Jake Rayner

(Re)Creating the Classics 16651251

Table of Contents

**What is a finaite state machine/A\*2**

Problem and solution to bunner can do3

**Testing**4-5

**diagram**6

**Code 7-9**

**What is a finite state machine /A\***

**Finite State machine**

A finite state machine is a technique that is used in computation that models sequential logic. It is an abstract machine that can be only one finite-state at any given time. The finite state machine can change from different states via inputs this is called a transition. Examples of this kind of state machine would be vending machines, traffic lights, and elevators. Its commonly used to program A.I which are used in the production of games and algorithms. A use for this in game creation would be the state trastions from walking, runing, attacking.

**A\* Algorithm**

This is an artificial intelligence algorism technique used to find the shortest paths from whichever state is the start to the end state. An example of this could be a character solving a puzzle. The way this technique works is there's a start state also known as a goal state where the program begins and where it aims to get to the goal. The algorithm works by generating paths that can lead to words to the goal. These paths are also referred to as children the children are generated into a list then are stored in a priority queue. The order of the queue is based on the child's distance to the goal as in how close is to accomplish it. The algorithm will test the closest child and continue to do this until the goal is reached or there are no more children whichever comes first. A use for the algorithm would be finding a way to get to point from point A to point B

**Problem and solution to bunner**

**Code Analysis**

After looking over the Python code for the bunner game I have come with several things that I would like to implement in the form of A.I to the game. Some of the ideas I have are creating an A.I that would detect if the space in front of the bunner is safe to travel. For example, if the space is water don’t move forward and if its grass, road, or a log continue to advance. While this is a basic solution it is effective but does have a few flaws as it would need a way to move around bushes that are blocking the way forward. A fix for this would be making the bunner move left or right along the bush until there's an opening. These are just primarily just ideas right now however I have decided I will be using the finite state machine technique. I have decided to use this technique over the A\* algorithms because I feel Iike have a better understanding of it. I feel that using the finite state machine is the best chance for getting a high score with the A.I.

**Soulution**

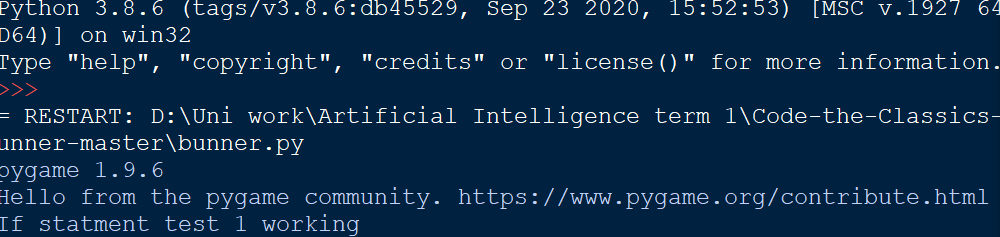
After having a look at the code I have come up with a solution that I am pleased with and have implemented. Using the finite state technique as a base I have added several lines of code to the program to do several things. First I have added code that stops bunner from advancing if there is water in front of him. While this works pretty well there are a few problems with this. The major problem being the bunner will attempt to cross the road into the path of a car. This makes getting a high score a lot more challenging. Another thing I have added to the game is a way for the bunner to navigate around bushes. This is done by the bunner moving left or right and keeps turning until an opening is found in the hedge. While this is functional it's not a very smart way of doing this and is more of a blunt force kinda way.

If I had longer I would make several improvements to these new additions to the code. First I would improve the bunner ability to detect cars on the road as it crosses the road with little cation. Another improvement to be made is when the bunner uses the logs to cross the river. The bunner can jump on to a log but then jumps again without a log to land on. This came up a number of times in testing a fix for this would be to make the bunner only jump when the log is in front of it and if I had time would implement it. Along with that was another issue which was the bunner waiting for logs. The bunner would only jump on the logs directly in front of so it would remain stationary until a log was in its view. I would fix this by making the bunner go either left or right until it is at the closet log then jump on it instead of waiting for it to be in front of the bunner.

**Testing**

**If statements**

To test that my code the added ran implemented a print statement to output a line of text if the code ran as quick test id the multiple times for each piece of code I added. These results are shown below.







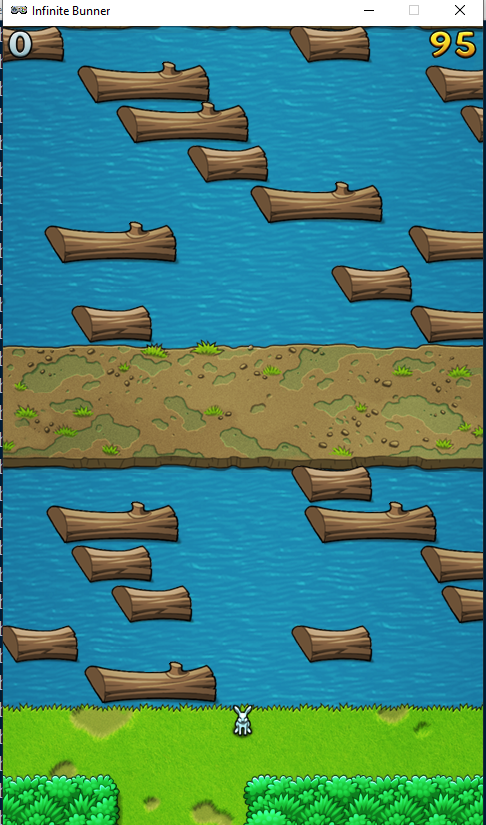
**Hedge test**

Running the program the bunner started making it way through the level until it got to a hedge. When it got to this point the bunner stopped then started running right look for an opening in the hedge which it did find and continued on through the level. This is shown below.

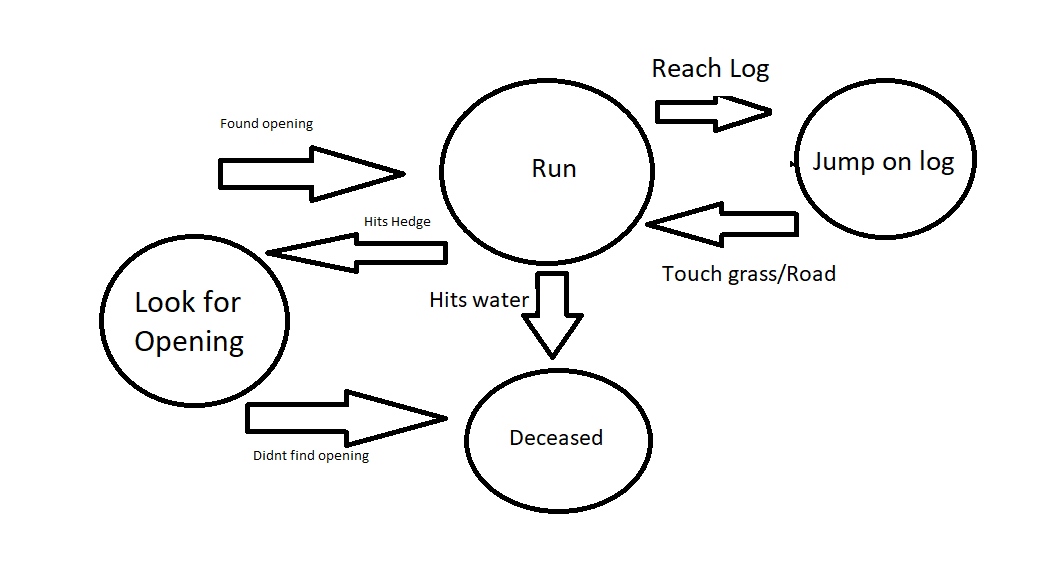


**Stop at water**

After the program is run when the bunner got to the water it did remain there until a log passed in front of it as you can see below. After which it hopped on to and made it across the water with no issue.



**Finite state Diagram**



**Code**

def update(self):

# Check each control direction

activeY = None

lastY = None

active\_found = False

active\_row = None

next\_row = None

jumpPath = 0

for row in game.rows:

if active\_found:

next\_row = row

break

if row.y==self.y:

active\_row = row

active\_found = True

#code to check if bunner is dead and if so advance to next state

if next\_row:

NextState = next\_row.check\_collision(self.x)

PlayerNextState = str(NextState)

if str(NextState) == "SPLAT":

print(NextState)

#tells the user that bunner is alive however this prints multiple times

if(PlayerNextState.find("ALIVE") == -1):

print("Bunny alive")

else:

rowType = type(next\_row).\_\_name\_\_

self.input\_queue.append(0)

lastY = activeY

activeY = self.y

if rowType == "Grass":

for child in next\_row.children:

if next\_row.collide(self.x, 0):

for i in range(16):

#print(currentY)

self.input\_queue.append(1)

activeY = self.y

self.input\_queue.append(0)

#print(currentY)

if self.y > activeY:

break

#self.input\_queue.append(0)

if self.y == self.y and type(child).\_\_name\_\_ == "Hedge":

for i in range(16):

# print(currentY)

self.input\_queue.append(3)

currentY = self.y

self.input\_queue.append(0)

#print(currentlY)

if self.y > activeY:

break

# self.input\_queue.append(0)