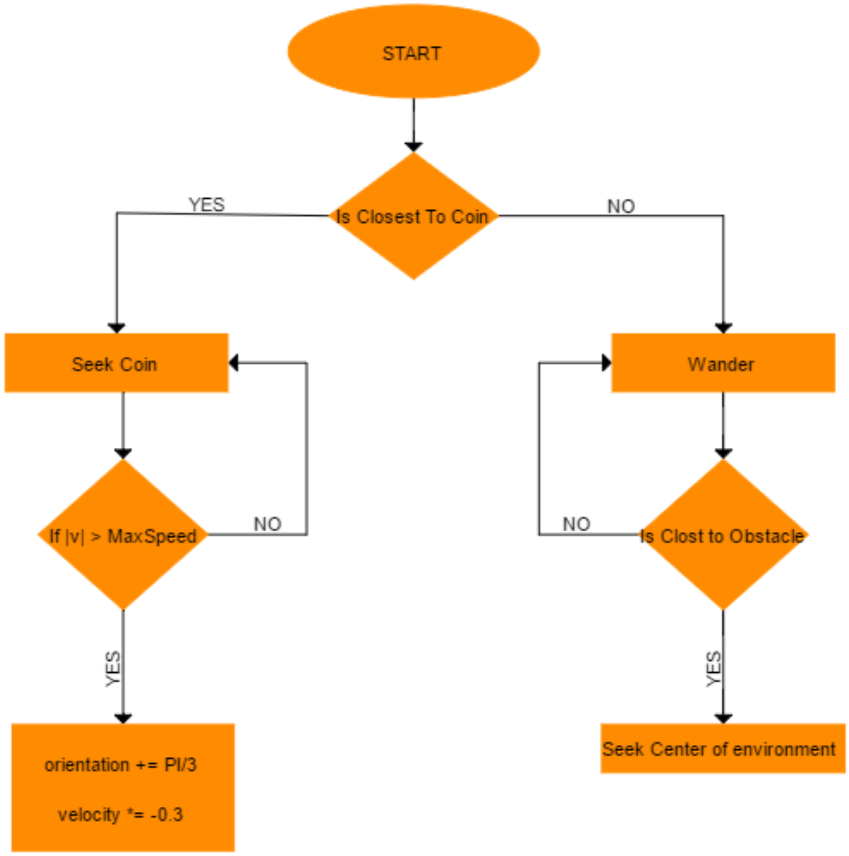
CSC 584/484 Spring 17 Homework 3: Pathfinding and Path Following

Due: 4/26/17 by the start of class

Decision Tree(20pts)

In this part of the assignment we are asked to design a decision tree and use this decision tree to control the player's movement in the environment. Following is my decision tree.



The way this decision tree works is as follows. First it starts at the START, then it checks if there are any coins close to the player. If the coin is within the reach of the player then it seeks the coin, by finding the path to the coin using A* algorithm, which outputs the list of ordered points. This list is fed to path follower and the character is made to follow the path until it retrieves the coin.

While following the path if the player's velocity magnitude goes above the max speed, then it receives a "SLAP" which brings the velocity under the speed limit. A punishment for over

speeding. This SLAP makes the player rotate 60 degrees in the direction of motion and reduces its velocity by 30% in the direction opposite of its motion. This makes the player look like sleepy driver, who gets awake every time the it reaches max speed. Also, it does a little dance on some corners of the obstacles. This has been added so that it helps the monster to catch the player.

If the none of the coin is in the range of the player, then it starts to wander. Wander with obstacles was hard to achieve. So, what I did is that I used a hack. I used path Finder to find the path to the random location returned by the wander method, parallelly checking if the location doesn't intersect with any of the obstacles. If it intersects with any obstacle, then the player starts to move to the center of the environment. Although the player doesn't entirely travel to the center. If at any point wanderer returns a valid random location then the character seeks that location. This approach gave the same results as wanderer should give.

The value of how close a coin should be set to 300. After experimenting with many other values, I found that I if this value is low then it ends up wandering for a longer period, and thus not achieving coins as quickly as it should be. On the other end if the value is set to high, then the player just seeks out the coins and doesn't end up wandering at all.

After all the coins are collected, new coins are generated, and the player respawn to its initial position.

Thus, my approach includes changes in targets and movement behaviors.

Behavior Tree (20pts)

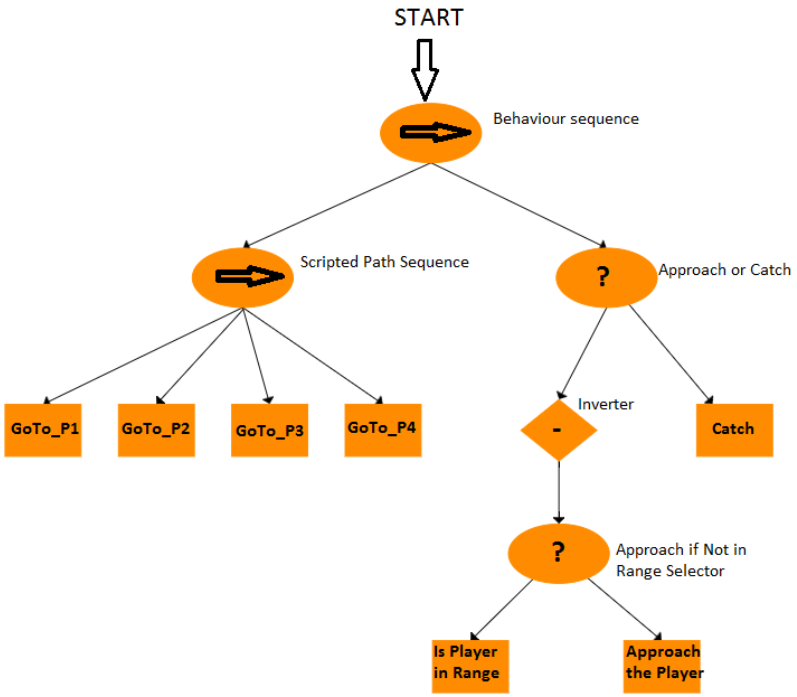
In this section, we were asked to design a monster that behaves in a fashion. My monster follows a scripted path and later tries to catch the player. Below is the behavior tree that I've implemented in this assignment.

The way in which the behavior tree works as follows. Parent node is the sequence node. The first node to the left of the parent node is the Scripted Path Behavior, which is a sequence of movements. When in this node, the monster, first execute the action "GOTO_P1" which is a position in the scripted path. When the monster reaches P1, then it returns true, and next node of the sequence is executed, i.e. "GOTO_P2", position 2 in the scripted path. Once it reaches position P2, it returns true and consequently the sequence node, runs rest of the two nodes, "GOTO_P3" & "GOTO_P4". Once these nodes are executed, they return true, thus sequence returns true and the parent node, which also the sequence goes to its right child for the execution.

The right child of the parent node is a selector for whether monster should approach or catch the player. This node further as an inverter as its left child which inverts the result from its child which is a selector. This selector has two children, it checks if the player is in the range. If it is true then selector returns true, inverter returns false and thus the monster proceeds to catch him. If the player was not in range, then the selector will run its next node, which is approach the Player.

In "Catch" action the monster seeks the player's current location and its speed increases substantially when it is pursuit. Although it does slow down when it gets near to the character,

until the player receives a slap or slows down or change the direction of its motion, it catches it. Once it catches the player it sets the catch state to true, thus making the “Approach or catch” state to true and ultimately the behavior tree returns true.

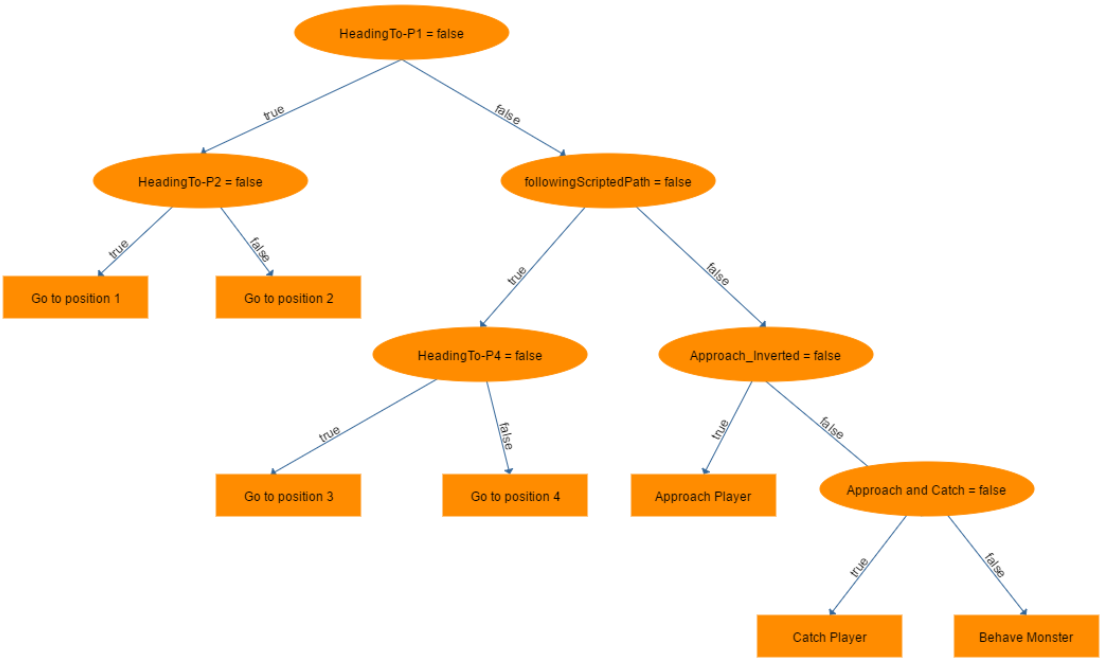


When the behavior tree returns true, monster goes back to follow the same routine till the end of the run. When player is being pursuit it doesn’t try to avoid it. But if it gets caught, it loses all the coins and must collect them all again.

Decision Tree Learning(20pts)

Looking ahead for the implementation of this step I’ve recorded data in some intelligent fashion, by doing some preprocessing myself. Since we need to have the information of states and the actions taken. I’ve encoded state flags, which if true represents that the state is being processed or processed. If it false then that means that we haven’t entered that state yet.

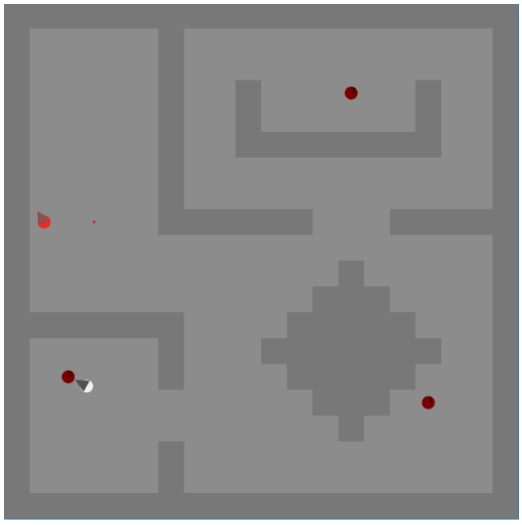
There are total 9 attributes that I’m generating and applying decision tree learning algorithm over it. These 9 attributes are pertaining to the states. They are: monster following scripted path(T/F), monster heading to position1(T/F), monster heading to position2(T/F), heading to position3(T/F), heading to position 4(T/F), monster approaching the player (T/F), monster catching player(T/F), etc. and the actions are, seek position1, seek position2, seek position3, seek position4, approach player, catch player, behave monster (which marks the completion of catching the player). The learning algorithm gave me the same results as that by my behavior tree. The Decision tree generated is as shown below.



The data set on which the decision tree has been applied has more than 2000 samples. Completing one run successfully, were the monster catches the player, will generate around 1000 samples. To make sure that the decision tree generated is accurate you need to complete at least one complete cycle of the behavior tree, i.e. until it returns true.

While working on decision tree, I faced a lot of difficulties in selecting the attributes. I tried various attributes, to state few, monsters position from player, their velocity, current seek Location of both, etc. This led me to think of much simpler attributes. So, I used the states of my Behavior tree as my attributes. The state of behavior is true if it is being initiated or being processed, else it is false. Taking this value and generating the data for the decision tree, seem to give more accurate results. It almost depicted the same behavior. Following is the image of my game environment. Where the red character is the monster, currently on its scripted path, and the white character is the player who is currently collecting the coin.

Please check the video attached along with this document, where, it runs both behavior tree and decision tree, at the same time. Red colored is running behavior tree and the green colored one is running the learnt decision tree.



The output shown by the decision tree, is shown below. I haven't labelled the column names, but you can check in the text file generated by the program. Please delete the first line of the generated text file, as it contains labels to each column and may affect the output of the decision tree algorithm. The below results are same as represented above.

```
E:\Sem1-2\Game-AI\asgavane_hw3>py decisionTree.py
Column2 : false?
Left->
Column1 : false?
  Left->
{'GOTO_P1': 1579}
  Right->
{'GOTO_P2': 299}
Right->
Column4 : false?
  Left->
Column3 : false?
    Left->
{'GOTO_P3': 491}
    Right->
{'GOTO_P4': 286}
  Right->
Column6 : false?
    Left->
{'APPROACH': 324}
    Right->
Column5 : false?
      Left->
{'CATCHING': 578}
      Right->
{'BEHAVE_MONSTER': 2}
```