

Troubleshooting

The console is outputting the fact that the code is telling the bunny to jump, therefore executing the if statement. However, the bunny is not actually moving to direction 0 (up).

When called outside of the if statement, the direction 0 works correctly. Also, I changed the text to prove that it was indeed this if statement, and it was.   
I even tested it with direction 3 (down) and it would not execute the movement. It goes through to the else statement, and moves left or right randomly.

If the very first move the bunny can make is onto a log, it will. So, it is detecting it can run forward. It then runs back and fore on the log until it reaches the end of the screen as it cannot move onto another log in front of it even if that log is detected (proven by using a print function).

The bunny detects cars, but does not effectively cross between them and does not predict the movement of the car so enters a gap, but then gets run over from the side.

Implementation

FSM was the most sensible and relevant technique to use for the Bunner AI. The life state of the bunny can be considered one FSM and the movement state of the bunny could be considered another. The concurrent FSMs could even be combined by running the movements FSM WHILE the life FSM is in the ALIVE state.

The state of character: Alive or Dead(split into EAGLE, SPLAT and SPLASH) can be viewed as an FSM. There are 2 true states, alive or dead, much like an on-off switch. The way to get from off to on (dead to alive) in this case is restarting the game. The state cannot be changed during gameplay. The death state can be broken down, creating more states within the machine that essentially operate in an identical way.

Restarting

Not meeting death criteria



Off Screen

State: ALIVE

State: SPLAT

State: SPLASH

State: EAGLE

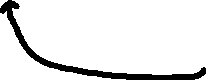
State: ALIVE

State: DEATH



Dying

Restarting



Restarting

Restarting

Hit by vehicle

Drown

Not meeting death criteria



Reaches hedge

Reaches a rail with a train in front

Reaches water with no log in front



Reaches a road with a car in front

Finds log

Finds gap in hedge

Finds gap in rail

Finds gap in road

Has no clear path



State: Random Left/Right

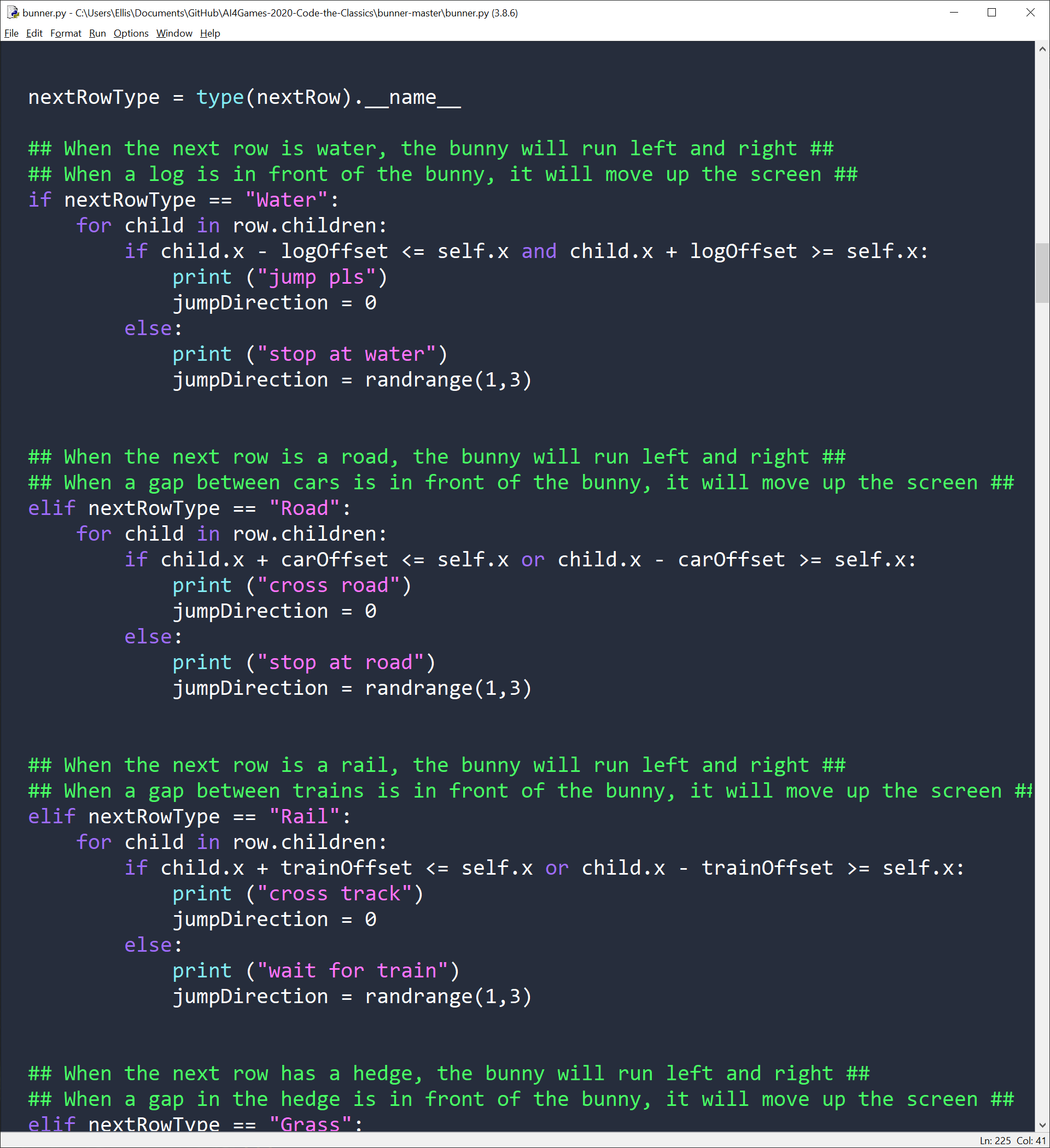
State: Move Up

Has clear path



The movement of the bunny can be reduced to 2 states that it cycles between. Technically the left and right could be split into 2 states, but they are only called through random generation – so it is one state of “random movement in the X direction”.

The bunny will repeat the “Move Up” state and automatically move up the screen. If it detects a car, train etc in front of it, the bunny will swap to the “Random left/right” state and repeat that state until the bunny detects a clear path – at which point it will swap back to the “Move up” state.



The code above has the 2 options of the FSM as an if statement. If the bunny approaches water and there is no log to jump to, the “Random left/right” state will enable and the bunny will run parallel to the water until a log is found in front of it. Then it will switch states and jump forward onto the log.

As the code stands, the bunny does not account for being at the edge of a log. To make the AI more accurate, the bunny would need to include an idle state in the FSM. As shown below:

Reaches hedge

Reaches a rail with a train in front

Reaches water with no log in front



Reaches a road with a car in front

Finds log

Finds gap in hedge

Finds gap in rail

Finds gap in road

Has no clear path



State: Random Left/Right

State: Move Up

Has clear path



Finds log in front

Is on a log

State: Idle

The idle state could be implemented instead of using a “Random left/right” state, but the movement means the bunny does not sit around waiting for a log on the initial row of water, or does not sit in the middle of a road.

Currently the bunny does not account for traffic in its own row heading towards it. There would need to be a separate detection to have the bunny run specifically away from a car in its row. The FSM once more would grow.

Reaches hedge

Reaches a rail with a train in front

Reaches water with no log in front



Reaches a road with a car in front

Has no clear path



State: Random Left/Right

State: Move Up

Has clear path

Is on a log

State: Idle

Finds log in front



Finds gap in rail

Finds gap in hedge

Finds log



Has clear path

Finds gap in road

Car is approaching



State: Move away from car

I did not think A\* was as suitable for this application as an FSM. Though technically the application is a grid using x and y coordinates for positions, it does not particularly lend itself nicely to an A\* implementation as they are not necessarily “nodes”. An A\* algorithm would work best for the game if it was not a randomly generated map. It could map out the perfect path that took the least amount of moves to reach a high score. If it were to be implemented in the random, infinitely generated map, it would likely take a little more processing power than the simple if statements of the FSM. The heuristic value would be calculated for every space in the row in front of the bunny, therefore the time complexity would increase. It could be limited to just up, left, or right, but then will not predict the better path and just choose the best option at the time.

The perfect AI for Bunner would scan far further in front of the bunny, not one row at a time. This would increase processing time considerably, however. To gain a heuristic value for hundreds or thousands of nodes every tick, process them, calculate the ideal path (taking into consideration the prediction of collisions with objects) and track information about all objects, it will likely lose frames.

As the current system only scans one line in front, the likelihood of the bunny being forced off the back of the screen or going of the screen on a log is unfortunately high. The bunny will move back and for searching for a log for too long, to the point the EAGLE death state is met as you drop off the bottom of the screen.

APPENDIX

def update(self):

# MANUAL CONTROLS #

for jumpDirection in range(4):

if key\_just\_pressed(direction\_keys[jumpDirection]):

self.input\_queue.append(jumpDirection)

if self.state == PlayerState.ALIVE:

# While the player is alive, the timer variable is used for movement. If it's zero, the player is on

# the ground. If it's above zero, they're currently jumping to a new location.

# Are we on the ground, and are there inputs to process?

if self.timer == 0 and len(self.input\_queue) > 0:

# Take the next input off the queue and process it

self.handle\_input(self.input\_queue.pop(0))

land = False

if self.timer > 0:

# Apply movement

self.x += DX[self.direction]

self.y += DY[self.direction]

self.timer -= 1

land = self.timer == 0 # If timer reaches zero, we've just landed

current\_row = None

for row in game.rows:

if row.y == self.y:

current\_row = row

break

jumpDirection = 0

logOffset = 10

carOffset = 40

trainOffset = 160

moveBool = False

lateralCooldown = 0

myRow =None

nextRow = None

myFound = None

for row in game.rows:

if myFound:

nextRow = row

break

if row.y == self.y:

myRow = row

myFound = True

nextRowType = type(nextRow).\_\_name\_\_

## When the next row is water, the bunny will run left and right ##

## When a log is in front of the bunny, it will move up the screen ##

if nextRowType == "Water":

for child in row.children:

if child.x - logOffset <= self.x and child.x + logOffset >= self.x:

print ("jump pls")

jumpDirection = 0

else:

print ("stop at water")

jumpDirection = randrange(1,3)

## When the next row is a road, the bunny will run left and right ##

## When a gap between cars is in front of the bunny, it will move up the screen ##

## It currently does not account for direction of traffic flow ##

elif nextRowType == "Road":

for child in row.children:

if child.x + carOffset <= self.x or child.x - carOffset >= self.x:

print ("cross road")

jumpDirection = 0

else:

print ("stop at road")

jumpDirection = randrange(1,3)

## When the next row is a rail, the bunny will run left and right ##

## When a gap between trains is in front of the bunny, it will move up the screen ##

elif nextRowType == "Rail":

for child in row.children:

if child.x + trainOffset <= self.x or child.x - trainOffset >= self.x:

print ("cross track")

jumpDirection = 0

else:

print ("wait for train")

jumpDirection = randrange(1,3)

## When the next row has a hedge, the bunny will run left and right ##

## When a gap in the hedge is in front of the bunny, it will move up the screen ##

elif nextRowType == "Grass":

for child in row.children:

if child.x + carOffset <= self.x or child.x - carOffset >= self.x:

jumpDirection = randrange(1,3)

else:

jumpDirection = 0

self.input\_queue.append(jumpDirection)