CSC 584 Building Game AI
Homework 4

hbharat2

Question 1:

The given task involves creating a Decision Tree[1] to simulate a behavior of our choice for the boid on the screen. Let me take a few sentences to set up the context and we can then talk about the implementation I have used. I have used a grid with 40x30 size from the last assignment, each cell being 20x20 pixels. There are 3 distinct rooms with doorways. The Game state is tracked by tracking the current behaviour, duration in the current state, and the current movement target. The overarching behaviour would be to switch between path following and wandering behaviour based on a few decisions made. The variables taken to implement the Decision tree are duration in a particular state, distance from a wall/ wall proximity, and the speed and maximum velocity. This helps us adequately summarize the local idea of what the boid is currently up to. The three primary decision nodes are: TimeINState node, WallProximity node and a Velocity node. The Time in State node ensures a behaviour is not locked in forever and there is some random NPC like movement associated with the character. When I initially designed it without a TimeInState Node, once the boid reaches one of the walls and starts wandering, it just keeps wandering forever as we don't have any control over how it wanders and it just wanders within the wall proximity. This led me to introduce a timeInState node that forcefully changes the behaviour of the boid and thus make it look more natural and NPC like.
The path following behaviour follows A* search algorithm which progresses to a particular waypoint, taken from the previous assignment.
The Pathfollowing behavior involves an A star search which has speed adjustments next to turns where the speed is reduced and the waypoint is reached.
The Wander behaviour initially implemented in assignment 2 did not have any obstacle/ Collision avoidance coded into it. This caused some issues and thus I included a collision avoidance algorithm to make sure the wander does not go into the walls.
The obstacle avoidance was implemented to check a 2 cell radius around the boid and inform whether a wall is there. This when triggered, gave a repulsive force to steer exactly opposite the lineOfSight from the wall.
The decision tree involves deliberately switching up the order of path following and wandering in the leaf nodes. This is because when the boid is path following and suddenly hits a wall, it shouldn't start wandering and stray, thus the decision tree has such a condition in the leaf node.

Question 2:

The given task involves implementing a Behaviour tree[2] for a monster character which collides with our player character. I envisioned the Monster to be a spirit which has 4 patrol points around the map. When a boid is in the vicinity of its search radius it exits its patrol behaviour and starts chasing the boid.
The Behaviour tree implemented uses 3 types of composite nodes namely Selector, Sequence and Parallel. The behaviour tree implemented in this tries all the children in order until one succeeds. Therefore the first node tried is the Victore Node which then launches a sequence of Victory dance after collision and a wait timer. This is ensured by a sequence node situated at the Victory Dance node which first checks if collisions happened, and then the victory dance is performed (In this case it is a huge circular path where it scales up and down in size until it reaches the initial collision point and then waits for 2 seconds) after which the monster resets. A Parallel node used here is the Chase sequence where it continuously does both actions of trying to stay in the line of sight of the character and move towards the character i.e move towards the player.
A few problems solved were that the player kept spawning inside obstacles or randomly in the same location, thus the player was made to respawn in a random safe location in one of the

three rooms. After this the grid information was given as points where the player would not be allowed to respawn. The monster's reset position was always kept the same.
Another problem faced was the hierarchy of actions performed by the monster. Since the monster's behavior is determined by a behaviour tree which dictates that it maintain the line of sight to the player but at the same time the steering behaviours used have obstacle avoidance in them, The monster got confused and just kept hitting the wall aimlessly trying to go to the player in the next room. Since this happened I had kept a 5 second timer after which the monster mandatorily switched behaviours for a while and then it would come back to chasing (The behaviour switched to was patrol). This was rarely observed afterwards as the player boid usually comes somewhere near the radius of detection of the monster and thus the monster gets a different angle to chase it and eventually catches it.

Question 3:
This was a very interesting question which took a lot of time for me to develop. The given task involved parametrizing the monster's behaviour influenced by the behavior tree and then converting that into a usable set of data. The task then asked us to train a decision tree[3] based on this data and subsequently run the new monster and compare it with the old monster. Let us get into the meat of this write up then!
The first task was to choose variables to parametrize the state space from the behavior tree. The initial variables chosen were only currentRoom, distanceToPlayer, hasLineofSight, NearestWall. I quickly realized these weren't enough and went on to track a few more variables to parametrize such as TimeinState, PlayerinSameRoom, AngleToPlayer, PlayerSpeed, Action currently being performed. To record all these features I had to find a way to quantify them. Some harder ones to quantify were: WallDistance - for which I used the line of sight I had already coded and found the nearby wall on the grid. Action timers had to be integrated to record state changes. The player in Same room was implemented by recording the players room as a game state and updating the monster's state to check with the player every frame. Once these were done I tried running the file to get 3000 lines of data thinking it'd be enough. The decision tree poorly underfit and just kept dancing around in the simulation after. I ended up recording and training 7282 observations for the decision Tree.
Some metrics as seen in[4] show us that all the samples recorded are equally distributed and thus there is no class imbalance when training the Decision tree[3]. One interesting observation I made in terms of sanity check for checking whether the generated Decision tree made sense was to check the mean values of important metrics. My hunch was correct as the mean values +/- epsilon seemed to be the inflection point between different states and thus this acted as a sanity check for me to confidently state that the Decision tree worked.
The Decision tree is trained using Entropy and Information Gain. Instead of pruning the tree I implemented a minimum gain threshold and a maximum depth to the tree. The initial tree was generated with a minimum information gain of 0.5 and it gave a very general and shallow tree without a lot of discriminatory features. The second tree was trained with a minimum gain threshold of 0.001 and it was deeper and finer and I ended up choosing the deeper tree as it was able to move the monster a little more accurately. The final maxDepth was set to 8 and was generous enough allowing a maximum of 256 child nodes.
The final Decision tree was tested using a few test cases which has been listed below in [8] and shows its correct nature.
The monster is trained when the file is run and then the state is predicted according to the data currently fed. The initial underfit model for the monster had an average time to collision as 10.78 seconds as it kept wandering aimlessly, compared to a benchmark value of 6.39 seconds. The final model fit was able to reach a lesser time to collision of 5.2 seconds but the newly learned

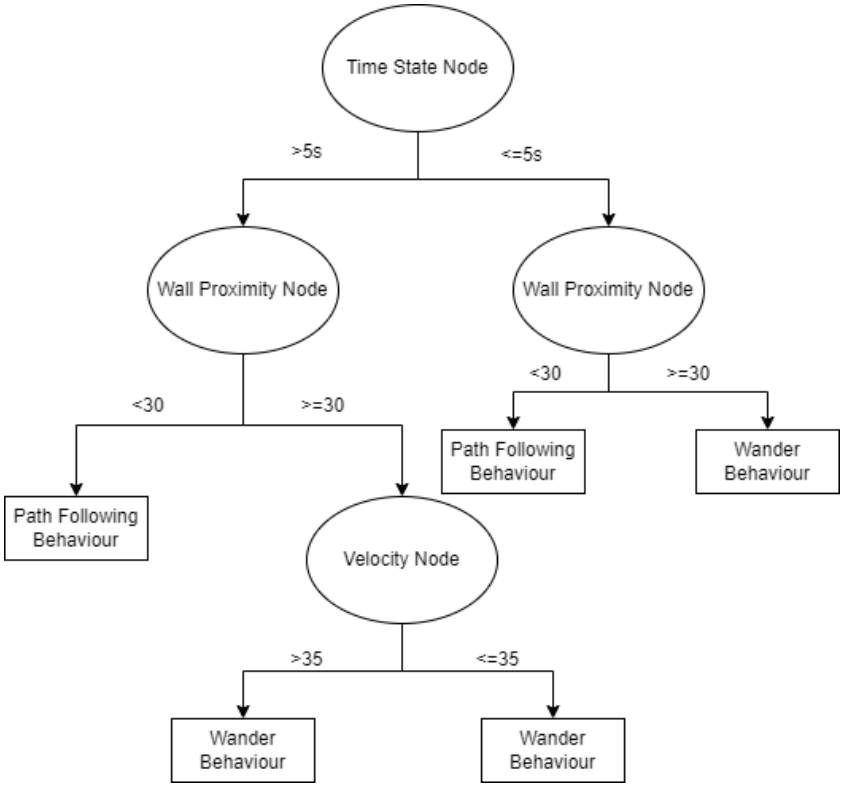
tree has a few flaws.(The averages were only taken from around 5 to 7 observations and might not be very accurate)

Flaw 1: The decision Tree learner did not learn to dance properly. The original victory dance involved going around the window on a huge circle scaling from small to big size and then reaching back to the collision point and waiting for 2 seconds. The current dance seems to jitter a lot and not complete the dance circle as a whole. Reason was that the states kept changing when dancing and thus the dance did not last a whole 2 seconds. This interim change in state caused a jittery dance which does not seem perfect.

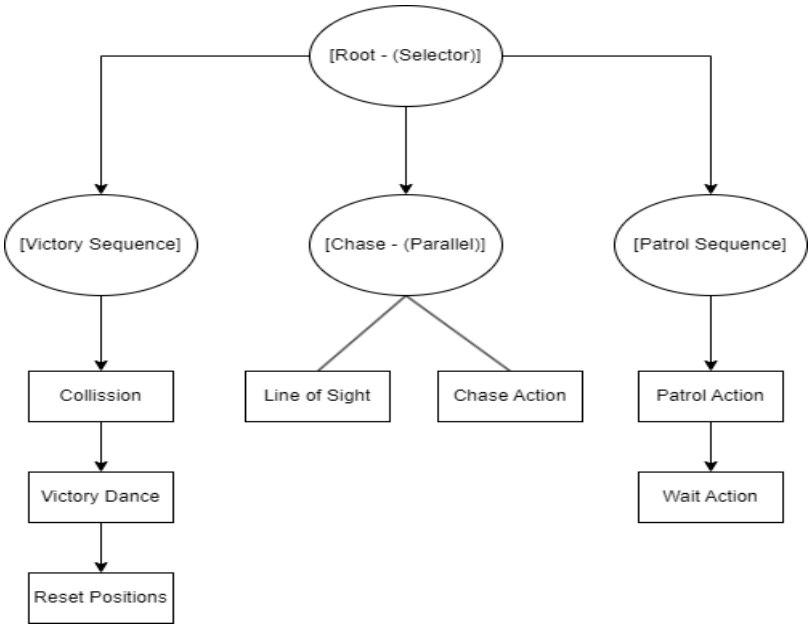
Flaw 2: After a chase sequence is complete, the monster seems to jitter and move very randomly. I think this can be again attributed to the fact that after the chase sequence the next state predictor was not trained perfectly and thus kept giving random states thus causing jittery behaviour.

APPENDIX:

[1]



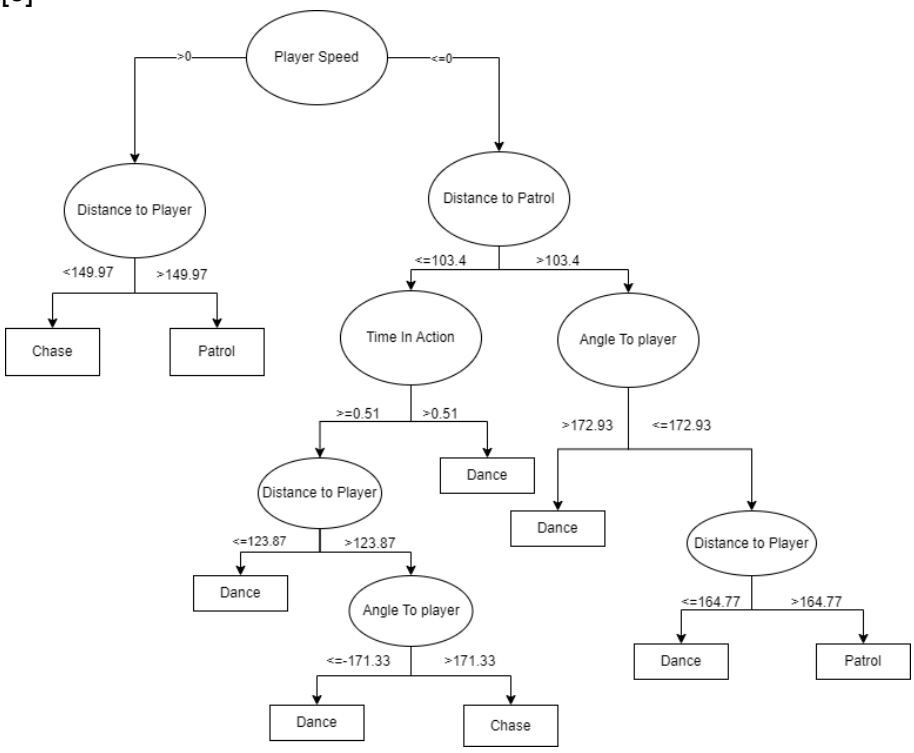
[2]



[4]

```
=== Running train_dt.cpp to train and display decision tree ===

./train_dt
Analyzing training data...
Training Data Analysis:
Total samples: 7286

Action distribution:
     CHASE:  2037 samples (28.0%)
     DANCE:  2413 samples (33.1%)
    PATROL:  2836 samples (38.9%)

Feature statistics:
          Room: min=    1.00 max=    3.00 mean=    1.87 unique=   3
   DistToPlayer: min=   23.30 max=  441.81 mean=  215.59 unique=6627
    LineOfSight: min=    0.00 max=    1.00 mean=    0.35 unique=   2
     DistToWall: min=    0.35 max= 1000.00 mean=  360.66 unique=2612
   TimeInAction: min=    0.00 max=    5.96 mean=    1.79 unique= 562
       SameRoom: min=    0.00 max=    1.00 mean=    0.41 unique=   2
   DistToPatrol: min=    0.35 max=  608.80 mean=  171.18 unique=6344
  AngleToPlayer: min= -180.00 max=  180.00 mean=  -62.44 unique=4911
    PlayerSpeed: min=    0.00 max=16007.56 mean=   28.87 unique=2558
```

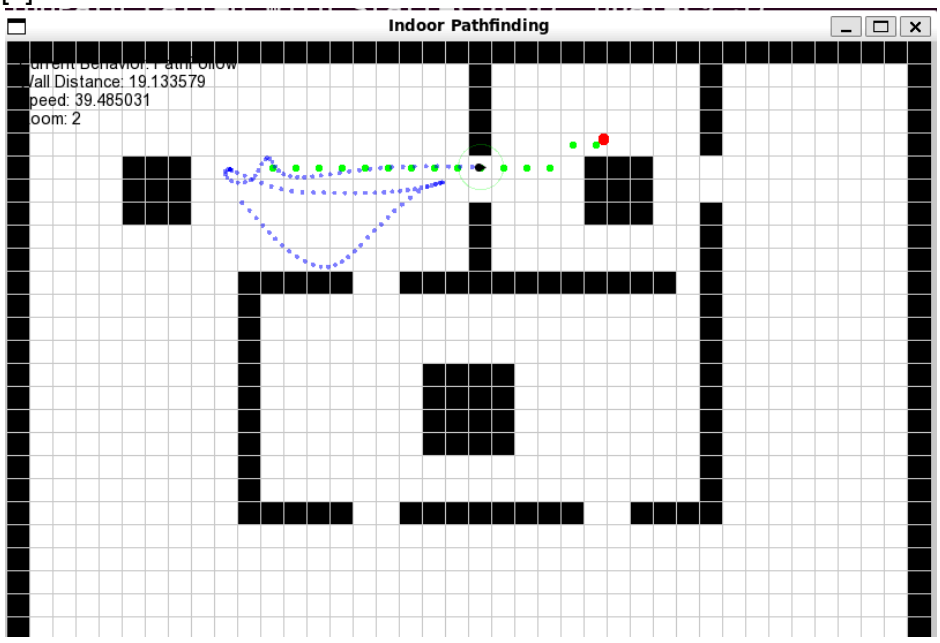
[3]



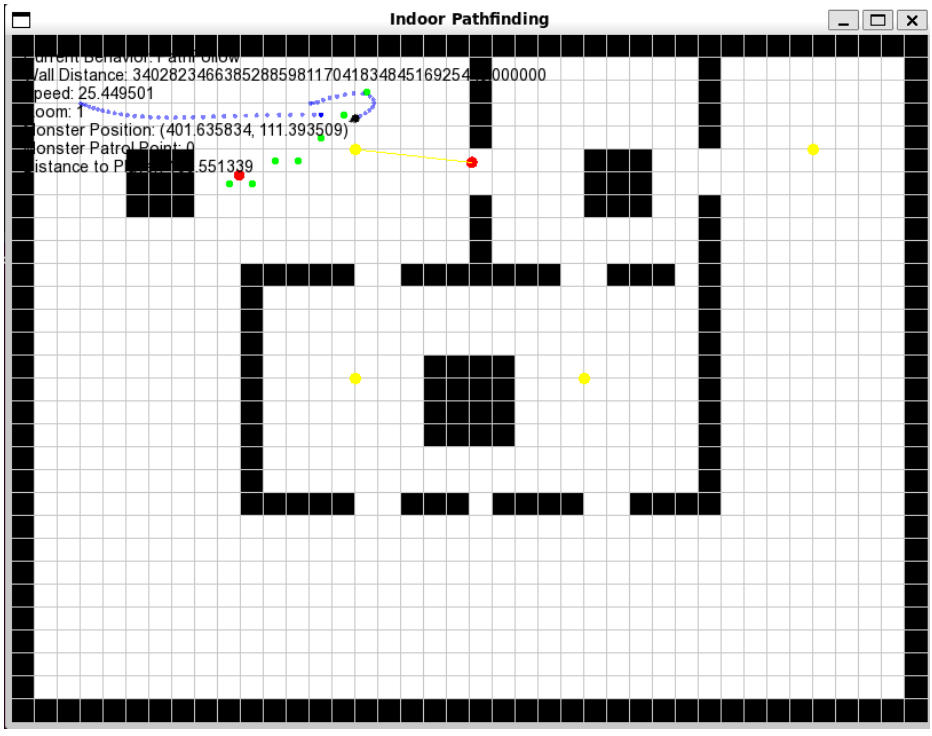
[8]

```
Testing various scenarios:

Test Case 1:
Room: 2.00, DistToPlayer: 100.00, LineOfSight: Yes, DistToWall: 50.00
TimeInAction: 1.00, SameRoom: Yes, DistToPatrol: 200.00, PlayerSpeed: 0.00
Predicted Action: DANCE

Test Case 2:
Room: 1.00, DistToPlayer: 300.00, LineOfSight: No, DistToWall: 100.00
TimeInAction: 2.00, SameRoom: No, DistToPatrol: 50.00, PlayerSpeed: 30.00
Predicted Action: PATROL

Test Case 3:
Room: 3.00, DistToPlayer: 20.00, LineOfSight: Yes, DistToWall: 80.00
TimeInAction: 0.50, SameRoom: Yes, DistToPatrol: 150.00, PlayerSpeed: 0.00
Predicted Action: DANCE

Test Case 4:
Room: 2.00, DistToPlayer: 150.00, LineOfSight: Yes, DistToWall: 10.00
TimeInAction: 1.50, SameRoom: Yes, DistToPatrol: 100.00, PlayerSpeed: 20.00
Predicted Action: PATROL

Test Case 5:
Room: 1.00, DistToPlayer: 200.00, LineOfSight: No, DistToWall: 60.00
TimeInAction: 5.00, SameRoom: No, DistToPatrol: 75.00, PlayerSpeed: 15.00
Predicted Action: PATROL
```

[7]


Indoor Pathfinding
Wall Distance: 19.133579
Speed: 39.485031
Zoom: 2

[5]


Indoor Pathfinding
Wall Distance: 340282346638528859811704183484516925440000000
Speed: 25.449501
Zoom: 1
Monster Position: (401.635834, 111.393509)
Monster Patrol Point: 0

[6]