small", "2) Medium", and "3) Large". The prompt "Enter passenger bay size :" is at the bottom.

```

Traction types :

1) Wheels
2) Tracks
3) Skis

Enter traction type : 1

Passenger bay size :

1) Small
2) Medium
3) Large

Enter passenger bay size :
```

2:2

```

Traction types :
```

- 1) Wheels
- 2) Tracks
- 3) Skis

```
Enter traction type : 2
```

```

Passenger bay size :
```

- 1) Small
- 2) Medium
- 3) Large

```
Enter passenger bay size : |
```

2:3

```

Traction types :
```

- 1) Wheels
- 2) Tracks
- 3) Skis

```
Enter traction type : 3
```

```

Passenger bay size :
```

- 1) Small
- 2) Medium
- 3) Large

```
Enter passenger bay size : |
```



2:4

```
Welcome to BotMod, you must first
modify your bot
```

- 1) Start Game
- 2) Modify Bot
- 3) Quit

```
What would you like to do? 2
```

```

```

```
Traction types :
```

- 1) Wheels
- 2) Tracks
- 3) Skis

```
Enter traction type : 4
```

```
Enter traction type : |
```

2:5

```
Welcome to BotMod, you must first
modify your bot
```

- 1) Start Game
- 2) Modify Bot
- 3) Quit

```
What would you like to do? 2
```

```

```

```
Traction types :
```

- 1) Wheels
- 2) Tracks
- 3) Skis

```
Enter traction type : h
```

```
Enter traction type :
```

2:6

```
Welcome to BotMod, you must first
modify your bot
```

- 1) Start Game
- 2) Modify Bot
- 3) Quit

```
What would you like to do? 2
```

```

```

```
Traction types :
```

- 1) Wheels
- 2) Tracks
- 3) Skis

```
Enter traction type : *
```

```
Enter traction type : |
```

3:1

```

```

```
Passenger bay size :
```

- 1) Small
- 2) Medium
- 3) Large

```
Enter passenger bay size : 1
```

```

```

- 1) Start Game
- 2) Modify Bot
- 3) Quit

```
What would you like to do?
```

Ln: 46 Col: 27

3:2

```

Passenger bay size :

1) Small
2) Medium
3) Large

Enter passenger bay size : 2

1) Start Game
2) Modify Bot
3) Quit

What would you like to do? |
Ln: 46 Col: 27
```

3:3

```

Passenger bay size :

1) Small
2) Medium
3) Large

Enter passenger bay size : 3

1) Start Game
2) Modify Bot
3) Quit

What would you like to do? |
Ln: 46 Col: 27
```

3:4

```

Passenger bay size :

1) Small
2) Medium
3) Large

Enter passenger bay size : 4

Enter passenger bay size : |
```

Ln: 36 Col: 27

3:5

```

Passenger bay size :

1) Small
2) Medium
3) Large

Enter passenger bay size : h

Enter passenger bay size : |
```

3:6

```

Passenger bay size :

1) Small
2) Medium
3) Large

Enter passenger bay size : *

Enter passenger bay size : |
```

4:1

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 150

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : w

Direction : a

Direction :

4:2

```

g	g	i	i	i	i	g	g	i	X
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 129

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : w

Direction : d

Direction : |

4:3

```

g	g	i	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
X	g	g	g	g	g	g	g	i	r

```

The current energy level is : 96

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : s

Direction : a

Direction :

4:4

```

g	g	i	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	X

```

The current energy level is : 84

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : s

Direction : d

Direction :

5:X

There's no print screen(s) for this section as there would have to be 27 different images to show every possibility.



6:1

Before:

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 150

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction :

---

After:

```

g	g	i	X	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 143

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

6:2

Before:

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 150

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction :

---

After:

```

g	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	r	X	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 128

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : |

---

6:3

Before:

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 150

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction :

---

After:

```

g	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	g	X	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 130

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction :

6:4

Before:

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 150

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction :

---

After:

```

g	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	X
r	r	r	r	P	i	i	i	g	g
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 113

There are 1 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : |



7:1

Would you like to play the next level ? (yes/no) : yes

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 140

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck TEster

Direction :

7:2

Would you like to play the next level ? (yes/no) : no

- 1) Start Game
- 2) Modify Bot
- 3) Quit

What would you like to do? |

8:1

Would you like to play the next level ? (yes/no) : yes

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 130

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up  
'a' = left  
's' = down  
'd' =right

Good luck Tester

Direction : |

8:2

Would you like to play the next level ? (yes/no) : no

- 1) Start Game
- 2) Modify Bot
- 3) Quit

What would you like to do? |

---

9:1

```

g	g	P	i	i	i	g	g	i	i
X	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 140

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : w

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i

```

```

g	g	P	i	i	i	g	g	i	i
X	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 147

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : d

```

g	g	P	i	i	i	g	g	i	i
g	X	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i

```

```

g	g	P	i	i	i	g	g	i	i
g	X	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 143

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : a

```

g	g	P	i	i	i	g	g	i	i
X	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i

```

```

X	g	P	i	i	i	g	g	i	i
g	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g
r	r	g	i	g	i	r	r	g	g
r	r	r	i	r	r	r	r	i	g
r	r	r	r	P	i	i	i	g	P
g	r	r	r	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i
g	g	g	g	g	g	g	g	i	r

```

The current energy level is : 150

There are 0 passengers in the bay

You have saved 0 passengers

The player is marked as 'X'

Directions :

'w' = up

'a' = left

's' = down

'd' =right

Good luck Tester

Direction : s

```

g	g	P	i	i	i	g	g	i	i
X	r	r	i	i	i	g	g	i	i
r	r	r	g	g	i	g	P	i	i
i	r	g	g	g	g	B	g	g	g

```

## **Final solution evaluation.**

### **Tasks,**

1) Develop a start menu for the game. This must give the user the options of modifying their robot, playing the game or exiting the program.

I have completed this task, my main menu has the options; Start Game, Modify Bot and Quit. These can be seen in 'Bot Mod Comments' section 4.

2) Develop the part of the program where the player modifies their robot. The player should be able to select their choices and return to the start menu when they have finished. They should be able to make the following choices:

- type of traction they want (wheels, tracks or skis)
- size of the passenger bay (large, medium or small).

I have fulfilled this task the user has the option of the three listed bay sizes and three listed traction types. See 'Bot Mod Comments' section 6.

3) Develop the program so that when the player chooses to play the game from the start menu the landscape and initial positions of the robot, people and base camp are displayed

I've completed this, an image of the board is below:

```

X	g	P	i	i	i	i	g	g	i	i
g	x	x	i	i	i	i	g	g	i	i
x	x	x	g	g	i	g	P	i	i	
i	x	g	g	g	g	B	g	g	g	
x	x	g	i	g	i	x	x	g	g	
x	x	x	i	x	x	x	i	g		
x	x	x	x	P	i	i	i	i	g	P
g	x	x	x	i	i	i	i	i	g	i
g	g	g	g	i	g	g	i	i	i	
g	g	g	g	g	g	g	g	i	x	

```

4) Develop the part of the program to enable the player to move the robot as described.

For evidence see 'Bot Mod Testing and Evaluation' ID 9:1-9:4.

5) Develop the program so that the robot automatically picks up a person when it moves into the square they are occupying if there is enough room in its passenger bay. It should also automatically drop off all its passengers when it moves into the square containing the base camp.

For evidence see 'Bot Mod Testing and Evaluation' ID 6:1-6:4

6) Develop the part of the program that calculates and displays the power units and number of passengers after each move. At the start of every game the robot should have 150 power units.



I've completed this task, see 'Bot Mod Comments' section 13 & 14.

7) Develop the part of the program that checks if the player has won or lost the game.

a. The player has won if all of the people have been taken to the base camp.

b. The player has lost if the number of power units runs out.

I've done these three tasks, for evidence see 'Bot Mod Comments' (a: section 17, b: section 16)

8) Extend the program by creating more levels for the game that increase the difficulty for the player. Each time a player wins a game, they move up a level and the robot starts with fewer power units than on the previous level.

I've completed this task, for evidence see 'Bot Mod Comments' section 17.