AQA Computer Science GCSE Practical Programming

Scenario 3: Game application

Centre Number: 69772

Candidate number: 5211

Lloyd Clowes

Contents

Design of solution

What the problem involves

In this turn-based game, two players compete to move both of their pieces to the end of the board, which consists of 11 spaces. The aim of the game is to win the game by getting both of their pieces to the finish space, the 11th one (FINISH). Each player has two pieces, which both start one on the 1st space (START). The first, fifth and eleventh spaces on the board are safe spaces, where the players pieces cannot be sent back to the start.

The aim of this puzzle is to increase the user's strategic abilities, which can be useful for people with learning disabilities, or who want to think more logically.

The puzzle can be split into nine main tasks:
- Develop a main menu, where the options are 'Enter player names', 'Play game' and 'Quit'
- Allow the users to enter their names and be shown the main menu again. If they don't enter names, make 'Player 1' and 'Player 2', the default names
- When the user chooses 'Play game', the game starts, a board of 11 spaces is displayed. On it, the current positions of the four pieces and which pieces belong to which player
- Generate and displays a random number from 1 to 4 and the action that corresponds to the result.
- Tell them if the selected piece cannot be moved and say to whom the message is directed towards
- Develop the part of the program that allows the player to select which one of their pieces they want to move. If the piece does not meet the criteria in the rules, it cannot be moved
- Move the players position accordingly and if it lands on the same space as an opponents, if it is a safe space, do nothing, however if it isn't, move the opponents position back to the start
- Check if the user has won the game after each turn. If so, show them a congratulations message and go back to the main menu
- Extend the program so that it uses a five sided dice, and if the user gets 5, they can select one of their pieces and move that piece to the next unoccupied space after the space. The piece cannot be moved if there are no unoccupied spaces after the space

Python code has the required features to make this application run through the command prompt on Windows and terminal on OS X. Python is freely available and much online help is available, which means that almost anyone can run it with ease. Python code can also be easily checked and debugged, making it programs less vulnerable to having an error. Python code can be run through emulators on many operating systems. Below is a quote on the official Python website.

"Python is powerful… and fast; plays well with others; runs everywhere; is friendly & easy to learn; is Open" - Python website - https://www.python.org/about/

Storyboard Application

1. The user is presented on screen with the menu
    A. The user chooses to 'Enter player names', enters them and is returned to the menu
    B. The user chooses to 'Play game'. 2-7 (inclusive) show what happens if this is selected
    C. The user chooses to 'Quit' and the program ends
    D. The user chooses to 'View instructions', and they are shown how to play
    E. The user chooses to 'Play with five-sided die'. 3A shows what happens if this is selected
2. The initial state of the game is displayed to the user
3. The user is shown the result of a four-sided dice being thrown, the action that corresponds to the result of the die roll and who the message is directed towards
    A. If the user is playing with five-sided die, and they roll a 5, they can move a chosen piece to move to the next unoccupied space
4. The user is told if no legal move is possible as a result of the die roll
5. The user selects the piece that they want to move and is shown an appropriate message if that piece cannot legally move
6. The user is shown the board, with the result of their move and the opponent's piece's new position at the start if the user landed on the opponents space, while not in a safe space
7. The user is congratulated if they win and if not, it becomes the other player's turn and tasks 3-7 are repeated

The user's needs

Meeting the user's needs is, in my opinion, the most important part of the task, or else there simply wouldn't be any point in undergoing the task. As well as meeting the user's needs, the program needs to be simple, intuitive, easy to navigate and run smoothly and efficiently without any problems. To achieve all these points, I set out to make it exceed the user's needs and make it more useful and beneficial, while still being spontaneous, navigable and relevant. Of course, I plan to complete the instructions given to me (and do some extra), like the following tasks:

1. Display the menu and allow the user choose an option
2. Allow the users to enter their names and be returned to the menu
3. Display the initial state of the game if the user chooses to 'Play game' at the menu
4. Display a randomly generated number and show who the message is directed towards
5. Check if the user cannot move any pieces. If not, display a message telling the user so
6. Allow the user to choose what piece they want to move. If that piece cannot be moved, tell them so
7. Make the move selected by the player
8. Check if the user has won, and if not swap turns
9. Add on an extra bit allowing the user to use a five sided die

Data structures that will be used

String – A sequence of characters that store text as a 'string of bytes', and so the string type can also be used to store binary data. It is implemented to represent an array of Unicode characters, in other words, a text string

Integer - An integer stores a whole number. These can be used to do calculations with, for example to use as a loop counter, so that you can tell how many times a loop has looped
Boolean - Contains a 'True' or 'False' value. In my code, I use this in 'while' loops so that the loop knows whether or not to loop again. If the variable 'repeat' is 'True' it will loop again
Array – Represents an ordered collection of objects, and it provides a high-level interface for sorting and otherwise manipulating lists of data. In python, arrays are mutable, so you can dynamically add or remove items

Use of functions & parameters

In my program, will be about 10 functions. I will use functions to not only split the code into 'blocks' to make it easier to read and debug, but also for when parts of the code must be run multiple times. For example, my code will have a function that will roll and display the dice, showing who the message is directed towards. This must be run multiple times because once a turn is over, it will switch players and roll the dice again. This will repeat until the game ends, when one of the users has gotten both of their pieces on the FINISH space. Here is the text-based algorithm for this function:

```
Get a random number, from 1 to 4
Display who the message is directed towards
Display the result of the die roll
Return the die roll so that the actions can be told to the user
```

The code consists of a single parameter. This parameter is designed to display the instructions, and since this is all it does, there is no point in returning anything, which is what a function does. It does not use any variables from another functions, in fact it does not use any variables. All it contains is the 'print' statement.

Use of built-in functions

The code will make use of Python's multiple built in features, such as:

- input() - getting information from the user, like what piece they want to move
- print() - displaying information to the user, like the board
- int() - returns an integer object constructed from a number or string
- exit() - exits the program, which is used when a user chooses to quit at the menu
- isinstance() - checks if a specific variable is a stated type and returns Boolean value (e.g. checks if a variable is an integer. If it is, it is True)

Flowchart

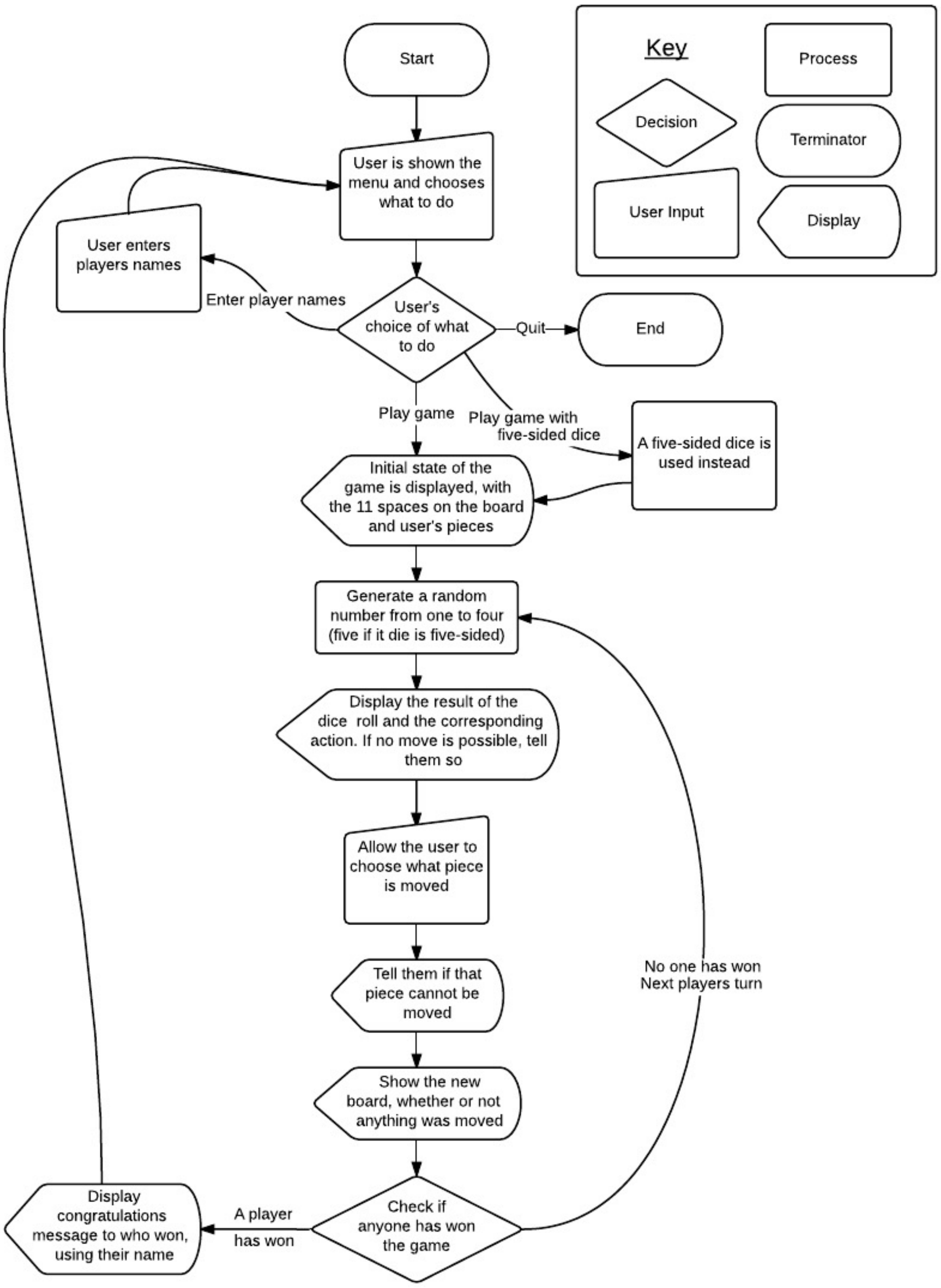

Key
Process
Decision
Terminator
User Input
Display
Start
User is shown the menu and chooses what to do
User's choice of what to do
Quit
End
Enter player names
User enters players names
Play game
Play game with five-sided dice
A five-sided dice is used instead
Initial state of the game is displayed, with the 11 spaces on the board and user's pieces
Generate a random number from one to four (five if it die is five-sided)
Display the result of the dice roll and the corresponding action. If no move is possible, tell them so
Allow the user to choose what piece is moved
Tell them if that piece cannot be moved
Show the new board, whether or not anything was moved
Check if anyone has won the game
No one has won Next players turn
A player has won
Display congratulations message to who won, using their name

FIGURE 1.1 - A FLOWCHART OF THE PROGRAM

Connecting functions

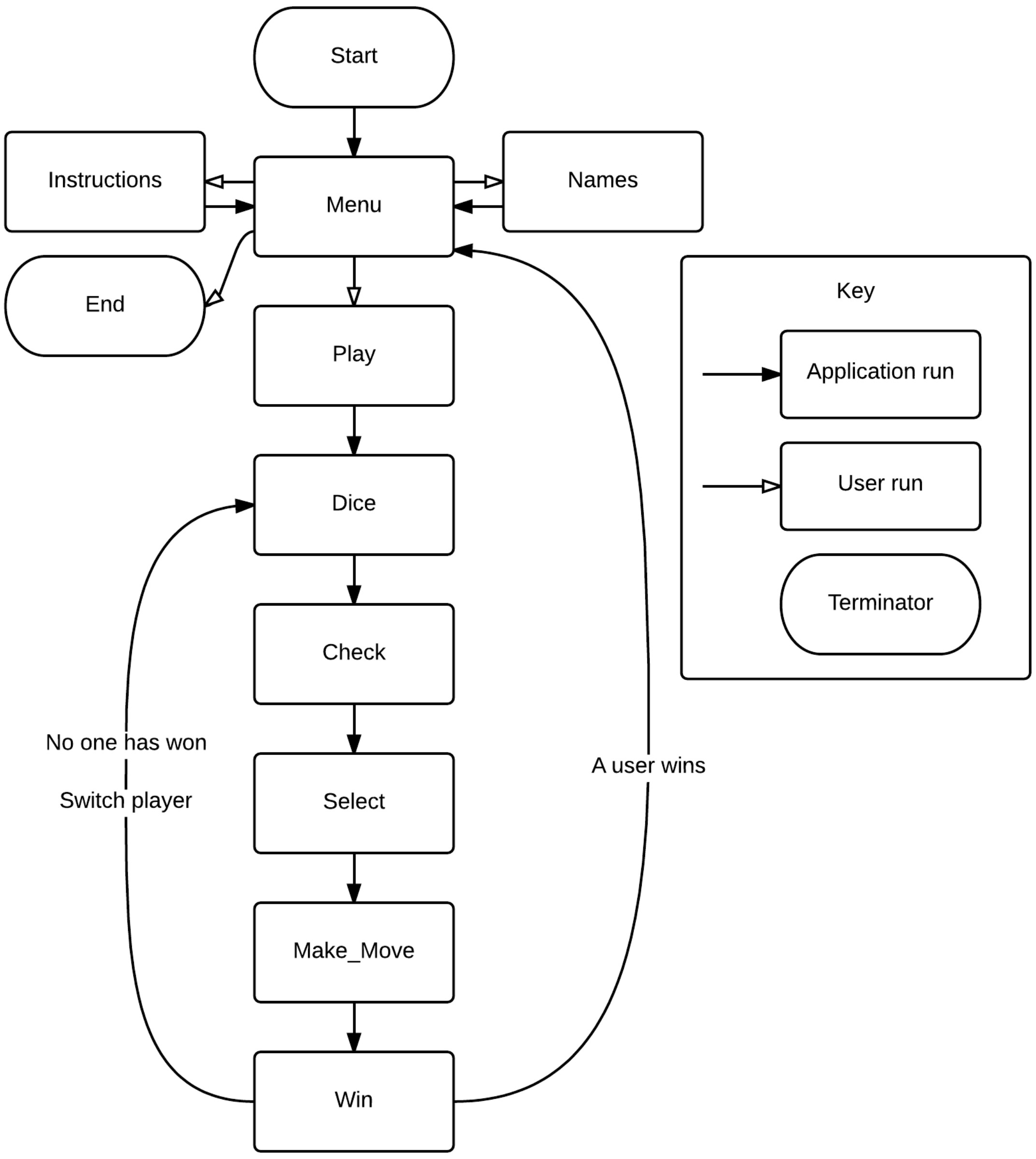


FIGURE 1.2 - A FLOWCHART SHOWING HOW THE FUNCTIONS ARE CONNECTED

The flowchart above shows when a function is run and how the functions are conjoined. The ones that have black arrow are run automatically and the user does not have a choice on this. The white arrows however (look at key for user called) are not run automatically and are run depending on what the user wants to do. The central function is the menu function, called 'Menu'. This is because this is where the user choses what to do next. For example, if they choose to play the game, the function 'Play_game' is run.

Function breakdown

Menu

The flowchart to the right shows the processes involved in the first function that will show the menu. It will start of with displaying and receiving the input for the menu. It will then validate the input to make sure errors are avoided and decide what to do next depending on what the user wants to do. Most of will them result in another function being run, hence why it is a short function.

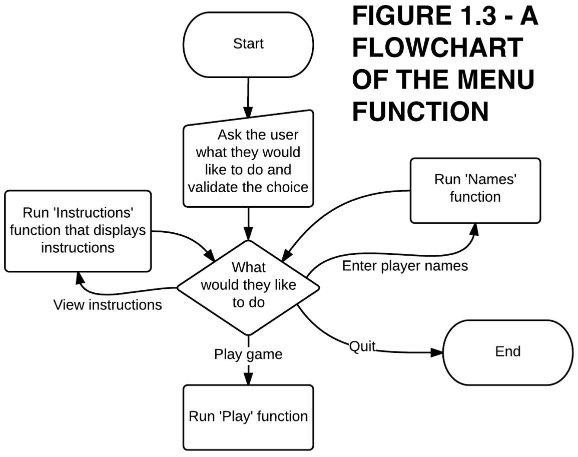

Start
Ask the user what they would like to do and validate the choice
Run 'Instructions' function that displays instructions
Run 'Names' function
What would they like to do
View instructions
Enter player names
Quit
End
Play game
Run 'Play' function

FIGURE 1.3 - A FLOWCHART OF THE MENU FUNCTION

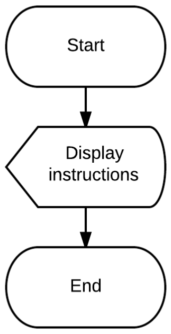


FIGURE 1.4 - A FLOWCHART OF THE INSTRUCTIONS FUNCTION

Instructions

This parameter will just display the rules that a new player would need to know when run. It will be run from the menu function and will lead back to the menu once finished. On the right is the flowchart for this function.

Names

The point of this function is to receive the names of the two players and validate that the names aren't the same and if they are, they will be asked again. The names of the two players will be returned.

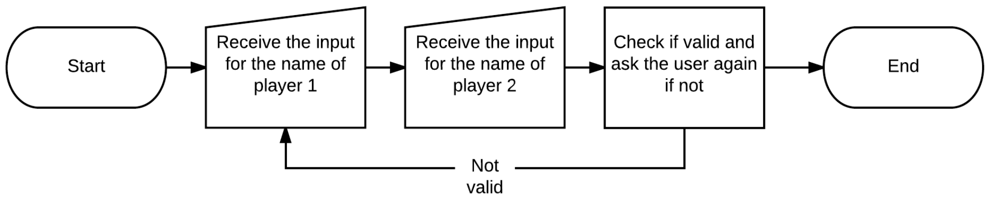

Start
Receive the input for the name of player 1
Receive the input for the name of player 2
Check if valid and ask the user again if not
End
Not valid

FIGURE 1.5 - A FLOWCHART OF THE NAMES FUNCTION

Play

This function will simply display the board, which will probably contain a for loop with a negative stride (so the loop goes from 11 to 1)

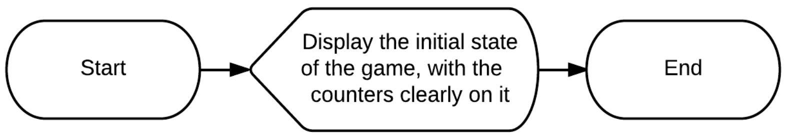

Start
Display the initial state of the game, with the counters clearly on it
End

FIGURE 1.6 - A FLOWCHART OF THE PLAY FUNCTION

Dice

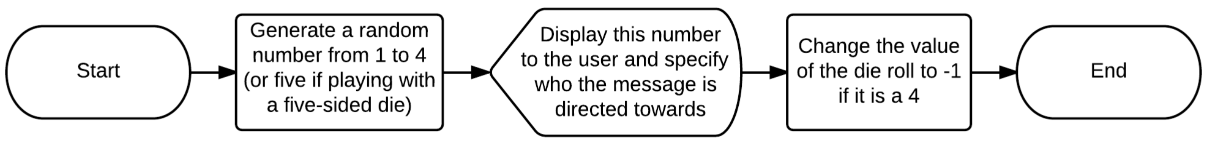

Start
Generate a random number from 1 to 4 (or five if playing with a five-sided die)
Display this number to the user and specify who the message is directed towards
Change the value of the die roll to -1 if it is a 4
End

FIGURE 1.7 - A FLOWCHART OF THE DICE FUNCTION

This function will start off by generating a random number from 1 to 4, however if the user has previously chosen to play with a five-sided die, a random number will be generated from 1 to 5.

The name of the user that is receiving the message will be displayed and they will told what they got on the dice roll. Finally to prevent further errors, the value of the die roll.

Check

This function will check if the result of the die roll will allow a player to move either of their pieces this turn. If no move is possible then an message will be displayed telling them so and the player's turn will end.

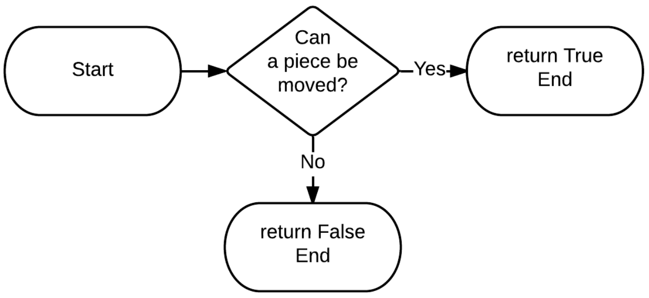


FIGURE 1.8 - A FLOWCHART OF THE CHECK FUNCTION

Select

This function will the user to enter what piece they would like to move, validate it and make sure that the piece can be moved as a result of the die roll. If their selected piece cannot be moved, they will be told so. A piece cannot be moved if the space the piece would be moved to is already occupied by the same player's other piece (unless it is a safe space), if the piece is on the starting space and the result of the die roll is a four, if the piece selected is one of the opponent's pieces or if the piece is on the last space and the result of the die roll is not a four.

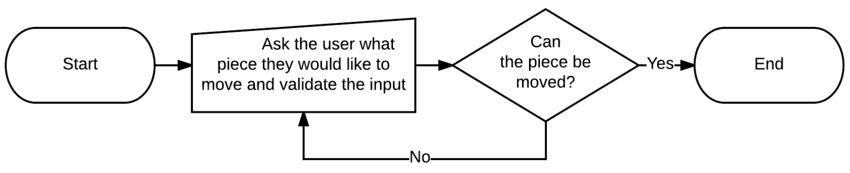

Start
Ask the user what piece they would like to move and validate the input
Can the piece be moved?
Yes
End
No

FIGURE 1.9 - A FLOWCHART OF THE SELECT FUNCTION

Make_move

This function will make the move chosen by the user once it is made sure that the move is valid. A different counter will be used for the different players. If the piece lands on another player's piece that the opponents piece will be moved back to the start.

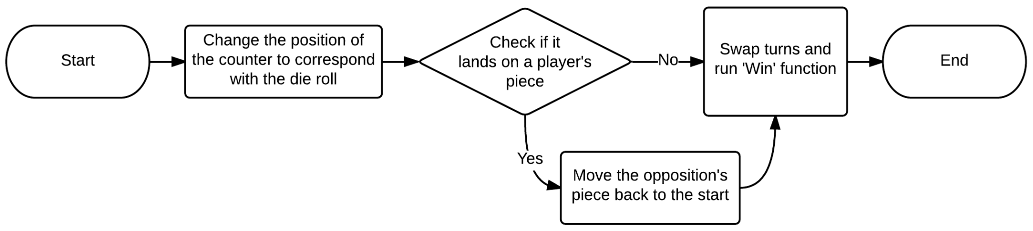

Start
Change the position of the counter to correspond with the die roll
Check if it lands on a player's piece
No
Swap turns and run 'Win' function
End
Yes
Move the opposition's piece back to the start

FIGURE 1.10 - A FLOWCHART OF THE MAKE_MOVE FUNCTION

Win

This function will simply check if anyone has won and display a message if anyone has, showing who the message is directed towards. They are then returned to the menu. Otherwise, the turns swap and the board is displayed again.

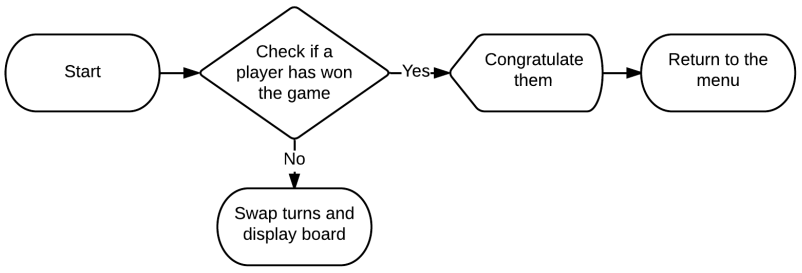

Start
Check if a player has won the game
Yes
Congratulate them
Return to the menu
No
Swap turns and display board

FIGURE 1.11 - A FLOWCHART OF THE WIN FUNCTION

CPU

This function will allow the player to play with a computer. This computer will make it's own decisions based on it's pieces and the other player's pieces. The decisions will be based on whether or not a piece lands on space 11, if a piece lands on another players piece or if a player's piece lands on a safe space. If neither of these apply, a random piece will be chosen.

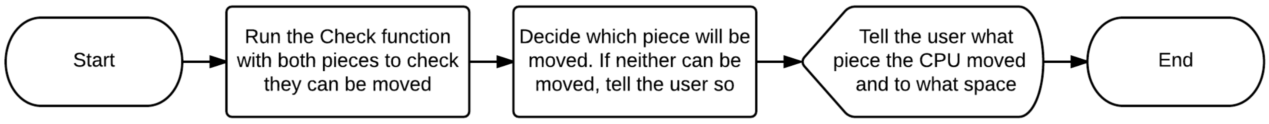

Start
Run the Check function with both pieces to check they can be moved
Decide which piece will be moved. If neither can be moved, tell the user so
Tell the user what piece the CPU moved and to what space
End

FIGURE 1.12 - A FLOWCHART OF THE CPU FUNCTION

Solution development

How it meets the user's needs

The program meets the user's needs in all ways. As described in 'The User's Needs' in the 'Design of solution' section, the program needs to be simple, intuitive, easy to navigate and run smoothly and efficiently without any problems. I also said that I would have to do all of the tasks given, or else I won't have fulfilled the user's needs. My code not only does all of the tasks efficiently, but also is easy to use and understand, and is quick and intuitive. No matter what the situation, no red errors will occur and everything will work as it is meant to.

How the original tasks have been catered for

Each of the original tasks have been split into different functions, making it easier to view and debug. Below is a list of the tasks with the line numbers on them showing where each of the tasks have been fulfilled. This references to the end of the document, containing the entire final solution.

1. Display a menu allowing the user to enter the player names, play the game and quit - Function: Menu
2. Allow the user to enter their names if the user chooses to do so - Function: Names
3. Display the initial state of the game if the user chooses to play game - Function: Play
4. Generate a random number from 1-4 (5 if using extended dice) and tell them what can be done - Function: Dice
5. Check if the user can move any of their pieces and if not, end their turn - Function: Check
6. Allow the user to choose what piece they want to move and do so if that move is legal - Function: Select + Valid
7. Make the move selected by the player, and if it lands on another player's piece, move that piece back to the start - Function: Make_move
8. Check if the player has won after each move. If so, congratulate them and if not, end their turn - Function: Win
9. Extend the program to so that it uses a five sided die. If a player rolls 5, move to the next unoccupied space - Function: Check + Make_move

Annotated code

```
import random #Make sure a random number can be accessed for the dice roll
import time  #Allow there to be a delay before doing certain things
```

Here, I use what's called double quotes so that it displays it exactly as I have written it out, making it more efficient so that I don't have to keep using different print statements

```
    choice=input('''What would you like to do?
    1. Enter player names
    2. Play game
    3. Quit
    4. View instructions
    5. Play with a five-sided die
    6. Play with a computer\n''')#Show the menu to the user and allow them to enter their option
    while choice!='1' and choice!='2' and choice!='3' and choice!='4' and choice!='5' and choice!='6':#Validate the input
        print('\aPlease enter a valid answer: 1, 2, 3, 4, 5 or 6')
        choice=input('''What would you like to do?
    1. Enter player names
    2. Play game
    3. Quit
    4. View instructions
    5. Play with a five-sided die
    6. Play with a computer\n''')


    if choice=='1':#Ask the user to enter their names if they choose to do so
        (info)=Names()
    if choice=='2' or choice=='5' or choice=='6':#Allow the user to start the game
        if choice=='6':
            info.name2, info.CPU='CPU', True
        elif choice=='5':
            info.roll=5
        Play(info, position)
    elif choice=='3':#Allow the user to quit the game
        exit()
    elif choice=='4':#Allow the user to view the instructions
        Instructions()
    Menu(info) #Return to the main menu


def Instructions():#Display the instructions
    print('''The players take it in turns to roll a four-sided die
The result of the die roll determines the move that you are allowed to make on the board
e.g. if you roll a 1, you can move one of your pieces one space nearer to FINISH
However, if you roll a 4, you move one of your pieces one space back towards START
The aim of the game is to move both of the player\'s pieces to the end of the 11-space board
If your piece lands on a space with an opponent's piece on, then the opponent's piece is moved back to START
BUT, a piece on a safe space cannot be sent back to START
You can\'t move your piece onto a space already that has your other piece on unless the space is a safe space
A piece cannot move backwards if it is on START or forwards if it is on FINISH
If a player can make a move then they must do so
A player wins the game when both of their pieces are on FINISH\n''')
```

```
class Info:
    def __init__(self, name1, name2, CPU, roll, turn):
        self.name1=name1 #Make a class variable for player 1's name
        self.name2=name2 #Make a class variable for player 2's name
        self.CPU=CPU #Make a class variable for whether or not the CPU is playing
        self.roll=roll #Make a class variable for the maximum roll that can be gotton (4 or 5)
        self.turn=turn #Make a class variable for whose turn it is


#Task 2 - Allow the players to enter their names
def Names():
    info.name1=input('Please enter the name of player 1: ')#Allow the users to enter their names
    info.name2=input('Please enter the name of player 2: ')
    while info.name1==info.name2:#Make sure that they don't enter the same name twice
        print('\aPlease do not enter the same name twice...')
        info.name1=input('Please enter the name of player 1: ')
        info.name2=input('Please enter the name of player 2: ')
    return(info)


#Task 3 - Display the board
def Play(info, position):
    for count in range(11, 0, -1):#Create a for loop with a negative stride so that it counts downwards (11 is at the top)
        array=['X', 'X', 'O', 'O'] #Make an array containing the 4 counters
        for counter in range(4):#Make a for loop that loops 4 times
            if position[counter]!=count:#If the space number isn't the position of the counter...
                array[counter]=' ' #Change the counter of this space for that column to an empty space


    if count==5 or count==4 or count==1 or count==11 or count==10:#Print this if there is a safe space line (asterisk) above it
        print('*'*29)
    else:#Print this if there is a safe space line above it (dash line)
        print('-'*26)
    print('|  '+str(count)+'  |  '+array[0]+'  |  '+array[1]+'  |  '+array[2]+'  |  '+array[3] +'  |')
print('*'*29) #Print the final asterisked line at the end
Win(position, info)#Run the funtion that checks if the player has won
Dice(info, position)


#Task 4 - Generate a random number to simulate the rolling of a four-sided die
def Dice(info, position):
    roll=random.randint(1, info.roll)#Generate a radom number using a variable containing the highest roll it can do
    if info.turn==True:#Display the dice roll and the player that the message is directed towards
        print(info.name1+' got '+str(roll)+' on the dice roll!')
    else:
        print(info.name2+' got '+str(roll)+' on the dice roll!')
    if roll==4:#Make the die roll -1 if it rolls a 4 so that it goes back one space
        roll=-1


if info.CPU==True and info.turn==False:#Run the function that allows the computer to choose if the player chose to do so
    CPU(position, roll, info)
else:
    if info.turn==True:
        valid=[Check(position, 1, roll), Check(position, 2, roll)]#Check if player1's pieces can be moved
    else:
        valid=[Check(position, 3, roll), Check(position, 4, roll)]
    if valid[0]==False and valid[1]==False:#If neither of the pieces can move, swap goes and display board
        print('\aNeither of your pieces can be moved...'), time.sleep(1)
        info.turn = not info.turn#Swap turns
        Play(info, position)#Print the board again
    if valid[1]==False:#If only piece 1 can, tell the user and swap goes
        print('\aPiece 2 cannot be moved')
    elif valid[0]==False:#If only piece 2 can, tell the user and swap goes
        print('\aPiece 1 cannot be moved\n')
Select(roll, position, info)
```

```
#Task 5 - Check if the result of the die roll will allow a player to move either of their pieces
def Check(position, move, roll):
    piece=position[move-1]#Make a variable for this commonly used position to make it more efficient
    if roll!=-1 and piece==11:#Make sure that if they get 4 while on 11, they can't move it foward
        return False
    if roll==5:#Allow them to use a 5 sided dice
        loopcount=1#Make a loop counter variable


        while piece+loopcount<11:#Make sure they don't go past 11
            if piece+loopcount==position[0] or piece+loopcount==position[1] or piece+loopcount==position[2] or piece+loopcount==position[3]:#Check if that move lands on another player's piece
                loopcount+=1
            else:
                return loopcount
        return False
    if piece+roll>=11:#Allow them to move a piece past 11
        return True


    if piece+roll!=5 and piece+roll!=1:#Do the following if not on a safe space
        if piece+roll<1:#Make sure that it won't go under 1 or over 11
            return False
        if move-1==0 and position[0]+roll==position[1]:#Make sure that it won't equal the other piece of that player
            return False
        elif move-1==1 and position[1]+roll==position[0]:#Make sure that it won't equal the other piece of that player
            return False
        elif move-1==2 and position[2]+roll==position[3]:#Make sure that it won't equal the other piece of that player
            return False
        elif move-1==3 and position[3]+roll==position[2]:#Make sure that it won't equal the other piece of that player
            return False#If the piece cannot be moved, return a False boolean value
    return True#If the piece cannot be moved, return a True boolean value
```

```
#Task 6 - Allow the player to select which one of their pieces they want to move
def Select(roll, position, info):
    valid, move=Valid(position, roll)
    while valid==False:#Validate the input by checking if that piece can be moved
        print('\a\nThat piece cannot be moved...')
        valid, move=Valid(position, roll)
    Make_move(move, roll, position, info, valid)


def Valid(position, roll):
    move=input('Which piece would you like to move? ') #Ask the user what piece they want to move
    while (move!='1' and move!='2'):#Validate the input
        print('\a\nPlease enter a valid option (1 or 2)...')
        move=input('Which piece would you like to move? ')
    move=int(move)#Change it to an integer so that the number can be changed easily
    if info.turn==False:#Add 2 to player 2's piece moves, as theirs is 2 to the right of player 1's
        move+=2
    valid=Check(position, move, roll)
    return valid, move
```

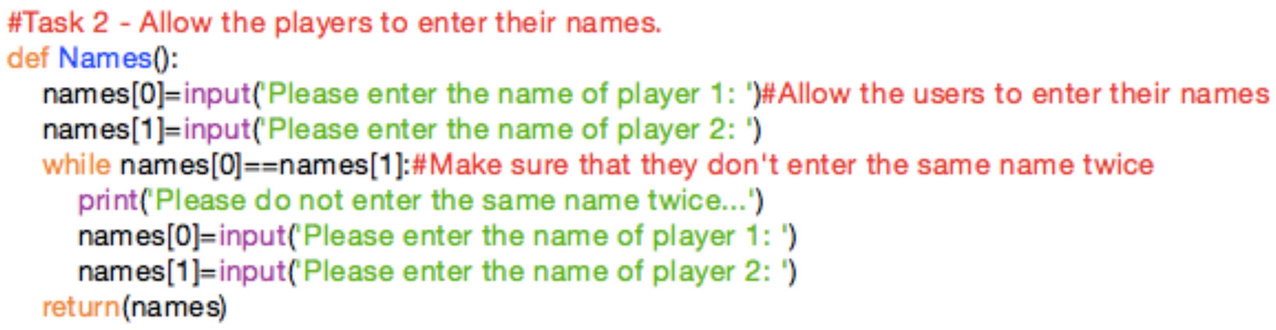
```
#Task 7 - Make the legal move selected by the player
def Make_move(move, roll, position, info, valid):
    if info.CPU==True and info.turn==False: #Do the following if it is the CPU's turn
        valid=True
        print('The CPU moves piece ' + str(move-2)+' to space '+str(position[move-1]+roll)), time.sleep(1) #Tell the user what the CPU got
    if position[move-1]+roll>11:
        position[move-1]=11 #If their move lands over 11, make it land on 11
    else:
        if isinstance(valid, bool)==False: #If valid is not a boolean value...
            position[move-1]=position[move-1]+valid #Add this valid value onto the piece's position


        else:
            position[move-1]=position[move-1]+roll#Change the position of the counter
        if position[move-1]!=5 and position[move-1]!=11:#Do the following when not on a safe space
            if (move-1==0 or move-1==1) and (position[move-1]==position[2]):#Check if any of player 1's pieces are on player 2's piece 1
                position[2]=1#If so, take the opponent's piece back to the start
            elif (move-1==0 or move-1==1) and (position[move-1]==position[3]):#Check if any of player 1's pieces are on player 2's piece 2
                position[3]=1#If so, take the opponent's piece back to the start
            elif (move-1==2 or move-1==3) and (position[move-1]==position[0]):#Check if any of player 2's pieces are on player 1's piece 1
                position[0]=1#If so, take the opponent's piece back to the start
            elif (move-1==2 or move-1==3) and (position[move-1]==position[1]):#Check if any of player 2's pieces are on player 1's piece 2
                position[1]=1#If so, take the opponent's piece back to the starts
    info.turn=not info.turn#Swap the turn to the other player
    Play(info, position)#If no one has won, the code prints the board again


#Task 8 - Extra - Check if a player has won the game after each move
def Win(position, info):
    if position[0]==11 and position[1]==11:#Check if player 1 has won
        print('\a\n\aCongratulations! '+ info.name1+' has won!\n'), time.sleep(1)#Display a congratulations message if player 1 has won
        Menu(info) #Return to the main menu
    if position[2]==11 and position[3]==11:#Check if player 2 has won
        print('\a\n\aCongratulations! '+ info.name2+' has won!\n'), time.sleep(1)#Display a congratulations message if player 2 has won
        Menu(info) #Return to the main menu

#Extra - Allow the user to play with a computer
def CPU(position, roll, info):
    valid=[Check(position, 3, roll), Check(position, 4, roll)]
    if valid[0]==False and valid[1]==False:
        info.turn=not info.turn
        print('\aNeither of the CPU\'s pieces can be moved'), time.sleep(1)
        Play(info, position)#Print the board again
    if position[2]+roll==position[0] or position[2]+roll==position[1] and valid[0]==True:#Check if piece 1 lands on another player's piece
        Make_move(3, roll, position, info, valid)
    elif position[3]+roll==position[0] or position[3]+roll==position[1] and valid[1]==True:#Check if piece 2 lands on another player's piece
        Make_move(4, roll, position, info, valid)


    elif position[2]+roll==5 or position[2]+roll==1 and valid[0]==True:#Check if piece 1 lands on a safe space
        Make_move(3, roll, position, info, valid)
    elif position[3]+roll==5 or position[3]+roll==1 and valid[1]==True:#Check if piece 2 lands on a safe space
        Make_move(4, roll, position, info, valid)
    elif valid[0]==False and valid[1]==True:
        move=4
    elif valid[1]==False and valid[0]==True:
        move=3
    if valid[0]==True and valid[1]==True:
        move=random.randint(3, 4)#Generate a random number
    Make_move(move, roll, position, info, valid)


info=Info('Player 1', 'Player 2', False, 4, True)
Menu(info)
```

Programming techniques

Programming techniques used

Here, I will evaluate the programming techniques used within the program, explaining where they were and why they were used over other methods.

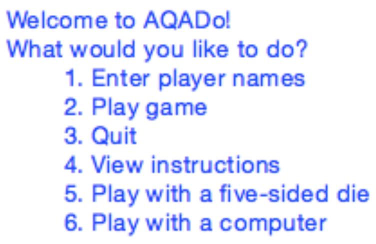
Firstly, I used functions because some parts of the program were very repetitive. For example, the program has to display the board multiple times, so I put the functionality of this into a single function, so that the program can efficiently display the board using the same code.
I also used functions as an attempt to simplify the program a bit. This is because each function is easily distinguishable and stands out. With the help of comments in the program using '#', anyone looking at the code will understand the layout of it and know how the tasks are split up.



```
#Task 2 - Allow the players to enter their names.
def Names():
    names[0]=input('Please enter the name of player 1: ')#Allow the users to enter their names
    names[1]=input('Please enter the name of player 2: ')
    while names[0]==names[1]:#Make sure that they don't enter the same name twice
        print('Please do not enter the same name twice...')
        names[0]=input('Please enter the name of player 1: ')
        names[1]=input('Please enter the name of player 2: ')
    return(names)
```

FIGURES 3.1 - AN EXAMPLE OF A FUNCTION

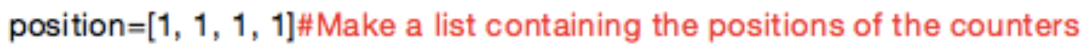
The output format I used was chosen that way to make the program as simple and easy to understand as possible, while still not looking like paragraphs of information. For example, instead of displaying all of the instructions, I simplified them to make it less lines that are still easy to understand and don't look as cluttered. Also, I used '\n' to place empty lines between different parts of the program. This makes it look a lot neater and less compact.



```
Welcome to AQADo!
What would you like to do?
        1. Enter player names
        2. Play game
        3. Quit
        4. View instructions
        5. Play with a five-sided die
        6. Play with a computer
```

FIGURE 3.2 - AN EXAMPLE OF THE OUTPUT FORMAT

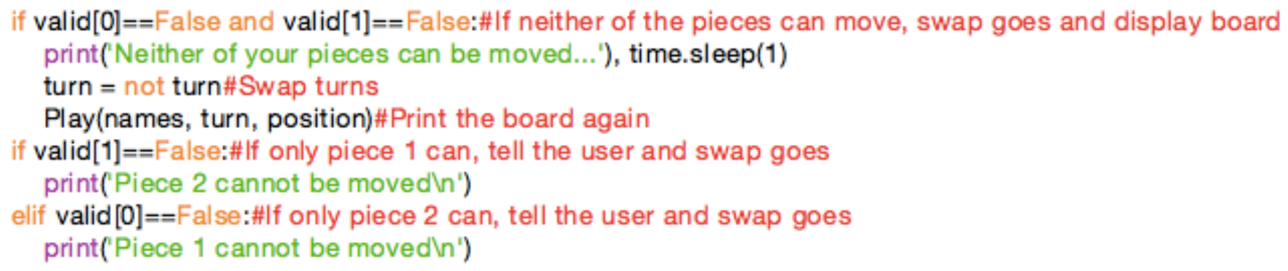
I decided to use arrays in parts of my program instead of strings to reduce the amount of variables, so that I didn't have to put a long list of strings into separate variables and make the code a lot longer, which would be less efficient and would make it harder to call on when using functions. This is because there would be a lot more parameters, increasing the risk of a human error and making the code more prone to errors.



```
position=[1, 1, 1, 1]#Make a list containing the positions of the counters
```

FIGURE 3.3 - AN EXAMPLE OF ARRAYS

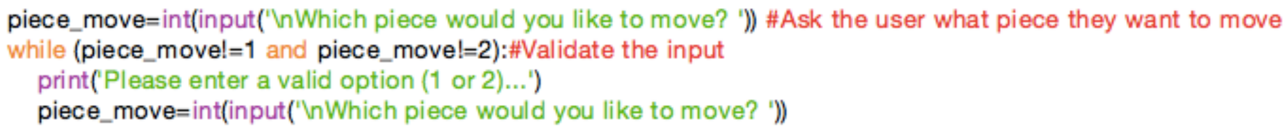
The ‘if’, ‘elif’ and ‘else’ functions were used in my program, mainly to decide what to do when the user enters an input. For example, when the user is asked what they would like to do next at the menu, an ‘if’ statement was used, so that a different algorithm can be run depending on what the user enters.



```
if valid[0]==False and valid[1]==False:#If neither of the pieces can move, swap goes and display board
    print('Neither of your pieces can be moved...'), time.sleep(1)
    turn = not turn#Swap turns
    Play(names, turn, position)#Print the board again
if valid[1]==False:#If only piece 1 can, tell the user and swap goes
    print('Piece 2 cannot be moved\n')
elif valid[0]==False:#If only piece 2 can, tell the user and swap goes
    print('Piece 1 cannot be moved\n')
```

FIGURE 3.5 - AN EXAMPLE OF ‘IF’, ‘ELIF’ AND ‘ELSE’

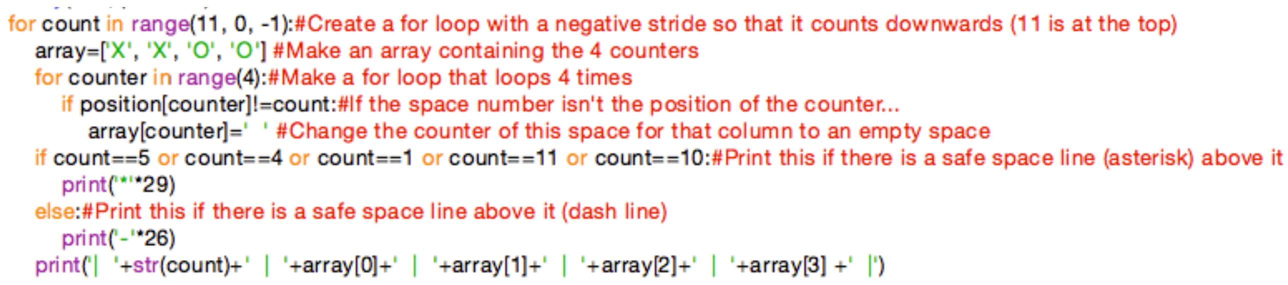
‘while’ loops were also used. This due to the fact that the program must repeat a lot, and lots of inputs must be validated. One example of this is to check if a user has won the game. Another example is if they answer a question with an invalid response, as well as displaying an error message. I used a ‘while’ loop over other methods because it is the most efficient way to do these tasks.



```
piece_move=int(input('\nWhich piece would you like to move? ')) #Ask the user what piece they want to move
while (piece_move!=1 and piece_move!=2):#Validate the input
    print('Please enter a valid option (1 or 2)...')
    piece_move=int(input('\nWhich piece would you like to move? '))
```

FIGURE 3.6 - AN EXAMPLE OF A ‘WHILE’ LOOP

I used ‘for’ loops in my program so that a section of code can be run multiple times, while being able to know how many times that code has run for. An essential example of this is displaying the board, where each space is printed for every time the code runs. It must use a negative stride, so that the ‘for’ loop counts downwards (as the highest number is printed at the top of the board)



```
for count in range(11, 0, -1):#Create a for loop with a negative stride so that it counts downwards (11 is at the top)
    array=['X', 'X', 'O', 'O'] #Make an array containing the 4 counters
    for counter in range(4):#Make a for loop that loops 4 times
        if position[counter]!=count:#If the space number isn't the position of the counter...
            array[counter]='  ' #Change the counter of this space for that column to an empty space
    if count==5 or count==4 or count==1 or count==11 or count==10:#Print this if there is a safe space line (asterisk) above it
        print('*'*29)
    else:#Print this if there is a safe space line above it (dash line)
        print('-'*26)
    print('|  '+str(count)+'  |  '+array[0]+'  |  '+array[1]+'  |  '+array[2]+'  |  '+array[3] +'  |')
```

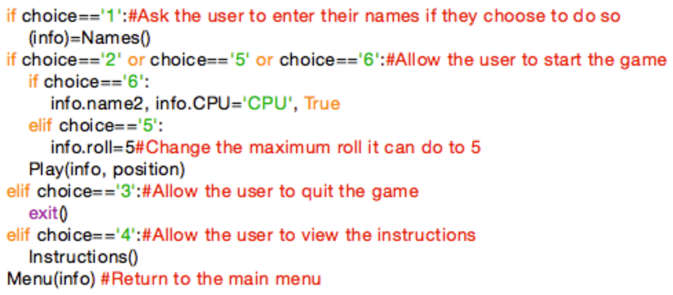
FIGURE 3.7 - AN EXAMPLE OF A ‘FOR’ LOOP

As well as putting the different sections of the program into functions, I used comments throughout the program to make it easier to read, not only for another person trying to understand the code, but also to help me by making it easier to see where I went wrong in the code if there is a problem and how I could fix the problem.

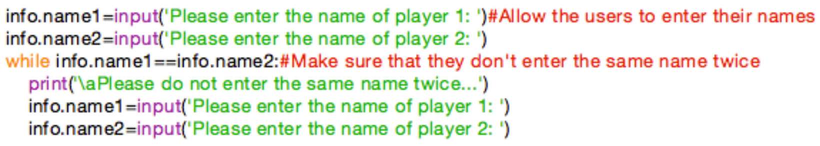
In my program I used 1 module which is imported at the start: random. I did this to allow the user to get a random number from 1 to 4 (5 if playing with a five sided die) which decides what the user can do next, and is necessary to the purpose of the game.

How the parts work together

The menu function uses a selection process to decide what to do after the user has inputted what they want to do. For example, if the want to enter the player's names, the Names function is run. If the code run leads back to the menu again (like in the code below where 'names' is returned), the Menu function is run again and it goes back to the start of the function.

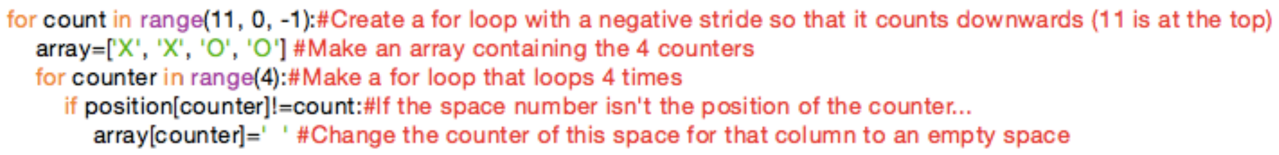


```
if choice=='1':#Ask the user to enter their names if they choose to do so
   (info)=Names()
if choice=='2' or choice=='5' or choice=='6':#Allow the user to start the game
   if choice=='6':
      info.name2, info.CPU='CPU', True
   elif choice=='5':
      info.roll=5#Change the maximum roll it can do to 5
   Play(info, position)
elif choice=='3':#Allow the user to quit the game
   exit()
elif choice=='4':#Allow the user to view the instructions
   Instructions()
Menu(info) #Return to the main menu
```



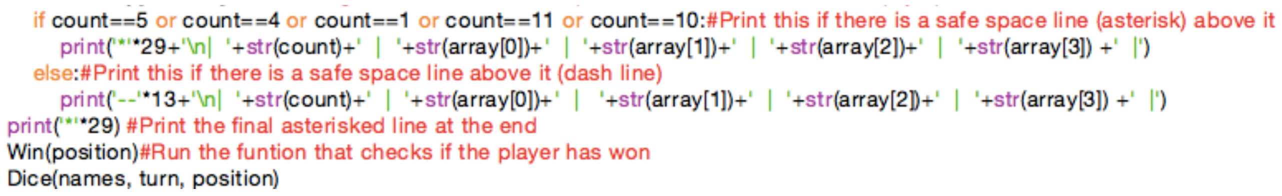
```
info.name1=input('Please enter the name of player 1: ')#Allow the users to enter their names
info.name2=input('Please enter the name of player 2: ')
while info.name1==info.name2:#Make sure that they don't enter the same name twice
   print('\aPlease do not enter the same name twice...')
   info.name1=input('Please enter the name of player 1: ')
   info.name2=input('Please enter the name of player 2: ')
```

In the function to the left, a simple 'while' loop checks if the names are the same, and while they are, loop back again and ask them again.



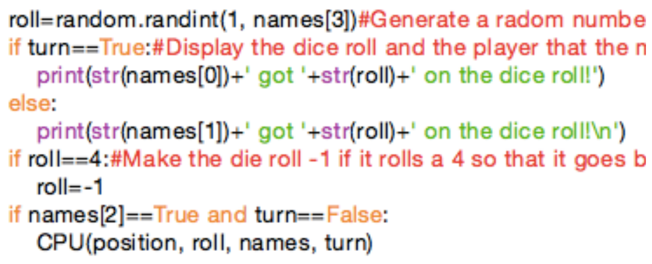
```
for count in range(11, 0, -1):#Create a for loop with a negative stride so that it counts downwards (11 is at the top)
   array=['X', 'X', 'O', 'O'] #Make an array containing the 4 counters
   for counter in range(4):#Make a for loop that loops 4 times
      if position[counter]!=count:#If the space number isn't the position of the counter...
         array[counter]=' ' #Change the counter of this space for that column to an empty space
```

The code above shows the first part of the program that displays the board. This part essential shows what should be on each space of each line. Using the outer and inner 'for' loop and the 'if' statement, it decides for each line whether a counter should be there or if an empty space should be there. If there should be an empty space there, it overrides the list 'array' and changes that space to an empty space. It does this by checking if the space that the outer 'for' loop (the first line) matches a position in the array 'position'.



```
   if count==5 or count==4 or count==1 or count==11 or count==10:#Print this if there is a safe space line (asterisk) above it
      print('*'*29+'\n| '+str(count)+' | '+str(array[0])+' | '+str(array[1])+' | '+str(array[2])+' | '+str(array[3]) +' |')
   else:#Print this if there is a safe space line above it (dash line)
      print('--'*13+'\n| '+str(count)+' | '+str(array[0])+' | '+str(array[1])+' | '+str(array[2])+' | '+str(array[3]) +' |')
print('*'*29) #Print the final asterisked line at the end
Win(position)#Run the funtion that checks if the player has won
Dice(names, turn, position)
```

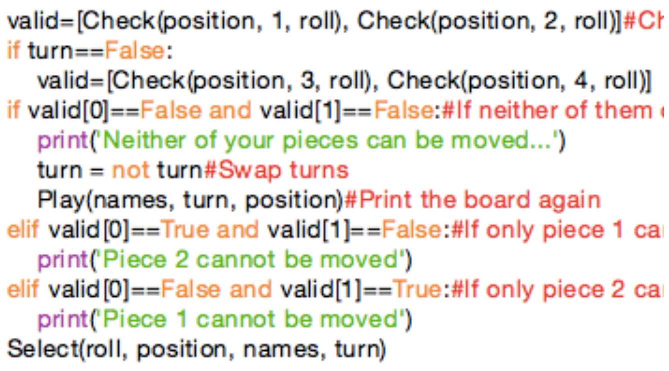
Above is the second part of printing the board. This is still part of the outer 'for' loop in the code above this one. 'count' is the space that the 'for' loop is currently on. The first line of this code checks if there should be a safe space line above the space currently on, for example space 10 and 11 must have an asterisked line (safe space line) above it so that they are around space 11. Otherwise, a normal dashed line is printed. The last print statement in this code is an asterisked line which is printed because usually the line printed is the one above it, so without this print statement, there would be no line below the bottom space.



```
roll=random.randint(1, names[3])#Generate a radom numbe
if turn==True:#Display the dice roll and the player that the n
    print(str(names[0])+' got '+str(roll)+' on the dice roll!')
else:
    print(str(names[1])+' got '+str(roll)+' on the dice roll!\n')
if roll==4:#Make the die roll -1 if it rolls a 4 so that it goes b
    roll=-1
if names[2]==True and turn==False:
    CPU(position, roll, names, turn)
```

This code simply gets a random number using the 'random' module used earlier and displays it depending on whose turn it is. If it is a 4, it is then turned to -1 so that it goes back a space, however this is done after displaying the roll so that it doesn't display -1.

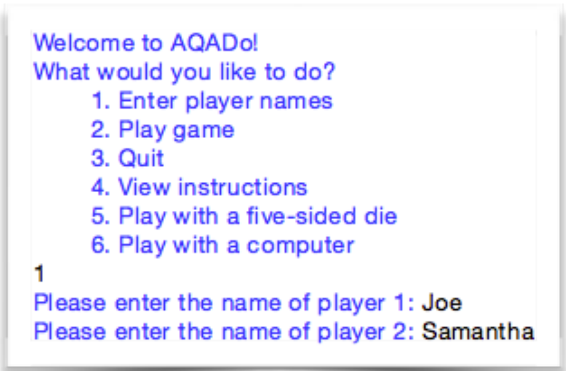
The code to the left checks if each of the pieces are valid by running the Check function and telling the user the corresponding action that can be done. This is done by using a list to store two Boolean values (one for each piece) and telling them which pieces can't be moved.



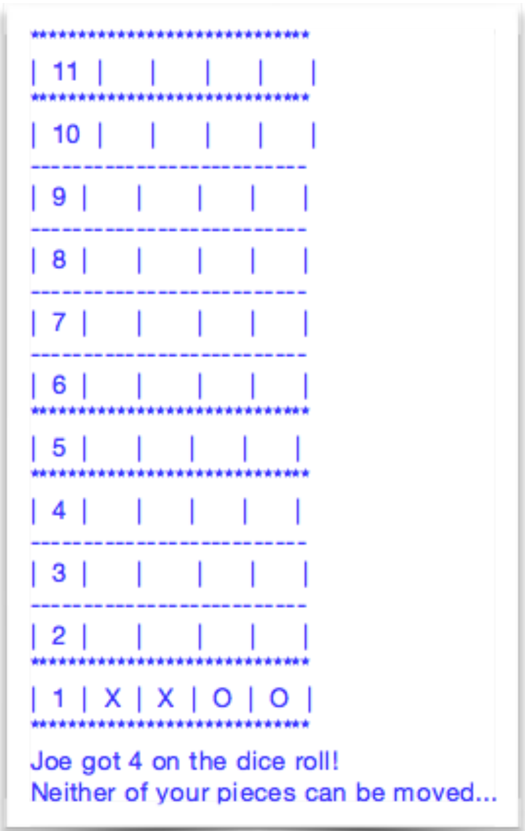
```
valid=[Check(position, 1, roll), Check(position, 2, roll)]#Ch
if turn==False:
    valid=[Check(position, 3, roll), Check(position, 4, roll)]
if valid[0]==False and valid[1]==False:#If neither of them
    print('Neither of your pieces can be moved...')
    turn = not turn#Swap turns
    Play(names, turn, position)#Print the board again
elif valid[0]==True and valid[1]==False:#If only piece 1 ca
    print('Piece 2 cannot be moved')
elif valid[0]==False and valid[1]==True:#If only piece 2 ca
    print('Piece 1 cannot be moved')
Select(roll, position, names, turn)
```

Proof of success

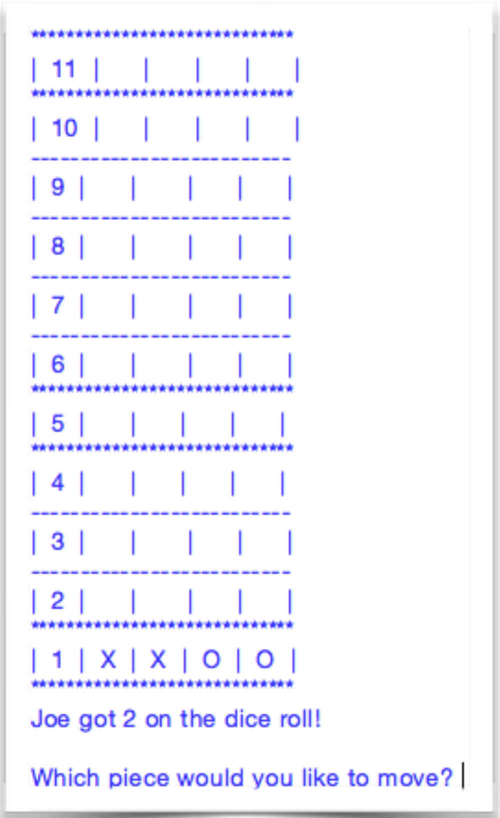
Here I will show that all areas of the solution work as intended, or else it wouldn't meet the user's needs. The best way to do this is to show screenshots of the program running, so below are the parts of the running code that I thought are important. To be completely sure that it works fine, run the code at the bottom of the document.



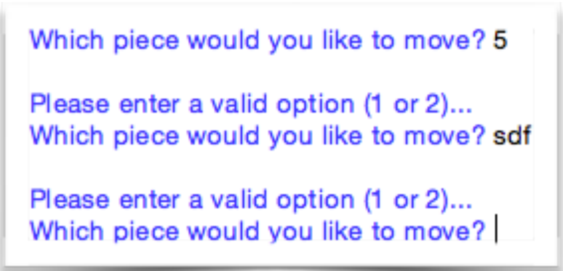
```
Welcome to AQADo!
What would you like to do?
        1. Enter player names
        2. Play game
        3. Quit
        4. View instructions
        5. Play with a five-sided die
        6. Play with a computer
1
Please enter the name of player 1: Joe
Please enter the name of player 2: Samantha
```



```
*****************************
| 11 |    |    |    |    |
*****************************
| 10 |    |    |    |    |
-----------------------------
| 9 |    |    |    |    |
-----------------------------
| 8 |    |    |    |    |
-----------------------------
| 7 |    |    |    |    |
-----------------------------
| 6 |    |    |    |    |
*****************************
| 5 |    |    |    |    |
*****************************
| 4 |    |    |    |    |
-----------------------------
| 3 |    |    |    |    |
-----------------------------
| 2 |    |    |    |    |
*****************************
| 1 | X | X | O | O |
*****************************
Joe got 4 on the dice roll!
Neither of your pieces can be moved...
```



```
*****************************
| 11 |    |    |    |    |
*****************************
| 10 |    |    |    |    |
-----------------------------
| 9 |    |    |    |    |
-----------------------------
| 8 |    |    |    |    |
-----------------------------
| 7 |    |    |    |    |
-----------------------------
| 6 |    |    |    |    |
*****************************
| 5 |    |    |    |    |
*****************************
| 4 |    |    |    |    |
-----------------------------
| 3 |    |    |    |    |
-----------------------------
| 2 |    |    |    |    |
*****************************
| 1 | X | X | O | O |
*****************************
Joe got 2 on the dice roll!

Which piece would you like to move?
```



```
Which piece would you like to move? 5

Please enter a valid option (1 or 2)...
Which piece would you like to move? sdf

Please enter a valid option (1 or 2)...
Which piece would you like to move?
```

Efficiency

The code that I have produced is efficient because it is beneficial, simple and is generally formatted nicely, but is still uses the least amount of code. To do this, I have used multiple programming techniques that I have chosen, planned, coded and set out to see how I could make it more efficient.

One way I did this is by making every line of the code unique. Instead of repeating lines of coded for the same purpose, I have used functions. This allows me to call lines of code, but use parameters to change the resources that it uses to carry out that code. For example, my function Play_game, where the board is printed. Instead of having different code for different situations, there is bit of code that does it all. This reduces the amount of code and makes it more efficient.

Next, I broke the code down into smaller chunks once finished and looked again. I looked for the little things that seem simple but are things that need to be done all the time. Even if they only took a line or two of code. I thinned it down because these small things add up over time.

As well as these, I tried not to make all of the code fit for any circumstance, so that no matter what the user enters and no matter what the situation, the code still functions on the same code and not separate code that runs depending on the situation. An example of this in my code is input validation. Each input is validated, but in every single case in my code, the input is run off the same code. The only different line is the error message that comes up if they enter an invalid input, however it is still run on the same code after they have been asked again, until they answer a valid input.

The best example of efficiency in my code is the fact that I stored the positions of the counters in a single array containing four numbers, each the position of a counter. Another way of doing this is using multiple dictionaries to store them, however this is unnecessarily inefficient, time consuming and confusing. Also, two-dimensional arrays could have been used which is a slightly better way than dictionaries, however is still unnecessary when all that is needed is something as simple as a single array. It means that a counter can be moved to any position in a single line instead of using multiple, needless lines of having to remove the counter, add the dice roll and add a new one, or in some extreme cases, having to clear the board and store their positions in separate variables before adding them back in again.

Another example of efficiency in my code is the use of Boolean values. These are the most efficient data types, full stop. They take up a single bit, compared to integers which take up 8 bits. Some people, for some reason, used an integer to store whose turn it is and whether a piece is valid or not. This not only meant that more space was taken up and that it is less efficient, but also that it was harder to swap around whose turn it is. In my code, swapping turns is done in a single line (turn=not turn), whereas when using integers it takes up 4 lines. This is because they need to decide if a number should be deducted from it or added on to it to swap turns.

Finally, the use of modules in my code makes it a lot more efficient. It saved me having to code some parts of the code, most helpfully when a random number is picked. Instead of having to write the code for getting a random number myself, I could use code that someone else had written and stated which numbers I want the numbers to be between. This is good because it makes it more reliable and it saves a lot of time.

In conclusion to the topic of efficiency, I have discussed the methods used to make my code more efficient: Using the least amount of code and still perform just as well, make every line of code unique, made the code work for any circumstance, removed every useless bit of code possible, which can add up if not removed and lead to inefficiency, the use of a single array to store the positions and using modules to save time, make it more reliable and make it more efficient.

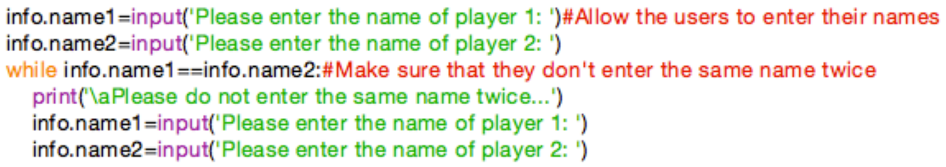
Data structures used

Variable name	Data type	Purpose
info.turn	Boolean	Decides whose turn it is (if it is True, it is player 1's turn)
info.CPU	Boolean	Shows whether a CPU is playing (True if one is)
count	Integer	Used when printing the board in the for loop. It shows the space that it is currently on (e.g. when printing space 7, count will equal 7)
counter	Integer	Used in a for loop inside the for loop printing the board. It makes sure that the space the outer for loop is on should have a counter in it. Otherwise, it is made an empty space
info.roll	Integer	Stores the maximum value a dice roll can get (5 when playing with a 5-sided die)
move	Integer	Stores the piece that the player wants to move
roll	Integer	Stores the result of the die roll
array	List	Contains the four counters. Each time the space is printed it is checked if a counter should be there. If it shouldn't, the counter is removed
position	List	Stores the positions of the counters
valid	List	Contains 2 boolean values, saying whether or not the player's pieces can be moved (True means it can be and vise versa)
choice	String	The user's choice of what they would like to do at the menu (e.g. play the game)
info.name1	String	Stores the names of player 1
info.name2	String	Stores the names of player 2
piece	Integer	Makes it more efficient by storing a commonly used phrase - the position of the piece being checked
loopcounter	Integer	Acts as a loop counter when checking where to move to when a 5 is rolled

TABLE 3.1 - THE DATA STRUCTURES USED IN THE PROGRAM

Robustness

The final code is completely robust in the sense that no errors are made because all inputs made by the user are validated, in some cases more than one way. For example, when the user enters what piece they want to enter, it not only checks that either a 1 or a 2 is entered, but also whether or not that piece can legally be moved. Another example of robustness in the code is shown below:



```
info.name1=input('Please enter the name of player 1: ')#Allow the users to enter their names
info.name2=input('Please enter the name of player 2: ')
while info.name1==info.name2:#Make sure that they don't enter the same name twice
    print('\aPlease do not enter the same name twice...')
    info.name1=input('Please enter the name of player 1: ')
    info.name2=input('Please enter the name of player 2: ')
```

In this example, the names are inputted by the users and they are validated by checking that they are not the same. If they are then they are asked again, however if they aren't, they are returned at the end of the function and used in the rest of the code. Although this doesn't prevent errors like when the piece that they want to move is entered, it does prevent confusion between the users.

Testing and Evaluation

Test plan

The below tables show the testing processes I went through in the first stages of my code. It shows the test number (which comes into use in debugging and I need to refer back to them), the test purpose (showing what I hope to accomplish by the end of the full test) the test data (the data that will be tested), the expected outcome (what should happen according to my predictions) and the actual outcome (what actually happens).

Version 1 (Task 1)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
1.1	Display menu and allow user to enter option	1 Valid input	Allow the user to enter their names	As expected
1.2		2 Valid input	Display the initial state of the game	As expected
1.3		3 Valid input	Quit the game	As expected
1.4		4 Valid input	Display instructions and return to the menu	As expected
1.5		0 or 5 Invalid input	Display error message and allow them to enter another option	As expected
1.6		! User enters erroneous value	Display error message and allow them to enter another option	As expected

TABLE 4.1 - TEST DATA FOR TASK 1

Version 2 (Task 2)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
2.1	Allow the user to enter their names	Player 1 and Player 2 Valid input	Input is accepted and user is returned to the menu	As expected
2.2		Bob and Bob Invalid input (both names are same)	Error message is displayed and user is allowed to enter names again	As expected

TABLE 4.2 - TEST DATA FOR TASK 2

Version 3 (Task 3)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
3.1	Display the initial state of the game	None	Initial state of game is displayed in an appropriate way	It does so, but very slowly

TABLE 4.3 - TEST DATA FOR TASK 3

Version 4 (Task 4)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
4.1	Generate a random number from 1-4 (five if extension is selected) and display the corresponding action	None	A random number is made and user is told the corresponding action	As expected

TABLE 4.4 - TEST DATA FOR TASK 4

Version 5 (Task 5)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
5.1	Check if any of their pieces can be moved	None of them can be moved	Tell them that none of them can be moved and swap turns	As expected
5.2		Piece 1 can be moved	User is told that piece 1 can be moved	User is told that piece 2 can be moved
5.3		Piece 2 can be moved	User is told that piece 2 can be moved	User is told that piece 1 can be moved
5.4		Both can be moved	User is told that both pieces can be moved	As expected

TABLE 4.5 - TEST DATA FOR TASK 5

Version 6 (Task 6)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
6.1	Allow the user to enter what piece they want to move	1 Valid input	Input is accepted	As expected
6.2		2 Valid input	Input is accepted	As expected
6.3		1 Piece cannot be moved	User is told that that piece cannot be moved	Piece is moved anyway
6.4		3 Invalid input	User is told that that piece cannot be moved	As expected
6.5		? Erroneous value	User is told that that piece cannot be moved	As expected

TABLE 4.6 - TEST DATA FOR TASK 6

Version 7 (Task 7)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
7.1	Make the legal move selected by the player	Player lands on an opponent's piece an not a safe space	Opponent's piece is moved back to start	As expected
7.2		Player lands on an opponent's piece on a safe space	Piece is moved by no piece is moved back	As expected

TABLE 4.7 - TEST DATA FOR TASK 7

Version 8 (Task 8)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
8.1	Check if a player has won and swap turns if not	A player has won	User is congratulated and the game ends	As expected
8.2		A player has not won	The user's turn ends and the next player's turn starts	As expected

TABLE 4.8 - TEST DATA FOR TASK 8

Version 9 (Task 9 - Extra)

Test number	Test purpose	Test data	Expected outcome	Actual outcome
9.1	Use a five sided die and if a player rolls a five, they can move on of their pieces to the next unoccupied space	User rolls a five and piece can be moved	Piece is moved to next unoccupied space	As expected
9.2		User rolls a five and piece cannot be moved	User is told that that piece cannot be moved and are asked again	As expected
9.3		User rolls a five and neither of their pieces can be moved	User is told that neither of their pieces can be moved and it becomes the other player's turn	As expected

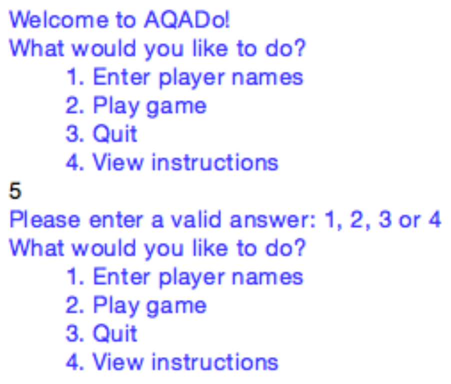
TABLE 4.9 - TEST DATA FOR TASK 9

Version 10 (Extra task)

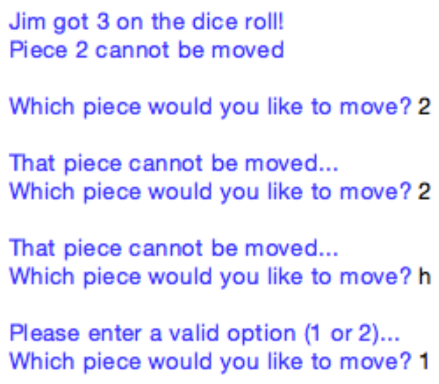
Test number	Test purpose	Test data	Expected outcome	Actual outcome
10.1	Allow the player to play with a computer that makes it's own decision based on it's and the player's pieces	Neither of the CPU's pieces can be moved	User is told that neither of the CPU's pieces could move and turn is swapped	As expected
10.2		One of the CPU's pieces lands on FINISH	User is told that the CPU moves this piece and it is moved	As expected
10.3		One of CPU's pieces lands on another player's piece	User is told that the CPU moves this piece and it is moved	As expected
10.4		One of CPU's pieces lands on a safe space	User is told that the CPU moves this piece and it is moved	As expected
10.5		Both of the CPU's pieces can be moved and none of the above apply	A random piece is chosen and the user is told that the CPU moves this piece and it is moved	As expected

TABLE 4.10 - TEST DATA FOR CPU EXTENSION

Testing screenshots



```
Welcome to AQADo!
What would you like to do?
        1. Enter player names
        2. Play game
        3. Quit
        4. View instructions
5
Please enter a valid answer: 1, 2, 3 or 4
What would you like to do?
        1. Enter player names
        2. Play game
        3. Quit
        4. View instructions
```



```
Jim got 3 on the dice roll!
Piece 2 cannot be moved

Which piece would you like to move? 2

That piece cannot be moved...
Which piece would you like to move? 2

That piece cannot be moved...
Which piece would you like to move? h

Please enter a valid option (1 or 2)...
Which piece would you like to move? 1
```

FIGURE 4.1 - EXAMPLES OF TESTING SCREENSHOTS

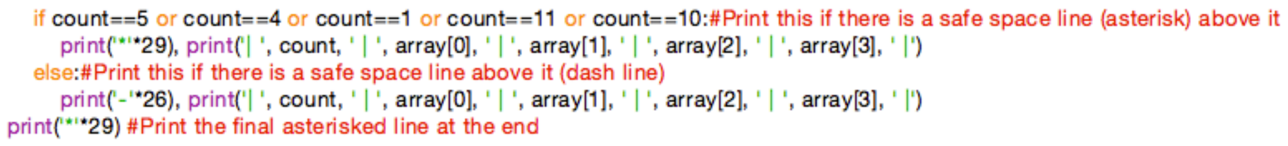
Beta testing

Beta testing is a test that possible user see what they think of it prior to its commercial release. This is the last stage of testing, which commences once alpha testing has finished. Beta testing is normally the stage where the product is sent outside of the programmer(s) and to other people for real-world exposure. This can from physically handing over the code on a USB, to putting it on beta testing websites for a specific amount of people to download for free and trial.

In my case, I handed the code over on a USB to some of my friends, who didn't know anything about the task or anything about programming. This meant that they could give me crucial feedback that helped to make my program more user-friendly and fix some of the problems. It also lead me to believe that my instructions are clear and understandable.

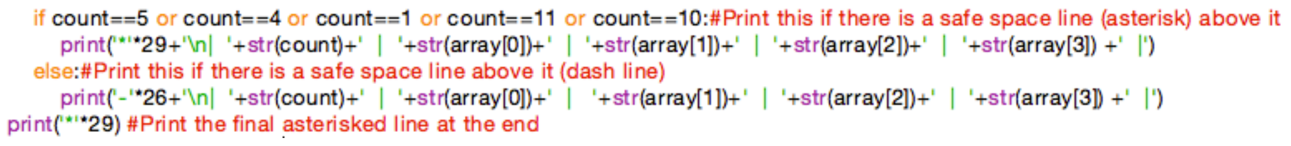
Debugging

Test 3.1



```
    if count==5 or count==4 or count==1 or count==11 or count==10:#Print this if there is a safe space line (asterisk) above it
        print('*'*29), print('| ', count, ' | ', array[0], ' | ', array[1], ' | ', array[2], ' | ', array[3], ' |')
    else:#Print this if there is a safe space line above it (dash line)
        print('-'*26), print('| ', count, ' | ', array[0], ' | ', array[1], ' | ', array[2], ' | ', array[3], ' |')
print('*'*29) #Print the final asterisked line at the end
```

Above is the code that I used before debugging. It was extremely slow and took a while to print the whole board. It took me a while to figure out that the problem was that I was using commas to separate the variables and the strings when printing. It meant that it was interpreting them as separate print statements, which is not efficient and is the reason why it was so slow. This meant that the user's needs were not met and I knew that it would not be very good code unless this was changed. To solve it, I used pluses instead of the commas and turn the variables into string so that an error did not occur. Below is the code used after debugging.



```
    if count==5 or count==4 or count==1 or count==11 or count==10:#Print this if there is a safe space line (asterisk) above it
        print('*'*29+'\n|  '+str(count)+'  |  '+str(array[0])+'  |  '+str(array[1])+'  |  '+str(array[2])+'  |  '+str(array[3]) +'  |')
    else:#Print this if there is a safe space line above it (dash line)
        print('-'*26+'\n|  '+str(count)+'  |  '+str(array[0])+'  |   '+str(array[1])+'  |  '+str(array[2])+'  |  '+str(array[3]) +'  |')
print('*'*29) #Print the final asterisked line at the end
```

Test 5.2 and 5.3

```
elif valid[0]==False and valid[1]==True:#If only piece 1 can, tell the user and swap goes
    print('Piece 2 cannot be moved')
elif valid[0]==True and valid[1]==False:#If only piece 2 can, tell the user and swap goes
    print('Piece 1 cannot be moved')
```

This error was a very simple mistake and did not take long to fix. The reason it said the pieces cannot be moved the wrong way around is because I got the boolean values the wrong way around. The 'Check' function returns False if it can't be moved and True if it can, so for the first 'if' statement above, it would print out that piece 2 can't be moved, but in fact piece 1 can't be moved (valid is an array containing boolean values for each of the pieces saying whether or not it can be moved). Below is the new code

```
elif valid[0]==True and valid[1]==False:#If only piece 1 can, tell the user and swap goes
    print('Piece 2 cannot be moved')
elif valid[0]==False and valid[1]==True:#If only piece 2 can, tell the user and swap goes
    print('Piece 1 cannot be moved')
```

Test 6.3

```
Player 2 got 4 on the dice roll!
Traceback (most recent call last):
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 150, in <module>
    Menu(names, position)
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 24, in Menu
    Play(names, turn, position)
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 66, in Play
    Dice(names, turn, position)
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 83, in Dice
    Play(names, turn, position)
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 66, in Play
    Dice(names, turn, position)
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 79, in Dice
    valid=[Check(position, 3, roll), Check(position, 4, roll)]
  File "/Users/Lloyd/Library/Mobile Documents/com~apple~CloudDocs/CS Cont
rolled assessment 2015/Controlled Assessment 2015.py", line 93, in Check
    if ((position[move]+roll)<1 or (position[move-1]+roll)>11):#Make sure that it wo
n't go under 1 or over 11
IndexError: list index out of range
```

The reason this happened was because I wanted to make it easier to test by starting the position array at 1,1,1,1 instead of the 0 that lists start at in Python. This means that each time it was used, 1 must be subtracted from it. In this case, I forgot to. Below is the new code

```
if (position[move-1]!=5 and position[move-1]!=11):#Do the following if not on a safe space
```

Overview

Task	Status of completion
Display a menu allowing the user to enter the player names, play the game and quit	Completed
Allow the user to enter their names if the user chooses to do so	Completed
Display the initial state of the game if the user chooses to play game	Completed
Generate a random number from 1-4 (5 if using extended dice) and tell them what can be done	Completed
Check if the user can move any of their pieces and if not, end their turn	Completed
Allow the user to choose what piece they want to move and do so if that move is legal	Completed
Make the move selected by the player, and if it lands on another player's piece, move that piece back to the start	Completed
Check if the player has won after each move. If so, congratulate them and if not, end their turn	Completed
Extend the program to so that it uses a five sided die. If a player rolls 5, move to the next unoccupied space	Completed

TABLE 7.1 - SEEING WHETHER OR NOT THE PROGRAM HAS COMPLETED THE TASKS

In detail

Overall, I think that the program has completed and exceeded the user's needs, everything is laid out nicely using functions and the program is well commented. Some elements of the program would normally be considered hard that I added in, like being universal by letting the user enter their own ten words, as can be seen below by part of the code:

The hardest bit of the program for me to code however was setting up all the rules and making sure that they are all called for in the code. Although this wasn't too hard, it took a long time and a lot of debugging.

Although there were some hold-backs, I think the program is simple, efficient and meets the user's needs.

Meeting the user's needs

At the start of this document I talked about meeting the user's needs (ref. Page 10). In it, I said that it is the most important part of the task was meeting the user's needs and to accomplish this I said that it needs to meet and exceed the tasks that we were given while still being simple, intuitive, easy to navigate and run smoothly and efficiently without any problems. Not only did I complete the tasks given, but I went further and added extra features to make it even more beneficial. There are discussed below.
As well as coding the features, I aimed to make it simple and easy to navigate. I attained this by making a menu from which the user can choose what to do next and I used the '\n' feature in python to make it look neat by splitting up the lines.

Taking the application further

As an extension to the program that was instructed, some features I have added in include:

- Using a five-sided die. If a player rolls a five then they can select one of their pieces and move that piece to the next unoccupied space after the space it is currently on
- Allowing the user to see instructions at the main menu
- Allowing the user to play single-player (with a computer), so that the computer makes it's own decisions

Some features that I could add in in the future are:

- Storing a database (maybe in a text file using dictionaries) of the different player's names and how many times they have won. This would allow for a leaderboard to be implemented and games would be more competitive

Final solution

```
import random #Make sure a random number can be accessed for the dice roll
import time  #Allow there to be a delay before doing certain things

#Task 1 - Display a main menu for the program
print('Welcome to AQADo!')#Greet the user
def Menu(info):#Show the menu to the user
    position=[11, 8, 9, 10]#Make a list containing the positions of the counters
    choice=input('''What would you like to do?
        1. Enter player names
        2. Play game
        3. Quit
        4. View instructions
        5. Play with a five-sided die
            6. Play with a computer\n''')#Show the menu to the user and allow them to enter their option
        while choice!='1' and choice!='2' and choice!='3' and choice!='4' and choice!='5' and choice!='6':#Validate the input
        print('\aPlease enter a valid answer: 1, 2, 3, 4, 5 or 6')
        choice=input('''What would you like to do?
        1. Enter player names
        2. Play game
        3. Quit
        4. View instructions
        5. Play with a five-sided die
        6. Play with a computer\n''')
    if choice=='1':#Ask the user to enter their names if they choose to do so
        (info)=Names()
    if choice=='2' or choice=='5' or choice=='6':#Allow the user to start the game
        if choice=='6':
            info.name2, info.CPU='CPU', True
        elif choice=='5':
            info.roll=5#Change the maximum roll it can do to 5
        Play(info, position)
    elif choice=='3':#Allow the user to quit the game
        exit()
    elif choice=='4':#Allow the user to view the instructions
        Instructions()
    Menu(info) #Return to the main menu

def Instructions():#Display the instructions
    print('''The players take it in turns to roll a four-sided die
The result of the die roll determines the move that you are allowed to make on the board
e.g. if you roll a 1, you can move one of your pieces one space nearer to FINISH
However, if you roll a 4, you move one of your pieces one space back towards START
The aim of the game is to move both of the player\'s pieces to the end of the 11-space board
If your piece lands on a space with an opponent’s piece on, then the opponent’s piece is moved back to START
BUT, a piece on a safe space cannot be sent back to START
You can\'t move your piece onto a space already that has your other piece on unless the space is a safe space
A piece cannot move backwards if it is on START or forwards if it is on FINISH
If a player can make a move then they must do so
A player wins the game when both of their pieces are on FINISH\n''')

class Info:
    def __init__(self, name1, name2, CPU, roll, turn):
        self.name1=name1 #Make a class variable for player 1's name
        self.name2=name2 #Make a class variable for player 2's name
        self.CPU=CPU #Make a class variable for whether or not the CPU is playing
        self.roll=roll #Make a class variable for the maximum roll that can be gotton (4 or 5)
        self.turn=turn #Make a class variable for whose turn it is

#Task 2 - Allow the players to enter their names
def Names():
    info.name1=input('Please enter the name of player 1: ')#Allow the users to enter their names
    info.name2=input('Please enter the name of player 2: ')
    while info.name1==info.name2:#Make sure that they don't enter the same name twice
        print('\aPlease do not enter the same name twice...')
        info.name1=input('Please enter the name of player 1: ')
        info.name2=input('Please enter the name of player 2: ')
    return(info)

#Task 3 - Display the board
def Play(info, position):
        for count in range(11, 0, -1):#Create a for loop with a negative stride so that it counts downwards (11 is at the top)
        array=['X', 'X', 'O', 'O'] #Make an array containing the 4 counters
        for counter in range(4):#Make a for loop that loops 4 times
            if position[counter]!=count:#If the space number isn't the position of the counter...
                    array[counter]='  ' #Change the counter of this space for that column to an empty space
         if count==5 or count==4 or count==1 or count==11 or count==10:#Print this if there is a safe space line (asterisk) above it
            print('*'*29)
        else:#Print this if there is a safe space line above it (dash line)
            print('-'*26)
        print('|  '+str(count)+'  |  '+array[0]+'  |  '+array[1]+'  |  '+array[2]+'  |  '+array[3] +'  |')
    print('*'*29) #Print the final asterisked line at the end
    Win(position, info)#Run the funtion that checks if the player has won
    Dice(info, position)

#Task 4 - Generate a random number to simulate the rolling of a four-sided die
def Dice(info, position):
        roll=random.randint(1, info.roll)#Generate a radom number using a variable containing the highest roll it can do
    if info.turn==True:#Display the dice roll and the player that the message is directed towards
        print(info.name1+' got '+str(roll)+' on the dice roll!')
    else:
        print(info.name2+' got '+str(roll)+' on the dice roll!')
    if roll==4:#Make the die roll -1 if it rolls a 4 so that it goes back one space
        roll=-1
    if info.CPU==True and info.turn==False:#Run the function that allows the computer to choose
if the player chose to do so
        CPU(position, roll, info)
    else:
        if info.turn==True:
            valid=[Check(position, 1, roll), Check(position, 2, roll)]#Check if player1's pieces can be
moved
        else:
            valid=[Check(position, 3, roll), Check(position, 4, roll)]
            if valid[0]==False and valid[1]==False:#If neither of the pieces can move, swap goes and
display board
            print('\aNeither of your pieces can be moved...'), time.sleep(1)
            info.turn = not info.turn#Swap turns
            Play(info, position)#Print the board again
        if valid[1]==False:#If only piece 1 can, tell the user and swap goes
            print('\aPiece 2 cannot be moved\n')
        elif valid[0]==False:#If only piece 2 can, tell the user and swap goes
            print('\aPiece 1 cannot be moved\n')
    Select(roll, position, info)

#Task 5 - Check if the result of the die roll will allow a player to move either of their pieces
def Check(position, move, roll):
      piece=position[move-1]#Make a variable for this commonly used position to make it more
efficient
    if roll!=-1 and piece==11:#Make sure that if they get 4 while on 11, they can't move it foward
        return False
    if roll==5:#Allow them to use a 5 sided dice
        loopcount=1#Make a loop counter variable
        while piece+loopcount<11:#Make sure they don't go past 11
                        if piece+loopcount==position[0] or piece+loopcount==position[1] or piece
+loopcount==position[2] or piece+loopcount==position[3]:#Check if that move lands on another
player's piece
                loopcount+=1
            else:
                return loopcount
        return False
    if piece+roll>=11:#Allow them to move a piece past 11
        return True
    if piece+roll!=5 and piece+roll!=1:#Do the following if not on a safe space
        if piece+roll<1:#Make sure that it won't go under 1 or over 11
            return False
        if move-1==0 and position[0]+roll==position[1]:#Make sure that it won't equal the other piece of that player
            return False
        elif move-1==1 and position[1]+roll==position[0]:#Make sure that it won't equal the other piece of that player
            return False
        elif move-1==2 and position[2]+roll==position[3]:#Make sure that it won't equal the other piece of that player
            return False
        elif move-1==3 and position[3]+roll==position[2]:#Make sure that it won't equal the other piece of that player
            return False#If the piece cannot be moved, return a False boolean value
    return True#If the piece cannot be moved, return a True boolean value

#Task 6 - Allow the player to select which one of their pieces they want to move
def Select(roll, position, info):
    valid, move=Valid(position, roll)
    while valid==False:#Validate the input by checking if that piece can be moved
        print('\a\nThat piece cannot be moved...')
        valid, move=Valid(position, roll)
    Make_move(move, roll, position, info, valid)
def Valid(position, roll):
    move=input('Which piece would you like to move? ') #Ask the user what piece they want to move
    while (move!='1' and move!='2'):#Validate the input
        print('\a\nPlease enter a valid option (1 or 2)...')
        move=input('Which piece would you like to move? ')
    move=int(move)#Change it to an integer so that the number can be changed easily
    if info.turn==False:#Add 2 to player 2's piece moves, as theirs is 2 to the right of player 1's
        move+=2
    valid=Check(position, move, roll)
    return valid, move

#Task 7 - Make the legal move selected by the player
def Make_move(move, roll, position, info, valid):
    if info.CPU==True and info.turn==False: #Do the following if it is the CPU's turn
        valid=True
            print('The CPU moves piece ' + str(move-2)+' to space '+str(position[move-1]+roll)), time.sleep(1) #Tell the user what the CPU got
    if position[move-1]+roll>11:
        position[move-1]=11 #If their move lands over 11, make it land on 11
    else:
        if isinstance(valid, bool)==False: #If valid is not a boolean value...
            position[move-1]=position[move-1]+valid #Add this valid value onto the piece's position
        else:
            position[move-1]=position[move-1]+roll#Change the position of the counter
            if position[move-1]!=5 and position[move-1]!=11:#Do the following when not on a safe space
            if (move-1==0 or move-1==1) and (position[move-1]==position[2]):#Check if any of
player 1's pieces are on player 2's piece 1
                position[2]=1#If so, take the opponent's piece back to the start
            elif (move-1==0 or move-1==1) and (position[move-1]==position[3]):#Check if any of
player 1's pieces are on player 2's piece 2
                position[3]=1#If so, take the opponent's piece back to the start
            elif (move-1==2 or move-1==3) and (position[move-1]==position[0]):#Check if any of
player 2's pieces are on player 1's piece 1
                position[0]=1#If so, take the opponent's piece back to the start
            elif (move-1==2 or move-1==3) and (position[move-1]==position[1]):#Check if any of
player 2's pieces are on player 1's piece 2
                position[1]=1#If so, take the opponent's piece back to the starts
    info.turn=not info.turn#Swap the turn to the other player
    Play(info, position)#If no one has won, the code prints the board again

#Task 8 - Extra - Check if a player has won the game after each move
def Win(position, info):
    if position[0]==11 and position[1]==11:#Check if player 1 has won
            print('\a\n\aCongratulations! '+ info.name1+' has won!\n'), time.sleep(1)#Display a
congratulations message if player 1 has won
        Menu(info) #Return to the main menu
    if position[2]==11 and position[3]==11:#Check if player 2 has won
            print('\a\n\aCongratulations! '+ info.name2+' has won!\n'), time.sleep(1)#Display a
congratulations message if player 2 has won
        Menu(info) #Return to the main menu

#Extra - Allow the user to play with a computer
def CPU(position, roll, info):
    valid=[Check(position, 3, roll), Check(position, 4, roll)]
    if valid[0]==False and valid[1]==False:
        info.turn=not info.turn
        print('\aNeither of the CPU\'s pieces can be moved'), time.sleep(1)
        Play(info, position)#Print the board again
        if position[2]+roll==position[0] or position[2]+roll==position[1] and valid[0]==True:#Check if
piece 1 lands on another player's piece
        Make_move(3, roll, position, info, valid)
      elif position[3]+roll==position[0] or position[3]+roll==position[1] and valid[1]==True:#Check if
piece 2 lands on another player's piece
        Make_move(4, roll, position, info, valid)
     elif position[2]+roll==5 or position[2]+roll==1 and valid[0]==True:#Check if piece 1 lands on a
safe space
        Make_move(3, roll, position, info, valid)
     elif position[3]+roll==5 or position[3]+roll==1 and valid[1]==True:#Check if piece 2 lands on a
safe space
        Make_move(4, roll, position, info, valid)
    elif valid[0]==False and valid[1]==True:
        move=4
    elif valid[1]==False and valid[0]==True:
        move=3
    if valid[0]==True and valid[1]==True:
        move=random.randint(3, 4)#Generate a random number
    Make_move(move, roll, position, info, valid)

info=Info('Player 1', 'Player 2', False, 4, True)
Menu(info)
```