Part 07

Code your own Pac-Man game: part 2

Pac-Man captured the hearts and pocket money of many young people in the eighties. In part two of our guide, we add some groovy features to the basic game created last month

MAKER

Mark Vanstone

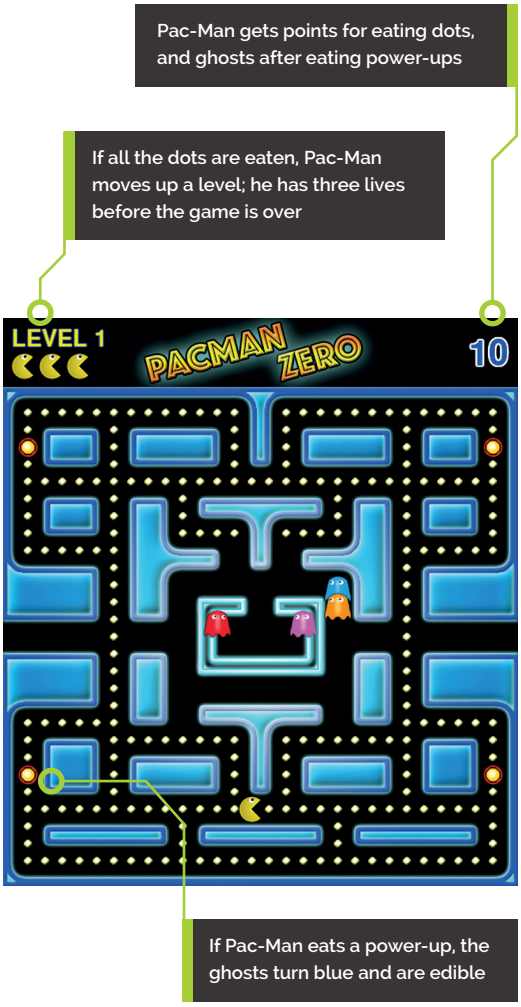
Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!

magpi.cc/YiZnxl

@mindexplorers

You'll Need

- Raspbian Jessie or newer
- An image manipulation program such as GIMP, or images available from **magpi.cc/nBSXKz**
- The latest version of Pygame Zero (1.2)
- USB joystick or gamepad (optional)
- Headphones or speakers



In part one, we created a maze for our player to move around, and restricted movement to just the corridors. We provided some dots to eat and some ghosts to avoid. In this part we are going to give the ghosts some more brains so that they are a bit more challenging to the player. We will also add the bonus power-ups which turn the ghosts into tasty edibles, give Pac-Man some extra levels to explore and some extra lives. So far in this series we have not dealt with music, so we will have a go at putting some music and sound effects into the game.

01 Need more brains

In part one, we left our ghosts wandering around the maze randomly without much thought for what they were doing, which was a bit unfair as Pac-Man could evade them without too much trouble. In the original game, each ghost had a program that it followed to characterise its movements. We are going to add some brains to two of the ghosts. The first we will make follow Pac-Man, and the second we will get to ambush by moving ahead of Pac-Man. We will still leave in some random movement, otherwise it may get a bit too difficult.

02 Follow the leader

First, let's get the red ghost to follow Pac-Man. We already have a `moveGhosts()` function

Adding a brain to a ghost to follow the player

from part one and we can add a condition to see if we are dealing with the first ghost: `if g == 0: followPlayer(g, dirs)`. This calls `followPlayer()` if it's the first ghost. The `followPlayer()` function receives a list of directions that the ghost can move in. It then tests the x coordinate of the player against the x coordinate of the ghost and, if the direction is valid, sets the ghost direction to move toward the player. Then it does the same with the y coordinates.

03 Y over x

The keen-witted among you will have noticed that if x and y movements towards the player are both valid, then the y direction will always win. We could throw in another random number to choose between the two, but in testing this arrangement it doesn't cause any significant problem with the movement. See **figure1.py** for the `followPlayer()` function. You will see there is a special condition `aboveCentre()` when we check the downward movement. We are checking that the ghost is not just above the centre, otherwise it will go back into its starting enclosure.

04 The central problem

If we go back to the `moveGhosts()` function, we need another centre-related condition: `if inTheCentre(ghosts[g])`. This is because if we leave the ghost to randomly move around our centre enclosure, it may take a long time to get out. In part one, you may have noticed that from time to time one ghost would get stuck in the centre. What we do is, if we detect that a ghost is in the centre, we always default to direction 3, which is up. If we run the game with this condition and the `followPlayer()` function, we should see all the ghosts making their way straight out of the centre and then the red ghost making a bee-line towards Pac-Man.

figure1.py

```
001. def followPlayer(g, dirs):
002.     d = ghosts[g].dir
003.     if d == 1 or d == 3:
004.         if player.x > ghosts[g].x and dirs[0] == 1:
005.             ghosts[g].dir = 0
006.         if player.x < ghosts[g].x and dirs[2] == 1:
007.             ghosts[g].dir = 2
008.     if d == 0 or d == 2:
009.         if player.y > ghosts[g].y and dirs[1] == 1 and not
    aboveCentre(ghosts[g]): ghosts[g].dir = 1
010.         if player.y < ghosts[g].y and dirs[3] == 1:
011.             ghosts[g].dir = 3
012.
013.
014. def aboveCentre(ga):
015.     if ga.x > 220 and ga.x < 380 and ga.y > 300 and ga.y
    < 320:
016.         return True
017.     return False
```

figure2.py

```
001. # This code goes in the update() function
002.
003.     if player.status == 1:
004.         i = gameinput.checkInput(player)
005.         if i == 1:
006.             player.status = 0
007.             player.x = 290
008.             player.y = 570
009.
010. # This code goes in the gameinput module
011. # in the checkInput() function
012.
013.     if joystick_count > 0:
014.         jb = joyin.get_button(1)
015.     else:
016.         jb = 0
017.     if p.status == 1:
018.         if key.get_pressed()[K_RETURN] or jb:
019.             return 1
```

05 It's an ambush!

So, the next brain to implant is for the second ghost. We will add a function `ambushPlayer()` in the same way we did for the first ghost, but this time `if g == 1:`. The `ambushPlayer()` function works very much like the `followPlayer()` function, but this time we just check the direction that Pac-Man is currently moving and try to move in that direction. We, of course, cannot know which direction the player is going to move, and this may seem a bit of a simplistic approach to ambushing the player, but it is surprising how many times Pac-Man ends up wedged between these two ghosts with this method.

06 Scores on the doors

Brain functions could be added to all the ghosts, but we are going to leave the ghost brains for now as there is plenty more to do to get our

▲ Checking to see if **ENTER** or button A has been pressed and resetting the player actor

▼ Pac-Man gets three lives. You can use the gamepad or joystick buttons to ask for input from the player

game completed. Before we go any further, we ought to get a scoring system going and reward Pac-Man for all the dots eaten. We can attach the score variable to the player actor near the top of our code with `player.score = 0` and then each time a dot is eaten we add 10 to the score with `player.score += 10`. We can also display the score in the `draw()` function (probably top right is best) with `screen.draw.text()`.

07 Three strikes and you're out!

As is the tradition in arcade games, you get three lives before it's game over. If you followed our previous tutorial for Space Invaders, you will know how we do this. We just add a lives variable to the player actor and then each time Pac-Man is caught by a ghost, we take a life off, set `player.status = 1`, and print a message to say press **ENTER**. When pressed, we set `player.status = 0` and send Pac-Man back to the starting place. Then we continue. Have a look at **figure2.py** to see the code we add to reset Pac-Man to the start.

08 Printing lives

We have the system for keeping track of the `player.lives` variable, but we also need to show the player how many lives they have left. We can do this with a simple loop like we used in the previous Space Invaders tutorial. We can have a `drawLives()` function which we call from our `draw()` function. In that function, we go round a loop for the number of lives we have by saying `for l in range(player.lives):` and then we can use the same image that we use for the player and say `screen.blit("pacman_o", (10+(l*32),40))`.

09 Which button to press

You may notice in **figure2.py** that in our gameinput module we are checking a joystick button as well as the **ENTER** key. You may want to do a few tests with the gamepads or joysticks that you're using, as the buttons may have different numbers. You can also prompt the player to press (in this case) the A button to continue. If you were designing a game that relied on several buttons being used, you might want to set up a way of mapping the buttons to values depending on what type of gamepad or joystick is being used.

figure3.py

```
001. # This code is in our main code file (pacman2.py)
002.
003. def initDots():
004.     global pacDots
005.     pacDots = []
006.     a = x = 0
007.     while x < 30:
008.         y = 0
009.         while y < 29:
010.             d = gamemaps.checkDotPoint(10+x*20, 10+y*20)
011.             if d == 1:
012.                 pacDots.append(Actor("dot",(10+x*20,
    90+y*20)))
013.                 pacDots[a].status = 0
014.                 pacDots[a].type = 1
015.                 a += 1
016.             if d == 2:
017.                 pacDots.append(Actor("power",(10+x*20,
    90+y*20)))
018.                 pacDots[a].status = 0
019.                 pacDots[a].type = 2
020.                 a += 1
021.             y += 1
022.         x += 1
023.
024. # This code is in the gamemaps module
025.
026. def checkDotPoint(x,y):
027.     global dotimage
028.     if dotimage.get_at((int(x), int(y))) ==
    Color('black'):
029.         return 1
030.     if dotimage.get_at((int(x), int(y))) == Color('red'):
031.         return 2
032.     return False
```

▲ Updated code to include the creation of power-ups

10 I have the power!

The next item on our list is power-ups. These are large glowing dots that, when eaten, turn all the ghosts dark blue. In their blue form they can be eaten for bonus points and they return to the centre of the maze. First, let's devise a way to place the power-ups in the maze. We have updated the **pacmandotmap.png** image to include some red squares, instead of black, in the positions where we want our power-ups to be. Then, when we initialise our dots and call `checkDotPoint(x,y)`, we look for red as well as black – **figure3.py** shows how we change our code to do this.

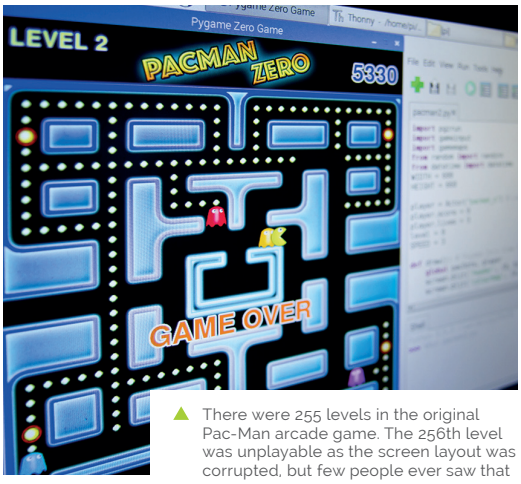
gamemaps.py

> Language: **Python 3**

```
001. from pygame import image, surface, Color
002. moveimage = image.load('images/pacmanmovemap.png')
003. dotimage = image.load('images/pacmandotmap.png')
004.
005. def checkMovePoint(p):
006.     global moveimage
007.     if p.x+p.movex < 0: p.x = p.x+600
008.     if p.x+p.movex > 600: p.x = p.x-600
009.     if moveimage.get_at((int(p.x+p.movex), int(p.y+
     p.movey-80))) != Color('black'):
010.         p.movex = p.movey = 0
011.
012. def checkDotPoint(x,y):
013.     global dotimage
014.     if dotimage.get_at((int(x), int(y))) ==
     Color('black'):
015.         return 1
016.     if dotimage.get_at((int(x), int(y))) ==
     Color('red'):
017.         return 2
018.     return False
019.
020. def getPossibleDirection(g):
021.     global moveimage
022.     if g.x-20 < 0:
023.         g.x = g.x+600
024.     if g.x+20 > 600:
025.         g.x = g.x-600
026.     directions = [0,0,0,0]
027.     if g.x+20 < 600:
028.         if moveimage.get_at((int(g.x+20),
     int(g.y-80))) == Color('black'): directions[0] = 1
029.     if g.x < 600 and g.x >= 0:
030.         if moveimage.get_at((int(g.x), int(g.y-60)))
     == Color('black'): directions[1] = 1
031.     if g.x-20 >= 0:
032.         if moveimage.get_at((int(g.x-20),
     int(g.y-80))) == Color('black'): directions[2] = 1
033.     if g.x < 600 and g.x >= 0:
034.         if moveimage.get_at((int(g.x), int(g.y-100)))
     == Color('black'): directions[3] = 1
035.     return directions
```

11 Not all dots are the same

We now have a system to place our power-ups in the maze. The next thing to do is to change what happens when Pac-Man eats a power-up compared to a normal dot. At the moment we just add ten points to the player's score if a dot is eaten, so we need to add more code to handle the event of a power-up being eaten. In the `draw()` function, where we look to see if the player has collided with a dot using `collidepoint()`, we then check the status of the dot (to make sure it's still there) and after this we can add a new condition: `if pacDots[a].type == 2:`.



▲ There were 255 levels in the original Pac-Man arcade game. The 256th level was unplayable as the screen layout was corrupted, but few people ever saw that

12 High status ghosts

As we have determined that we are dealing with a power-up (type 2), we can add a loop that goes through the list of ghosts and changes the status of the ghost. Normally the status for a ghost is 0. What we are going to do is change the status to a fairly high number (try 1200 to start with). This will indicate that the ghosts are in their alternate state and we will use the status as a countdown. We will decrement this value each time `update()` is called; when it reaches 0, the ghosts will turn back to normal.

13 Why so blue?

To make our ghost turn blue, we are going to add some conditions to our `drawGhosts()` function. We want them to be blue when the status is more than 0, but just to make it interesting we will make them flash when they are about to turn back. So we can write `if ghosts[g].status > 200 or (ghosts[g].status > 1 and ghosts[g].status%2 == 0): ghosts[g].image = "ghost5"`. What this is saying is that if the status is over 200 then make the ghost blue, but if it's less that 200 but greater than 1 then make it blue every other frame. We then have an `else` condition underneath that will set the image to its normal colour.

gameinput.py

Language: **Python 3**

```
001. from pygame import joystick, key
002. from pygame.locals import *
003.
004. joystick.init()
005. joystick_count = joystick.get_count()
006.
007. if(joystick_count > 0):
008.     joyin = joystick.Joystick(0)
009.     joyin.init()
010.
011. def checkInput(p):
012.     global joyin, joystick_count
013.     xaxis = yaxis = 0
014.     if p.status == 0:
015.         if joystick_count > 0:
016.             xaxis = joyin.get_axis(0)
017.             yaxis = joyin.get_axis(1)
018.         if key.get_pressed()[K_LEFT] or xaxis < -0.8:
019.             p.angle = 180
020.             p.movex = -20
021.         if key.get_pressed()[K_RIGHT] or xaxis > 0.8:
022.             p.angle = 0
023.             p.movex = 20
024.         if key.get_pressed()[K_UP] or yaxis < -0.8:
025.             p.angle = 90
026.             p.movey = -20
027.         if key.get_pressed()[K_DOWN] or yaxis > 0.8:
028.             p.angle = 270
029.             p.movey = 20
030.     if joystick_count > 0:
031.         jb = joyin.get_button(1)
032.     else:
033.         jb = 0
034.     if p.status == 1:
035.         if key.get_pressed()[K_RETURN] or jb:
036.             return 1
037.     if p.status == 2:
038.         if key.get_pressed()[K_RETURN] or jb:
039.             return 1
```

14 The tables have turned

Now we have our ghosts all turning blue when a power-up is eaten, we need to change what happens when Pac-Man collides with them. Instead of taking a life from the `player.lives` variable, we are going to add to the `player.score` variable and send the ghost back to the centre. So, the first job is to add a condition in `update()` when we check the ghost `collidepoint()` with the player, which would be `if ghosts[g].status > 0:`. We then add 100 to the `player.score` and `animate()` the ghost back to the centre. See **figure4.py** for the updated code.

15 Back to the start

You will notice that when Pac-Man comes into contact with a dark blue ghost, we just animate the actor straight back to the centre in the same time that we normally animate a ghost from one position to the next. This is so that we don't hold up the animation on the other ghosts waiting for the eaten one to get back to the centre. In the original game, the ghosts would turn into a pair of eyes and then make their way back to the centre along the corridors, but that would take too much extra code for this tutorial.

figure4.py

```
001. # This code is in the update() function
002.
003.         for g in range(len(ghosts)):
004.             if ghosts[g].status > 0: ghosts[g].status -= 1
005.             if ghosts[g].collidepoint((player.x,
     player.y)):
006.                 if ghosts[g].status > 0:
007.                     player.score += 100
008.                     animate(ghosts[g], pos=(290, 370),
     duration=1/SPEED, tween='linear',
     on_finished=flagMoveGhosts)
009.                 else:
010.                     player.lives -= 1
011.                     if player.lives == 0:
012.                         player.status = 3
013.                     else:
014.                         player.status = 1
```

▲ Updated ghost collision code to send them back to the centre if Pac-Man eats them

16 Time for some music

So far in this series, we have not covered adding music to games. In the documentation of Pygame Zero, music is labelled as experimental, so we will just have to try it out and see what happens. In the sample GitHub files for this

tutorial, there is a directory called **music** and in that directory is an MP3 file that we can use as eighties arcade game background music. To start our music, all we need to do is write `music.play("pm1")` in our `init()` function to start the **music/pm1.mp3** file. You may also want to set the volume with `music.set_volume(0.3)`.

17 More sound effects

The MP3 file will continue playing in a loop until we stop it, so when the game is over (`player.lives = 0`) we can fade the music out with `music.fadeout(3)`. At this stage we can also add some sound effects for when Pac-Man is eating dots. We have a sound in our **sounds** directory called **pac1.mp3** which we will use for this purpose and we can add a line of code just before we animate the player: `sounds.pac1.play()`. This will play the sound every time Pac-Man moves. We can do the same with **pac2.mp3** when a life is lost.

18 Level it up

The last thing we need to put into our game is to allow the player to progress to the next level when all the dots have been eaten. We could incorporate several things to make each level harder, but for the moment let's concentrate on resetting the screen and changing the level. If we define our level variable near the top of our code as `level = 0`, then inside our `init()` function we say `level += 1`, then each time we call `init()` we will increase our level variable. This means that instead of saying that the player has won, we just prompt them to continue, and call `init()` to reset everything and level up.

19 So much to do

The Pac-Man game has many more things that can be added to it. The original had bonus fruits to collect, the ghosts would move faster as the levels continued, there were animations between some of the levels, and the power-ups would run out quicker. You could add all of these things to this game, but we will have to leave you to do that yourself. Take a look into the history of the Pac-Man game – it's fascinating – and we will be starting a new Pygame Zero game in the next instalment of this series.

pacman2.py

> Language: **Python 3**

```
001. import pgzrun
002. import gameinput
003. import gamemaps
004. from random import randint
005. from datetime import datetime
006. WIDTH = 600
007. HEIGHT = 660
008.
009. player = Actor("pacman_o") # Load in the player Actor image
010. player.score = 0
011. player.lives = 3
012. level = 0
013. SPEED = 3
014.
015. def draw(): # Pygame Zero draw function
016.     global pacDots, player
017.     screen.blit('header', (0, 0))
018.     screen.blit('colourmap', (0, 80))
019.     pacDotsLeft = 0
020.     for a in range(len(pacDots)):
021.         if pacDots[a].status == 0:
022.             pacDots[a].draw()
023.             pacDotsLeft += 1
024.         if pacDots[a].collidepoint((player.x, player.y)):
025.             if pacDots[a].status == 0:
026.                 if pacDots[a].type == 2:
027.                     for g in range(len(ghosts)): ghosts[g].status = 1200
028.                 else:
029.                     player.score += 10
030.             pacDots[a].status = 1
031.     if pacDotsLeft == 0: player.status = 2
032.     drawGhosts()
033.     getPlayerImage()
034.     player.draw()
035.     drawLives()
036.     screen.draw.text("LEVEL "+str(level) , topleft=(10, 10), owidth=0.5,
    ocolor=(0,0,255), color=(255,255,0) , fontsize=40)
037.     screen.draw.text(str(player.score) , topright=(590, 20), owidth=0.5,
    ocolor=(255,255,255), color=(0,64,255) , fontsize=60)
038.     if player.status == 3: drawCentreText("GAME OVER")
039.     if player.status == 2: drawCentreText(
    "LEVEL CLEARED!\nPress Enter or Button A\nto Continue")
040.     if player.status == 1: drawCentreText(
    "CAUGHT!\nPress Enter or Button A\nto Continue")
041.
042. def drawCentreText(t):
043.     screen.draw.text(t , center=(300, 434), owidth=0.5,
    ocolor=(255,255,255), color=(255,64,0) , fontsize=60)
044.
045. def update(): # Pygame Zero update function
046.     global player, moveGhostsFlag, ghosts
```

DOWNLOAD THE FULL CODE:
magpi.cc/TDtRaV

```
    if player.status == 0:
        if moveGhostsFlag == 4: moveGhosts()
        for g in range(len(ghosts)):
            if ghosts[g].status > 0: ghosts[g].status -= 1
            if ghosts[g].collidepoint((player.x,
player.y)):
                if ghosts[g].status > 0:
                    player.score += 100
                    animate(ghosts[g], pos=(290, 370),
duration=1/SPEED, tween='linear',
on_finished=flagMoveGhosts)
                else:
                    player.lives -= 1
                    sounds.pac2.play()
                    if player.lives == 0:
                        player.status = 3
                        music.fadeout(3)
                    else:
                        player.status = 1
        if player.inputActive:
            gameinput.checkInput(player)
            gamemaps.checkMovePoint(player)
            if player.movex or player.movey:
                inputLock()
                sounds.pac1.play()
                animate(player, pos=(player.x + player.
movex, player.y + player.movey), duration=1/SPEED,
tween='linear', on_finished=inputUnLock)
    if player.status == 1:
        i = gameinput.checkInput(player)
        if i == 1:
            player.status = 0
            player.x = 290
            player.y = 570
    if player.status == 2:
        i = gameinput.checkInput(player)
        if i == 1:
            init()

def init():
    global player, level
    initDots()
    initGhosts()
    player.x = 290
    player.y = 570
    player.status = 0
    inputUnLock()
    level += 1
    music.play("pm1")
    music.set_volume(0.2)

def drawLives():
    for l in range(player.lives): screen.blit("pacman_o",
(10+(l*32),40))

def getPlayerImage():
    global player
    dt = datetime.now()
    a = player.angle
    tc = dt.microsecond%(500000/SPEED)/(100000/SPEED)
    if tc > 2.5 and (player.movex != 0 or player.movey
!=0):
        if a != 180:
            player.image = "pacman_c"
        else:
            player.image = "pacman_cr"
    else:
        if a != 180:
            player.image = "pacman_o"
        else:
            player.image = "pacman_or"
    player.angle = a

def drawGhosts():
    for g in range(len(ghosts)):
        if ghosts[g].x > player.x:
            if ghosts[g].status > 200 or (ghosts[g].status
> 1 and ghosts[g].status%2 == 0):
                ghosts[g].image = "ghost5"
            else:
                ghosts[g].image = "ghost"+str(g+1)+"r"
        else:
            if ghosts[g].status > 200 or (ghosts[g].status
> 1 and ghosts[g].status%2 == 0):
                ghosts[g].image = "ghost5"
            else:
                ghosts[g].image = "ghost"+str(g+1)
        ghosts[g].draw()

def moveGhosts():
    global moveGhostsFlag
    dmoves = [(1,0),(0,1),(-1,0),(0,-1)]
    moveGhostsFlag = 0
    for g in range(len(ghosts)):
        dirs = gamemaps.getPossibleDirection(ghosts[g])
        if inTheCentre(ghosts[g]):
            ghosts[g].dir = 3
        else:
            if g == 0: followPlayer(g, dirs)
            if g == 1: ambushPlayer(g, dirs)

        if dirs[ghosts[g].dir] == 0 or randint(0,50) == 0:
```

```
            d = -1
            while d == -1:
                rd = randint(0,3)
                if aboveCentre(ghosts[g]) and rd == 1:
                    rd = 0
                if dirs[rd] == 1:
                    d = rd
            ghosts[g].dir = d
        animate(ghosts[g], pos=(ghosts[g].x
+ dmoves[ghosts[g].dir][0]*20, ghosts[g].y +
dmoves[ghosts[g].dir][1]*20), duration=1/SPEED,
tween='linear', on_finished=flagMoveGhosts)

def followPlayer(g, dirs):
    d = ghosts[g].dir
    if d == 1 or d == 3:
        if player.x > ghosts[g].x and dirs[0] == 1:
ghosts[g].dir = 0
        if player.x < ghosts[g].x and dirs[2] == 1:
ghosts[g].dir = 2
    if d == 0 or d == 2:
        if player.y > ghosts[g].y and dirs[1] == 1 and not
aboveCentre(ghosts[g]): ghosts[g].dir = 1
        if player.y < ghosts[g].y and dirs[3] == 1:
ghosts[g].dir = 3


def ambushPlayer(g, dirs):
    d = ghosts[g].dir
    if player.movex > 0 and dirs[0] == 1: ghosts[g].dir = 0
    if player.movex < 0 and dirs[2] == 1: ghosts[g].dir = 2

    if player.movey > 0 and dirs[1] == 1 and not
aboveCentre(ghosts[g]): ghosts[g].dir = 1
    if player.movey < 0 and dirs[3] == 1: ghosts[g].dir = 3

def inTheCentre(ga):
    if ga.x > 220 and ga.x < 380 and ga.y > 320 and ga.y <
420:
        return True
    return False

def aboveCentre(ga):
    if ga.x > 220 and ga.x < 380 and ga.y > 300 and ga.y <
320:
        return True
    return False

def flagMoveGhosts():
    global moveGhostsFlag
    moveGhostsFlag += 1

def ghostCollided(ga,gn):
    for g in range(len(ghosts)):
        if ghosts[g].colliderect(ga) and g != gn:
            return True
    return False

def initDots():
    global pacDots
    pacDots = []
    a = x = 0
    while x < 30:
        y = 0
        while y < 29:
            d = gamemaps.checkDotPoint(10+x*20, 10+y*20)
            if d == 1:
                pacDots.append(Actor("dot",(10+x*20,
90+y*20)))
                pacDots[a].status = 0
                pacDots[a].type = 1
                a += 1
            if d == 2:
                pacDots.append(Actor("power",(10+x*20,
90+y*20)))
                pacDots[a].status = 0
                pacDots[a].type = 2
                a += 1
            y += 1
        x += 1

def initGhosts():
    global ghosts, moveGhostsFlag
    moveGhostsFlag = 4
    ghosts = []
    g = 0
    while g < 4:
        ghosts.append(Actor("ghost"+str(g+1),(270+(g*20),
370)))
        ghosts[g].dir = randint(0, 3)
        ghosts[g].status = 0
        g += 1

def inputLock():
    global player
    player.inputActive = False

def inputUnLock():
    global player
    player.movex = player.movey = 0
    player.inputActive = True

init()
pgzrun.go()
```