Part 03

Code an isometric adventure game: AmazeBalls

Pygame Zero in 3D. Let's make some baddies and dynamite in this last part of the series

MAKER

Mark Vanstone

Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!

magpi.cc/YiZnxL

@mindexplorers

We'll start from where we left off in the last part and add some extra elements to make a more challenging game. We're going to add some baddie balls that roam around the maze, pushing walls about – so even if you know how to get to the finish, you may find your path is blocked. To give our player an antidote to being blocked in, we'll add some dynamite for them to pick up and use.

01 Changing colours

Previously we had our ball bouncing around the maze by moving the drawing position of the map so that we are always viewing the area around the ball. Now we'll add some more balls, but these will be working against the player so we need to make them a different colour. We can do this quite easily with a paint app like GIMP. Just load each frame and use a tool called 'colorize' (in the Colors menu in GIMP). Make sure you save the frames as a different name; for example, put an 'e' for enemy in front of each file name.

02 Recycling code

We already have one ball bouncing around the maze – to get more balls, we'll try to reuse the code that we already have. We can duplicate the dictionary data at the top of our code that we have for the player and call it `enemy1` instead of `player`. You will want to change the `x` and `y` values in the data to something like 13, which will put the enemy ball near the middle of the maze. Now that we have our enemy defined, we can recycle some code.

03 Common update code

We can use exactly the same `updateBall()` function as we do for the player, and that will deal with all the animation and movement of the enemy from one block to the next. All we need to do is add another call to `updateBall()` after the one we have in our Pygame Zero `update()` function. But this time, rather than passing the player dictionary to the function, we pass the `enemy1` dictionary by writing `updateBall(enemy1)`. This means that if we set our enemy ball moving, all the changes to the data will be done in the same way as the player ball.

04 Drawing the enemy

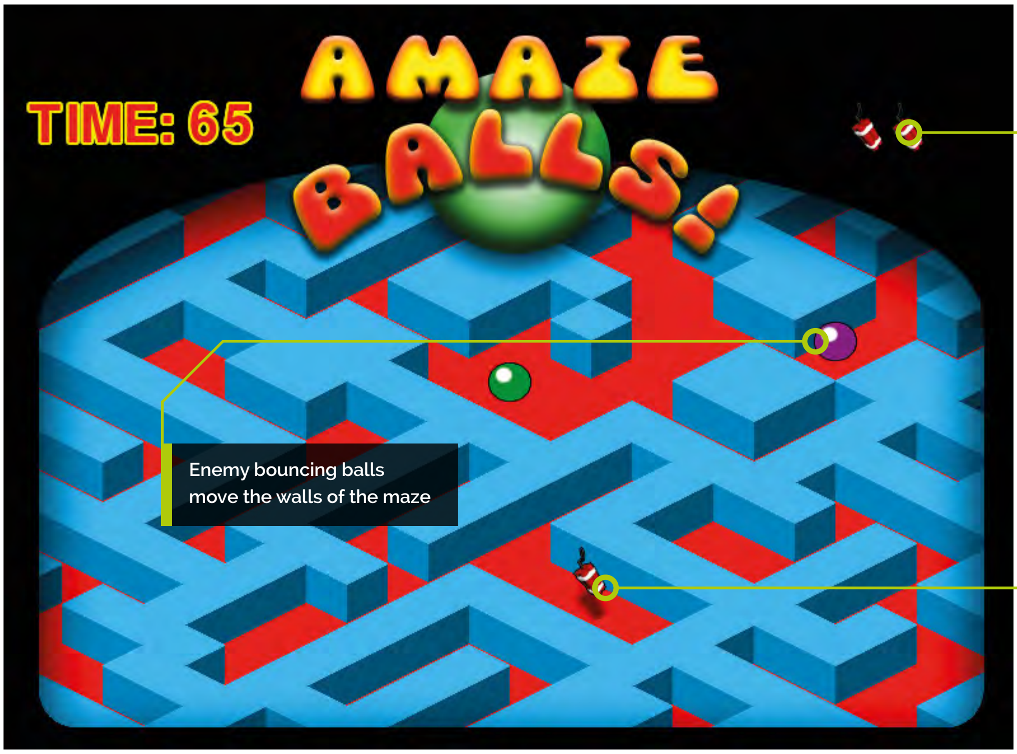
Although we now have a way to update the animation of our enemy ball, we also need to write some code to draw it on the screen. This needs to be a bit different than the player ball because the enemy ball needs to move as the map is scrolled. We have to use both the map position and the `sx` and `sy` values of the ball to work out where it needs to be drawn. See **figure1.py** for the updates to the `drawMap()` function. You will see we are calculating the block position (`bx` and `by`) and then adding the `sx` and `sy` values.

05 Enemy brains

If we run our program now, we should see an enemy ball bouncing away in the middle of the maze. You'll notice that both balls bounce at exactly the same time – if you wanted to have the bounces non-synchronised, you could change the

You'll Need

- Raspbian Jessie or newer
- Tiled (free map editor) **mapeditor.org**
- An image manipulation program such as GIMP, or images available from **magpi.cc/fPBrhM**
- The latest version of Pygame Zero (1.2)



Collected dynamite can be used to demolish walls that block the player

The player can pick up sticks of dynamite

Enemy bouncing balls move the walls of the maze

initial frame value where `enemy1` is declared at the top of the code. Now let's define a function called `updateEnemy()`; this will make the enemy ball move and also push some walls around.

06 Getting random

We are going to get the enemy to move around in a random way so, as we have done previously in this series, let's use the random module to generate some random movement. At the top of our program we import the module with `from random import randint`. Let's define our `updateEnemy()` function as `def updateEnemy(e):`. The `e` variable is the enemy dictionary that we will pass into the function when we call it. Now let's define some directions. We can do this with a list of x and y directions; for example, if we had x and y written as `[0,1]`, that would mean move no blocks in the x direction and one block in the y direction.

07 One direction

So, we can define all four directions as `edirs = [[-1,0],[0,1],[1,0],[0,-1]]`. And then all we need to do is pick one of them with a random number. To choose a random integer between 0 and 3, we write `r = randint(0,3)`. Now we can reuse

▼ Changes to `drawMap()` to incorporate the enemy ball

figure1.py

▶ Language: **Python 3**

```python
def drawMap():
    psx = OFFSETX
    psy = OFFSETY-32
    mx = psx - player["sx"]
    my = psy - player["sy"]+32

    for x in range(player["x"]-12, player["x"]+16):
        for y in range(player["y"]-12, player["y"]+16):
            if onMap(x,y):
                b = mapData["data"][y][x]
                td = findData(mapData["tiles"], "id", b)
                block = td["image"]
                bheight =  td["imageheight"]-34
                bx = (x*32)-(y*32) + mx
                by = (y*16)+(x*16) + my
                if -32 <= bx < 800 and 100 <= by < 620:
                    screen.blit(block, (bx, by - bheight))
                if x == player["x"] and y == player["y"]:
                    screen.blit("ball"+str(player["frame"]),
(psx, psy))
                if x == enemy1["x"] and y == enemy1["y"]:
                    screen.blit("eball"+str(enemy1[
"frame"]),(bx + enemy1["sx"],(by-32)+enemy1["sy"]))
```

figure2.py

Language: **Python 3**

```python
def doMove(p, x, y):
    global mazeSolved
    if onMap(p["x"]+x, p["y"]+y):
        mt = mapData["data"][p["y"]+y][p["x"]+x]
        if mt == 1 or mt == 3:
            p.update({"queueX":x, "queueY":y,
"moveDone":False})
            if mt == 3 and p == player:
                mazeSolved = True
        return mt
```

Changes to `doMove()` to make sure that the enemy ball doesn't trigger the finish condition

Top Tip

Dynamic map data

You can change any block on the map by changing the `id` in `mapData`. You could have lots of fun animating map elements in the `update()` function.

There are several ways of creating images for games. This dynamite was created in a free 3D modelling program called Blender

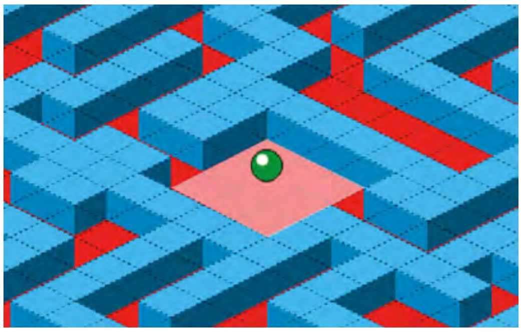
the `doMove()` function that moves the player, but use it for our enemy. We do need to make a couple of alterations to the `doMove()` function. The first is to return the `id` of the block that is being moved to. Then we can detect if a wall block is in the way; if it is, we can really mess things up for the player by moving the blocks around – sneaky, huh?

08 Making a move

The other problem that we have with the `doMove()` function is that it detects if the player has landed on the finish and if we use the same code for the enemy, we may get the finished condition triggered by the enemy instead of the player, so we need to change the `mazeSolved` condition to include a test to see if it's the player we are dealing with. Have a look at **figure2.py** to see these two changes to the `doMove()` function. When we've made those changes, we can go back to our `updateEnemy()` function.

09 Dynamic blocks

So far, all the blocks have stayed in the same place, but because they're all represented as numbers in our map data, we can change the numbers in the data and we'll see the maze change on screen. For example, if a block has `id` 2, it is a wall block. If we change the data to say that it is `id` 1, we'll see the wall disappear and there will be a floor block in its place. So, we can make changes to the maze as the enemy ball bounces around.


When the dynamite is used, it changes all the blocks around the player (shown in pink here) to floor blocks

10 Moving the walls

So, going back to our `updateEnemy()` function, we need to first generate a random number for our direction and then move the enemy ball in that direction. But if it moves towards a wall, then we attempt to move that wall block in the direction the enemy ball is moving. The other thing we need to check is that there is a space for the wall to move into. We need to make a call to `doMove()` using the `enemy1` dictionary (passed into `updateEnemy()`) and then, if the block `id` is 2 (a wall), call another function called `moveBlock()`.

11 Changing places

We need to define the `moveBlock()` function and we will pass it the x and y block position in the map data and also the direction values that we are using to move the enemy ball. First, the function will check that we are moving data within the map area and then it will check that the block we are moving the wall to is a floor block (`id` 1). If this all checks out, then we copy the `id` of the block we are moving to the new position. Have a look at **figure3.py** to see the `updateEnemy()` function and the `moveBlock()` function.

12 What are we doing?

The **figure3.py** code may look a bit daunting in places, so let's have a look at the detail. The `updateEnemy()` function is basically defining four directions to move in and then we are saying: if the enemy is not currently moving, then get a random direction and pass the x and y values of that direction to the `doMove()` function. If the block we are moving to is `id` 2, then move the block using the location we are moving to and the direction values. Then zero the screen x and y coordinates (`sx` and `sy`) of the enemy dictionary.

13 It's all relative

You will notice that if the enemy is moving, then we check to see if we are on frame seven (when the ball actually moves from one block to another in the data) and if so, we fix up the coordinates so they are now relative to the new map location rather than the old one. The `moveBlock()` function just checks directly with the `mapData` data to check that the block can be moved, moves the data from the source location to the target location, and sets the source location to be a floor block.

14 Multiple enemies!

When all that is done, we just need to add `updateEnemy(enemy1)` after our `updateBall()` calls in the Pygame Zero `update()` function. Now, it may be that we consider that one enemy ball is not enough to make the game interesting and to make a second one is very easy now. We just need to duplicate the `enemy1` dictionary and call it `enemy2`, change the starting `x` and `y` to perhaps 25, make calls to `updateBall(enemy2)` and `updateEnemy(enemy2)` in the `update()` function and before you know it you have a second enemy ball. You could make as many as you like, or maybe put them in a list to be more efficient if there are more than three.

15 A bit one-sided

Now that we have our baddies messing up our maze, it's going to get pretty difficult for our player to get through to the finish, so it's time to level the playing field, in this case quite literally. Let's introduce some dynamite into the mix! We'll need to make a tile graphic for the dynamite that we can use in the Tiled map editor, and also an icon that we can use to show how many sticks of dynamite the player has collected. If you want to use ready-made graphics and map data, they are available from the GitHub repo: **magpi.cc/NvafjA**.

16 Handling the explosives

First, let's add a variable to hold the number of sticks of dynamite being held by the player, which can be done by writing `"dynamite":0` as part of the `player` dictionary. Then, assuming that we

figure3.py

Language: **Python 3**

```python
def updateEnemy(e):
    edirs = [[-1,0],[0,1],[1,0],[0,-1]]
    if e["moveX"] == 0 and e["moveY"] == 0:
        r = randint(0,3)
        if doMove(e, edirs[r][0], edirs[r][1]) == 2:
            moveBlock(e["x"]+edirs[r][0],e["y"]+edirs[r]
[1],edirs[r][0],edirs[r][1])
        e["sx"] = e["sy"] = 0
    else:
        if e["frame"] == 7 and e["movingNow"] == True:
            if e["sx"] == 12: e["sx"] -= 32
            if e["sx"] == -12: e["sx"] += 32
            if e["sy"] == 6: e["sy"] -= 16
            if e["sy"] == -6: e["sy"] += 16

def moveBlock(mx,my,dx,dy):
    if onMap(mx+dx,my+dy):
        d = mapData["data"][my+dy][mx+dx]
        if d == 1:
            mapData["data"][my+dy][mx+dx] =
mapData["data"][my][mx]
            mapData["data"][my][mx] = 1
```

▲ Updating the enemy ball and moving blocks if walls are in the way

▼ The updated `update()` function to include two enemy balls

figure4.py

Language: **Python 3**

```python
def update(): # Pygame Zero update function
    global player, timer
    mt = 0
    if player["moveDone"] == True:
        if keyboard.left: mt = doMove(player, -1, 0)
        if keyboard.right: mt = doMove(player, 1, 0)
        if keyboard.up: mt = doMove(player, 0, -1)
        if keyboard.down: mt = doMove(player, 0, 1)
    if mt == 4:
        mapData["data"][ player["y"] + player["queueY"]][
player["x"] + player["queueX"]] = 1
        player["dynamite"] += 1
    updateBall(player)
    updateBall(enemy1)
    updateBall(enemy2)
    updateEnemy(enemy1)
    updateEnemy(enemy2)
```

have added some dynamite to our map data (see the previous part of this series for details on editing with Tiled), we need to detect if our player has moved onto a dynamite block and, if so, add `1` to our `dynamite` count and make the dynamite block into a floor block, which will make it disappear from the map. We can do this in our `update()` function.

17 Stockpiling ammo

To handle the picking up, we just need to test the value of `mt` after our keyboard checks. See **figure4.py** to view the revised `update()` function. When our player has picked up some dynamite, we can display the number held with icons, as we have done before (for example with lives), in the `draw()` function by writing `for l in range(player["dynamite"]): screen.blit("dmicon", (650+(l*32),80))`, which will draw our dynamite icons in the top right of the screen. So now we have the ammunition for our player to blow a path through the blockages that the baddies have put in the way.

18 Going off with a bang

All we have left to do now is to code a mechanism to set off the dynamite. We will do this with the Pygame Zero `on_key_down()` function. We need to test if the **SPACE** bar has been pressed and, if so, clear a space around the player, making all the blocks into floor blocks. This can be done with a nested `for` loop. Have a look at the full **amazeballs3.py** listing to see this last bit of code.

Now is the time to test how the game is set up, is it too easy or too hard? Do you need more or fewer enemies? You could try making a range of different maps with other objects to collect.

19 And finally

Well, sadly that's all we have time for in this series. We hope you have learned a lot about Pygame Zero and writing games in Python. We must at this stage give a big shout out to the creator of Pygame Zero, Daniel Pope: without his excellent work, this series would not have existed. We hope you would agree that the Pygame Zero framework is an ideal starting place to learn game coding on the Raspberry Pi.

amazeballs3.py

Language: **Python**

```python
import pgzrun
import map3d
from random import randint

player = {"x":3, "y":3, "frame":0, "sx":0, "sy":96,
          "moveX":0, "moveY":0, "queueX":0, "queueY":0,
          "moveDone":True, "movingNow":False, "animCounter":0, "dynamite":0}
enemy1 = {"x":13, "y":13, "frame":0, "sx":0, "sy":0,
          "moveX":0, "moveY":0, "queueX":0, "queueY":0,
          "moveDone":True, "movingNow":False, "animCounter":0}
enemy2 = {"x":25, "y":25, "frame":0, "sx":0, "sy":0,
          "moveX":0, "moveY":0, "queueX":0, "queueY":0,
          "moveDone":True, "movingNow":False, "animCounter":0}
OFFSETX = 368
OFFSETY = 300
timer = 0
mazeSolved = False

mapData = map3d.loadmap("maps/map1.json")

def draw(): # Pygame Zero draw function
    screen.fill((0, 0, 0))
    drawMap()
    screen.blit('title', (0, 0))
    screen.draw.text("TIME: "+str(timer) , topleft=(20, 80), owidth=0.5, ocolor=(255,255,0), color=(255,0,0) , fontsize=60)
    for l in range(player["dynamite"]): screen.blit("dmicon", (650+(l*32),80))
    if mazeSolved:
        screen.draw.text("MAZE SOLVED in " + str(timer) + " seconds!" , center=(400, 450), owidth=0.5, ocolor=(0,0,0), color=(0,255,0) , fontsize=60)


def update(): # Pygame Zero update function
    global player, timer
    mt = 0
    if player["moveDone"] == True:
        if keyboard.left: mt = doMove(player, -1, 0)
        if keyboard.right: mt = doMove(player, 1, 0)
        if keyboard.up: mt = doMove(player, 0, -1)
        if keyboard.down: mt = doMove(player, 0, 1)
    if mt == 4:
        mapData["data"][ player["y"] + player["queueY"]][player["x"] + player["queueX"]] = 1
        player["dynamite"] += 1
    updateBall(player)
    updateBall(enemy1)
    updateBall(enemy2)
    updateEnemy(enemy1)
    updateEnemy(enemy2)

def on_key_down(key):
    if player["dynamite"] > 0 and key.name == "SPACE":
        player["dynamite"] -= 1
        for x in range(player["x"]-1, player["x"]+2):
```

DOWNLOAD THE FULL CODE:

magpi.cc/ODgUCg

```python
            for y in range(player["y"]-1, player["y"]+2):
                mapData["data"][y][x] = 1

def timerTick():
    global timer
    if not mazeSolved:
        timer += 1

def drawMap():
    psx = OFFSETX
    psy = OFFSETY-32
    mx = psx - player["sx"]
    my = psy - player["sy"]+32

    for x in range(player["x"]-12, player["x"]+16):
        for y in range(player["y"]-12, player["y"]+16):
            if onMap(x,y):
                b = mapData["data"][y][x]
                td = findData(mapData["tiles"], "id", b)
                block = td["image"]
                bheight =  td["imageheight"]-34
                bx = (x*32)-(y*32) + mx
                by = (y*16)+(x*16) + my
                if -32 <= bx < 800 and 100 <= by < 620:
                    screen.blit(block, (bx, by - bheight))
                if x == player["x"] and y == player["y"]:
                    screen.blit("ball"+str(player["frame"]), (psx, psy))
                if x == enemy1["x"] and y == enemy1["y"]:
                    screen.blit("eball"+str(enemy1["frame"]), (bx + enemy1["sx"], (by-32)+enemy1["sy"]))
                if x == enemy2["x"] and y == enemy2["y"]:
                    screen.blit("eball"+str(enemy2["frame"]), (bx + enemy2["sx"], (by-32)+enemy2["sy"]))

def findData(lst, key, value):
    for i, dic in enumerate(lst):
        if dic[key] == value:
            return dic
    return -1

def onMap(x,y):
    if 0 <= x < mapData["width"] and 0 <= y < mapData["height"]:
        return True
    return False

def doMove(p, x, y):
    global mazeSolved
    if onMap(p["x"]+x, p["y"]+y):
        mt = mapData["data"][p["y"]+y][p["x"]+x]
        if mt == 1 or mt == 3 or mt == 4:
            p.update({"queueX":x, "queueY":y, "moveDone":False})
            if mt == 3 and p == player:
                mazeSolved = True
        return mt

def updateEnemy(e):
    edirs = [[-1,0],[0,1],[1,0],[0,-1]]
    if e["moveX"] == 0 and e["moveY"] == 0:
        r = randint(0,3)
        if doMove(e, edirs[r][0], edirs[r][1]) == 2:
            moveBlock(e["x"]+edirs[r][0],e["y"]+edirs[r][1],edirs[r][0],edirs[r][1])
        e["sx"] = e["sy"] = 0
    else:
        if e["frame"] == 7 and e["movingNow"] == True:
            if e["sx"] == 12: e["sx"] -= 32
            if e["sx"] == -12: e["sx"] += 32
            if e["sy"] == 6: e["sy"] -= 16
            if e["sy"] == -6: e["sy"] += 16

def moveBlock(mx,my,dx,dy):
    if onMap(mx+dx,my+dy):
        d = mapData["data"][my+dy][mx+dx]
        if d == 1:
            mapData["data"][my+dy][mx+dx] = mapData["data"][my][mx]
            mapData["data"][my][mx] = 1

def updateBall(p):
    if p["movingNow"]:
        if p["moveX"] == -1: moveP(p,-1,-0.5)
        if p["moveX"] == 1: moveP(p,1,0.5)
        if p["moveY"] == -1: moveP(p,1,-0.5)
        if p["moveY"] == 1: moveP(p,-1,0.5)
    p["animCounter"] += 1
    if p["animCounter"] == 4:
        p["animCounter"] = 0
        p["frame"] += 1
        if p["frame"] > 7:
            p["frame"] = 0
        if p["frame"] == 4:
            if p["moveDone"] == False:
                if p["queueX"] != 0 or p["queueY"] !=0:
                    p.update({"moveX":p["queueX"], "moveY":p["queueY"], "queueX":0, "queueY":0, "movingNow": True})
            else:
                p.update({"moveDone":True, "moveX":0, "moveY":0, "movingNow":False})

        if p["frame"] == 7 and p["moveDone"] == False and p["movingNow"] == True:
            p["x"] += p["moveX"]
            p["y"] += p["moveY"]
            p["moveDone"] = True

def moveP(p,x,y):
    p["sx"] += x
    p["sy"] += y

clock.schedule_interval(timerTick, 1.0)
pgzrun.go()
```