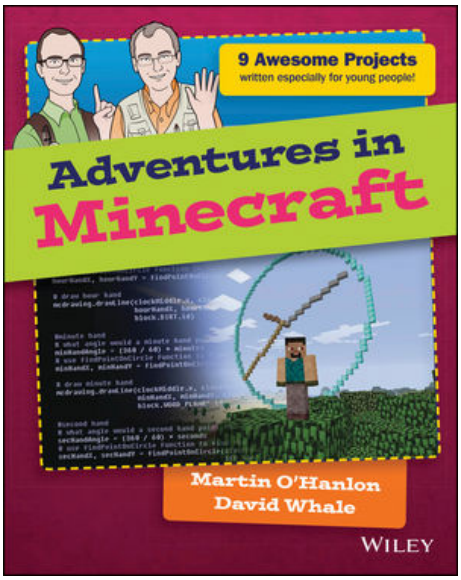
Adventures in Minecraft

Program Listings

Martin O'Hanlon and David Whale

November 2014



MARTIN O'HANLON has been designing and programming computer systems for all of his adult life. His passion for programming and helping others to learn led him to create the blog <Stuff about="code" />(www.stuffaboutcode.com) where he shares his experiences, skills and ideas. Martin regularly delivers presentations and workshops on programming Minecraft to coders, teachers and young people with the aim of inspiring them to try something new and making programming fun.

DAVID WHALE writes computer programs for devices you wouldnt imagine have computers inside them. He was bitten by the computer programming bug aged 11 when he was at school, and still thoroughly enjoys writing software and helping others to learn programming. He runs a software consultancy business in Essex, but also regularly volunteers for The Institution of Engineering and Technology (The IET) helping in schools, running weekend computing clubs, judging schools competitions, and running programming workshops for young people at community events all around the UK. You can follow his adventures on his blog at http://blog.whaleygeek.co.uk.

All programs are from the book: *Adventures in Minecraft* by Martin O'Hanlon and David Whale, Wiley, 2014

http://eu.wiley.com/WileyCDA/WileyTitle/productCd-111894691X.html

Contents

1 Adventure 1: Hello Minecraft World

1.1 Hello Minecraft World

```
import mcpi.minecraft as minecraft
mc = minecraft.Minecraft.create()
mc.postToChat("Hello Minecraft World")

# END
```

2 Adventure 2: Tracking Your Players as They Move

2.1 Where Am I

```
# This program works out where you are in the Minecraft world.
# It prints a message on the Minecraft chat, with your coordinates.

# The Minecraft API has to be imported before it can be used
import mcpi.minecraft as minecraft

# To communicate with a running Minecraft game,
# you need a connection to that game.
mc = minecraft.Minecraft.create()

# Ask the Minecraft game for the position of your player
pos = mc.player.getTilePos()

# Display the coordinates of the players position
mc.postToChat("x="+str(pos.x) + " y="+str(pos.y) +" z="+str(pos.z))

# END
```

2.2 Where Am I 2

```
# This program works out where you are in the Minecraft world.
# It uses a game loop to repeatedly display your position
# on the Minecraft chat.

# The Minecraft API has to be imported before it can be used
import mcpi.minecraft as minecraft

# The time module allows you to insert time delays
import time

# To communicate with a running Minecraft game,
# you need a connection to that game.
mc = minecraft.Minecraft.create()

# A game loop will make sure your game runs forever
while True:
    # delay, to prevent the Minecraft chat filling up too quickly
    time.sleep(1)

    # Ask the Minecraft game for the position of your player
    pos = mc.player.getTilePos()

    # Display the coordinates of the players position
    mc.postToChat("x="+str(pos.x) + " y="+str(pos.y) +" z="+str(pos.z))

# END
```

2.3 Welcome Home

```
# This program creates a magic doormat that says "welcome home"
# when you stand on it.

# Import necessary modules
import mcpi.minecraft as minecraft
import time

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# The main game loop
while True:
    # Short delay to prevent chat filling up too quickly
    time.sleep(1)

    # Get player position
    pos = mc.player.getTilePos()

    # Check whether your player is standing on the mat
    if pos.x == 10 and pos.z == 12:
        mc.postToChat("welcome home")

# END
```

2.4 Rent

```
# This program is a game using geo-fencing.
# You are challenged to collect objects from a field in the shortest
# time possible. All the time you are in the field you are charged
# rent. You have to collect/destroy blocks by spending the minimum
# amount of rent. If you stay in the field for too long, your player
# is catapulted up into the sky and out of the field, which makes the
# game harder to play and much more fun!

# Import necessary modules
import mcpi.minecraft as minecraft
import time

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Define constants for the 4 coordinates of the geo-fence
X1 = 10
Z1 = 10
X2 = 20
Z2 = 20

# Constants for the place to move through when catapulted!
HOME_X = X2 + 2
HOME_Y = 10 # up in the sky
HOME_Z = Z2 + 2

# A variable to keep a tally of how much rent you have been charged
rent = 0

# A variable to hold the number of seconds the player is in the field
inField = 0

# The main game loop ticks round once every second
while True:
    # Slow the program down a bit, this also helps with timing things
    time.sleep(1) # the loop runs once every second

    # Get the players position
    pos = mc.player.getTilePos()

    # If the player is in the field, charge him rent
    if pos.x>X1 and pos.x<X2 and pos.z>Z1 and pos.z<Z2:
        # Charge 1 pound rent every time round the loop (1sec per loop)
        rent = rent + 1
        # Tell player how much rent they owe
        mc.postToChat("You owe rent:" + str(rent))
        # Count number of seconds player is in the field (1sec per loop)
        inField = inField + 1
    else:
        # Not inside the field...
        inField = 0 # ...so reset the counter to zero

    # If player in field for more that 3 seconds...
    if inField>3:
        mc.postToChat("Too slow!")
        # ...catapult player outside of the field
        mc.player.setPos(HOME_X, HOME_Y, HOME_Z)

# END
```

3 Adventure 3: Building Anything Automatically

3.1 Block

```
# This program builds a block next to your player.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Get the player position
pos = mc.player.getTilePos()

# Create a block in front of the player
mc.setBlock(pos.x+3, pos.y, pos.z, block.STONE.id)

# END
```

3.2 Build

```
# This program builds a block at an absolute coordinate.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Build a block
mc.setBlock(1, 2, 3, block.DIAMOND_BLOCK.id)

# END
```

3.3 Dice

```
# This program builds the face of a dice near your player.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Get the player position
pos = mc.player.getTilePos()

# Create 6 blocks in front of the player, that look like a dice
mc.setBlock(pos.x+3, pos.y,   pos.z,   block.STONE.id)
mc.setBlock(pos.x+3, pos.y+2, pos.z,   block.STONE.id)
mc.setBlock(pos.x+3, pos.y+4, pos.z,   block.STONE.id)
mc.setBlock(pos.x+3, pos.y,   pos.z+4, block.STONE.id)
mc.setBlock(pos.x+3, pos.y+2, pos.z+4, block.STONE.id)
mc.setBlock(pos.x+3, pos.y+4, pos.z+4, block.STONE.id)

# END
```

3.4 Tower

```
# This program builds a tall tower inside Minecraft, using a for loop.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Get the player position
pos = mc.player.getTilePos()

# Build a tower 50 blocks high
for a in range(50):
    # Each block is at height: y+a
    mc.setBlock(pos.x+3, pos.y+a, pos.z, block.STONE.id)

# END
```

3.5 Clear Space

```
# This program clears some space near your player.
# This is so that it is then easier for you to build other things.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Get the player position
pos = mc.player.getTilePos()

# Ask the user how big a space to clear
size = int(raw_input("size of area to clear? "))

# Clear a space size by size*size*size, by setting it to AIR
mc.setBlocks(pos.x, pos.y, pos.z, pos.x+size, pos.y+size, pos.z+size,
————→————→————→ block.AIR.id)

# END
```

3.6 Build House

```
# This program builds a single house, with a doorway, windows,
# a roof, and a carpet.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to Minecraft
mc = minecraft.Minecraft.create()

# A constant, that sets the size of your house
SIZE = 20

# Get the players position
pos = mc.player.getTilePos()

# Decide where to start building the house, slightly away from player
x = pos.x + 2
y = pos.y
z = pos.z

# Calculate the midpoints of the front face of the house
midx = x+SIZE/2
midy = y+SIZE/2

# Build the outer shell of the house
mc.setBlocks(x, y, z, x+SIZE, y+SIZE, z+SIZE, block.COBBLESTONE.id)

# Carve the insides out with AIR
mc.setBlocks(x+1, y, z+1, x+SIZE-2, y+SIZE-1, z+SIZE-2, block.AIR.id)

# Carve out a space for the doorway
mc.setBlocks(midx-1, y, z, midx+1, y+3, z, block.AIR.id)

# Carve out the left hand window
mc.setBlocks(x+3, y+SIZE-3, z, midx-3, midy+3, z, block.GLASS.id)

# Carve out the right hand window
mc.setBlocks(midx+3, y+SIZE-3, z, x+SIZE-3, midy+3, z, block.GLASS.id)

# Add a wooden roof
mc.setBlocks(x, y+SIZE, z, x+SIZE, y+SIZE, z+SIZE, block.WOOD.id)

# Add a woolen carpet, the colour is 14, which is red.
mc.setBlocks(x+1, y-1, z+1, x+SIZE-2, y-1, z+SIZE-2, block.WOOL.id, 14)

# END
```

3.7 Build House 2

```
# This program builds a house with a doorway, windows, a roof
# and a carpet. It defines a house() function that you can easily
# reuse in other programs.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to Minecraft
mc = minecraft.Minecraft.create()

# A constant, that sets the size of your house
SIZE = 20

# define a new function, that builds a house
def house():
    # Calculate the midpoints of the front face of the house
    midx = x+SIZE/2
    midy = y+SIZE/2

    # Build the outer shell of the house
    mc.setBlocks(x, y, z, x+SIZE, y+SIZE, z+SIZE, block.COBBLESTONE.id)

    # Carve the insides out with AIR
    mc.setBlocks(x+1, y, z+1, x+SIZE-2, y+SIZE-1, z+SIZE-2, block.AIR.id)

    # Carve out a space for the doorway
    mc.setBlocks(midx-1, y, z, midx+1, y+3, z, block.AIR.id)

    # Carve out the left hand window
    mc.setBlocks(x+3, y+SIZE-3, z, midx-3, midy+3, z, block.GLASS.id)

    # Carve out the right hand window
    mc.setBlocks(midx+3, y+SIZE-3, z, x+SIZE-3, midy+3, z, block.GLASS.id)

    # Add a wooden roof
    mc.setBlocks(x, y+SIZE, z, x+SIZE, y+SIZE, z+SIZE, block.WOOD.id)

    # Add a woolen carpet, the colour is 14, which is red.
    mc.setBlocks(x+1, y-1, z+1, x+SIZE-2, y-1, z+SIZE-2, block.WOOL.id, 14)

# Get the players position
pos = mc.player.getPos()

# Decide where to start building the house, slightly away from player
x = pos.x + 2
y = pos.y
z = pos.z

# run the house() function that was defined by def house():
# this will build a house at x,y,z
house()

# END
```

3.8 Build Street

```
# This program builds a street of identical houses.
# It uses a for loop.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to Minecraft
mc = minecraft.Minecraft.create()

# A constant, that sets the size of your house
SIZE = 20

# define a new function, that builds a house
def house():
    # Calculate the midpoints of the front face of the house
    midx = x+SIZE/2
    midy = y+SIZE/2

    # Build the outer shell of the house
    mc.setBlocks(x, y, z, x+SIZE, y+SIZE, z+SIZE, block.COBBLESTONE.id)

    # Carve the insides out with AIR
    mc.setBlocks(x+1, y, z+1, x+SIZE-2, y+SIZE-1, z+SIZE-2, block.AIR.id)

    # Carve out a space for the doorway
    mc.setBlocks(midx-1, y, z, midx+1, y+3, z, block.AIR.id)

    # Carve out the left hand window
    mc.setBlocks(x+3, y+SIZE-3, z, midx-3, midy+3, z, block.GLASS.id)

    # Carve out the right hand window
    mc.setBlocks(midx+3, y+SIZE-3, z, x+SIZE-3, midy+3, z, block.GLASS.id)

    # Add a wooden roof
    mc.setBlocks(x, y+SIZE, z, x+SIZE, y+SIZE, z+SIZE, block.WOOD.id)

    # Add a woolen carpet, the colour is 14, which is red.
    mc.setBlocks(x+1, y-1, z+1, x+SIZE-2, y-1, z+SIZE-2, block.WOOL.id, 14)

# Get the players position
pos = mc.player.getTilePos()

# Decide where to start building the house, slightly away from player
x = pos.x + 2
y = pos.y
z = pos.z

# build 5 houses, for a whole street of houses
for h in range(5):
    # build one house
    house()
    # move x by the size of the house just built
    x = x + SIZE

# END
```

3.9 Build Street 2

```
# This program builds a street of houses with random carpets.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# A module to generate random numbers
import random

# Connect to Minecraft
mc = minecraft.Minecraft.create()

# A constant, that sets the size of your house
SIZE = 20

# define a new function, that builds a house
def house():
    # Calculate the midpoints of the front face of the house
    midx = x+SIZE/2
    midy = y+SIZE/2

    # Build the outer shell of the house
    mc.setBlocks(x, y, z, x+SIZE, y+SIZE, z+SIZE, block.COBBLESTONE.id)

    # Carve the insides out with AIR
    mc.setBlocks(x+1, y, z+1, x+SIZE-2, y+SIZE-1, z+SIZE-2, block.AIR.id)

    # Carve out a space for the doorway
    mc.setBlocks(midx-1, y, z, midx+1, y+3, z, block.AIR.id)

    # Carve out the left hand and right hand windows
    mc.setBlocks(x+3, y+SIZE-3, z, midx-3, midy+3, z, block.GLASS.id)
    mc.setBlocks(midx+3, y+SIZE-3, z, x+SIZE-3, midy+3, z, block.GLASS.id)

    # Add a wooden roof
    mc.setBlocks(x, y+SIZE, z, x+SIZE, y+SIZE, z+SIZE, block.WOOD.id)

    # Add a woolen carpet, the colour is 14, which is red.
    mc.setBlocks(x+1, y-1, z+1, x+SIZE-2, y-1, z+SIZE-2, block.WOOL.id, 14)

    # Add a random carpet
    c = random.randint(0, 15)
    mc.setBlocks(x+1, y+1, z+1, x+SIZE-1, y+1, z+SIZE-1, block.WOOL.id, c)


# Get the players position
pos = mc.player.getTilePos()

# Decide where to start building the house, slightly away from player
x = pos.x + 2
y = pos.y
z = pos.z

# build 5 houses, for a whole street of houses
for h in range(5):
    # build one house
    house()
    # move x by the size of the house just built
    x = x + SIZE

# END
```

4 Adventure 4: Interacting with Blocks

4.1 Safe Feet

```
# This program works out if your feet are safe or not.
# If you are in the air or in water, your feet are not safe.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import time

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Define a function to decide if your feet are safe or not
# This function will be reusable in other programs
def safeFeet():
    # Get the players position
    pos = mc.player.getTilePos()

    # Get the block directly below your player
    b = mc.getBlock(pos.x, pos.y-1, pos.z)

    # Is the player safe?
    if b == block.AIR.id or b == block.WATER_STATIONARY.id \
    or b == block.WATER_FLOWING.id:
        mc.postToChat("not safe")
    else:
        mc.postToChat("safe")

# Game loop
while True:
    # Run the game loop once every half a second
    time.sleep(0.5)
    # Check if your feet are safe
    safeFeet()

# END
```

4.2 Magic Bridge

```
# This program builds a bridge under your feet as you walk around.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Define a function to build a bridge if your feet are not safe
# This function will be reusable in other programs
def buildBridge():
    # Get the players position
    pos = mc.player.getTilePos()

    # Get the block directly below your player
    b = mc.getBlock(pos.x, pos.y-1, pos.z)
    # Is the player unsafe?
    if b == block.AIR.id or b == block.WATER_FLOWING.id \
    or b==block.WATER_STATIONARY.id:
        # If unsafe, build a glass block directly under the player
        mc.setBlock(pos.x, pos.y-1, pos.z, block.GLASS.id)

# Game loop
while True:
    # There is no delay here, this loop needs to run really fast
    # otherwise your bridge might not build quick enough and you will fall off
    buildBridge()

# END
```

4.3 Vanishing Bridge

```
# This program builds a bridge as you walk in the air or on water,
# and when your feet are safe again, the bridge magically disappears.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import time

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Create a bridge list that is empty.
# There are no blocks in your bridge at the moment,
# so it always starts empty
bridge = []

# Define a function to build a bridge if your feet are not safe
# This function will be reusable in other programs
def buildBridge():
    # Get the position of your player
    pos = mc.player.getTilePos()

    # Get the block directly below your player
    b = mc.getBlock(pos.x, pos.y-1, pos.z)

    # Is the player safe?
    if b == block.AIR.id or b == block.WATER_FLOWING.id \
    or b == block.WATER_STATIONARY.id:
        # Player is unsafe, build a glass block directly under the player
        mc.setBlock(pos.x, pos.y-1, pos.z, block.GLASS.id)
        # Use a second list to remember the three parts of this coordinate
        # but note pos.y-1 as you want to remember the block position below
        coordinate = [pos.x, pos.y-1, pos.z]

        # Add the new coordinate to the end of your bridge list
        bridge.append(coordinate)

    elif b != block.GLASS.id:
        # Player is safe and not on the bridge
        if len(bridge) > 0:
            # There is still some bridge remaining
            # Remove the last coordinate from the bridge list
            coordinate = bridge.pop()

            # Set the block at that coordinate to AIR, to delete the bridge
            mc.setBlock(coordinate[0], coordinate[1], coordinate[2],
                block.AIR.id)

            # Delay a short time so that the bridge vanishes slowly
            time.sleep(0.25)

# Game loop
while True:
    # Don't run the game loop too fast, you might crash your computer
    # But don't run the game loop too slow, you might fall off the bridge
    time.sleep(0.25)

    # Decide what to do with the bridge
    buildBridge()

# END
```

4.4 Block Hit

```
# This program senses that a block has been hit.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import time

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Work out the position of the player
diamond_pos = mc.player.getTilePos()

# Move the diamond slightly away from the player position
diamond_pos.x = diamond_pos.x + 1

# Build a diamond treasure block
mc.setBlock(diamond_pos.x, diamond_pos.y, diamond_pos.z,
----->block.DIAMOND_BLOCK.id)

# Define a function that checks if the diamond treasure has been hit
# You can reuse this function in other programs
def checkHit():
----->events = mc.events.pollBlockHits()

----->for e in events:
----->----># Get the coordinate that the hit happened at
----->----->pos = e.pos

----->----># If the diamond was hit
----->----->if pos.x == diamond_pos.x and pos.y == diamond_pos.y \
----->----->and pos.z == diamond_pos.z:
----->----->----->mc.postToChat("HIT")

# Game loop
while True:
----># Run the game loop once every second
----># Don't run it too fast or your computer might crash
----->time.sleep(1)

----># Check to see if the diamond treasure has been hit
----->checkHit()

# END
```

4.5 Sky Hunt

```
# This program is a treasure hunt game.
# It places treasure randomly near your player in the sky.
# You have to find this treasure in the fewest moves possible,
# because as you move you leave a trail of gold behind you.
# Each block of gold costs you points from your score.
# When you find the treasure, the gold trail melts away leaving
# holes in the ground for you to fall down, and another block
# of treasure is placed at a random location.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
import random # used generate random numbers

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Create a variable to keep track of your score
score = 0

# Create a constant that sets how far away to create the treasure
RANGE = 5

# There is no treasure yet, but create the global variables to use later
# Set these to None as they have no value yet, but the variables need
# to be created here to prevent an error later
treasure_x = None
treasure_y = None
treasure_z = None


# Define a function that places a new piece of treasure
# It will place the treasure at a random location
# but this location will be close to your player
def placeTreasure():
    # This function will change the values in these global variables
    # so you must list them as global here
    global treasure_x, treasure_y, treasure_z

    # Get the position of the player
    pos = mc.player.getTilePos()

    # Calculate a random position for the treasure
    # This is within 'RANGE' blocks from the player position
    # and also always slightly higher up in the sky
    # Note how the treasure_ variables are given a new value here
    # which means they no longer have 'None' stored in them.
    # This signals to the main loop that the treasure is now placed
    treasure_x = random.randint(pos.x,   pos.x+RANGE)
    treasure_y = random.randint(pos.y+2, pos.y+RANGE)
    treasure_z = random.randint(pos.z,   pos.z+RANGE)

    # Place the treasure in the world at the new position
    mc.setBlock(treasure_x, treasure_y, treasure_z, block.DIAMOND_BLOCK.id)


# Define a function that checks if the diamond treasure has been hit
# You wrote this function in an earlier program, it is similar
def checkHit():
    # This function will change the values in these global variables
```

```
    # so you must list them as gloal here
    global score
    global treasure_x

    # Get a list of hit events that have happened
    events = mc.events.pollBlockHits()

    # Process each event in turn
    for e in events:
        # Get the coordinate that the hit happened at
        pos = e.pos

        # If the diamond treasure was hit
        if pos.x == treasure_x and pos.y == treasure_y and pos.z == treasure_z:
            mc.postToChat("HIT!")
            # Increase the score
            score = score + 10

            # Delete the treasure so it disappears
            mc.setBlock(treasure_x, treasure_y, treasure_z, block.AIR.id)

            # Set treasure_x to None so that the game loop knows that there
            # is no longer any treasure, and it will place a new block of
            # treasure when it is ready
            treasure_x = None


# Create a timer variable that counts 10 loops of the game loop
TIMEOUT = 10
timer = TIMEOUT

# Define a function that displays the homing beacon on the Minecraft chat
def homingBeacon():
    # This function changes the value in the timer variable
    # so it must be listed as global here
    global timer

    # Treasure will be present in the sky if the treasure_x variable
    # has a value. If it has no value (None), then there is no treasure.
    # Check to see if a piece of treasure has been placed
    if treasure_x != None:
        # There is a piece of treasure in the world somewhere

        # You only want to update the homing beacon every 10 times
        # round the game loop, so count down from 10
        timer = timer - 1
        if timer == 0:
            # Once you have counted 10 game loops, 1 second has passed
            # re-set the timer so that it starts counting another 10 game loops
            timer = TIMEOUT

            # Get the position of the player
            pos = mc.player.getTilePos()

            # Calculate a rough number that estimates distance to the treasure
            diffx = abs(pos.x - treasure_x)
            diffy = abs(pos.y - treasure_y)
            diffz = abs(pos.z - treasure_z)
            diff = diffx + diffy + diffz

            # Display score and estimate to treasure location on chat
            mc.postToChat("score:" + str(score) + " treasure:" + str(diff))
```

```
# Just like in your vanishingBridge.py program, create an empty bridge list
# there are no blocks in the bridge yet
bridge = []

# Define a function to build a bridge of gold blocks
# This is similar to the one you wrote in vanishingBridge.py
def buildBridge():
    # This function will change the value in the score variable
    # so it has to be listed as global here
    global score

    # Get the position of the player
    pos = mc.player.getTilePos()

    # Get the id of the block directly below the player
    b = mc.getBlock(pos.x, pos.y-1, pos.z)

    # Has the treasure been found and deleted?
    if treasure_x == None:
        # Are there still blocks remaining in the gold bridge?
        if len(bridge) > 0:
            # Remove the last coordinate of the bridge from the bridge list
            coordinate = bridge.pop()

            # Delete that block of the bridge by setting it to AIR
            mc.setBlock(coordinate[0], coordinate[1], coordinate[2],
                block.AIR.id)

            # Display a helpful countdown message on the Minecraft chat
            mc.postToChat("bridge:" + str(len(bridge)))

            # Delay for a while, so that the bridge disappears slowly
            time.sleep(0.25)

    elif b != block.GOLD_BLOCK.id:
        # There is treasure, and you are not already standing on gold
        # build another gold block below your player
        mc.setBlock(pos.x, pos.y-1, pos.z, block.GOLD_BLOCK.id)

        # Remember the coordinate of this new gold block...
        coordinate = [pos.x, pos.y-1, pos.z]
        # ...by adding it to the end of the bridge list
        bridge.append(coordinate)

        # You loose one point for each block of the bridge that is built
        score = score - 1

# Main game loop
while True:
    # Run the loop 10 times every second
    # It needs to run quite fast so that the program responds well
    time.sleep(0.1)

    # If there is no treasure placed yet,
    # and if a bridge has not yet been built
    if treasure_x == None and len(bridge) == 0:
        # Place a new treasure block randomly
        placeTreasure()

    # Perform any bridge-building activities
    buildBridge()
```

```
————→# Perform any homing-beacon activitites
————→homingBeacon()

————→# Perform any hit-checking activities
————→checkHit()

# END
```

5 Adventure 5: Interacting with Electronic Circuits

5.1 Test LED

```
# This program tests that you can flash an LED connected to your computer.
# This works on Raspberry Pi, PC and Mac.

# Import necessary modules
import time
import RPi.GPIO as GPIO     # use this for Raspberry Pi
#import anyio.GPIO as GPIO  # use this for Arduino on PC/Mac

# This is the GPIO number that the LED is attached to
# It's not the same as the pin number on the connector though!
LED = 15

# Use Broadcom pin numbering (GPIO numbers, not connector pin numbers)
GPIO.setmode(GPIO.BCM)

# Configure the LED GPIO to be an output, so that you can
# change the voltage on it and thus turn the LED on and off
GPIO.setup(LED, GPIO.OUT)

# Define a function that flashes the LED once
# The 't' variable holds the time to flash for
def flash(t):
    # True puts 3.3 Volts on the pin connected to the LED, the LED goes on
    GPIO.output(LED, True)
    time.sleep(t)

    # False puts 0 Volts on the pin connected to the LED, the LED goes off
    GPIO.output(LED, False)
    time.sleep(t)

# Game loop
try: # Try to run this code, jump to "finally" if it fails
    while True:
        # Flash your LED 0.5 seconds ON, 0.5 seconds OFF
        flash(0.5)

finally: # always gets here if your code fails (e.g. you CTRL-C the program)
    # Always use this function, to leave the GPIO hardware in a safe state
    GPIO.cleanup()

# END
```

5.2 Welcome LED

```
# This program creates a welcome home mat that flashes an LED
# when you stand on the mat.
# This works on Raspberry Pi, PC and Mac.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
import RPi.GPIO as GPIO     # use this for Raspberry Pi
#import anyio.GPIO as GPIO  # use this for Arduino on PC/Mac

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# This is the GPIO number that the LED is attached to
# It's not the same as the pin number on the connector though!
LED = 15

# Set some constants that will be the position of your welcome home mat
# Just like in Adventure 2, you need to change these numbers to somewhere
# sensible inside your Minecraft world
HOME_X = 0
HOME_Y = 0
HOME_Z = 0

# build a door mat at the location of your home.
# Colour number 15 for WOOL is the colour black
mc.setBlock(HOME_X, HOME_Y, HOME_Z, block.WOOL.id, 15)

# Use Broadcom pin numbering (GPIO numbers, not connector pin numbers)
GPIO.setmode(GPIO.BCM)

# Configure the LED GPIO to be an output, so that you can
# change the voltage on it and thus turn the LED on and off
GPIO.setup(LED, GPIO.OUT)

# Define a function that flashes the LED once, 't' holds the flash time
def flash(t):
    # True puts 3.3 Volts on the pin connected to the LED, the LED goes on
    GPIO.output(LED, True)
    time.sleep(t)

    # False puts 0 Volts on the pin connected to the LED, the LED goes off
    GPIO.output(LED, False)
    time.sleep(t)

# Game loop
try: # Try to run this code, jump to "finally" if it fails
    while True:
        # Get the position of your player
        pos = mc.player.getTilePos()

        # Geo-fence the door mat, work out if your player is on the mat
        if pos.x == HOME_X and pos.z == HOME_Z:
            # They are on the mat,flash your LED 0.5 seconds ON,0.5 seconds OFF
            flash(0.5)
finally: # always gets here if your code fails (e.g. you CTRL-C the program)
    # Always use this function, to leave the GPIO hardware in a safe state
    GPIO.cleanup()

# END
```

5.3 Test Display

```
# This program shows how to write patterns to a 7-segment LED display.
# This works on Raspberry Pi, PC and Mac.

# Import necessary modules
import RPi.GPIO as GPIO  # use this for Raspberry Pi

# Define a list of GPIO numbers. The order here is important
LED_PINS = [10,22,25,8,7,9,11,15]  # Use this for Raspberry PI

#import anyio.GPIO as GPIO  # Use this for Arduino on PC/Mac
# Define a list of GPIO numbers. The order here is important
#LED_PINS = [7,6,14,16,10,8,9,15]  # Use this for Arduino on PC/Mac

# Use Broadcom pin numbering (GPIO numbers, not connector pin numbers)
GPIO.setmode(GPIO.BCM)

# The suggested display is common-anode
# This means all the positive sides of the LEDs are connected together
# and thus you set the output to False (0V) to turn the LED on.
# A common-anode display has its common pin connected to 3.3V

# If you set this to True, it drives a common-cathode display
# This means all the negative sides of the LEDs are connected together
# and thus you set the output to True (3.3V) to turn the LED on.
# A common-cathode display has its common pin connected to 0V

ON = False # False=common-anode. Set to True for a common-cathode display

# loop through all 8 GPIO's setting them to outputs.
for g in LED_PINS:
    GPIO.setup(g, GPIO.OUT)

# Make a list of on/off values, one value for each LED in the display.
# The True/False values in this list represent the segments of the display
# in the following order: [A, B, C, D, E, F, G, Dp]
pattern = [True, True, True, True, True, True, True, False] # ABCDEFG No Dp

# Loop through all 8 values in your pattern list
for g in range(8):
    # if the pattern for index 'g' is True, the LED needs to be on
    if pattern[g]:
        # Turn the LED on
        GPIO.output(LED_PINS[g], ON)
    else:
        # Turn the LED off
        GPIO.output(LED_PINS[g], not ON)

# Wait for the ENTER key to be pressed on the keyboard
# This is needed because the next GPIO.cleanup() will turn off all LEDs!
raw_input("finished?")

# Put all of the GPIO hardware in a safe state. This turns off all LEDs
GPIO.cleanup()

# END
```

5.4 Test Display 2

```
# This program shows how to write patterns to a 7-Segment LED display.
# It uses a display module written by David, that makes the use
# of different patterns for different letters and numbers and symbols
# very easy to generate.
# This works on Raspberry Pi, PC and Mac.

# Import necessary modules

# The anyio.seg7 module is written by David, and provided for free
# It makes it really easy to display any pattern on a 7-Segment display
# Get it from here:
#   https://github.com/whaleygeek/anyio/blob/master/anyio/seg7.py

import anyio.seg7 as display
import time

import RPi.GPIO as GPIO  # use this for Raspberry Pi
# Define a list of GPIO numbers. The order here is important
LED_PINS = [10,22,25,8,7,9,11,15]  # Use this for Raspberry Pi

#import anyio.GPIO as GPIO  # Use this for Arduino on PC/Mac
# Define a list of GPIO numbers. The order here is important
#LED_PINS = [7,6,14,16,10,8,9,15]  # Use thsi for Arduino on PC/Mac

# Use Broadcom pin numbering (GPIO numbers, not connector pin numbers)
GPIO.setmode(GPIO.BCM)

# The suggested display is common-anode
# This means all the positive sides of the LEDs are connected together
# and thus you set the output to False (0V) to turn the LED on.
# A common-anode display has its common pin connected to 3.3V

# If you set this to True, it drives a common-cathode display
# This means all the negative sides of the LEDs are connected together
# and thus you set the output to True (3.3V) to turn the LED on.
# A common-cathode display has its common pin connected to 0V

ON = False # False=common-anode. Set to True for a common-cathode display

# Setup the display module by giving it access to your GPIOs,
# giving it a list of the pin numbers to use for each segment of the display,
# and telling it whether you are using common-anode or common-cathode
display.setup(GPIO, LED_PINS, ON)

# Game loop
try: # if this code fails, jumps to "finally"
----->while True:
----->----->
----->----->
----->----->
----->----->
----->----->
----->----->
----->----->
finally: # Gets here if the above program fails (e.g. you press CTRL-C)
----->
----->

# END
```

5.5 Detonator

```
# This program creates a big red detonator button.
# When the button is pressed, there is a countdown 4,3,2,1,0
# on the 7-Segment LED display, then the bomb goes off and a huge
# crater appears in Minecraft. You can use this to clear lots of
# space in the world to make it easier to build things.
# This works on Raspberry Pi, PC and Mac.

# Import necessary modules

import mcpi.minecraft as minecraft
import mcpi.block as block
import time

# The anyio.seg7 module is written by David, and provided for free
# It makes it really easy to display any pattern on a 7-Segment display
# Get it from here:
#   https://github.com/whaleygeek/anyio/blob/master/anyio/seg7.py
import anyio.seg7 as display

import RPi.GPIO as GPIO  # use this for Raspberry Pi
# Define a list of GPIO numbers. The order here is important
LED_PINS = [10,22,25,8,7,9,11,15]  # Use this for Raspberry Pi
BUTTON = 4 # Button connected to GPIO4

#import anyio.GPIO as GPIO  # Use this for Arduino on PC/Mac
# Define a list of GPIO numbers. The order here is important
#LED_PINS = [7,6,14,16,10,8,9,15]  # Use thsi for Arduino on PC/Mac
#BUTTON = 4 # Button connected to GPIO4

# Use Broadcom pin numbering (GPIO numbers, not connector pin numbers)
GPIO.setmode(GPIO.BCM)

# Setup the button GPIO to be an input (so you can read it's voltage)
GPIO.setup(BUTTON, GPIO.IN)

# The suggested display is common-anode
# This means all the positive sides of the LEDs are connected together
# and thus you set the output to False (0V) to turn the LED on.
# A common-anode display has its common pin connected to 3.3V

# If you set this to True, it drives a common-cathode display
# This means all the negative sides of the LEDs are connected together
# and thus you set the output to True (3.3V) to turn the LED on.
# A common-cathode display has its common pin connected to 0V

ON = False # False=common-anode. Set to True for a common-cathode display

# Setup the display module by giving it access to your GPIOs,
# giving it a list of the pin numbers to use for each segment of the display,
# and telling it whether you are using common-anode or common-cathode
display.setup(GPIO, LED_PINS, ON)

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Define a function that places and detonates a bomb
# You can reuse this function in other programs
# x,y,z are variables that indicate the coordinates for the bomb
def bomb(x, y, z):
    # Place a block of TNT
    mc.setBlock(x+1, y, z+1, block.TNT.id)
```

```
    # Count 6 numbers (from 0 to 5)
    for t in range(6):
        # Display the number sequence 5,4,3,2,1,0 on the 7-Seg display
        display.write(str(5-t))
        # Wait so that you can see the number
        time.sleep(1)

    mc.postToChat("BANG!")
    # Just like clearSpace.py, clear a "crater" around the player
    # The crater is centered around your player
    mc.setBlocks(x-10, y-5, z-10, x+10, y+10, z+10, block.AIR.id)

# game loop
try: # if this code fails, jumps to "finally"
    while True:
        # Read the switch 10 times per second (0.1 of a second delay)
        time.sleep(0.1)
        if GPIO.input(BUTTON) == False:
            # If the input reads false (0V) the button is pressed
            # Get the position of the player
            pos = mc.player.getTilePos()

            # Drop a bomb where the player is standing
            bomb(pos.x, pos.y, pos.z)

finally: # Gets here if the above program fails (e.g. you pressed CTRL-C)
    # Put all of the GPIO hardware in a safe state. This turns off the LEDs
    GPIO.cleanup()

# END
```

6 Adventure 6: Using Data Files

6.1 Tip Chat

```
# This program posts random hints and tips to the Minecraft chat.
# It reads those hits and tips from a separate text file.

# Import necessary modules
import mcpi.minecraft as minecraft
import time
import random

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Set a constant for the filename, so that you can easily change it
# to read a different file later
FILENAME = "tips.txt"

# Open the file in read mode
f = open(FILENAME, "r")

# Read all lines from the file into a list called 'tips'
tips = f.readlines()

# Close the file when you have finished with it.
f.close()

# Game loop
while True:
    # Wait a random time between 3 and 7 seconds
    time.sleep(random.randint(3, 7))

    # Choose a random message from the 'tips' list
    msg = random.choice(tips)

    # Give the tip to the player. strip() strips out unwanted newlines
    mc.postToChat(msg.strip())

# END
```

6.2 CSV Build

```
# This program builds mazes in the Minecraft world.
# The maze data is stored in a CSV file.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Define some constants for the different blocks you will use
# to build your maze out of. This allows you to experiment with
# different blocks and create interesting mazes
GAP   = block.AIR.id
WALL  = block.GOLD_BLOCK.id
FLOOR = block.GRASS.id

# This is the name of the file to read maze data from.
# It is a constant, so that you can change it easily to read from
# other data files in the future
FILENAME = "maze1.csv"

# Open the file containing your maze data
f = open(FILENAME, "r")

# Get the player position, and work out coordinates of the bottom corner
# The x and z are moved slightly to make sure that the maze doesn't get
# built on top of your players head
pos = mc.player.getTilePos()
ORIGIN_X = pos.x+1
ORIGIN_Y = pos.y
ORIGIN_Z = pos.z+1

# The z coordinate will vary at the end of each line of data
# So start it off at the origin of the maze
z = ORIGIN_Z

# Loop around every line of the file
for line in f.readlines():
    # Split the line into parts, wherever there is a comma
    data = line.split(",")

    # Restart the x coordinate every time round the "for line" loop
    x = ORIGIN_X

    # This is a nested loop (a loop inside another loop)
    # It loops through every item in the "data" list
    # It loops through cells, just like cells in a spreadsheet
    for cell in data:
        # Check if this cell of data is a gap or a wall
        if cell == "0": # f.readlines read it in as a string, so use quotes
            # choose the gap block id
            b = GAP
        else:
            # otherwise choose the wall block id
            b = WALL

        # build two layers of the required block id (held in the b variable)
        mc.setBlock(x, ORIGIN_Y, z, b)
        mc.setBlock(x, ORIGIN_Y+1, z, b)
```

```
————→————→# build one layer of floor underneat it
————→————→mc.setBlock(x, ORIGIN_Y-1, z, FLOOR)

————→————→# move to the next x coordinate at the end of this "for cell" loop
————→————→x = x + 1

————→# move to the next z coordinate at the end of this "for line" loop
————→z = z + 1

# END
```

6.3 Scan 3D

```
# This program scans a region of the Minecraft world and stores
# a representation of the object in a CSV file.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Create some constants for the name of the file you want to scan to
# and the size of the object you want to scan
FILENAME = "tree.csv"
SIZEX = 5
SIZEY = 5
SIZEZ = 5

# Define a function that will scan at an x,y,z position to a file
# filename - name of the file to create
# originx,originy,originz - x,y,z coordinates of start of object to scan
def scan3D(filename, originx, originy, originz):
    # Open the file in write mode
    f = open(filename, "w")

    # Write the metadata line that has 3 sizes in it seprated by commas
    # the \n is a newline character (it presses ENTER for you at end)
    f.write(str(SIZEX) + "," + str(SIZEY) + "," + str(SIZEZ) + "\n")

    # Loop through all the y coordinates of the space to be scanned
    for y in range(SIZEY):
        # Write a blank line at the start of each layer of data
        f.write("\n")

        # Loop through all x coordinates of the space to be scanned
        for x in range(SIZEX):
            # Build up a "line" variable, it starts off empty
            line = ""

            # Loop through all z coordinates of the space to be scanned
            for z in range(SIZEZ):
                # get the block id at this x,y,z position
                blockid = mc.getBlock(originx+x, originy+y, originz+z)

                # If the "line" variable is empty, this is the first block
                # on this line, so it doesn't need a comma
                if line != "":
                    # 'line' is not empty, so you must need a comma, so add it
                    line = line + ","

                # Add the block id of this block to the end of 'line' variable
                line = line + str(blockid)

            # Note the indention. This is part of the 'for x' loop
            # write the whole line and press 'ENTER' at the end of it
            f.write(line + "\n")

    # Note the indentation. This is not part of any loop
    # Close the file when finished, so other programs can use it
    f.close()
```

```
# Get the position of the player
pos = mc.player.getTilePos()

# Scan an area into the file FILENAME
# It spans SIZEX,SIZEY,SIZEZ centered around the player,
# but note that it does not "dig down" under the player,
# which is why pos.y is used here
scan3D(FILENAME, pos.x-(SIZEX/2), pos.y, pos.z-(SIZEZ/2))

# END
```

6.4 Print 3D

```
# This program reads from a CSV file and builds blocks in the Minecraft world.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Create a constant for the name of your data file
# This makes it easier to use a different file in the future
FILENAME = "object1.csv"

# Define a function that will print the contents of any file
# at any x,y,z coordinate in the Minecraft world
# filename - is the name of the file to open
# x,y,z - these are the coordinates of the origin (bottom corner)
def print3D(filename, originx, originy, originz):
    # Open the data file in read mode
    f = open(filename, "r")

    # Read all lines from the file into a 'lines' list variable
    lines = f.readlines()

    # The first line in the file is the metadata, it holds the 3 sizes
    # Split this line into parts, and set 3 variables, one for each size
    coords = lines[0].split(",")
    sizex = int(coords[0])
    sizey = int(coords[1])
    sizez = int(coords[2])

    # Use the 'lineidx' variable to keep track of the line number
    # in the loop below. Each time arount the loop, this will set to
    # the next line number, so that your loop processes one line at a time
    lineidx = 1

    # This first for loop reads through each vertical layer of the file
    for y in range(sizey):
        mc.postToChat(str(y)) # just so you can see what it is doing

        # Advance to the next line index
        lineidx = lineidx + 1

        # This inner loop reads through each east/west position in a line
        for x in range(sizex):
            # Get the whole line at this 'lineidx'
            line = lines[lineidx]

            # Advance to the next line index
            lineidx = lineidx + 1

            # Split this line into separate cells, at each comma
            data = line.split(",")

            # This (third) inner loop reads through each north/south position
            for z in range(sizez):
                # Get the blockid from the line at this z position
                # Convert it to a number by using int()
                blockid = int(data[z])
                # Build the block at the correct x,y,z position
                mc.setBlock(originx+x, originy+y, originz+z, blockid)
```

```
# Get the player position
pos = mc.player.getTilePos()

# "3D-print" the contents of the FILENAME file just to the side
print3D(FILENAME, pos.x+1, pos.y, pos.z+1)

# END
```

6.5 Duplicator

```
# This program builds a huge 3D duplicating machine.
# The machine can be used to scan and to print 3D objects all around
# the Minecraft world. Objects can be saved to a CSV file, loaded
# back into the duplicator room, and duplicated all over the world.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import glob # needed to handle filename wildcards
import time
import random

# Connect to the Minecraft game
mc = minecraft.Minecraft.create()

# Set some constants for the size of the duplicator room
SIZEX = 10
SIZEY = 10
SIZEZ = 10

# Set a default location for the duplicator room
roomx = 1
roomy = 1
roomz = 1


# Define a function that builds the duplicator room
# It will build it at x,y,z
def buildRoom(x, y, z):
    # These 3 variables will have their value changed by this function
    # so they have to be listed as global here
    global roomx, roomy, roomz

    # Remember where the room has been built
    # so that it can be demolished later on
    roomx = x
    roomy = y
    roomz = z

    # Build the duplicator room out of glass
    mc.setBlocks(roomx, roomy, roomz,
        roomx+SIZEX+2, roomy+SIZEY+2, roomz+SIZEZ+2, block.GLASS.id)

    # Carve the insides and front face out with air
    mc.setBlocks(roomx+1, roomy+1, roomz,
        roomx+SIZEX+1, roomy+SIZEY+1, roomz+SIZEZ+1, block.AIR.id)


# Define a function that demolishes the room
def demolishRoom():
    # Set the whole area, including walls and contents, to air
    # Note how the roomx, roomy and roomz are used to remember
    # where the room was built in the first place
    mc.setBlocks(roomx, roomy, roomz,
        roomx+SIZEX+2, roomy+SIZEY+2, roomz+SIZEZ+2, block.AIR.id)


# Define a function to clean just the contents of the room
def cleanRoom():
    # Set the whole inside area to AIR
    mc.setBlocks(roomx+1, roomy+1, roomz+1,
```

```
————→————→roomx+SIZEX+1, roomy+SIZEY+1, roomz+SIZEZ+1, block.AIR.id)


# Define a function that lists all CSV files
def listFiles():
————→print("\nFILES:")

————→# Get a list of files matching the wildcard *.csv into 'files'
————→files = glob.glob("*.csv")

————→# Loop through each filename in the list
————→for filename in files:
————→————→print(filename)

————→# Print a blank line at the end
————→# This is so that it separates the file list from the menu display
————→print("\n")


# Define a function that will scan at an x,y,z position to a file
# filename - name of the file to create
# originx,originy,originz - x,y,z coordinates of start of object to scan
def scan3D(filename, originx, originy, originz):
————→# Open the file in write mode
————→f = open(filename, "w")

————→# Write the metadata line that has 3 sizes in it seprated by commas
————→# the \n is a newline character (it presses ENTER for you at end)
————→f.write(str(SIZEX) + "," + str(SIZEY) + "," + str(SIZEZ) + "\n")

————→# Loop through all the y coordinates of the space to be scanned
————→for y in range(SIZEY):
————→————→mc.postToChat("scan:" + str(y)) # so you can see it working

————→————→# Write a blank line at the start of each layer of data
————→————→f.write("\n")

————→————→# Loop through all x coordinates of the space to be scanned
————→————→for x in range(SIZEX):
————→————→————→# Build up a "line" variable, it starts off empty
————→————→————→line = ""

————→————→————→# Loop through all z coordinates of the space to be scanned
————→————→————→for z in range(SIZEZ):
————→————→————→————→# get the block id at this x,y,z position
————→————→————→————→blockid = mc.getBlock(originx+x, originy+y, originz+z)

————→————→————→————→# If the "line" variable is empty, this is the first block
————→————→————→————→# on this line, so it doesn't need a comma
————→————→————→————→if line != "":
————→————→————→————→————→# 'line' is not empty, you must need a comma, so add it
————→————→————→————→————→line = line + ","

————→————→————→————→# Add the block id of this block to the end of 'line' variable
————→————→————→————→line = line + str(blockid)

————→————→————→# Note the indention. This is part of the 'for x' loop
————→————→————→# write the whole line and press 'ENTER' at the end of it
————→————→————→f.write(line + "\n")

————→# Note the indentation. This is not part of any loop
————→# Close the file when finished, so other programs can use it
————→f.close()
```

```
# Define a function that will print the contents of any file
# at any x,y,z coordinate in the Minecraft world
# filename - is the name of the file to open
# x,y,z - these are the coordinates of the origin (bottom corner)
def print3D(filename, originx, originy, originz):
    # Open the data file in read mode
    f = open(filename, "r")

    # Read all lines from the file into a 'lines' list variable
    lines = f.readlines()

    # The first line in the file is the metadata, it holds the 3 sizes
    # Split this line into parts, and set 3 variables, one for each size
    coords = lines[0].split(",")
    sizex = int(coords[0])
    sizey = int(coords[1])
    sizez = int(coords[2])

    # Use the 'lineidx' variable to keep track of the line number
    # in the loop below. Each time arount the loop, this will set to
    # the next line number, so that your loop processes one line at a time
    lineidx = 1

    # This first for loop scans through each vertical layer of the file
    for y in range(sizey):
        mc.postToChat("print:" + str(y)) # so you can see what it is doing

        # Advance to the next line index
        lineidx = lineidx + 1

        # This inner loop reads through each east/west position in a line
        for x in range(sizex):
            # Get the whole line at this 'lineidx'
            line = lines[lineidx]

            # Advance to the next line index
            lineidx = lineidx + 1

            # Split this line into separate cells, at each comma
            data = line.split(",")

            # This (third) inner loop reads through each north/south position
            for z in range(sizez):
                blockid = int(data[z])
                mc.setBlock(originx+x, originy+y, originz+z, blockid)


# Define a function that displays the menu
# This function will also ask the user for a choice number
# and it will check if the number in range or not.
# If it is not in range it will display the menu and try again
def menu():
    while True:
        print("DUPLICATOR MENU")
        print(" 1. BUILD the duplicator room")
        print(" 2. LIST files")
        print(" 3. SCAN from duplicator room to file")
        print(" 4. LOAD from file into duplicator room")
        print(" 5. PRINT from duplicator room to player.pos")
        print(" 6. CLEAN the duplicator room")
        print(" 7. DEMOLISH the duplicator room")
```

```
        print(" 8. QUIT")

        choice = int(raw_input("please choose: "))
        if choice < 1 or choice > 8:
            print("Sorry, please choose a number between 1 and 8")
        else:
            return choice


# Game loop

# The anotherGo boolean variable decides if we want to go round
# the game loop another time
anotherGo = True

while anotherGo:
    # Display the menu and get a choice from the user
    choice = menu()

    if choice == 1: # BUILD
        # Build the room at the players position
        pos = mc.player.getTilePos()
        buildRoom(pos.x, pos.y, pos.z)

    elif choice == 2: # LIST
        # List all matching CSV files
        listFiles()

    elif choice == 3: # SCAN
        # Ask the user for a file to scan into
        filename = raw_input("filename?")

        # Scan from the duplicator room into the file
        scan3D(filename, roomx+1, roomy+1, roomz+1)

    elif choice == 4: # LOAD
        # Ask the user for a file to load from
        filename = raw_input("filename?")

        # Load the file into the duplicator room
        print3D(filename, roomx+1, roomy+1, roomz+1)

    elif choice == 5: # PRINT
        # Note, this is a really neat solution here, by reusing
        # the scan3D and print3D functions, and scanning via a temporary
        # file called 'scantemp' - it means you don't have to write
        # lots of extra printing and scanning code in order to offer this
        # option to your user

        # Scan the room into a temporary file called 'scantemp'
        scan3D("scantemp", roomx+1, roomy+1, roomz+1)

        # Get the position of the player
        pos = mc.player.getTilePos()

        # Print the contents of the temporary file 'scantemp' here
        print3D("scantemp", pos.x+1, pos.y, pos.z+1)

    elif choice == 6: # CLEAN
        # Clean out just the insides of the room
        cleanRoom()

    elif choice == 7: # DEMOLISH
```

```
————→————→# Remove all trace of the room by demolishing it
————→————→# This makes sure you don't leave lots of "dead" rooms in your world
————→————→demolishRoom()

————→elif choice == 8: # QUIT
————→————→# Finish the program
————→————→# When anotherGo is False, the while loop (above) will finish
————→————→anotherGo = False

# END
```

7 Adventure 7: Building 2D and 3D Structures

7.1 Lines Circles and Spheres

```
# This program creates some lines, circles and spheres.

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time

#create the minecraft api
mc = minecraft.Minecraft.create()

#create the minecraft drawing object
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#get the players position
pos = mc.player.getTilePos()

#draw 3 lines
mcdrawing.drawLine(pos.x, pos.y, pos.z, pos.x, pos.y+20, pos.z,
↪block.WOOL.id,1)
mcdrawing.drawLine(pos.x, pos.y, pos.z, pos.x+20, pos.y, pos.z,
↪block.WOOL.id,2)
mcdrawing.drawLine(pos.x, pos.y, pos.z, pos.x + 20, pos.y + 20, pos.z,
↪block.WOOL.id, 3)

#sleep so the player can move to a different position
time.sleep(5)

#draw a circle above the player
pos = mc.player.getTilePos()
mcdrawing.drawCircle(pos.x, pos.y + 20, pos.z, 20, block.WOOL.id, 4)
time.sleep(5)

#draw a sphere above the player
pos = mc.player.getTilePos()
mcdrawing.drawSphere(pos.x, pos.y + 20, pos.z, 15, block.WOOL.id, 5)
time.sleep(5)

# END
```

7.2 Minecraft Clock

```
# This program creates a huge Minecraft clock
# big enough for you to stand on the hands as they go round.

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
import datetime
import math

def findPointOnCircle(cx, cy, radius, angle):
    x = cx + math.sin(math.radians(angle)) * radius
    y = cy + math.cos(math.radians(angle)) * radius
    x = int(round(x, 0))
    y = int(round(y, 0))
    return(x,y)

#create connection to minecraft
mc = minecraft.Minecraft.create()

#create minecraft drawing object
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#get the players position
pos = mc.player.getTilePos()

#create clock face above the player
clockMiddle = pos
clockMiddle.y = clockMiddle.y + 25

CLOCK_RADIUS = 20
HOUR_HAND_LENGTH = 10
MIN_HAND_LENGTH = 18
SEC_HAND_LENGTH = 20

#use mc drawing object to draw the circle of the clock face
mcdrawing.drawCircle(clockMiddle.x, clockMiddle.y, clockMiddle.z,
    CLOCK_RADIUS, block.DIAMOND_BLOCK.id)

#loop forever
while True:
    #Find the time
    # get the time
    timeNow = datetime.datetime.now()
    # hours
    hours = timeNow.hour
    if hours >= 12:
        hours = timeNow.hour - 12
    # minutes
    minutes = timeNow.minute
    # seconds
    seconds = timeNow.second

    #hour hand
    # what angle would a hour hand point to
    hourHandAngle = (360 / 12) * hours
    # use findPointOnCircle function to find the angle
    hourHandX, hourHandY = findPointOnCircle(clockMiddle.x, clockMiddle.y,
        HOUR_HAND_LENGTH, hourHandAngle)
    # draw hour hand
    mcdrawing.drawLine(clockMiddle.x, clockMiddle.y, clockMiddle.z,
```

```
————→————→hourHandX, hourHandY, clockMiddle.z, block.DIRT.id)

————→#minute hand
————→# what angle would a minute hand point to
————→minHandAngle = (360 / 60) * minutes
————→# use findPointOnCircle function to find the angle
————→minHandX, minHandY = findPointOnCircle(clockMiddle.x, clockMiddle.y,
————→————→MIN_HAND_LENGTH, minHandAngle)
————→# draw minute hand
————→mcdrawing.drawLine(clockMiddle.x, clockMiddle.y, clockMiddle.z - 1,
————→————→minHandX, minHandY, clockMiddle.z-1, block.WOOD_PLANKS.id)

————→#second hand
————→# what angle would a second hand point to
————→secHandAngle = (360 / 60) * seconds
————→# use findPointOnCircle function to find the angle
————→secHandX, secHandY = findPointOnCircle(clockMiddle.x, clockMiddle.y,
————→————→SEC_HAND_LENGTH, secHandAngle)
————→# draw second hand
————→mcdrawing.drawLine(clockMiddle.x, clockMiddle.y, clockMiddle.z+1,
————→————→secHandX, secHandY, clockMiddle.z+1, block.STONE.id)

————→#wait for 1 second
————→time.sleep(1)

————→#clear the time by drawing over the hands with AIR
————→mcdrawing.drawLine(clockMiddle.x, clockMiddle.y, clockMiddle.z,
————→————→hourHandX, hourHandY, clockMiddle.z, block.AIR.id)
————→mcdrawing.drawLine(clockMiddle.x, clockMiddle.y, clockMiddle.z-1,
————→————→minHandX, minHandY, clockMiddle.z-1, block.AIR.id)
————→mcdrawing.drawLine(clockMiddle.x, clockMiddle.y, clockMiddle.z+1,
————→————→secHandX, secHandY, clockMiddle.z+1, block.AIR.id)

# END
```

7.3 Polygon

```
# This program creates some very big interesting shapes (aka Polygons).

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time

#create connection to minecraft
mc = minecraft.Minecraft.create()

#create minecraftstuff drawing object
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#get the players position
pos = mc.player.getTilePos()

#draw 2d shapes
# draw a triangle
points = []
points.append(minecraft.Vec3(pos.x, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 20, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 10, pos.y + 20, pos.z))
mcdrawing.drawFace(points, True, block.WOOL.id, 6)

#move the position on a bit
pos.x = pos.x + 25

#draw a square
points = []
points.append(minecraft.Vec3(pos.x, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 20, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 20, pos.y + 20, pos.z))
points.append(minecraft.Vec3(pos.x, pos.y + 20, pos.z))
mcdrawing.drawFace(points, False, block.WOOL.id, 7)

#move the position on a bit
pos.x = pos.x + 25

#4 sided odd shape
points = []
points.append(minecraft.Vec3(pos.x, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 15, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 20, pos.y + 15, pos.z))
points.append(minecraft.Vec3(pos.x, pos.y + 20, pos.z))
mcdrawing.drawFace(points, True, block.WOOL.id, 8)

#move the position on a bit
pos.x = pos.x + 25

#5 sided odd shape
points = []
points.append(minecraft.Vec3(pos.x, pos.y, pos.z))
points.append(minecraft.Vec3(pos.x + 20, pos.y + 5, pos.z))
points.append(minecraft.Vec3(pos.x + 15, pos.y + 20, pos.z))
points.append(minecraft.Vec3(pos.x + 5, pos.y + 15, pos.z))
points.append(minecraft.Vec3(pos.x, pos.y + 5, pos.z))
mcdrawing.drawFace(points, False, block.WOOL.id, 9)

# END
```

7.4 Minecraft Pyramids

```
# This program allows you to create enormous pyramids
# of any size and number of sides.

#import python modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math

#find point on circle function
def findPointOnCircle(cx, cy, radius, angle):
    x = cx + math.sin(math.radians(angle)) * radius
    y = cy + math.cos(math.radians(angle)) * radius
    x = int(round(x, 0))
    y = int(round(y, 0))
    return(x,y)

#create connection to minecraft
mc = minecraft.Minecraft.create()

#create minecraft drawing object
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#pyramid variables
#the midde of the pyramid will be the position of the player
pyramidMiddle = mc.player.getTilePos()

PYRAMID_RADIUS = 20
PYRAMID_HEIGHT = 10
PYRAMID_SIDES = 4

#loop through the number of sides, each side will become a triangle
for side in range(0,PYRAMID_SIDES):
    #find the first point of the triangle
    point1Angle = int(round((360 / PYRAMID_SIDES) * side,0))
    point1X, point1Z = findPointOnCircle(pyramidMiddle.x, pyramidMiddle.z,
        PYRAMID_RADIUS, point1Angle)

    #find the second point of the triangle
    point2Angle = int(round((360 / PYRAMID_SIDES) * (side + 1),0))
    point2X, point2Z = findPointOnCircle(pyramidMiddle.x, pyramidMiddle.z,
        PYRAMID_RADIUS, point2Angle)

    #draw the triangle
    # create the triangle points
    trianglePoints = []
    trianglePoints.append(minecraft.Vec3(point1X, pyramidMiddle.y, point1Z))
    trianglePoints.append(minecraft.Vec3(point2X, pyramidMiddle.y, point2Z))
    trianglePoints.append(minecraft.Vec3(pyramidMiddle.x,
        pyramidMiddle.y + PYRAMID_HEIGHT, pyramidMiddle.z))
    # draw the triangle
    mcdrawing.drawFace(trianglePoints, True, block.SANDSTONE.id)

# END
```

8 Adventure 8: Giving Blocks a Mind of Their Own

8.1 Block Friend

```
# This program simulates a block friend that will follow the player around,
# providing they don't make it sad.

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math
import time

#function get the distance between 2 points
def distanceBetweenPoints(point1, point2):
    xd = point2.x - point1.x
    yd = point2.y - point1.y
    zd = point2.z - point1.z
    return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))

#constants
# how far away the block has to be before he stops following
TOO_FAR_AWAY = 15

#create minecraft and minecraftstuff objects
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#set the blocks mood
blockMood = "happy"

#create the block next to the player
friend = mc.player.getTilePos()
friend.x = friend.x + 5
# find out the height of the world at x, y, so the friend is on top
friend.y = mc.getHeight(friend.x, friend.z)
# create the block
mc.setBlock(friend.x, friend.y, friend.z, block.DIAMOND_BLOCK.id)
# say hello
mc.postToChat("<block> Hello friend")
# the friends target is where he currently is
target = friend.clone()

while True:

    #get players position
    pos = mc.player.getTilePos()
    distance = distanceBetweenPoints(pos, friend)
    #apply the rules to work out where the block should be doing next
    # am I happy?
    if blockMood == "happy":
        #where is the player?
        #Are they near enough to me or should I move to them?
        if distance < TOO_FAR_AWAY:
            target = pos.clone()
        if distance >= TOO_FAR_AWAY:
            blockMood = "sad"
            mc.postToChat(
                "<block> Come back. You are too far away. I need a hug!")
    # am I sad?
    elif blockMood == "sad":
        #if Im sad, I'll wait for my friend to come close
        #and give me a hug before I'm happy again
```

```
————→————→#print(distance)
————→————→if distance <= 1:
————→————→————→blockMood = "happy"
————→————→————→mc.postToChat("<block> Awww thanks. Lets go.")

————→#move block to the target
————→if friend != target:
————→————→#get the blocks in between block friend and player,
————→————→#by 'drawing' an imaginary line
————→————→blocksBetween = mcdrawing.getLine(friend.x, friend.y, friend.z,
————→————→————→target.x, target.y, target.z)
————→————→#loop through the blocks in between the friend and the target
————→————→#loop to the last but 1 block (:-1)
————→————→#otherwise the friend will be sitting on top of the player
————→————→for blockBetween in blocksBetween[:-1]:
————→————→————→#move the block friend to the next block
————→————→————→# clear the old block by making it air
————→————→————→mc.setBlock(friend.x, friend.y, friend.z, block.AIR.id)
————→————→————→# set the position of the block friend
————→————→————→# to be the next block in-line
————→————→————→friend = blockBetween.clone()
————→————→————→# get the height of the land at the new position
————→————→————→friend.y = mc.getHeight(friend.x, friend.z)
————→————→————→# draw the block friend in the new position
————→————→————→mc.setBlock(friend.x, friend.y, friend.z, block.DIAMOND_BLOCK.id)
————→————→————→# time to sleep between each block move
————→————→————→time.sleep(0.25)
————→————→# we have reached our target,so set the target to be friend's position
————→————→target = friend.clone()

————→#sleep for a little bit to give the computer a rest!
————→time.sleep(0.25)

# END
```

8.2 Random Number

```
# This program shows an example of how to create random numbers.

import random
randomNo = random.randint(1,10)
print randomNo

# END
```

8.3 Random Number If

```
# This program shows how to use random numbers to test probability.

import random
if random.randint(1,10) == 10:
——→print "This happens about 1 time in 10"
else:
——→print "This happens about 9 times out of 10"

# END
```

8.4 Block Friend Random

```
# This program introduces some random behavior into the block friend.

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math
import time
#import random module
import random

#function get the distance between 2 points
def distanceBetweenPoints(point1, point2):
    xd = point2.x - point1.x
    yd = point2.y - point1.y
    zd = point2.z - point1.z
    return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))

#constants
# how far away the block has to be before he stops following
TOO_FAR_AWAY = 15

#create minecraft and minecraftstuff objects
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#set the blocks mood
blockMood = "happy"

#create the block next to the player
friend = mc.player.getTilePos()
friend.x = friend.x + 5
# find out the height of the world at x, y, so the friend is on top
friend.y = mc.getHeight(friend.x, friend.z)
# create the block
mc.setBlock(friend.x, friend.y, friend.z, block.DIAMOND_BLOCK.id)
# say hello
mc.postToChat("<block> Hello friend")
# the friends target is where he currently is
target = friend.clone()

while True:

    #get players position
    pos = mc.player.getTilePos()
    distance = distanceBetweenPoints(pos, friend)
    #apply the rules to work out where the block should be doing next
    # am I happy?
    if blockMood == "happy":
        #where is the player?
        #Are they near enough to me or should I move to them?
        if distance < TOO_FAR_AWAY:
            target = pos.clone()
        if distance >= TOO_FAR_AWAY:
            blockMood = "sad"
            mc.postToChat(
            "<block> Come back. You are too far away. I need a hug!")

    # am I sad?
    elif blockMood == "sad":
        #if Im sad, I'll wait for my friend to come close
        #and give me a hug before I'm happy again
```

```
        #print distance
        if distance <= 1:
            blockMood = "happy"
            mc.postToChat("<block> Awww thanks. Lets go.")
            if random.randint(1,100) == 100:
                blockMood = "hadenough"
                mc.postToChat("<block> That's it. I have had enough.")

    #Challenge
    #elif blockMood == "hadenough":
    #    if random.randint(1,50) = 50:
    #        blockMood = "sad"
    #        mc.postToChat("<block> I forgive you, can I have that hug now?")

    #move block to the target
    if friend != target:
        #get the blocks in between block friend and player,
        #by 'drawing' an imaginary line
        blocksBetween = mcdrawing.getLine(friend.x, friend.y, friend.z,
            target.x, target.y, target.z)
        #loop through the blocks in between the friend and the target
        # loop to the last but 1 block (:-1)
        #otherwise the friend will be sitting on top of the player
        for blockBetween in blocksBetween[:-1]:
            #move the block friend to the next block
            # clear the old block by making it air
            mc.setBlock(friend.x, friend.y, friend.z, block.AIR.id)
            # set the position of the block friend
            # to be the next block in-line
            friend = blockBetween.clone()
            # get the height of the land at the new position
            friend.y = mc.getHeight(friend.x, friend.z)
            # draw the block friend in the new position
            mc.setBlock(friend.x, friend.y, friend.z, block.DIAMOND_BLOCK.id)
            # time to sleep between each block move
            time.sleep(0.25)
        # we have reached our target,so set the target to be friend's position
        target = friend.clone()

    #sleep for a little bit to give the computer a rest!
    time.sleep(0.25)

# END
```

8.5 Wooden Horse

```
# This program shows how to create complex minecraft shapes and move them,
# by building a wooden horse.

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time

#create minecraft and minecraftdrawing objects
mc = minecraft.Minecraft.create()

#create the shape for our flying saucer
horseBlocks = [
    minecraftstuff.ShapeBlock(0,0,0,block.WOOD_PLANKS.id),
    minecraftstuff.ShapeBlock(-1,0,0,block.WOOD_PLANKS.id),
    minecraftstuff.ShapeBlock(1,0,0,block.WOOD_PLANKS.id),
    minecraftstuff.ShapeBlock(-1,-1,0,block.WOOD_PLANKS.id),
    minecraftstuff.ShapeBlock(1,-1,0,block.WOOD_PLANKS.id),
    minecraftstuff.ShapeBlock(1,1,0,block.WOOD_PLANKS.id),
    minecraftstuff.ShapeBlock(2,1,0,block.WOOD_PLANKS.id)
]

#set the horses position
horsePos = mc.player.getTilePos()
horsePos.z = horsePos.z + 1
horsePos.y = horsePos.y + 1

#create the horseShape
horseShape = minecraftstuff.MinecraftShape(mc, horsePos, horseBlocks)

#make the horse move
for count in range(1,10):
    time.sleep(1)
    horseShape.moveBy(1,0,0)

horseShape.clear()

# END
```

8.6 Alien Invasion

```
# This program creates an alien invasion,
# where an alien craft tries to abduct the player and beam him onboard.

import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math
import time
import random

#function get the distance between 2 points
def distanceBetweenPoints(point1, point2):
    xd = point2.x - point1.x
    yd = point2.y - point1.y
    zd = point2.z - point1.z
    return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))

#constants
# will go here!
HOVER_HEIGHT = 15
ALIEN_TAUNTS = [
    "<aliens>You cant run forever",
    "<aliens>Resistance is useless",
    "<aliens>We only want to be friends"
]

#create minecraft and minecraftdrawing objects
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)

#set the aliens position
alienPos = mc.player.getTilePos()
alienPos.y = alienPos.y + 50

#set the flying saucer's mode
mode = "landing"

#create the shape for our flying saucer
alienBlocks = [
    minecraftstuff.ShapeBlock(-1,0,0,block.WOOL.id, 5),
    minecraftstuff.ShapeBlock(0,0,-1,block.WOOL.id, 5),
    minecraftstuff.ShapeBlock(1,0,0,block.WOOL.id, 5),
    minecraftstuff.ShapeBlock(0,0,1,block.WOOL.id, 5),
    minecraftstuff.ShapeBlock(0,-1,0,block.GLOWSTONE_BLOCK.id),
    minecraftstuff.ShapeBlock(0,1,0,block.GLOWSTONE_BLOCK.id)
]

alienShape = minecraftstuff.MinecraftShape(mc, alienPos, alienBlocks)

#loop
while mode != "missionaccomplished":
    #get the players position
    playerPos = mc.player.getTilePos()

    # if its landing, set target to be above players head
    if mode == "landing":
        mc.postToChat("<aliens> We dont come in peace - please panic")
        alienTarget = playerPos.clone()
        alienTarget.y = alienTarget.y + HOVER_HEIGHT
        mode = "attack"
    elif mode == "attack":
```

```
        #check to see if the alien ship is above the player
        if alienPos.x == playerPos.x and alienPos.z == playerPos.z:
            #the alienship is above the player
            mc.postToChat("<aliens>We have you now!")

            # create a room
            mc.setBlocks(0,50,0,6,56,6,block.BEDROCK.id)
            mc.setBlocks(1,51,1,5,55,5,block.AIR.id)
            mc.setBlock(3,55,3,block.GLOWSTONE_BLOCK.id)

            #beam up player
            mc.player.setTilePos(3,51,5)
            time.sleep(10)
            mc.postToChat(
                "<aliens>Not very interesting at all - send it back")
            time.sleep(2)

            #send the player back to the original position
            mc.player.setTilePos(playerPos.x, playerPos.y, playerPos.z)

            #clear the room
            mc.setBlocks(0,50,0,6,56,6,block.AIR.id)
            mode = "missionaccomplished"

        else:
            #the player got away
            mc.postToChat(ALIEN_TAUNTS[random.randint(0,len(ALIEN_TAUNTS)-1)])
            alienTarget = playerPos.clone()
            alienTarget.y = alienTarget.y + HOVER_HEIGHT

    #move the flying saucer towards its target
    if alienPos != alienTarget:
        #get the blocks in between block friend and player,
        #by 'drawing' an imaginary line
        blocksBetween = mcdrawing.getLine(
            alienPos.x, alienPos.y, alienPos.z,
            alienTarget.x, alienTarget.y, alienTarget.z)
        #loop through the blocks in between the alien and the target
        for blockBetween in blocksBetween:
            #move the block friend to the next block
            # move the alien shape to the new position
            alienShape.move(blockBetween.x, blockBetween.y, blockBetween.z)
            # time to sleep between each block move
            time.sleep(0.25)
        # we have reached our target,so set the target to be friend's position
        alienPos = alienTarget.clone()

#clear the alien ship
alienShape.clear()

# END
```

9 Adventure 9: The Big Adventure: Crafty Crossing

9.1 Crafty Crossing

```
# This program is a complete mini game, called Crafty Crossing.
# The player has to get to the other side of the arena,
# collecting diamond blocks along the way and avoiding the obstacles.

#import modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
import random
import thread

#arena constants
ARENAX = 10
ARENAZ = 20
ARENAY = 3

#Create the Arena
def createArena(pos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #Create the floor
    mc.setBlocks(pos.x - 1 , pos.y, pos.z - 1,
        pos.x + ARENAX + 1, pos.y - 3, pos.z + ARENAZ + 1, block.GRASS.id)

    #Create the walls on the outside of the arena
    mc.setBlocks(pos.x - 1, pos.y + 1, pos.z - 1,
        pos.x + ARENAX + 1, pos.y + ARENAY, pos.z + ARENAZ + 1,
        block.GLASS.id)
    mc.setBlocks(pos.x, pos.y + 1, pos.z,
        pos.x + ARENAX, pos.y + ARENAY, pos.z + ARENAZ, block.AIR.id)

#Build the obstacles
#The Wall
def theWall(arenaPos, wallZPos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #create the wall shape
    wallPos = minecraft.Vec3(
        arenaPos.x, arenaPos.y + 1, arenaPos.z + wallZPos)
    #create the wall blocks
    wallBlocks = []
    for x in range(0, ARENAX + 1):
        for y in range(1, ARENAY):
            wallBlocks.append(
                minecraftstuff.ShapeBlock(x, y, 0, block.BRICK_BLOCK.id))
    wallShape = minecraftstuff.MinecraftShape(mc, wallPos, wallBlocks)

    #move the wall up and down
    while not gameOver:
        wallShape.moveBy(0,1,0)
        time.sleep(1)
        wallShape.moveBy(0,-1,0)
        time.sleep(1)

#The River
def theRiver(arenaPos, riverZPos):
```

```
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #constants
    RIVERWIDTH = 4
    BRIDGEWIDTH = 2

    #create the river
    mc.setBlocks(arenaPos.x, arenaPos.y - 2, arenaPos.z + riverZPos,
        arenaPos.x + ARENAX, arenaPos.y,
        arenaPos.z + riverZPos + RIVERWIDTH - 1, block.AIR.id)
    #fill with water
    mc.setBlocks(arenaPos.x, arenaPos.y - 2, arenaPos.z + riverZPos,
        arenaPos.x + ARENAX, arenaPos.y - 2,
        arenaPos.z + riverZPos + RIVERWIDTH - 1, block.WATER.id)
    #create the bridge shape
    bridgePos = minecraft.Vec3(
        arenaPos.x, arenaPos.y, arenaPos.z + riverZPos + 1)
    #create the bridge blocks
    bridgeBlocks = []
    for x in range(0, BRIDGEWIDTH):
        for z in range(0, RIVERWIDTH - 2):
            bridgeBlocks.append(
                minecraftstuff.ShapeBlock(x, 0, z, block.WOOD_PLANKS.id))
    bridgeShape = minecraftstuff.MinecraftShape(mc, bridgePos, bridgeBlocks)

    #move the bridge left and right
    #how many steps are there between the left and right side of the arena
    steps = ARENAX - BRIDGEWIDTH
    while not gameOver:
        for left in range(0, steps):
            bridgeShape.moveBy(1,0,0)
            time.sleep(1)
        for right in range(0, steps):
            bridgeShape.moveBy(-1,0,0)
            time.sleep(1)

def theHoles(arenaPos, holesZPos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #constants
    HOLES = 15
    HOLESWIDTH = 3

    while not gameOver:
        #create random holes which open up for a few seconds,
        #close, then change to a different set of holes
        holes = []
        #find some random holes
        for count in range(0,HOLES):
            x = random.randint(arenaPos.x, arenaPos.x + ARENAX)
            z = random.randint(arenaPos.z + holesZPos,
                arenaPos.z + holesZPos + HOLESWIDTH)
            holes.append(minecraft.Vec3(x, arenaPos.y, z))
        #turn the holes black before opening them up
        for hole in holes:
            mc.setBlock(hole.x, hole.y, hole.z, block.WOOL.id, 15)
        time.sleep(0.25)
        #open up the holes
        for hole in holes:
            mc.setBlocks(hole.x, hole.y, hole.z,
                hole.x, hole.y - 2, hole.z, block.AIR.id)
```

```
————→————→time.sleep(2)
————→————→#close up the holes
————→————→for hole in holes:
————→————→————→mc.setBlocks(hole.x, hole.y, hole.z,
————→————→————→————→hole.x, hole.y - 2, hole.z, block.GRASS.id)
————→————→time.sleep(0.25)

#Place the diamonds
def createDiamonds(arenaPos, number):
————→#create connection to minecraft
————→mc = minecraft.Minecraft.create()

————→#create the number of diamonds required
————→for diamond in range(0, number):
————→————→#create a random position
————→————→x = random.randint(arenaPos.x, arenaPos.x + ARENAX)
————→————→z = random.randint(arenaPos.z, arenaPos.z + ARENAZ)
————→————→#create the diamond block
————→————→mc.setBlock(x, arenaPos.y + 1, z, block.DIAMOND_BLOCK.id)

#Main program
#create minecraft object
mc = minecraft.Minecraft.create()

#create the gameOver flag
gameOver = False

#arena pos
arenaPos = mc.player.getTilePos()

#build the arena
createArena(arenaPos)

#create the wall
WALLZ = 10
#theWall(arenaPos, WALLZ)
thread.start_new_thread(theWall, (arenaPos, WALLZ))

#create the river
RIVERZ = 4
#theRiver(arenaPos, RIVERZ)
thread.start_new_thread(theRiver, (arenaPos, RIVERZ))

#create the holes
HOLESZ = 15
#theHoles(arenaPos, HOLESZ)
thread.start_new_thread(theHoles, (arenaPos, HOLESZ))

#level constants
LEVELS = 3
DIAMONDS = [3,5,9]
TIMEOUTS = [30,25,20]

#set level and points
level = 0
points = 0

#game loop, while not game over
while not gameOver:
————→#create the diamonds
————→createDiamonds(arenaPos, DIAMONDS[level])
————→diamondsLeft = DIAMONDS[level]
```

```
    #position the player at the start of the arena
    mc.player.setPos(arenaPos.x + 1, arenaPos.y + 1, arenaPos.z + 1)

    #start the clock
    start = time.time()

    #set the level complete flag
    levelComplete = False
    #level loop
    while not gameOver and not levelComplete:
        #sleep for a bit
        time.sleep(0.1)

        #has player hit a diamond?
        hits = mc.events.pollBlockHits()
        for hit in hits:
            blockHitType = mc.getBlock(hit.pos.x, hit.pos.y, hit.pos.z)
            if blockHitType == block.DIAMOND_BLOCK.id:
                #turn the block to AIR
                mc.setBlock(hit.pos.x,hit.pos.y, hit.pos.z, block.AIR.id)
                #reduce the diamonds to collect by 1
                diamondsLeft = diamondsLeft - 1

        #get the players position
        pos = mc.player.getTilePos()

        #has player fallen down
        if pos.y < arenaPos.y:
            #put them back to the start
            mc.player.setPos(arenaPos.x + 1, arenaPos.y + 1, arenaPos.z + 1)

        #has the player got to the end of the arena and got all the diamonds?
        if pos.z == arenaPos.z + ARENAZ and diamondsLeft == 0:
            levelComplete = True

        #has the time expired
        secondsLeft = TIMEOUTS[level] - (time.time() - start)
        if secondsLeft < 0:
            gameOver = True
            mc.postToChat("Out of time...")

    #level complete?
    if levelComplete:
        #calculate points
        #1 points for every diamond
        #1 x multipler for every second left on the clock
        points = points + (DIAMONDS[level] * int(secondsLeft))
        mc.postToChat("Level Complete - Points = " + str(points))
        #set it to the next level
        level = level + 1
        #if its the last level, set it to gameOver
        if level == LEVELS:
            gameOver = True
            mc.postToChat("Congratulations - All levels complete")

#its game over
mc.postToChat("Game Over - Points = " + str(points))

# END
```

9.2 Crafty Crossing With Hardware

```
# This program adds a physical start button to the game,
# uses a 7 segment display to count down the number of diamonds to collect,
# and lights up the LED on the display when the time is running out.

#import modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
import random
import thread

#HARDWARE - START
#7 segment display
import anyio.seg7 as display

#Raspberry Pi Hardware
import RPi.GPIO as GPIO
BUTTON = 4
LED_PINS = [10,22,25,8,7,9,11,15]

#Arduino Hardware
#import anyio.GPIO as GPIO
#BUTTON = 4
#LED_PINS = [7,6,14,16,10,8,9,15]

#setup hardware
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN)
ON = False # common-anode, set to True for common cathode
display.setup(GPIO, LED_PINS, ON)

#HARDWARE - END

#arena constants
ARENAX = 10
ARENAZ = 20
ARENAY = 3

#Create the Arena
def createArena(pos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #Create the floor
    mc.setBlocks(pos.x - 1 , pos.y, pos.z - 1,
        pos.x + ARENAX + 1, pos.y - 3, pos.z + ARENAZ + 1, block.GRASS.id)

    #Create the walls on the outside of the arena
    mc.setBlocks(pos.x - 1, pos.y + 1, pos.z - 1,
        pos.x + ARENAX + 1, pos.y + ARENAY, pos.z + ARENAZ + 1,
        block.GLASS.id)
    mc.setBlocks(pos.x, pos.y + 1, pos.z,
        pos.x + ARENAX, pos.y + ARENAY, pos.z + ARENAZ,
        block.AIR.id)

#Build the obstacles
#The Wall
def theWall(arenaPos, wallZPos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()
```

```
    #create the wall shape
    wallPos = minecraft.Vec3(
        arenaPos.x, arenaPos.y + 1, arenaPos.z + wallZPos)
    #create the wall blocks
    wallBlocks = []
    for x in range(0, ARENAX + 1):
        for y in range(1, ARENAY):
            wallBlocks.append(
                minecraftstuff.ShapeBlock(x, y, 0, block.BRICK_BLOCK.id))
    wallShape = minecraftstuff.MinecraftShape(mc, wallPos, wallBlocks)

    #move the wall up and down
    while not gameOver:
        wallShape.moveBy(0,1,0)
        time.sleep(1)
        wallShape.moveBy(0,-1,0)
        time.sleep(1)

#The River
def theRiver(arenaPos, riverZPos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #constants
    RIVERWIDTH = 4
    BRIDGEWIDTH = 2

    #create the river
    mc.setBlocks(arenaPos.x, arenaPos.y - 2, arenaPos.z + riverZPos,
        arenaPos.x + ARENAX, arenaPos.y,
        arenaPos.z + riverZPos + RIVERWIDTH - 1, block.AIR.id)
    #fill with water
    mc.setBlocks(arenaPos.x, arenaPos.y - 2, arenaPos.z + riverZPos,
        arenaPos.x + ARENAX, arenaPos.y - 2,
        arenaPos.z + riverZPos + RIVERWIDTH - 1, block.WATER.id)
    #create the bridge shape
    bridgePos = minecraft.Vec3(
        arenaPos.x, arenaPos.y, arenaPos.z + riverZPos + 1)
    #create the bridge blocks
    bridgeBlocks = []
    for x in range(0, BRIDGEWIDTH):
        for z in range(0, RIVERWIDTH - 2):
            bridgeBlocks.append(
                minecraftstuff.ShapeBlock(x, 0, z, block.WOOD_PLANKS.id))
    bridgeShape = minecraftstuff.MinecraftShape(mc, bridgePos, bridgeBlocks)

    #move the bridge left and right
    #how many steps are there between the left and right side of the arena
    steps = ARENAX - BRIDGEWIDTH
    while not gameOver:
        for left in range(0, steps):
            bridgeShape.moveBy(1,0,0)
            time.sleep(1)
        for right in range(0, steps):
            bridgeShape.moveBy(-1,0,0)
            time.sleep(1)

def theHoles(arenaPos, holesZPos):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #constants
```

```
    HOLES = 15
    HOLESWIDTH = 3

    while not gameOver:
        #create random holes which open up for a few seconds,
        #close, then change to a different set of holes
        holes = []
        #find some random holes
        for count in range(0,HOLES):
            x = random.randint(arenaPos.x, arenaPos.x + ARENAX)
            z = random.randint(arenaPos.z + holesZPos,
                arenaPos.z + holesZPos + HOLESWIDTH)
            holes.append(minecraft.Vec3(x, arenaPos.y, z))
        #turn the holes black before opening them up
        for hole in holes:
            mc.setBlock(hole.x, hole.y, hole.z, block.WOOL.id, 15)
        time.sleep(0.25)
        #open up the holes
        for hole in holes:
            mc.setBlocks(hole.x, hole.y, hole.z,
                hole.x, hole.y - 2, hole.z, block.AIR.id)
        time.sleep(2)
        #close up the holes
        for hole in holes:
            mc.setBlocks(hole.x, hole.y, hole.z,
                hole.x, hole.y - 2, hole.z, block.GRASS.id)
        time.sleep(0.25)

#Place the diamonds
def createDiamonds(arenaPos, number):
    #create connection to minecraft
    mc = minecraft.Minecraft.create()

    #create the number of diamonds required
    for diamond in range(0, number):
        #create a random position
        x = random.randint(arenaPos.x, arenaPos.x + ARENAX)
        z = random.randint(arenaPos.z, arenaPos.z + ARENAZ)
        #create the diamond block
        mc.setBlock(x, arenaPos.y + 1, z, block.DIAMOND_BLOCK.id)

#Main program
#create minecraft object
mc = minecraft.Minecraft.create()

#create the gameOver flag
gameOver = False

#arena pos
arenaPos = mc.player.getTilePos()

#build the arena
createArena(arenaPos)

#create the wall
WALLZ = 10
#theWall(arenaPos, WALLZ)
thread.start_new_thread(theWall, (arenaPos, WALLZ))

#create the river
RIVERZ = 4
#theRiver(arenaPos, RIVERZ)
thread.start_new_thread(theRiver, (arenaPos, RIVERZ))
```

```
#create the holes
HOLESZ = 15
#theHoles(arenaPos, HOLESZ)
thread.start_new_thread(theHoles, (arenaPos, HOLESZ))

#level constants
LEVELS = 3
DIAMONDS = [3,5,9]
TIMEOUTS = [30,25,20]

#set level and points
level = 0
points = 0

#HARDWARE - START
#wait for the button to be pressed
mc.postToChat("Press the button to start")
while GPIO.input(BUTTON):
    time.sleep(0.1)
#HARDWARE - END

#game loop, while not game over
while not gameOver:
    #create the diamonds
    createDiamonds(arenaPos, DIAMONDS[level])
    diamondsLeft = DIAMONDS[level]
    #HARDWARE - update 7 segment display
    display.write(str(diamondsLeft))

    #position the player at the start of the arena
    mc.player.setPos(arenaPos.x + 1, arenaPos.y + 1, arenaPos.z + 1)

    #start the clock
    start = time.time()

    #set the level complete flag
    levelComplete = False
    #level loop
    while not gameOver and not levelComplete:
        #sleep for a bit
        time.sleep(0.1)

        #has player hit a diamond?
        hits = mc.events.pollBlockHits()
        for hit in hits:
            blockHitType = mc.getBlock(hit.pos.x, hit.pos.y, hit.pos.z)
            if blockHitType == block.DIAMOND_BLOCK.id:
                #turn the block to AIR
                mc.setBlock(hit.pos.x,hit.pos.y, hit.pos.z, block.AIR.id)
                #reduce the diamonds to collect by 1
                diamondsLeft = diamondsLeft - 1
                #HARDWARE - update 7 segment display
                display.write(str(diamondsLeft))

        #get the players position
        pos = mc.player.getTilePos()

        #has player fallen down
        if pos.y < arenaPos.y:
            #put them back to the start
            mc.player.setPos(arenaPos.x + 1, arenaPos.y + 1, arenaPos.z + 1)
```

```
        #has the player got to the end of the arena and got all the diamonds?
        if pos.z == arenaPos.z + ARENAZ and diamondsLeft == 0:
            levelComplete = True

        #has the time expired
        secondsLeft = TIMEOUTS[level] - (time.time() - start)
        #HARDWARE - START
        #if there are less than 5 seconds left turn on the decimal point
        if secondsLeft < 5:
            display.setdp(True)
        else:
            display.setdp(False)
        #HARDWARE - END
        if secondsLeft < 0:
            gameOver = True
            mc.postToChat("Out of time...")

    #level complete?
    if levelComplete:
        #calculate points
        #1 points for every diamond
        #1 x multipler for every second left on the clock
        points = points + (DIAMONDS[level] * int(secondsLeft))
        mc.postToChat("Level Complete - Points = " + str(points))
        #set it to the next level
        level = level + 1
        #if its the last level, set it to gameOver
        if level == LEVELS:
            gameOver = True
            mc.postToChat("Congratulations - All levels complete")

#its game over
mc.postToChat("Game Over - Points = " + str(points))

#HARDWARE - START
#clear the display and cleanup the GPIO
display.clear()
GPIO.cleanup()
#HARDWARE - END

# END
```

10 Adventure 10: The Minecraft Lift

10.1 Lift

```
# This program builds a fully functional passenger lift.
# The lift is a footplate that moves up and down a lift shaft.
# You can press hardware buttons on a breadboard to choose which
# floor to travel to and it takes your player with you.
# You can also climb outside of the lift and hit one of many
# diamond "request blocks" on each floor, to request the lift to come
# to that floor to greet you.
# This program works on Raspberry Pi, PC and Mac.

# Import necessary modules
import mcpi.minecraft as minecraft
import mcpi.block as block
import time

# This module is needed to drive the 7-segment display
import anyio.seg7 as display

# Used on the Raspberry Pi
import RPi.GPIO as GPIO
# GPIO numbers of the 4 buttons
FLOOR_GPIO   = [4, 14, 23, 24] # the order is important
# GPIO numbers of the 8 LEDs inside the 7-seg display
DISPLAY_GPIO = [10, 22, 25, 8, 7, 9, 11, 15] # the order is important
# What type of display do you have?
ON           = False # True=common-cathode, False=common-anode

# Used on the Arduino on PC/Mac
#import anyio.GPIO as GPIO
# GPIO numbers of the 4 buttons
#FLOOR_GPIO   = [4, 5, 2, 3] # the order is important
# GPIO numbers of the 8 LEDs inside the 7-seg display
#DISPLAY_GPIO = [7, 6, 14, 16, 10, 8, 9, 15] # the order is important
# What type of display do you have?
#ON           = False # True=common-cathode, False=common-anode

# Create some constants for the floors.
# This will make it easier to add more floors later, or make the
# floors further apart or nearer to each other depending on the
# height of your building
NUM_FLOORS   = len(FLOOR_GPIO)
FLOOR_HEIGHT = 10

# Create constants for the 3 different states that the lift can be in
STOPPED      = 0
GOING_DOWN   = 1
GOING_UP     = 2

# Connect to the Minecraft game
mc           = minecraft.Minecraft.create()

# Get the player position
pos          = mc.player.getTilePos()

# Create some constants for the lift position
# The position is moved slightly in the x and z dimensions
# This is so that it is not built on top of the players head
# This will build the lift at a position relative to your player
# If you want the lift at an absolute position (always at the
# same position regardless of where the player is), you can set
```

```
# these 3 constants to the x,y,z coordinates of the insides of
# your building, so the lift never gets relocated elsewhere
LIFT_X       = pos.x+3
LIFT_Y       = pos.y
LIFT_Z       = pos.z+3

# The lift starts off stopped, right at the bottom
lift_state   = STOPPED
lift_y       = LIFT_Y

# These 4 Boolean variables keep track of which floors have a request
# outstanding, such as when the player has pressed one of the
# hardware buttons, or has hit one of the diamond request blocks
# outside the lift entrance at each floor
lift_requests = [False, False, False, False]

# Define a function that creates the lift
# It carves out a lift shaft using AIR, so that if you build your lift
# inside a building or mountain, you will have a long column of AIR
# that the lift can go up and down. It puts diamond blocks outside
# of the entrance to the lift at each floor, so you know where to
# attach the floors of your building to.
def createLift():
    # loop for each floor
    for floor in range(NUM_FLOORS):
        # calculate the y coordinate of the height of this floor
        y = LIFT_Y + floor*FLOOR_HEIGHT
        # carve out a cube of air for the lift shaft for this whole floor
        mc.setBlocks(LIFT_X-2, y, LIFT_Z-2, LIFT_X+2, y+FLOOR_HEIGHT, LIFT_Z+2,
            block.AIR.id)
        # place 3 diamond blocks at the entrance to the lift on this floor
        mc.setBlocks(LIFT_X+2, y, LIFT_Z-1, LIFT_X+2, y, LIFT_Z+1,
            block.DIAMOND_BLOCK.id)


# Define a function that draws the lift at a certain height,
# using a certain block id number.
# A lift in this simple program is just a slab you can stand on
# called the footplate. There are no walls or ceiling or doors.
# y - the height that the lift needs drawing at
# blockid - the blockid to use for the footplate of the lift
def drawLift(y, blockid):
    mc.setBlocks(LIFT_X-1, y, LIFT_Z-1, LIFT_X+1, y, LIFT_Z+1, blockid)

# Define a function that works out if the player is in the lift or not.
# This geo-fences the whole lift shaft and works out if the player
# is in that area or not. This is a bit of a cheat, as it doesn't
# mean they are actually in the lift, but it's good enough.
def playerInLift():
    # Get the player position
    pos = mc.player.getTilePos()

    # Check if the player is within the lift shaft or not
    if pos.x >= LIFT_X-1 and pos.x <= LIFT_X+1 \
    and pos.z >= LIFT_Z-1 and pos.z <= LIFT_Z+1:
        return True # They are in the lift
    return False # They are not in the lift


# Define a function that moves the lift up by one block
# It does this by un-building and re-building the footplate slab
# withPlayer - True if the player needs to move with the lift
def moveUp(withPlayer):
```

```
    # This function will change the value in the lift_y variable
    # so it must be listed as global here
    global lift_y

    # If the lift is moving with the player on the slab
    if withPlayer:
        # Move the player up by one block
        # This prevents the new lift slab being built on top of their feet
        mc.player.setPos(LIFT_X, lift_y+2, LIFT_Z)

    # build a new footplate 1 block higher than the existing footplate
    drawLift(lift_y+1, block.STONE.id)

    # delete the old footplate that is below the new footplate
    drawLift(lift_y, block.AIR.id)

    # Move the player up again
    # On slow computers, sometimes the player falls down before the
    # new footplate is built, so this prevents them from falling
    # down the whole lift shaft, by moving them up one block again
    if withPlayer:
        mc.player.setPos(LIFT_X, lift_y+2, LIFT_Z)

    # The lift has moved up by one block, so adjust the lift_y variable
    # so that it always holds the y coordinate of the lift
    lift_y = lift_y+1


# Define a function that moves the lift down by one block
# It does this by un-building and re-building the footplate slab
# withPlayer - True if the player needs to move with the lift
def moveDown(withPlayer):
    # This function will change the value in the lift_y variable
    # so it must be listed as global here
    global lift_y

    # Build a new footplate below the existing one
    drawLift(lift_y-1, block.STONE.id)

    # Delete the existing footplate
    drawLift(lift_y, block.AIR.id)

    # If the player is in the lift
    if withPlayer:
        # Make sure they move down with the lift
        # The player will naturally fall by gravity
        # but if flying is enabled, this forces them to move down
        mc.player.setPos(LIFT_X, lift_y+1, LIFT_Z)

    # The lift has moved down by one block, so adjust the lift_y variable
    # so that it always holds the y coordinate of the lift
    lift_y = lift_y-1


# Define a function that calculates the floor number for a y position
# y - the y position to calculate the floor number of
# This uses knowledge of the size of the lift shaft and number of floors
# to calculate the floor number
def getFloor(y):
    # Divide the height up the lift shaft by the height of the floors
    # and check if there is any remainder after that division
    remainder = (y-LIFT_Y) % FLOOR_HEIGHT
```

```
    # If there is no remainder, the lift is at a floor
    if remainder == 0:
        # Calculate the actual floor number
        return (y-LIFT_Y) / FLOOR_HEIGHT

    # The lift is not at a floor, so return "None" (no value)
    return None


# Define a function that looks for new requests for the lift
# A request can come from a hardware button, or a diamond hit event
def checkRequests():
    # This function changes the value of lift_requests
    # so it has to be listed as global here
    global lift_requests

    # CHECK FOR ANY DIAMOND HITS

    # loop through any hit events that have happened
    for e in mc.events.pollBlockHits():
        pos = e.pos

        # Work out the floor number of the y coordinate of this hit
        # Note how the getFloor() function is used in multiple places
        # inside this program, for different purposes
        floor = getFloor(pos.y)

        # If the lift is at a floor
        if floor != None:
            # Set the appropriate Boolean in the lift_requests
            # This remembers that there is a request for this floor
            # which will be processed elsewhere in this program
            lift_requests[floor] = True


    # CHECK FOR ANY HARDWARE BUTTON PRESSES

    # Hardware buttons are only looked at if the player is in the lift
    # This is because your player can't reach the buttons
    # if they are not in the lift!
    # Note how the playerInLift() function is used in multiple places
    # inside this program, for different purposes
    if playerInLift():
        # Loop through all the floors, one by one
        for floor in range(NUM_FLOORS):
            # Read the hardware button for this floor number
            if GPIO.input(FLOOR_GPIO[floor]) == False:
                # False means 0V, which means the button is pressed
                # Set the appropriate Boolean in the lift requests
                # This remembers that there is a request for this floor
                # which will be processed elsewhere in this program
                lift_requests[floor] = True

    # To help us debug the program, the lift requests are printed
    # on the Python Shell Window every time this function is called
    print(str(lift_requests))


# Define a function that performs necessary actions when the lift
# is at a particular floor
# floor - the floor numbe that the lift is now at
def atFloor(floor):
    # This function will change the values in these variables
```

```
    # so they must be listed as global here
    global lift_requests, lift_state

    # If there is a lift request pending for this floor
    if lift_requests[floor]:
        # Set the lift request to False, so the lift doesn't stick here
        lift_requests[floor] = False

        # The lift is now stopped (this will be used elsewhere)
        lift_state = STOPPED

        # Update the display hardware with the floor number
        display.write(str(floor))

        # Wait a couple of seconds to allow the player to walk on/off
        time.sleep(2)


# Define a function that chooses the next direction to move the lift
# depending on what other requests are pending, and where the lift is.
# This is a very simple algorithm that prioritises floors below the
# player first in the journey. This is normal for a lift, because
# most buildings have a lobby on the ground floor that is more
# popular than all other floors
def chooseNext(floor):
    # This function changes the lift_state variable
    # so it must be listed as global here
    global lift_state

    # Loop through all floors from the lowest to the highest
    for f in range(NUM_FLOORS):
        # If there is a request pending for this floor number
        if lift_requests[f]:
            # work out which direction the lift needs to trave
            # in order to get to the floor
            if f > floor:
                lift_state = GOING_UP
            else:
                lift_state = GOING_DOWN

            # 'return' from function immediately once direction is chosen
            # This prevents the loop looking at all other floors and
            # possibly making the wrong decision on the direction to use
            return


# Define a function that can be used for service engineers
# This allows them to manually move the lift up and down
# Pressing the first hardware button moves it up 1 block
# Pressing the second hardware button moves it down 1 block
# The main game loop should either use manualLift() or it should
# use autoLift(), but never both at the same time
def manualLift():
    # This function will change the value in lift_requests
    # so it has to be listed as global here
    global lift_requests

    # If there is a request on floor 0 (first button pressed)
    if lift_requests[0]:
        # Move the lift up one block
        moveUp(True)

        # Clear the lift request, to prevent the lift moving forever
```

```
————→————→lift_requests[0] = False

————→# else if there is a request on floor 1 (second button pressed)
————→elif lift_requests[1]:
————→————→# Move the lift down one block
————→————→moveDown(True)

————→————→# Clear the lift request, to prevent the lift moving forever
————→————→lift_requests[1] = False

————→# Keep the display updated with the floor number
————→# This is useful so that the service engineer can test the display
————→floor = getFloor(lift_y)
————→if floor != None:
————→————→display.write(str(floor))


# Define a function that automatically moves the lift
# This is used in the main game loop, and you should either use
# autoLift() or use manualLift() but never both at the same time
def autoLift():
————→# If the player is in the lift, the movement will have to also
————→# move the player as well. Store this in a Boolean withPlayer
————→# that can be used to alter how the move functions work later
————→withPlayer = playerInLift()

————→# Work out if the lift is at a afloor, and if so, which floor
————→floor = getFloor(lift_y)

————→# If the lift is at a floor
————→if floor != None:
————→————→# Perform the at-floor processing
————→————→# which will stop the lift for a while
————→————→atFloor(floor)


————→# DECIDE WHAT TO DO NEXT
————→# The lift can be in one of 3 states: GOING_UP, GOING_DOWN, STOPPED
————→# Decide what to do for each of those 3 states

————→if lift_state == GOING_UP:
————→————→# Writing "up" to the display module will display a '^' symbol
————→————→# which looks a little bit like an up arrow
————→————→display.write("up")

————→————→# Move the lift up by one block (with the player too
————→————→# if the withPlayer variable is True)
————→————→moveUp(withPlayer)

————→elif lift_state == GOING_DOWN:
————→————→# Writing "down" to the display module will display a 'v' symbol
————→————→# which looks a little bit like a down arrow
————→————→display.write("down")

————→————→# Move the lift down by one blok (with the player too
————→————→# if the withPlayer variable is True)
————→————→moveDown(withPlayer)

————→else:
————→————→# Anything else, means the lift must be STOPPED
————→————→# So, choose which floor to travel to next.
————→————→# chooseNext() will change lift_state to GOING_UP or GOING_DOWN
————→————→# which will then be processed next time this function is used
```

```
————→————→chooseNext(floor)


# Define a function that destroys the lift
# When the game finishes, the lift needs to be destroyed
# otherwise you might get lots of "dead" inactive lifts all over
# the Minecraft world. If you want to always build the lift at
# a fixed location, you can change the constants at the top of this
# program to hold the absolute coordinates of your lift inside your
# building (rather than relative coordinates based on your position)
def destroyLift():
————→# Clear all the space in the lift shaft and around it
————→# which will remove the lift and the diamond request blocks
————→mc.setBlocks(LIFT_X-2, LIFT_Y, LIFT_Z-2, LIFT_X+2,
————→————→LIFT_Y+(NUM_FLOORS*FLOOR_HEIGHT), LIFT_Z+2, block.AIR.id)


# Game loop

try: # if this code fails, it jumps to "finally"
————→# Use Broadcom pin numbering scheme for GPIOs
————→GPIO.setmode(GPIO.BCM)

————→# Set all button GPIOs to be inputs so you can read their state
————→for p in FLOOR_GPIO:
————→————→GPIO.setup(p, GPIO.IN)

————→# Setup the display ready for use
————→display.setup(GPIO, DISPLAY_GPIO, ON)

————→# Create the lift shaft, diamond request blocks, and footplate
————→createLift()
————→drawLift(lift_y, block.STONE.id)

————→# Loop forever (or until CTRL-C)
————→while True:
————→————→# Delay a short time (to prevent computer being too busy!)
————→————→time.sleep(0.5)

————→————→# Check if there are any pending requests to process
————→————→checkRequests()

————→————→# Move the lift in automatic mode
————→————→autoLift() # change to manualLift() if you want "service mode"

finally: # if code fails it comes here (e.g. CTRL-C pressed)
————→# Make sure all GPIOs are left in a safe state
————→GPIO.cleanup()

————→# Remove all evidence of the lift
————→# This is so that you don't get lots of phantom lifts in the world!
————→# You can remove this if you always build your lift at same position
————→destroyLift()

# END
```