Okay so basically the "state" of the game is defined by...

oh, so by default we split up the board into like, a 12x12 grid, but any grid works really

1. The height of the paddle (defined by which of the 12 vertical "rows" it is in)

2. The y direction of the ball's movement, either 0, 1, or 2 for... "It's moving down a fair amount", "It's basically moving horizontal", and "It's moving up a fair amount"...  We wrote functions to determine which of these was true based on cutoffs

3. Whether the ball was moving left or right

4,5:  Which of the 12x12 boxes the ball was currently in

So at any given moment in time we check like... Call function (very simple) to get the height of the paddle 0-11... Call function (very simple) to return if the ball  is moving up,down, or basically horizontal....

The function was literally

def getymotind(ymot):  
       if (ymot < -.015):        # up  
               return 0  
       elif (ymot > .015):     #down  
               return 2  
       else:  
               return 1          #horiz

Get if the ball was moving left or right... etc

At the start of each time step we do

ballxindex = getind(ballx)  
               ballyindex = getind(bally)  
               xmotindex = getxmotind(xmot)  
               ymotindex = getymotind(ymot)  
               paddleindex = getpaddleind(paddley)

Yknow.... and all those functions are real simple

Anyway then we use those as indices in our 5D array, and that is the state of the game.... nah we never printed anything out really I don't think

ourstate = state[ballyindex][ballxindex][xmotindex][ymotindex][paddleindex] # the state we're in to begin with

Or yknow... however you wanna define your variables... whatever order

Anyway then we had like a 3% chance to move the paddle randomly... otherwise we chose how to move the paddle ourself

Either up, down, or stay

motion = getnextmotion(ourstate)

and so that function did a check for like... "have we tried this move from this state enough, or have we not done enough trials?"

possible = [0,0,0]  
  
if (state.counts[0] < 100):   # if we havent tried enough times  
        possible[0] = 2  
else:  
        possible[0] = state.moves[0]

If we chose a move, we'd update the state array of that state with like... so say we move "down"

We basically do ourstate.counts[2]++, when we move to track "hey we moved down (2), another time"

so if any of those counts was below 100 it meant we hadn't even tried that move 100 times yet, so we wanna choose that one (set it to some high value like 2 when looking for our next move)

Otherwise if each direction has been tried a decent amount from this state, we choose based on the actual "value" of the move

return possible.index(max(possible))       # return the best move for us, accounting for needing lots of attempts

So whatever that function basically just compares the "goodness" of each move (goodness is stored in currentstate.moves) if they've been chosen enough times (stored in currentstate.counts), or just chooses something that hasnt been tried enough, and returns that direction

Yeah

0, 1, or 2 is how we kinda "Encoded" that

So you move the paddle accordingly... and there's some checks to make sure like... "if you're at the top and try to move up, you just stay at the top" and stuff... but you adjust the paddle height based on whatever direction you decided to move

Update the counts... update some alpha value... forget what alpha even is

ourstate.counts[motion] += 1  # update count of how many times we've taken action  
alpha = 75 / (75 + ourstate.counts[motion])   # adjust alpha based on how many times we've taken this move

Basically updating based on "we were in this state and moved UP" or something... and you update based on how many times you've done that

Then you move the ball in the X dimension... so like... the box is bounded from 0 to 1

so if the x location + x motion is between 0 and 1, it's in "free space" and you just let it move by updating its x location

if its motion would carry it above 1, it means it's exiting the play space on the "paddle side" and either needs to bounce off the paddle or get missed

 elif (ballx + xmot > 1):  
                       if (paddley <= bally <= paddley+.2):        # if it hit off the paddle  
                               #REBOUNDS OFF PADDLE  
  
                               paddlecheck = 1

So if it rebounds yknow... you do some math... you need to randomize its motion and speed a little and stuff

ballx = 2 - (ballx+xmot)  
  
randomizex = (random.randint(-15,15)) / 1000  
adjustedx = (-xmot) + randomizex  
xmot = min (-.03, adjustedx)  
xmot = max(-1, xmot)         # Adjust and constrain x motion shit

We chose random numbers between some bounds, then adjusted its speed by those numbers, and basically set a min and max value it could be (it's always a negative number since the ball must be moving left after it rebounds)

else:  
                                #DO SOMETHING FOR MISSING THE BALL  
                               #WE MISSED THE BALL!  
                                 
                               ourstate.moves[motion] = ourstate.moves[motion] + alpha\*(-1)  
  
                               #print ("MISSED")  
                               break

Otherwise we missed the ball... it didn't hit the paddle... we update our state's value (punish it for being shitty), and then break out of our "While true" loop... The ball has "Died" so we're no longer stepping through time in a While(1) thing

in that case we just reset everything back to its starting location and go again... move the ball and paddle back to start, reset the ball's motion, and start the While(1) back up

But anyway............ as long as we don't fuck up and "lose" you need to keep track of like... if it bounces off the back wall... if it bounces off the top or bottom... etc

set checks for those and make sure you rebound the ball

and FINALLY the actually important last piece at the end of every time step

Get all the "next" values for the next time step.,.. once you've moved the paddle, updated everything about the ball, maybe changed speeds if it bounced off something...  get all those "next" values for where the ball is gonna be at the start of the next time step

Get the next state you're gonna be in, and then update the current state based on how good the next state is

nextstate = state[nextballyindex][nextballxindex][nextxmotindex][nextymotindex][nextpaddleindex]  
ourstate.moves[motion] = ourstate.moves[motion] + (alpha \* ( paddlecheck + gamma\*(max(nextstate.moves)) - ourstate.moves[motion]))

So that first line there is getting the next state we're gonna be in

The second line does as follows

1. Take the current "quality" of the move we just made.

2.  Get the best possible "quality" for any move in the next state (how good can the next state possibly get?)  ----   max(nextstate.moves)

3. Multiply that by some gamma value... that value is predetermined... we used .7

4.  Subtract that   "How good was the move we just made"  from  the adjusted "How good can the next state even get?" value

Add whether or not we got a hit on this time step.  Getting a hit is very good so we want to "reward" it by adding 1 if we got a hit (paddlecheck)

Multiply that whooole thing by "Alpha" which is smaller depending on how long we've been training... I put the code for that somewhere above all this

And that's how you update the "quality" of the move you just made

So basically... moves which result in a rebound are rewarded first... then moves which result in you getting into a rebound state are rewarded.... then moves that lead to... that lead to... etc

That's it, really...

Basic program flow is like...

Declare a big state array with "Count" and "Quality" of moves for each state in each direction

We made a "State" class and then made a 5D array holding a bunch of those

Train it by doing what I described... initialize... being a While(1) loop that goes until you miss the ball.... update ball motion/position and paddle stuff as I described... do your math at the end of every time step to update the "quality" of the move you just made based on the "Quality" of the next state you'll end up in

Eventually the Big-ass array will fill up with all sorts of numbers

Then run your "tests" on that big filled up array and see how good you do

DURING EACH TIME STEP

1. Determine how you're gonna move the paddle... during training there's some randomness to this

1. Also includes actually moving the paddle and updating the counts of that move you just did

2. Move the ball along the X and Y movement directions or w/e

2 A:  If it hit the paddle, you need to do all the adjustments for that... those are listed in the assignment but it involves changing its x and y speeds

2B: If it passed the paddle, punish the movement you just made with .....    "ourstate.moves[motion] = ourstate.moves[motion] + alpha\*(-1)"

IE: subtract alpha

3. Now that the ball has been moved, get the state for the new position

4. Update the "goodness" of the move you just made

ourstate.moves[motion] = ourstate.moves[motion] + (alpha \* ( paddlecheck + gamma\*(max(nextstate.moves)) - ourstate.moves[motion]))

Then just go back to the start of the loop

because at this point you've moved the paddle, then moved the ball, then updated how good the move you made was, and have moved on to the next state

Note: if you "miss" the ball you break out of the loop because this round of the game has ended