Ben Connick, Eric Stivala

AI ROADMAP

Path following

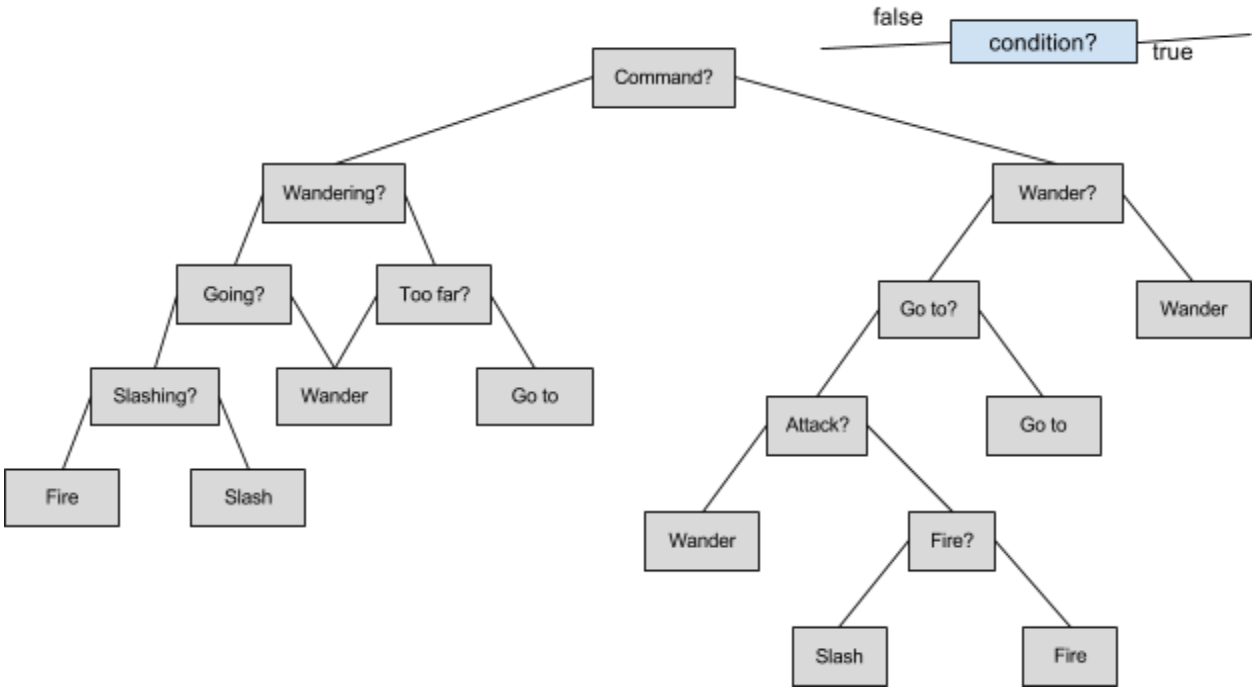
We implemented a simple waypoint path following algorithm and demonstrated it in class. Originally, it made the enemies follow the path to the tower. This aspect was removed to allow for enemies to have freedom of movement for GA and other behaviors. See commit d9d5a
Lesson: Useful game behavior, just not for these characters.

Path finding

Using Unity's navmesh, agents could plan their path to the tower. We demonstrated the functional version in class. We removed the navmesh in the final iteration because the navagent obstacle avoidance did not work well with custom agent avoidance code, and the custom obstacle avoidance looked more natural besides. See commit 1283a
Lesson: In complicated but static terrains, navmeshes are a good way to get characters from point A to point B. In wide-open areas they are less less useful.

Decision Tree

For our prototype of the dragon behavior, we used a behavior tree with manually-determined thresholds to make a decision of what action to take. We demonstrated this in class. We found Bayes is much more flexible and learning-oriented algorithm for the dragon's brain, so we removed the decision tree. See commit 9226a
Preorder traversal: ?command, ?wander, wander, ?goto, goto, ?attack, ?fire, fire, slash, wander, ?wandering, ?toofar, goto, wander, ?going, wander, ?slashing, slash, fire

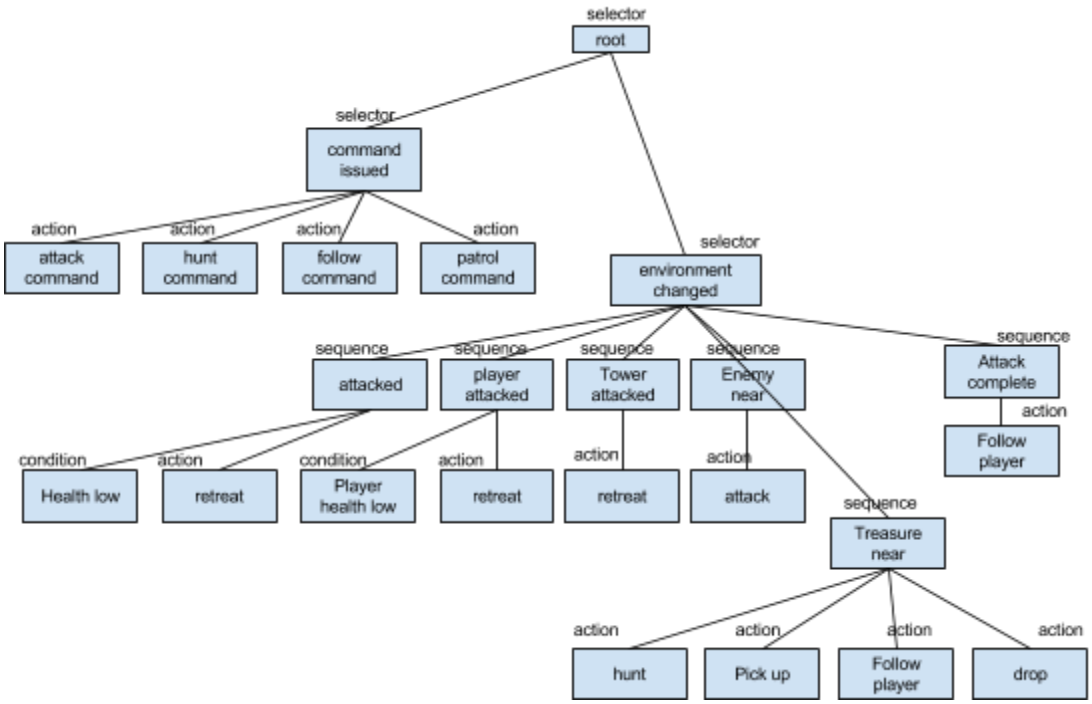


State Machine

Just for the exercise, we replaced the above decision tree with a state machine. Like the decision tree, this "brain" was replaced later. This state machine changes only on commands issued, not on environment changes. See commit 3f40b

```
//                          state 0:    state 1:    state 2:    state 3:    state 4:    state 5:    state 6:    state 7:
//                          idling      wandering   chasing     patrolling  following   slashing    burning     other
/* 0: no cmd        */   { 0,          1,          2,          3,          4,          5,          6,          7 },
/* cmd 1: wander    */   { 1,          1,          1,          1,          1,          1,          1,          1 },
/* cmd 2: go to     */   { 2,          2,          2,          2,          2,          2,          2,          2 },
/* cmd 3: patrol    */   { 3,          3,          3,          3,          3,          3,          3,          3 },
/* cmd 4: followW   */   { 4,          4,          4,          4,          4,          4,          4,          4 },
/* cmd 5: slash     */   { 5,          5,          5,          5,          5,          5,          5,          5 },
/* cmd 6: burn      */   { 6,          6,          6,          6,          6,          6,          6,          6 },
/* cmd 7: stay      */   { 7,          7,          7,          7,          7,          7,          7,          7 },
```

Behavior Tree




```
root :- command_issued.
root :- environment_changed.

command_issued :- attackcmd.
command_issued :- huntcmd.
command_issued :- followcmd.
command_issued :- patrolcmd.

attackcmd :- attack.
attack :- write('attack'), nl.

huntcmd :- hunt.
hunt :- write('hunt'), nl.

followcmd :- follow.
follow :- write('follow'), nl.

patrolcmd :- patrol.
patrol :- write('patrol'), nl.

environment_changed :- attacked.
```

```
environment_changed :- player_attacked.
environment_changed :- castle_attacked.
environment_changed :- enemy_near.
environment_changed :- treasure_near.
environment_changed :- attack_complete.

attacked :-
health_low.

player_attacked :-
player_health_low.

castle_attacked :-
retreat.

enemy_near :-
health_low,
attack.

treasure_near :-
hunt,
pick_up,
follow_player,
drop.

attack_complete :-
follow_player.

health_low :-
retreat.

player_health_low :-
write('is the player health high or low? '),
read(X),
(X == low),
write("the player is low on health"), nl.

retreat :-
write('retreat'), nl.

pick_up :-
write("picked something up"), nl.
```

```
follow_player :-
write("I'm following the player"), nl.

drop :-
write("dropped a thing"), nl.
```

Genetic Algorithm

We use the Genetic Algorithm in our game to evolve the enemy behavior. Specifically, in the final game, we are using the GA to evolve the safe distance from the dragon. We were initially running our experiments with 100 individuals, but we have since cut it down to 50 to make the game easier to play.

We are using a 10 byte chromosome with a phenotype range between 0 and 102.3 for the safe distance. Our fitness value for each individual is determined primarily by damage to the tower, and is slightly boosted by adding their total lifespan/500. This puts a small emphasis on how long they can survive, but ultimately favors those who can make it to the tower and deal some damage before death.

Evolving a Single Threshold

Initially, and in the final iteration of our game, we used the GA to evolve the safe distance from the dragon for the enemies. Our main experimentation was done before the dragon could be taught with Bayes, so we could just let the program run and observe the enemy learning. Because we randomly assign the phenotypes at the beginning, the enemy behaviors were quite sporadic. It typically takes between 20-25 generations for the enemies to reach a uniform level of decent behavior. However, there were easily noticeable improvements around the 15th generation in most of the experiments.

So how did the evolved values compare to our programmed values? We did two experiments to test this. In one experiment we took out all of the obstacles from the program and let it run for 25 generations. By the 25th generation, the majority of enemies had evolved to a genotype of 1000+, which translates to a safe distance around 100 units. This is more than twice our default programmed value of 40 units.

In the second experiment, we put the trees back into the environment, and got a completely different result. In this setup, which is much closer to our final game, the average enemy genotype was between 400 and 500, which translates to a phenotype of 40-50. This is much closer to our ideal programmed safe distance of 40 units.

Evolving Multiple Thresholds

We experimented by dividing the 10 bits of the genotype into a left and right side and using each side to evolve the safe distance and separation distance respectively. We let it run for about twenty rounds, but it didn't seem to approach optimal values as quickly as the single

threshold evolution. Some of the enemies liked to run each other over, while others would spawn in and immediately run as far away from each other as possible. Because the two thresholds were independent they seemed resistant to converging on decent behavior, at least in the time given.

Perhaps it is because the separation distance is not as directly related to the fitness as the safe distance from the dragon, but the separation distance seemed less uniform than the safe distance. Our programmed avoid distance was set to 8, but the enemies had evolved values as low as 4 and some as high as 14. The safe distance threshold was also more sporadic than in the past, but its range was relatively close to the ideal 40 units, falling between 30 and 80 units.

Viability in Our Game

Since our game revolves around stopping endless waves of enemies from destroying your tower, I'd say that this GA enhances the game play. The enemies attempting to evolve better behaviors to outsmart the player and the dragon leads to more variety in the gameplay, which in turn makes the game more fun. In one of my sessions with the game, the enemies started to evolve a large safe distance threshold, which meant that they were no longer walking the clear path straight to the tower. They ended up spread out in the trees, and it was much more of a challenge for me and the dragon. The dragon could no longer wipe them out en mass, as he had to navigate the forest, and I no longer had perfectly clear shots with my bow. This is just one example of how the GA can make the game more fun for players.

Bayes Classifier

Out Bayes classifier formed the final "brain" of the dragon, and it is where the learning happens. The observation table consists of 5 continuous conditions and 1 boolean action. The conditions are

1) the distance from the dragon to the enemy closest to the dragon
2) the number of enemies within a 20 unit circle of that enemy
3) the distance from the tower to the enemy closest to the tower
4) the number of enemies within a 20 unit circle of that enemy
5) the distance from the dragon to the tower.

The action is simply whether the dragon goes after the enemy closest to itself or the enemy closest to the tower. There is no neutral option, the dragon must choose one of those two actions.

Creating observations is the responsibility of the player, since this is a live-learning AI. The player takes the role (both narratively and literally) of the instructor/supervisor, and they are the sole judge of whether a decision was good. When the player approves or disapproves a decision, an observation is recorded and incorporated into future decisions.

If the player does not approve or disapprove of a decision, it is not recorded in the observation table.

Because we use a binary action, we record the observation with the opposite action when the player disapproves, and this seems to work well.

The dragon makes a new decision once every two seconds, unless it killed its target, in which case it makes a new decision immediately.

```
/* these observations are artificial */
174.423 3 245.678 2 71.49975 False
112.64 2 231.7521 3 119.5874 False
17.36357 3 24.30466 1 111.3498 True
23.23108 2 21.03123 2 97.56301 True
/* these observations are made in gameplay */
15.61122 1 111.1229 1 120.0496 False
31.87377 1 227.8942 1 199.0378 True
38.89787 2 213.5922 2 174.8364 False
20.26618 2 232.3264 2 214.3718 True
21.34716 1 186.8908 2 194.1762 False
36.84857 2 187.8197 2 231.4418 True
38.13211 3 178.4602 1 238.4471 False
14.34087 1 99.91768 1 135.8036 False
22.02424 1 80.90992 1 124.1399 False
29.74069 1 62.53941 1 118.3751 False
19.36756 1 39.9934 1 34.91599 False
67.59328 2 88.8091 2 25.74487 False
26.75761 3 93.85608 3 84.05164 False
51.6628 1 67.34476 1 67.97163 False
48.39218 1 74.81422 1 48.58844 False
33.82264 1 34.70156 1 62.84878 False
69.57124 2 71.85911 2 32.21437 False
175.3395 1 246.867 1 134.1002 False
170.72 1 233.2085 1 150.4832 False
118.6087 1 214.6763 1 168.3498 False
/* the impact from Bayes is undeniable, as the dragon almost never
defends the tower using this obs table */
```

The Bayesian classifier for a single decision as simple as this one works remarkably well, and the results are noticeable almost immediately. Players can change the dragon's behavior and see the results in just a few inputs.

The one drawback to this approach is that players water down the impact of an approval the more things they approve of, so where the natural tendency of players is to toy with the interactive thing, doing so will cause it to make odd and seemingly random decisions. Because a game using Bayes is blackboxed as far as the player is concerned, this means that interacting with the dragon has limited “play” value.

One solution that we discussed would be weighting observations so that recent observations are more important than older ones. Another could be periodically dumping old observations entirely. Ultimately, for the purposes of demonstration, we left the algorithm alone, so that we can demonstrate creating predictable learning patterns.

We were pleased to see that under a good instructor, the dragon learned surprisingly intelligent decisions. In one case, where Ben had predicted that a certain behavior pattern would be optimal (focusing on enemies near the tower when the tower was under attack), the dragon actually learned more efficient behaviors that involved targeting clumps of enemies at the expense of a little damage to the tower, which was more effective at preventing damage long-term; plus it looked more credible.

Because we were determined to have learning from the beginning, I can say that without Bayes, the game would have been incomplete, so it definitely improved the gameplay.

We tried Bayesian learning vs. the Enemies’ GA, and I can say that the effects of the GA are not nearly as clear. The GA was much more understandable in the context of a controlled environment. Back when the dragon had a simple behavior, the GA adapted and the result was higher individual and average fitness. Now that the dragon’s behavior is blackboxed, I can’t really discern whether the agents are becoming more optimal or just homogenizing arbitrarily. Their average fitness does not improve substantially after the third generation or so.

ID3

We did not use ID3 in our game, so we have created a hypothetical ID3 induced decision tree based off of the bayesian observations below. As stated earlier, the variables in the table are as follows:
1) the distance from the dragon to the enemy closest to the dragon
2) the number of enemies within a 20 unit circle of that enemy
3) the distance from the tower to the enemy closest to the tower
4) the number of enemies within a 20 unit circle of that enemy
5) the distance from the dragon to the tower.
And the final boolean is whether or not to attack the enemy closest to the tower

```
176.0269 2 246.4325 1 70.80947 False
```

```
43.3065 2 218.9598 1 176.1871 False
18.57281 2 185.7239 1 235.1993 False
44.13925 2 94.38057 1 199.431 False
24.77526 2 56.05809 2 180.1142 True
67.16964 2 37.88361 1 133.5094 True
26.44059 2 23.15842 1 76.26237 True
13.97845 1 27.79905 1 26.69146 True
```

These are observations the dragon made during play. For the sake of this example, I recorded all decisions as positive. All of our attributes have to be partitioned, since they are all continuous, so our observation table will become this:

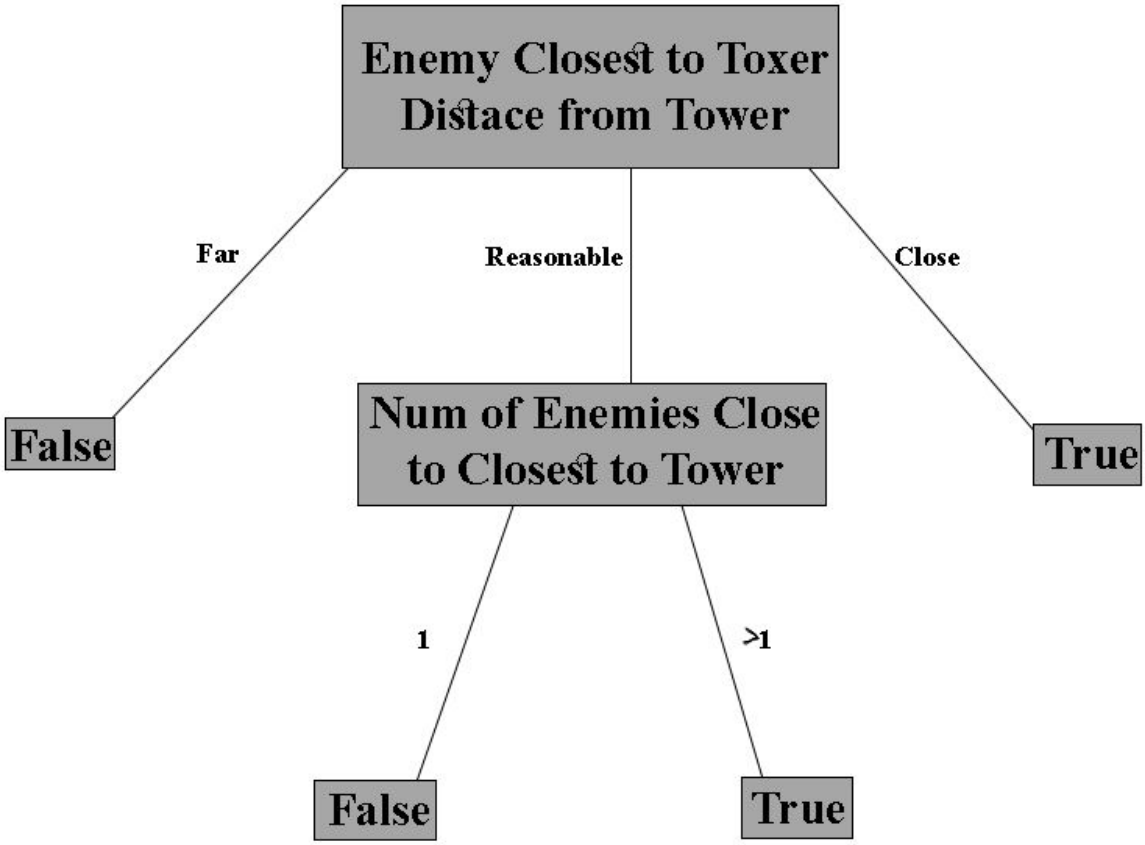
```
//For distances -- >100 far, <100 && >50 reasonable, <50 close
//For num of enemies -- 1 or >1

Far        >1  far         1  reasonable false
Close      >1  far         1  far        false
Close      >1  far         1  far        false
Close      >1  reasonable  1  far        false
Close      >1  reasonable >1  far        true
Reasonable >1  close       1  far        true
Close      >1  close       1  reasonable true
Close       1  close       1  close      true
```

The information gain for each variable is as follows:

```
the distance from the dragon to the enemy closest to the dragon: 0.25
the number of enemies within a 20 unit circle of that enemy: 0.138
the distance from the tower to the enemy closest to the tower: 0.75
the number of enemies within a 20 unit circle of that enemy: 0.138
the distance from the dragon to the tower: 0.143
```

This leads us to the following tree:



Does the Tree Make Sense?

The tree makes sense in a limited scope. It is pretty close to a tree we could have programed to come to a decision on whether or not to defend the tower, but I would have put more data into the tree, like ‘how close is the dragon to the tower’, as crossing the entire map may not make sense for the dragon.For the most part, however, it does capture the basic essence of whether or not the dragon should return to the tower to defend it.

How much Generalization Occurred?

The depth of this tree is significantly less than the number of attributes, seeing as we had 5 attributes and the tree only looks at 2 of them. The ID3 algorithm seems to think that The enemy closest to the dragon and the number of enemies around him are inconsequential. It also ignores the dragon’s distance from the tower. I think the tree is too overgeneralized, probably because you lose a lot of data when transitioning from continuous values to ones of finite cardinality.

Bayes or ID3?

So which is the better choice for our game? I would say Bayes. For starters, our game is about teaching the dragon as you play. I feel that this would be close to impossible with ID3. If it is possible, it seems incredibly computationally intensive. Also the dragon is supposed to help the player protect the tower, but the tree that ID3 constructed would lead to the dragon ignoring a single enemy chipping away at the tower's health until it is the only enemy left. This is less than ideal behavior for companion AI. Plus, with such a wide open world and so many enemies, I believe the continuous values of all 5 variables would lead to more informed decision making.