PROFESSOR: Today we're going to be talking about games. And I know you guys as well as I hope I do. The main thing that you guys want to talk about with games is how to do that alpha-beta thing. Because it's pretty confusing. And it's easy to get lost in a corner or something. Whereas doing the regular minimax, in my experience, most 6034 students can do that. And they do it right pretty much all the time.

However, we're going to focus on all the different components of games. And I put up two provocative silver star ideas up on the board, which will come into play here. The Snow White principle is a new name. And it has never been revealed until today. Because I made up name recently. So you will be the first people to hear it and decide if it works better than the term "grandfather clause" for the thing that I'm trying to describe. Because most grandfathers don't eat their children.

So here we've got a beautiful game tree. It has nodes from A through R. This is our standard game tree from 6034. We've got a maximizer up at the top who's trying to get the highest score possible. The minimizer is her opponent. And the minimizer is trying to get to the lowest score possible. And it's really unclear who wins or loses at each point. They're just trying to get it to the highest or the lowest score.

All right, so let's do a refresher. Hopefully the quiz didn't put people into such panic modes that they forgot Monday's lecture. So let's make sure that we can do regular minimax algorithm on this tree and figure out the minimax value at A.

So let's see how that works. All right, as you guys remember, the game search when using regular minimax is essentially a depth first search. And at each level, it chooses between all of the children whichever value that the parent wants. So here after it would choose the maximum of K and L, for instance. But that's getting ahead of ourselves. Because it's a depth first search. So we best start at the top.

I'll help you guys up for a while. So we're doing A. We need the maximum of B, C, D, depth first search. We go to B. We're looking for the minimum of E and F. So having looked at E, our current minimum of E and F is just 2 for the moment. So this is going to be less than or equal to 2.

All right, then we go down to F, which is a maximizer. And its children are K and L. So now I'm going to start making you guys do stuff. So what do you think? What is going to be the minimax value at F? The minimax value at F, what will that be?

AUDIENCE: [INAUDIBLE]

PROFESSOR: So that level is a maximizer-- max, min, max. F is a maximizer. K and L themselves are minimizers. But they're pretty impotent minimizers. Because they don't get to choose. They just have to do K or L. So the minimax value is three. And yeah, the path it would like to go is K. So we'll say that the minimax value here is 3. It's in fact exactly equal to 3. So if this is 3 and this is 2, then everyone, we know that the value of B is?

AUDIENCE: [INAUDIBLE]

PROFESSOR: I hear 3 and 2. Which one is it?

AUDIENCE: 2.

PROFESSOR: 2, that's right. So the value here is 2. Great, let's go down into this branch. So C is going to be the minimum of G and 6. But we don't see that yet. Because we're doing a depth first search. It's going to be the minimum of G. Now we need the maximum of M and N. We're going to need the minimum. M is the minimum of Q and R. So let's switch sides. The minimum of Q and R is?

AUDIENCE: 1.

PROFESSOR: Let's see. That's right, it's 1. So M has a value of 1. But I'm going to stay over here. Because M has a value of 1. Knowing that, then we know that G has a value of?

AUDIENCE: 7

PROFESSOR: That's right. 7 is higher than 1. And since G is a 7, we now know going up to C that C has a value of?

AUDIENCE: 6.

PROFESSOR: Yes, C has a value of 6. That's the minimum 6 and 7. So now I'm going to go back down. Because we've done one of the other sub-trees. This is a 6. All right, great. Now we're going to go down to D. Hopefully it won't be too bad.

These things usually aren't terrible. Because they're made to be pruned a lot in alpha-beta. So let's see, in D, we go down to I. And that's just a 1. We go down to J. And let's see, what's the minimax value of J?

AUDIENCE: It's 20.

PROFESSOR: That's right. 20 is the maximum of 20 and 2. Great, so what's the minimax value of D? Everyone said it-- 1. All right, so what's the minimax value at A?

AUDIENCE: 6.

PROFESSOR: 6 is right. 6 is higher than 2. It's higher than 1. Our value is 6. And our path is-- everyone-- A, C, H. That's it. Great, is everyone good with minimax? I know that usually a lot of people are. There's usually a few people who aren't. So if you're one of the people who would like some clarifications on minimax, raise your hand. There's probably a few other people who would like some too. OK.

AUDIENCE: When you're doing this minimax, whatever values are not showing, you keep going down the tree and then just look at whether you're trying to find the minimax. And just whatever values you get you go back up one?

PROFESSOR: Yes. The question was, when you go to do the minimax-- and let's say you got E was 2, and you know that B is going to be less than or equal to 2, but you don't know F yet. The question is, do you go down the tree, find the value at F, and then go back up? The answer is yes.

By default, we use a depth first search. However, in non alpha-beta version, just regular minimax, it turns out it probably doesn't matter what you do. I suggested doing a depth first search to get yourself in the mindset of alpha-beta. Because order is very, very important in alpha-beta.

But here, I don't know, you could do some weird bottom up search. Whatever you want, it's going to give you the right answer unless it asks what order they evaluated. But here's a hint. The order they're evaluated in is depth first search order. So without even doing anything, E, K, L, Q, R, N, H, I, O, P are the order of starting evaluation in this tree.

AUDIENCE: [INAUDIBLE]

PROFESSOR: So the question is, nodes like M and G we don't have to put values next to. Technically, if we were doing this very formally, and we couldn't remember, and I wasn't up there among the people, we would put 1 there. So at M, we would put a 1. But people remembered that. So we didn't do it.

But then at G, we would put a 7. So if we were writing it out very formally, we would have a 1 and a 7. And at this D, we would have a 1. And then at the A, we would put a 6. And then that's the answer.

Also, we've even put things like less than or greater than part way along the way. However, I believe that our alpha-beta search is going to definitely fulfill everyone's quota of pedantically putting lots of numbers next to nodes on a game tree. And so once you've done alpha-beta, if you can do it correctly, you'll think, oh, minimax, oh, those were the days. It's going to be easy. Because alpha-beta is a little bit more complicated.

There's a lot of things that trip people up here. For alpha-beta, however, I will erase some of these numbers for the moment. They're still right. But we do it a little differently.

So what do alpha-beta and beta add to this formula? Well, this is all sort of a winning formula, except for it's not. Because it takes too long. But it's a very nice formula. U is the maximizer, say. I would try to think, if I do this, what's he going to do? And then if he does that, what am I going to do? And then what is he going to do if I do that, et cetera, et cetera, all the way to the bottom. With alpha and beta, we add in what I like to call nuclear options.

I'd like in this game of maximizer minimizer-- you can think of it as like the Cold War or the Peloponnesian War, except the Peloponnesian War didn't have nukes, so probably the Cold War. And in the Cold War, or any situation where you're up against an adversary who-- actually, this doesn't really work as well for the Cold War. But in any situation where you're up against an adversary whose only goal in life is to destroy you, you always want to find out what the best thing you can possibly do is if they hit that button and send nukes in from Cuba, or if they send fighter pilots, or whatever is going on.

So the idea of alpha and beta is that they are numbers that represent the fail-safe, the worst case. Because obviously in the Cold War, sending nukes was not a good plan. But presumably, us sending nukes would be better than just being attacked and killed.

So the alpha and beta represent the worst possible outcome you'd be willing to accept for your side. Because right now, you know you're guaranteed to be able to force the conflict to that point or better. So the alpha is the nuclear option, the fail-safe, of the maximizer. Nuclear options-- alpha is maximizer's nuclear option. And beta is the minimizer's nuclear option.

So we ask ourselves-- and people who were paying attention at lecture or wrote stuff down know the answer already-- what could we possibly set to start off? Before we explore the tree and find anything, what will we set as our nuclear option, as our sort of fail-safe? We could always fall back on this number.

So you could set 0. You could try to set some low number for the maximizer. Because if you set a high number for the maximizer as its fail-safe, it's going to be really snooty and just say, oh, I won't take this path. I already have a fail-safe that's better than all these paths. If you set, like, 100, you have no tree.

Our default usually, in 6034, is to set negative infinity for alpha or negative some very large number if you're doing it in your lab. So if we set negative infinity as a default for alpha, that negative infinity is basically maximizer loses. So the maximizer goes in thinking, oh my god, if I don't look at this game tree, I automatically lose. He's willing to take the first path possibly presented. And that's why that negative infinity is a good default for alpha. Anyone have a good idea what a good default for beta is, or just remember?

Positive infinity, that's right. Because the minimizer comes in, and she's like, oh crap, the maximizer automatically wins if I don't look at this node here. That makes sure the maximizer and the minimizer both are willing to look at the first path they see every time. Because look on this tree. If 10 was alpha, the maximizer would just reject out of hand everything except for P. And then we wouldn't have a tree. The maximizer would lose, because he would be like, hmm, this test game is very interesting. However, I have another option-- pft, and then you throw over the table.

That's 10 for me, because you have to pick up the pieces. I don't own this set. I don't know. So that is why we set negative infinity and positive infinity as the default for alpha and beta. So how do alpha and beta propagate? And what do they do? The main purpose of alpha and beta is that, as we said, alpha-- let's say we have some chart of values. Alpha, which starts at negative infinity, is the worst that the maximizer is willing to accept. Because they know they can get that much or better.

It starts out, that's the worst thing you can have. So it's not a problem. Infinity is the highest that the minimizer is willing to accept. That's beta. As you go along, though, the minimizer sees, oh, look at that. I can guarantee that at best the maximizer gets 100. Haha, beta is now 100. The maximizer says, oh yeah? Well I can guarantee that the lowest you can get me to go to is 0. So it's going to be 0.

And this keeps going on until maybe at 6-- note, not drawn to scale. Maybe at 6, the maximizer said, haha, you can't make me go lower than 6. And the minimizer says, aha, you can't make me go higher than 6. And then 6 is the answer.

If you ever get to a point where beta gets lower than alpha, or alpha gets lower than beta, then you just say, screw this. I'm not even going to look at the remaining stuff. I'm going to just prune now and go somewhere else that's less pointless than this. Because if the alpha gets higher than the beta, what that's saying is the maximizer says, oh man, look at this, minimizer. The lowest you can make me go is, say, 50. And the minimizer says, that's strange. Because the highest that you can make me go is 40.

So something's generally amiss there. It usually means that one of the two of them doesn't even want to be exploring that branch at all. So you prune at that point.

All right, so given that that's what we're looking for, how do we move the alphas and betas throughout the tree? There's a few different ways to draw them. And some of them I consider to be very busy. Probably in recitation and tutorial you will see a way that's busier and has more numbers.

Technically, every node has both an alpha and a beta. However, the one that that node is paying attention to is the alpha, if it's a maximizer, and the beta if it's a minimizer. So I generally, for my purposes, only draw the alpha out for the maximizer and only draw the beta out for the minimizer. Very rarely, but it happens, they'll sometimes ask you, well, what's the beta of this node, which is a maximizer node? So it's good to know how it's derived. But I think that it wastes your time to write it out. That's my opinion. We'll see how it goes.

So the way that it works, the way that alpha and beta works, is the Snow White principal. So does everyone know the story of Snow White? So there's a beautiful princess. There's an evil queen stepmother. Mirror mirror on the wall, who's the fairest of them all, finds out that it's the stepdaughter. So much like in the real world, in *Snow White*, the stepdaughter, Snow White, had the beauty of her parents. She inherited those.

However, much like in the real world, maybe or perhaps not, the stepmother had an even better plan. She hired a hunter to sort of hunt Snow White, pull out Snow White's heart, and feed it to her so that she could gain Snow White's beauty for herself. How many people knew that version of the story? A few people. That's the original version of the story. Disney didn't put that in.

The hunter then brought the heart of a deer, which I think in Disney the hunter did kill a deer arbitrarily, but it was not explained that that's why he was doing it. So in alpha-beta, it's just like that. By which I mean you start by inheriting the alpha and beta of your parents. But if you see something that you like amongst your children, you take it for yourself-- the Snow White principle.

So let's see how that goes. Well, I told you guys that the default alpha was--

AUDIENCE: Negative infinity.

PROFESSOR: Negative infinity. So here alpha is negative infinity. And I told you that the default beta was positive infinity. We're doing a depth first search here. All right, beta is infinity. All right, so we come here to E. Now, we could put an alpha. But I never put an alpha or a beta for one of the terminal nodes. Because it can't really do anything. It's just 2.

So as we go down, we take the alpha and beta from our parents. But as we go up to a parent, if the parent likes what it sees in the child, it takes it instead. So I ask you all the question, would the minimizer prefer this 2 that it sees from its child or its own infinity for a beta?

AUDIENCE: 2

PROFESSOR: It likes the 2. That's absolutely right. So 2. All right, great, so now we go down to F. What is F's alpha? Who says negative infinity? Who says 2? No one-- oh, you guys are good. It's negative infinity.

Technically, it also will have a beta of 2. But we're ignoring the beta. And the alphas that have been progressing downward from the parents-- negative infinity. That's why I called it the grandfather clause before. Because you would often look up to your grandparent to see what your default number is. So we get an alpha of negative infinity. We then go down to the K. It's a static evaluation.

And now I'm going to start calling on people individually. So hopefully people paid attention to the mob, who were always correct. All right, so we go down to K. And we see a 3. F is a maximizer node. So what does F do now?

AUDIENCE: Switches its alpha to 3.

PROFESSOR: Yes, switches its alpha to 3, great. All right, so that's already quite good. It switches alpha to 3. It's very happy. It's got a 3 here. That's a nice value. So what does it do at L, the next node? It's gone to K, went back up to F. Depth first search, the next one would be L, right?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Well, technically F could take L's value of 0 if it liked it better than 3. But it's a maximizer. So does it want to take that?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK, that technically would be correct. But I'm sorry. I burdened you with a trick question. In fact, we don't look at L at all. Does everyone see that? I'll explain.

The alpha at F has reached 3. But the beta at B is 2. So B looks down and says, wait a minute. If I go down to F, my enemy's nuclear option, my enemy is the worst it can be for-- the best it can be for me is 3. F is trumpeting it around. I was thinking of eating his heart, or whatever, but I didn't want to. But it's going to be 3. It's going to be 3 or higher down there at F.

There's no way I want that. I already have my own default escape plan. And that's 2. That's going to be better than whatever comes out of that horrible F. So screw it. And we never look at L. Does everyone get that? That is the main principle of alpha-beta pruning. If you see an alpha that's higher than the beta above it-- as I said, if alpha goes up above the beta-- or if you see a beta, like if there's a beta down here, and it's lower than the alpha above it, prune it. Stop doing that.

And the question is, who prunes? Who decides that you don't look at L? The person who is thinking not to look at L is always up higher by at least two levels. So up here, B is saying, hmm, I don't want to look at L. Because F is already so terrible for me that it's just beyond belief. If this is 100, it might be 100. Even if it's lower, I'm still going to get a three.

There's a sanity check that I've written that I sort of came up with just in case you're not sure that you can skip it. Because on a lot of these tests, we ask you, which one's do you evaluate, which ones do you skip, right? Or we just say, which ones do you evaluate, and you don't write the ones that you skip.

Here's my sanity test to see if you can skip it. Ask yourself, if that node that I'm about to skip contained a negative infinity or some arbitrarily small number, negative infinity being the minimizer wins, would it change anything? Now that I've answered that, if it contained a positive infinity, would it change anything?

If the answer is no both times, then you're definitely correct in pruning it. So look at that 0. If it was a negative infinity, minimizer wins, what would happen? The maximizer would say, I'm not touching that was a 10 foot pole, choosing 3. The minimizer would say, screw that, I'll take E.

Let's say it was a positive infinity. The maximizer would say, eureka, holy grain, I win. The minimizer would say, yeah, if I'm a moron, and go down to F, and then would go to E and take 2. So no matter what was there, the minimizer would go to E.

And you could say, well, what if it was exactly 2? But still the maximizer would choose K. The minimizer would go to E. So there's no reason to go down there. We can just prune it off right now. Does everyone agree, everyone see what I'm talking about here?

Great, so we're now done with this branch. Because beta is 2. So now we're up at old grandpappy A. And he has an alpha of negative infinity. Everyone, what will he do? He'll take the 2. It's better than negative infinity for him. It's not wonderful. But certainly anything is better than an automatic loss.

All right, now our highest node is a 2. So let's keep that in mind for our alpha. OK, so let's go over here. Let's see, so what will be the value at C? What will be the beta value?

AUDIENCE: [INAUDIBLE]

PROFESSOR: You go back to which one? To G. I'm not at G yet. I'm actually just starting the middle branch. So I'm going to C. And what's going to be its starting beta before I go down?

AUDIENCE: Infinity.

PROFESSOR: Infinity, that's right-- default value. It's easier than it seemed. All right, so yes, beta is equal to infinity. This should be better erased. I think it's confusing people. Great, OK, so beta is equal to infinity at C. Now we go down depth first search to G. What's going to be our alpha at G?

AUDIENCE: Minus infinity.

PROFESSOR: Ahh, it would seem so. However, take a look up at the great-grandpappy A. It seems to have changed to 2. So this time it's 2. Why is it 2 instead of negative infinity? Why can we let A be so noxious and not start with saying, oh, I automatically lose? Well, A knows that no matter how awful things get in that middle branch, he can just say, screw the whole middle branch. I'm going to B.

That's something that the minimizer can't do. And we have to start at infinity for the minimizer. But the maximizer can. Because he has the choice at the top. Does everyone see that? He can just say, oh, I'm not even going to C. Yeah, shows you. I'm going to A and taking the 2. So therefore alpha is actually 2 at G.

All right, great, so we've got an alpha that's 2 at G. We're going to go down to M. It's a minimizer. All right, what's going to be our beta value at M?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Or which is the beta default, minus or positive infinity? What would be the minimizer?

AUDIENCE: Positive.

PROFESSOR: Positive infinity, that's right. M is going to be a positive infinity for beta. Again, it picks it up from C. Great, now we get to some actual values. So we're at some actual values. We are at Q. So what's going to happen at M when M sees that Q is 1?

AUDIENCE: [INAUDIBLE]

PROFESSOR: What is beta? It says infinity. I'm sorry, it's hard to read. Beta is infinity at M.

AUDIENCE: OK, so it's going to minimize, right? So it's going to be like, OK, [INAUDIBLE].

PROFESSOR: That's right. So they're going to put beta to 1. Because it sees Q. Great, so my next question is, what's going to happen at R?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Very smart. You've detected my trap. The question is, does it look at R? The answer is, no. It doesn't look at R. Why doesn't it look at R? Does everyone see? Yeah, alpha is now greater than the beta below it. Beta has gotten lower than alpha.

This is the same thing I was talking about before, when we figured out that the alpha here is 2. The maximizer says, wait a minute. The maximizer G says, if I go to M, the best I'm getting out of this is 1. Because if this is negative infinity, the minimizer will choose it. If this is positive infinity, he'll choose 1. The best I'm going to get out of here is 1.

If that's the case, I might as well have just gone to B and not even gone to C. So I'm not going to go to M. I'll go to N, maybe. Maybe N is better. Does everyone see that? Great, so let's say that the maximizer does go to N. So what's going to happen with this alpha?

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's right, it's going to be 7. 7 is better than 2. And the maximizer has control to get to that seven, at least if it gets to G. All right, now the minimizer at C-- we'll do everyone this time. The minimizer at C, seeing that 7, what does the minimizer do? Anyone?

So it sees the 7. What does it do to its beta? It takes the 7-- better than infinity, anyway. And yeah, then it checks H. And everybody, again, what happens at H? It takes the 6. It's lower than 7.

All right, now we'll go back to having people do it on their own. Well, all the way back to the top, what does A do when it sees the 6 coming out of C?

AUDIENCE: Changes to 6.

PROFESSOR: Changes to 6, that's right. Alpha equals 6. Great-- homestretch, people, homestretch. So the minimizer, everyone, has a beta of infinity. And if I wasn't a static node, it would have an alpha of 6. But it is a static node. So it just has a value of 1.

So since it has a value of 1, everyone, the beta becomes 1. And what next, everyone?

AUDIENCE: Prune.

PROFESSOR: Prune, that's right. Why prune? Well, this time it's A himself who can prune. A says, well darn, if I go to D, I'm going to get 1 or something even worse than 1. I might as well take my 6 while

I have it, prune all the rest all the way down.

Everyone see that? Everyone cool with that? It's not too bad if you take it one step at a time. We did it. Our question is, which nodes are evaluated in order? Our answer is, everyone-- E, K, Q, N, H, I. OK, not so obvious, I guess. A few people followed me. But it is E, K, Q, N, H, I. It's just depth first order. And we pruned some of them away.

Great, so that is alpha-beta. Any questions about that before I give some questions about progressive deepening? All right, we've got a bunch. So first question.

AUDIENCE: [INAUDIBLE] nodes like F, B, C, and D?

PROFESSOR: The question is, when asked for the order of evaluation, are we excluding F, B, C, and D? The answer is we're talking about here static evaluation. The static evaluator is a very important and interesting function. And I'll get back to something a few students have asked me about the static evaluator later and try to explain what it is. It's basically the thing that pops out those numbers at the bottom of the leaves.

So when we ask, what is the order of nodes that were statically evaluated, we mean leaves only. That's a good question. Any other questions? Let's see, there was one up here before. But it's gone. It might have been the same one. Question?

AUDIENCE: So a similar question. When you say, static nodes, that just means the leaf nodes?

PROFESSOR: Means the leaf nodes, that's right. The question is, does static nodes mean the leaf nodes. The answer is yes.

AUDIENCE: And so static evaluation is when you compare the value of a static node to something?

PROFESSOR: Static evaluation is when you get that number, the static node. Let me explain. Unless someone else has another question about alpha-beta, let me explain static values. Because I was about to do that. There is a question about alpha-beta. I'll come back to both of yours after I answer this.

AUDIENCE: You were mentioning [INAUDIBLE]. And I'm a little bit confused. If you're looking at one node, and you're seeing either grab the value from the grandparent or grab it from the--

PROFESSOR: So it always starts-- the question is, what is the Snow White principle? How does it work? Every node always starts off with taking the value of the same type, alpha or beta, from its grandparent. It always starts that way. Now, you say, why the grandparent? Wouldn't it take it from the parent? It actually does. But I'm not drawing out the alphas at all the minimizer levels. Because they don't do anything. They're only even there to pass them down.

So all of the values pass down, down, down, down, down to begin. Every node, in fact, starts off with its grandparents with its parents' values, OK? But then when the node sees a child, it's completely done evaluating. It's finished. It can't be in the process.

Let's say C. When C sees that G is completely done with all of its sub-branches and is ready to return a value, or if it's just a static evaluation, then it's automatically completely done. Because it has no children.

A static value like K of 3 is automatically completely done. It's got a 3. Similarly, when we came back to G after going to N, and we knew that the value was 7, that was completely done. The value was definitely 7. There was no other possibilities.

AUDIENCE: That's after looking at the children, right?

PROFESSOR: Yes. So once you're done with all the children of G, then G comes up and says, guess what? Guess what, guys? So technically before that, you would have said that G's alpha is greater than or equal to 1 when we looked at Q. And then we looked at M. We'd say, it's equal exactly to 7. We're done here.

And then at that point, when it's fresh and ripe and has all of its highest value or its best value, that's when the parent can eat its heart and gain that value itself. So that's when C says, for instance, oh man, I have an infinity. I really like that 7 better. And it takes the 7.

But then it saw H. And it said, oh man, that's a 6. That's even better than 7. So it took the 6.

AUDIENCE: So shouldn't the alpha take 7 then?

PROFESSOR: So alpha takes 6. Because C is a minimizer. C took the 7 from G, but then right after that C saw H and took the 6. Because 6 is even lower than 7. And then alpha took the 6. Because 6 was higher than 2.

AUDIENCE: So it's not going to look below the branch?

PROFESSOR: Yeah, the problem is that the maximizer doesn't have control there. The minimizer has got control at C. And the minimizer is going to make sure it's as low as possible. The maximizer at

A, his only control, or her only control, is the ability to send either way to B or C or D. And then at that point, at C, the minimizer gets to choose if we go to G or H. And it's never going to choose G. Because G is higher than H. All right, awesome, was there another question?

All right, let's go back to static evaluations. When I first took this class, I had some weird thoughts about static evolutions. I heard some students ask me this. I almost got a question about it onto one of the tests, but it was edited to some other weird question that was m to the b to the d minus 1 or something like that at the last minute. So I'm going to pose you guys the actual question that would have been on one of the older test, which is the following.

I had a student who came to me and said, you know, [INAUDIBLE], when we do this alpha-beta pruning, and all this other stuff, we're trying to assume that we're really saving that much time by getting rid of a few static evaluations. In fact, when we do progressive deepening, we're always just counting, how many static evaluations do we have to do?

And he said, I look at these static evaluations. And there's just a 3 there. It takes no time to do the static evaluation. It's on the board. It takes much longer to do the alpha-beta. It's faster by far to not do alpha-beta. So I then tried to explain to that student. I said, OK, we need to be clear about what static evaluations are. You guys get it easy. We put these numbers on the board.

A static evaluation-- let's say you're playing a game like chess. Static evaluation takes a long time. When I was in 6170, Java [INAUDIBLE], the class that used to exist, we had a program called Anti-Chess where I used my 6034 skills to write the AI. And the static evaluator took a long time. And we were timed.

So getting the static evaluator faster, that was the most important thing. Why does it take a long time? Well, the static evaluator is an evaluation of the board position, the state of the game, at a snapshot of time. And that's not as easy as just saying, oh, here's the answer.

Because in chess, first of all, not only did I have to look at how many pieces I had, what areas that I controlled. Also-- well, it was anti-chess. But that's not withstanding. Let's pretend it's regular chess. I also had to look, if it was in regular chess-- and I still had to do this in anti-chess-- if my king was in check.

And what that meant is I had to look at all of my opponent's moves, possible moves, to see if anyone of them could take my king. Because in regular chess, it's illegal to put your king into check. So you better not even allow that move. And regardless, getting into checkmate is negative infinity for you.

So it takes a really long time to do static evaluations, at least good ones, usually. You want to avoid them. Because they're not just some number on the page. They are some function you wrote that does a very careful analysis of the state of the game and says, I'm good to heuristically guess that my value is pi, or some other number, and then rates that compared to other states. Does that make sense to everyone?

So the answer to the hypothetical question that might have been on the old test, when the person said, I've got this great idea where you do tons of static evaluation, and you don't have to do this long alpha-beta, is, don't do that. The static evaluations actually take a long time.

Does that clear it up for people who asked me before about what is a static evaluation, why are the leaf nodes called static? And you might ask, why are some of these static just arbitrarily? The answer is, when you're running out of time to expand deeper, and you just need to stop that stage of the game-- maybe it's just getting too hairy, maybe it's spreading out too much, you have some heuristic that says, this is where I stop for now-- it's a heuristic guess of the value.

It's kind of like those heuristic values in the search tree. It's a guess of how much work you have left to get to the goal. Here, you say, well, I wish I could go deeper. But I just don't have the time. So here's how I think I'm doing at this level. It's not always right.

And that's going to lead us into the answer to one of the questions about progressive deepening. So I'll put up the progressive deepening question really quickly. So the question is this. Let me see, this is a maximizer-- yes. Suppose that we do progressive deepening on the tree that is only two levels deep. What is progressive deepening in a nutshell if you don't remember from the lecture?

The idea is this. In this tree, it doesn't work. But in trees that actually branch like 2 to the n, it doesn't take that much time to do some of the top levels first and then move on to the bottom levels. Just do them one at a time.

So let's say we only did it up through J. We only did the top two levels of the tree. We'd like to reorder the tree so that alpha-beta can prune as much as it possibly can, at least we hope. So let's pretend that we had a psychic awesome genius friend who told us that the static values when we went up to two levels-- remember, when we go to two levels, F, G, and J have to get a static value, right? Because we're not going down. We do a static evaluation. They get the exact correct numbers-- 3, 7, and 20. Genius, brilliant.

All right, so if that happens, what is the best way that we could reorder that tree? Oh yeah, so it's A, B, C, D with values of 2, 3, 7, 6, 1, 20. I'll draw that. This is the non-reordered tree.

Let's see, so it's 2, 3, 7, 6, 1, 20. So what's the best way to reorder? Well, first of all, does anyone remember what Patrick said when he talked about progressive deepening? Usually no one does, so don't worry about it. Because at that time you guys didn't think, oh, I have to do this for the quiz. You were just thinking, oh man, we've already heard alpha-beta and all this other stuff. And this is just a small fact. But it's a very important fact.

And now you know you have to do it for the quiz. So you're probably going to remember it. The way you do it is you try to guess, and you say, which one of these is going to be a winner? Whichever one I think is going to be a winner at that level, I put first. Why is that the case?

Well, something interesting you may have noticed here-- whenever you have a winner, like the middle node, or whenever you have whatever is the current best for your alpha, you sort of have to explore out a lot of that area. Like for instance, the left node was our current best at 2. The middle branch was our current best, at that time was 6. It was the total best.

We had to explore a good number of nodes. But on the right, we just saw, oh, there's 1. We're done. We cut everything off. In other words, the branch that turns out to be the one that you take, you have to do a pretty good amount of exploration to prove that it's the right one. Whereas if it's the wrong one, you can sometimes with just one node say, this is wrong, done.

So therefore, if the one that turns out to be the eventual winner is first of all, then it's really easy to reject all the other branches. Do people see that sort of conceptually a little bit, that if you get the best node right away, you can just reject all the wrong ones pretty quickly? That's our goal.

So how can we, quote, "get the right one," the best one right away? Well, here's how we do it. Let's say we're at B. Which one is the minimizer likely to pick assuming that our heuristic is good and that these guesses are pretty much close to the truth? It turns out they're perfect, so this is going to work. So which one will the minimizer pick if it has to choose between E and F, do we think?

AUDIENCE: E.

PROFESSOR: E, perfect. Which one will it pick between G and H?

AUDIENCE: H.

PROFESSOR: H. Which one will it pick between I and J?

AUDIENCE: I.

PROFESSOR: OK, so what we're saying is we think it's going to pick E. We think it's going to pick H. We think it's going to pick I. So first of all, we should put E before F, H before G, and I before J. Because we think it's going to pick those first. Those are probably our best ones to invalidate a poor branch. So now between 2, 6, and 1, which is what we think we're going to get, which one do we think the maximizer is going to take?

AUDIENCE: 6.

PROFESSOR: 6. Then if it couldn't take 6, what would be its next best choice? 2, then 1. That's just our order-- simple as that. It couldn't be anything easier that evolves really complex trees, a huge number of numbers, and reordering those trees.

So C-- you guys told me C, B, D. You told me C, B, D, I think? Yeah, those are the ones the maximizer likes. And then the ones the minimizer likes you told me was H, and before G. Because H is smaller than G. You guys told me E before F. And you guys told me I before J. And you guys would be correct in all regards.

We have 6, 7, 2, 3, 1, 20. All the minimizers choose from smallest to highest. The maximizer chooses from highest to lowest of the ones that the minimizers will take. And if we did that, you can see we would probably save some time.

Let's see how much time. Let's say we looked at H first. Well, if we looked at H first, we would still have actually had to look at Q and N. However, we would not have had to look at K. Do people see why? If we already knew this branch was 6, as soon as we saw 2 for the beta here-- 2 is less than 6-- we could have pruned. We still would have had to look at I over here. Because you have to look at at least one thing in the new sub-branch.

And it actually only would have saved us one node-- oops. So it winds up that in total, how many nodes would we have evaluated if we did that little scheme of reordering? Well, we normally had to do six-- E, K, Q, N, H, I. How many do we evaluate if we do this progressive deepening scheme? How many times do we run the static evaluator, which of course you know the static evaluator takes a long time? Anyone have a guess?

I told you the only one we don't evaluate is K. Raise your hand. I won't make anyone give this one. So I said the only one we save on is K. So we still do E, Q, N, H, and I over here. There's two possible answers that I will accept. So you have a higher chance of guessing it. Anyway?

Does everyone agree that we did six before? If we didn't do any progressive deepening, we just did E, K, Q, N, H, I. And now we're not doing K. OK, people are saying five. All right, good. That's not the right answer. But it at least shows that you can do taking away the one. We did at least five over here.

There's two possible answers, though. Because look over there. In order to do the progressive deepening, we had to do those static evaluations, right? So we either did all those static evaluations and these five-- E, K, Q, N, H, I-- static evaluations. Because we didn't do the K.

Or we might have saved ourselves. Because maybe we were smart and decided to cache the static values when we were going down the tree. It's an implementation detail that on this test when we asked that question we didn't say. What I mean by cache is when we did it here and saw that E was a 2, and then here-- oh, we have to do the static value at E. If we were smart, we might have made a little hash table or something and put down 2 so we didn't have to do a static evaluation at E. And if that happened, well, we save E, H, and I, and we do three fewer. Does everyone see that?

However, that's still more than six. So it didn't save us time. So you might say, oh, progressive deepening is a waste of time. But it's not. Because this is a very, very small, not very branchy tree that was made so that you guys could easily do alpha-beta and take the quiz, and it wouldn't be bad. If this was actually branching even double at each level, it would have, what, 16 nodes down here at the bottom. Then you would want to be doing that progressive deepening.

So now I ask you a conceptual riddle question. It's not really that much of a riddle. But we'll see if anyone wants to answer. Again, I won't call on you for this. According to this test, a student named Steve says, OK, I know I have to pay to do the progressive deepening here. But let's ignore that. Because it's small in a large tree, right? It's not going to take that much.

Let's ignore the costs of the progressive deepening and only look at how much we do here. He says, when it comes to performing the alpha-beta on the final level, I'm guaranteed to always prune at least as well or better if I rearrange the nodes based on the best result from progressive deepening. Do you agree?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Can I repeat it? OK, the question is, ignoring the cost that we pay progressively deepening here-- just forget about it-- at the final step, at the final iteration, the question is, am I guaranteed to do at least as well or better in my alpha-beta pruning when I reorder based on the best order for progressive deepening? Here certainly we did. But the question is, is Steve guaranteed? Answer?

AUDIENCE: [INAUDIBLE]

PROFESSOR: What did you say?

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's the answer and the why, which we asked to explain. The answer we got is, doesn't that depend on the heuristic? Perfectly correct. The answer is, no, we're not guaranteed, and it depends on the heuristic. So if we were guaranteed, that would be our heuristic was godlike, like this heuristic.

If your heuristic already tells you the correct answer no matter what, don't do game search. Just go to the empty chess board, put all the pieces in the front rows, and run static evaluator on that. And it'll say, oh, it looks like with this game not started that white is stupid, so black will win in 15 turns. And then you're done. And you don't do a search.

We know that our heuristic is flawed in some way. It could be very flawed. If it's flawed so badly that it tells us a very bad result of what's actually going to happen, even though we think the minimizer is going to go to H, maybe it's wrong by a lot and it goes to G. It could take us up an even worse path and make us take longer. Question?

AUDIENCE: If it's the heuristic, how could you cache the values so you didn't have to recalculate them later?

PROFESSOR: The question is, how can you cache the values if it's a heuristic so you don't have to recalculate them later? The answer is, it wouldn't help if there weren't these weird multi-level things where we stop at E for some reason, even though it goes down to five levels. The way you could cache it is it is a heuristic.

But it's consistent. And I don't mean consistent from search. I mean it's a consistent heuristic in-- the game state E is, let's say that's the state where I moved out my knight as the maximizer, and the minimizer said, you're doing the knight opening, really, and then did a counterattack.

No matter how we get to E, or where we go to get to E, that's always going to be state E. It's always going to have the same heuristic value. It's not like some guy who goes around and just randomly pulls a number out of a hat. We're going to have some value that gives us points based on state E. And it's going to be the same any time we go to state E. Does that make sense? It is a heuristic. But it's always going to give the same value at E no matter how you got to E.

But it could be really bad. In fact, you might consider a heuristic that's the opposite of correct and always tells us the worst move and claims it's the best. That's the heuristic that the minimizer program did to our computer, perhaps. In that case, when we do progressive deepening and we reorder, we'll probably get the worst pruning possible. We might not. But we may.

So in that case, you're not guaranteed. I hope that's given a few clues. In tutorial, you guys are going to see some more interesting problems that go into a few other details. I at least plan on doing [INAUDIBLE] interesting game problem from last year, which asked a bunch of varied things that are a little bit different from these. So it should be a lot of fun, hopefully, or at least useful, to do the next quiz. So have a great weekend. Don't stress out too much about the quiz.