PROFESSOR: All right. You guys ready for some more games? Today, we will finish off two-player games in particular.

So last time, we were in this world of two-player games with a polynomial-bounded number of moves. We proved that Othello reversi is PSPACE complete. All these games live in PSPACE.

And we ended with constraint logic in the two-player case. We proved that's PSPACE complete. I'll just show you that again in a moment.

And then, in the rest of today, we'll talk about various other games that don't fit in this table, but in particular two-player and unbounded number of moves. So exponential number of moves, where you get x time completeness for constraint logic and for various practical games like checkers, chess, and go. But I won't prove all of this.

So this is bounded to player constraint logic. Remember, we could simulate these formulas, where here the setup was you can only flip each edge once. And in particular, players could take turns deciding whether to-- white player can set variables to true by flipping these edges. Black player can set variables false by flipping those edges. Then, if the formula satisfied, white wins. And that was hard and in particular constraint logic could simulate that problem.

So what I want to show you next is three real-life games that can be reduced from bounded to player constraint logic. And for those, we'll need two additional facts. One is that planar-bounded two-player constraint logic is PSPACE complete with this crossover gadget. We just need the addition of a choice variable with three red edges, two in, one out. This is what we will need.

And also that we can make protected ORs, an OR where you're guaranteed at most one of these comes in by allowing an additional gadget which is a free edge. So only one of our constructions will use this. But in that setting, we can make a free edge and we need the protected OR. So they go together well . And so this is review from last class.

So first real game we're going to look at is called Amazons from the late '80s but became popular in the common material game theory world I think in the mid-2000s, at least that's when I heard about it. So the usual game is played on an 8x8 checkerboard-- chessboard, I should say-- with four black queens and four white queens. Those are the Amazons.

The Amazons, as you may know, carry bows and arrows. So a move in this game is to take a queen of your color, move it in a queen-like way. So in this example, we're moving the clean up to this position.

And then, you fire an arrow by another queen move from there. So from here, we're shooting an arrow to this position diagonally visible. You're not allowed to move through other players or through arrows. And you're not allowed to shoot through other players or through arrows.

So this the square is destroyed forever. So this is clearly a bounded game. Every move we destroy square. You have to do both of those things, move between by a non-zero amount and then shoot an arrow somewhere other than yourself. You're never allowed to land on a dead square.

So a typical game. After you play it for a while, it's really fun. You should try it. We'll end up the situation where you've partitioned the queens into independent components.

And so then, it's basically a bunch of Hamiltonian cycle problems, or longest path. Now, these guys have to walk around. It's a little bit weird, because you can shoot not exactly where you're landing. But in particular, you could shoot where you just came from. So if each of these components-- usually they're Hamiltonian or have a

Hamiltonian path-- and then with the queen moves.

And then, you just sum up the areas then, and whether white or black has more area is the winner. Because in general, once you can't move, you lose the game. The goal is to move last. So in general, your goal is to partition up the space so you have more territory.

So we're going to prove PSPACE completeness of this game using bounded two-player constraint logic. So here's the basic gadgets, in particular the idea of a wire. We're imagining this is an upward traveling signal.

And for it to travel up-- think of this as an edge reversing from downward pointing to upward pointing. If the A queen has to move down and also shoot down, so leaving two black squares, in that situation B can move down and shoot down one or move down two and shoot up one, either way consume these two squares but leave two for the next guy.

This is a parity shift. It also works if this guy moves down, shoots down, leaving these two squares. Then, B can move down and shoot diagonally and so on.

So that's our wire. That's how we're going to propagate the signal, because we only need single-use gadgets you can do a turn, A moves down, shoots down. B moves down, shoots down. C moves over, shoots diagonally. D moves over, shoots diagonally, and so on.

Variable gadget, so this is what we had on the bottom of the construction. This is a white or black edge. And we'll never have to connect black edges to anything, so they're just hanging out. Here, if black moves first here and obliterates these two squares, this wire's dead. If white moves first then, the white can activate the chain.

So that's going to simulate the usual setup, where white and black are taking turns, setting their variables the way they want them. And then, if white can succeed and satisfy the formula-- we have to get to AND and OR gadgets.

But if white can succeed and this guy can move down, then B can move to this place and shoot here, and then move over here and shoot into this giant room. And then, we'll spend tons of moves in here. And then, come back here, shoot there, back there, shoot there, ther, shoot there, and get to the other room

But if this guy does not move down first, this guy-- whichever way he goes-- he'll have to either destroy the room he is walking to or destroy the other room that he could have gone. So you only get half of the points if this is not activated. And we're going to set it up so that black has a bunch of other moves and some other room that's already been cut off. And it will be roughly equal to one of these rooms. And so only if you get both of them do you win. And because we can't control which way B goes, these two rooms have to be the same size. So you happen to win by a factor of 2 and the number of moves. But that's not essential to the proof, just for symmetry. So those are the basic gadgets.

Now, we have AND and OR and SPLIT and CHOICE. Usually, in NCL, we think of AND and SPLIT as being the same thing. But in this setting, especially in the bounded situation, they're really different. Remember, it corresponds to whether initially the red things are pointing in or the blue things are pointing in, red edges or the blue edge. And they behave quite differently in this setting.

So for the AND gate, these are the two inputs. If both of them move out of the way, then C can move here, shoot here. And D can move here and shoot there. And if you only move one of the two, you won't have enough room.

AUDIENCE: Wait. Why?

PROFESSOR: Why? If B moves out of the way, C can move and shoot here. But D only has one one square. If A moves out if the way, C can't move.

On the other hand, an OR gadget, these are the two inputs and this is the output. If either one of these moves, we sort of already have one square. We just need a second square for C to shoot into. It could be here or here. As long as one of them moves out of the way, we are happy.

As a CHOICE gadget, so this is three blues coming in but the orientations are different. So a CHOICE is we have an input coming up from the bottom. And we want to activate one of the two but only one.

So that corresponds to A moving out of the way leaving two squares. Either B can take it and hit those two squares or C can take it, but only one of them. So the same gadget does both. That makes sense because we're not really distinguishing between red and blue very much here.

SPLIT is a little bit trickier. Let's walk through it. A moves down. B moves over and shoots here. Then, C can move down here and shoot here. And then, D can move over here and shoot there.

Using our queen moves, so that leaves these four spaces unoccupied. Now, E can move down here and shoot there. F can move here and shoot there. And now, we've got the four squares necessary to trigger both H and G.

So that's a SPLIT. And that is Amazons. Therefore Amazons is PSPACE complete. Done. Question.

AUDIENCE: What is CHOICE needed for?

PROFESSOR: CHOICE is needed for the crossover gadget in particular.

Next game we'll look at-- these proofs are going to start to look similar. But I think it's helpful to see a few examples. This is Konane. It's a very old game-- we don't know how old-- invented by Hawaiian Polynesians. First documented in 1778 by Captain Cook. And this was a world championship. No. This is a regional championship in 2005 that I probably lost at. But we like to play it with go stones. That's not the original formulation. But you have two colors of stones on a square grid. And it's something like peg solitaire, but with two colors. So let me maybe show you a bigger image.

The type of move you can do, if you're white you can jump over a sequence of black stones. You can only jump over one at a time. So if I had two in a row, you wouldn't be able to jump it.

And furthermore, you're not allowed to change directions in a single move. In a single move, I could jump over any chain of black blank blank blank things. And they are removed. I captured them. And it has to be in a straight line. So that's the rules.

And of course, every move I make I capture somebody, so it's a polynomial-bounded number of moves. This is the general idea for a wire is that with white will be doing most of the action here. And so white will be jumping here. At this point, I have to change directions.

But luckily after I obliterate those two pieces, black can't do anything to me. I'm safe between these two black squares. So black will do something else, and then white is free to start jumping in the direction they want to. That will be critical.

Here is a variable where if black moves first in this scenario, it can capture the white thing and prevent it from activating this wire. If white moves first, they can activate the wire. So whoever goes there first sets the variable.

AUDIENCE: What's the start configuration for this game?

PROFESSOR: Oh, in the original game, the whole board is filled where the color of your pieces equal the color of the square of the checkerboard, except for the center two places are blank. But we're not considering it from the actual initial configuration. We're considering it from a half-played configuration. Because the original configuration, the answer is the function of the board size. And that's not enough information to be interesting from a complexity standpoint. Yeah?

AUDIENCE: Do you have any problems that have significant hardness results for sort of the standard start but generalized?

PROFESSOR: No. In general, none of those problems can be hard in the sense that we want, because they're sparse. They have very few inputs of a given size, so there is no such result. Unless you have a variable initial setup, like maybe games where you-- what do you call it-- draft your deck in the beginning. I mean, if you consider that part of the initial setup, then that could be much more interesting.

Right. Haven't done the OR and the CHOICE. So again if we-- let's do the OR. If we are activating from here, we can just keep going. It'd be bad to stop here, because then black could capture us. But we can go straight.

On the other hand, if we come from this position, we killed those two guys who were safe and then we can go up. And a CHOICE gadget would be coming-- the input for a CHOICE would be here. So you can jump here and then you have a choice whether to go up or down. So again, same gadget does OR and CHOICE.

And here's one gadget that does AND, SPLIT, and a SHIFT, which is convenient for various ways to use it. But what if what it really does is there's two traversal paths, input 1 to output 1 and input 2 to output 2. And they have to happen in that order.

If you try to activate path 2, you have to stop here. I mean, I guess you could jump here and then be captured that way. If you stop here, you'll be captured by that guy. So that's bad news.

But if we first do the input 1 traversal-- this is safe-- and then go to the output traversal, this guy's gone. And so then we can do input 2, pause, and then go up to output 2. So that's what the gadget does by itself.

If we ignore what happens out here, then we can think of this as an AND gate, where first this input has to come in, kill that guy, and then go off Neverneverland. And then, if the second input comes in, then we activate the output. So that would be the output of the AND gate. That's garbage. And these are the two inputs.

Now, there is an issue of you have to make this one trigger before this one. But in all these setups, we have these posits where the piece can wait. So if we have a turn gadget right before the input, it can just sit there and wait for the other input to the AND to activate. And then, this one can go ahead and activate the second one. So we're very flexible on timing. So we can make input 1 go before input 2 for AND.

For a SPLIT, we can just have a white token sitting here ready to go in that's in a safe position. And then, once this input comes in, it gets output there. Then, this one can trigger an output there. So we get both of the outputs for one input.

And for a SHIFT, we do both of these things. We treat this is garbage. And we have a safe white token here. So if this comes in, then that can go out. So we end up shifting by one. And that fixes any parity issues.

Cool. So that's Konane PSPACE complete.

AUDIENCE: How do you guarantee that black doesn't screw you?

PROFESSOR: By checking all the cases. In general, I mean, there's a question of what black is doing. Basically, within all these gadgets, we're preventing black from doing anything. So the main place where we care about black's motion is here.

So there's going to be an initial phase where everyone's setting variables. We're taking turns. And white's going to set half of them. Black's going to set half of them. Their black is screwing you, but in a controlled fashion.

After that, there's going to be a pile of extra moves that black can just do. And black will always be forced to do those extra useless moves. If we happen to satisfy the final output to the overall thing, then white will have a zillion free moves and then white will win. That's the idea.

But within these gadgets, as long as we don't do something intelligent like going here and then getting captured by black, and therefore destroying the whole construction. Then, white will lose, probably. We're always careful to go to positions that are safe. Black cannot jump us if we're there.

So by checking all those cases, you can guarantee black doesn't hurt you. I didn't mention what's black doing when we're not allowing it to play at all. Cool. So that's Konane.

Next, is a game called Cross Purposes. This is a fairly new game by Michael Albert who's a fun guy in the common material game theory scene.

So again we're going to play with go stones, but what you should imagine is that black stones are towers of two blocks. And then, you can push them over. So in this case, we took this tower, pushed it up, and so it occupied those two spots. But once it's pushed over, it's kind of dead. So we draw it in white. And then, for example, we could take this tower pop it over to the right and get those two.

Now, to make this a fun two-player game-- this would be a natural enough solitaire game. But to make it a two-player game, we're going to have two players, one called horizontal, which can only do this type of move, and another player called vertical, which can only do this type of move.

So what's fun here is that it's the same black pieces used by both players. But one of them can tip them left to right. And the other can tip them up or down. But again it's bounded, because once you top over a tower it's dead. So this is going to be PSPACE complete by reduction from bounded two-player constraint logic. So some-- question.

AUDIENCE: What's the goal of the game?

PROFESSOR: The goal the game is to move last. If you can't move, you lose. And that can be asymmetric. Some pieces are only going to be flippable up or down. A lot of games have that feature. It's called normal play, in fact, it's so common. That the last player to move wins.

So a wire is going to look something like this. In general, I think vertical is the player that we're rooting for, and we're trying to decide whether vertical can win. And we're going to set up horizontal-- unlike the previous game-- we're going to set up horizontal in a position that it never has any moves except the one that you give it right then.

So the idea is that as soon as we move A down, then horizontal has a move, namely the playing B. And then, we have a move and so on. So that will trigger. That better not stop at any point. We always want to set it up so when we're switching gadgets, it's our turn to move, because the horizontal player will never have another move to make. This is what it looks like after. I mean, things are just filling in if you've partially filled in this thing.

Here's how we make a variable. So here in the initial phase of the game, horizontal will have moves. Horizontal could play this, filling those two squares, preventing us from playing this wire. So whoever goes here first will claim the thing. And if it's vertical, then it triggers the true value.

We can also terminate a wire like this. This is always good for us. We can just trigger it whenever we want as vertical. So we need that for free edges, which we need for the protected OR.

So to clarify or to fix what I said before, for the initial n moves, horizontal and vertical are going to exchange variables, take turns picking them. At that point onwards, horizontal has no moves except the one right after-- we're forcing horizontal to always play the way we want.

So after this is done, then as soon as you move this, then horizontal has a unique move. So it will cooperate. Horizontal has to cooperate with us to fill in wires.

So here is a protected OR. So if we trigger this, then this triggers, then this triggers, then this triggers, then that triggers, and so on. So that's all good. The reason it's a protected OR-- the trouble if we activated this twice, so first we activate through here, and then we activate here, and then here. And then, as vertical, we move B down.

Now, horizontal has no move. So that would be bad news, if we sort of waste a trigger like that. And so that's why we need a protected OR, where only at most one of the two sides will trigger.

CHOICE gadget is the same, but rotated 90 degrees. And that makes a big difference, because horizontal and vertical are flipped. So input is here. So if we trigger this, and then this, and then this, and then this. Then, as vertical, we get to choose whether this one activates or this one activates. So it's important to make it the right player.

And here's the omega gadget that does AND, SPLIT, and SHIFT. If we try to activate input 2, these topple over. This topples down. And now, horizontal gets a choice. And horizontal's going to be mean and push B over and prevent this thing from activating, because it wants to prevent you from doing whatever you're doing.

Because eventually, you're going to get to a place where you win, where you have a single move and then horizontal doesn't have a move. So that's in horizontal's interest to block you.

But if you first did all of these toppling, topple this down, over down, over down, over down, over. Then, in particular, you put a white token here and so B won't be able to do that. So when you activate these guys, D will have to go over and out you go.

So and then, it's the same thing either putting a free wire in input 2 and/or ignoring the output from output 1, we get AND, SPLIT, and SHIFT. And I think that's it for Cross Purposes. Question.

AUDIENCE: I probably misinterpreted. Did you show TURN gadget?

PROFESSOR: I maybe did not show a TURN gadget. There's a TURN gadget. Good. I didn't talk about it. It works in the obvious way. But, yeah, it is definitely a little tricky to make sure you can do it. But it works.

AUDIENCE: Back to the normal play condition, if you use games different from the [INAUDIBLE] version of the rule set?

PROFESSOR: In general, those games are different when you can't move then you win. But from a PSPACE completeness perspective, it should be the same. It's really whether you are starting the quantifying sequence with an exists or a for all. And both end up with the class PSPACE.

So there may be a way to distinguish. But at our level of granularity, we're not distinguishing. Yeah.

AUDIENCE: Do you know whether it's PSPACE complete for arbitrary setups with only black starting tokens?

PROFESSOR: Oh. I see. No obstacles.

AUDIENCE: It's a more natural starting place, than starting with a--

PROFESSOR: Yeah. That's definitely not known. I think it would be cool to think about only black tokens, no initial whites.

All right. So that's all I want to say about PSPACE. I think we're done with PSPACE. We first did PSPACE here, then PSPACE here last class, and then PSPACE here.

So now we're going to move up to EXPTIME, 2 players, unbounded number of moves. EXPTIME turns out to be the right class.

Oh. I lied. I have one more PSPACE class, which I don't have slides for.

This is a short topic, but it's a cool one, I think something that deserves more study. But at the moment there are no natural games like real-life games that have been shown to fall into this class or to be complete in this class.

So this is stochastic games. The idea is you're playing against a random opponent. So you have two players, and one of the players is random. So it's hard to identify with randomness. So the other player is you. Your goal is to win the game.

And you know that your player's just flipping a coin every time and playing randomly, say uniformly or some distribution that's known to you. Think of games where you're rolling dice, or drawing cards, or there's some random aspect to the game.

How hard are these games? Well, what's been studied-- this is a [INAUDIBLE] paper-- is for bounded two-player stochastic games. The answer is PSPACE again.

So, for example, here's a game, a formula game, stochastic SAT. We want to know is there a move such that for a randomly chosen move-- I like this notation. We have exist and for all. Backwards R is a random quantifier. There exists an x3 such that a random x4, so this is uniform random over 0 and 1, the binary choice 01.

We want the probability-- this gives a distribution. We want the probability of the formula being satisfied being greater than a half. I'm guessing you could pump that up to be with high probability. But just deciding whether you could win by probability strictly more than half, is PSPACE complete.

So this is cool. And to me intuitively randomness should be weaker than an adversarial player, where you replace these with for all. But it turns out they behave roughly the same essentially by a probability amplification argument. It's a little too detailed to go into here.

And so far, there are no reductions from this to what I would call natural games. There are a few examples in the paper, but none around rolling dice.

I would love to apply this to like playing Tetris, or in the paper they mention backgammon as a candidate, something like that, where you'd like to get PSPACE completeness in a bounded setting. So throwing that out there is an interesting direction to study, but I'll end it there.

Now, we can go to EXPTIME, two players, unbounded game, adversarial player, no randomness. And we get EXPTIME completeness.

So we're going to start out with-- I guess, similar to what we do it down here-- I'm going to start with some formula games which are EXPTIME complete and then some graph games that are EXPTIME complete, then constraint logic game which is EXPTIME complete. And along the way will prove something some hardness.

There are fewer results here, in general, I would say. In particular, I don't have a good example of using constraint logic to prove a fun game hard, though they probably exist. We just haven't gotten there yet. So it's more open problems.

But there are a lot of cool base problems out there. And this field started by a paper by Stockmeyer and Chandra in 1979. And it's been the basis for pretty much all EXPTIME hardness proofs.

EXPTIME hardness is really cool, because it implies exponential time. Any problems EXPTIME hard cannot be solved faster than exponential time. Whereas all these other classes, we need to assume P does not equal NP or P is not equal to PSPACE. P we know does not equal EXPTIME.

So these games require exponential time. It's kind of fun. And I'm pretty sure the first results in this category were these games. So it's power of games.

So in general, we're going to have some formula, maybe a couple formulas. But definitely we have a variable assignment. And the state of the game is going to be a variable assignment and who's going to move next, player 1 or player 2, black or white. Again, we'll use that notation.

And we're going to start with some variable arbitrary variable assignment. That's part of the initial state of the game, the board if you will. And in general, you can set variables to 0 or 1 as many times as you like. That's what distinguishes this from bounded games. And all of these games are going to be partisan, a term we used last class, so meaning we're going to have black and white variables.

And I'm going to cheat a little bit. There's going to be one variable that's black and white, but only in one of the games. So we'll get there. So black player can only change the black variables. The white player can only change the white variables, except for how I'm going to cheat.

So let's start. These games are very creatively named. First one's called G1. And it's amazing how consistent the literature is about the naming of these games.

So in G1, I move my player is to set all the variables of your color. So if you're white, you're going to set all the white variables. If black, you're going to set all black variables. And set here means you get to choose whether it's 0 or 1. So you're just going to completely rewrite all of the variables of your color.

And then, here's the cheat. We're also going to have a single variable that's shared between black and white. Call it t. And it's going to be 0 if it's a player 2's move, 1 if it's player 1's move. I mean, it doesn't really matter. But I guess 1 and 1 is nice.

So that's the moves. What's the goal of the game? You're going to lose the game if, by making your move, you satisfy a common between the two players 4DNF formula.

So there's one formula in the game. It's DNF, which is very simple to think about. It's just a bunch of options. If this happens, you lose. If this happens, you lose. If this happens, you lose. It's an OR with various conditions.

So you want to move in such a way that you do not satisfy that formula every time. And in each move, you can rewrite all your variables. But this formula does involve t, which lets you care about who's turn it was.

So that's cool, very clean. I mean, the players have a lot of power here. This is the first game proved hard.

A little tricky to work with, because in most games you can't set all the variables at once. Generally, you're only setting one variable at a time. So all the other games are going to be setting one variable at a time. At this point, we're just defining. Then, we'll use these games to prove some hardness.

G2, the next game, is a move was going to be set one variable of your color. So in particular, if you set the variable to its original value, that's like passing. So passing is allowed in this game. Passing is also allowed here.

You could not change any of the variables. You wouldn't win in that case. You only win if your opponent loses in this setup. When the question is can you win. In this case, we're going to have a win condition.

In this case, white and black have their own winning formulas. They're in 12DNF, still DNF but a little bigger. And if you satisfy your formula at the end of your turn, you win and vice versa. And move is just change one variable at a time. So that's a very useful game.

Next is G3. A slight variation to G3, we're going to flip one variable of your color. And then, we're going to lose if you satisfy your 12DNF formula.

One thing to keep in mind. Losing and winning are quite different, because losing is essentially negating the formula, which turns DNF into DNF. So you can think of this as winning.

In the CNF condition, you could think of this as losing in a DNF condition. But they're not interchangeable.

So here we flipped between DNF and DNF. But we've also changed the definition of a move. A flip, you're not allowed to pass. You must change a variable to its opposite value. Here, we're allowed to just pass.

So we changed the move definition and the win condition. That is also hard. This is a common one. I would say most EXPTIME hardness proofs use this one.

G4. Move is going to be like this. You can set one variable of your color. And here, we're going to have a common win condition. This is nice. You win if satisfy a common 12DNF formula. Whoops. 13DNF. We need a little more.

So every move, we're both checking the same formula in this case. And here, the set happens to win. So it's a variation of G2.

G5 does not exist. G6-- there is a G5 in the paper, but it's a generalization of G6. So I'll tell you what is in a second. Not important.

So G6, same thing for the move. You set one of your variables to whatever you want. But we're going to make it asymmetric in the sense that-- and this will be very helpful for constraint logic-- player 1 wins-- player 1 is the one that we care about. We're trying to decide whether they win-- if anyone satisfies a particular formula.

So there's one CNF formula. If it's ever satisfied, player 1 wins. If it's never satisfied, you get a tie. And we want to know whether player 1 can force a win. So this is very compatible with this sort of thing we were doing before with it was the game problem from last class, but in particular the constraint logic games.

So we'll use G6 for constraint logic. G3 is most other proofs. So these are all EXPTIME complete formula games. Yeah?

AUDIENCE: All right. I'm a know; little confused about G1. So I can pass by just setting down any of my variables, right?

PROFESSOR: Yes.

AUDIENCE: So then, how could I ever lose? Because if--

AUDIENCE: You can lose if the turn variable changed satisfies the [INAUDIBLE].

AUDIENCE: Oh. Yeah.

[INTERPOSING VOICES]

AUDIENCE: Sorry.

PROFESSOR: Well, it changes every time so, yeah, right. I think you could set your variables in such a way when it's the opponents turn, no matter what they do, the formula becomes satisfied and then they lose. Yeah. Good.

AUDIENCE: But then it became satisfied on my turn. Sorry.

PROFESSOR: But t was a different value. So when it's your turn, you set t to 0 or set t to 1. And so then, maybe the formula's not satisfied. When it's the opponents turn, even if they pass, t is set to 0. Yes. t is part of the formula.

AUDIENCE: Where does the 12 come up? Is that from the previous?

PROFESSOR: I think it's from the previous result. Yeah. Didn't we have an 11 or 12?

AUDIENCE: It was 12.

PROFESSOR: Yeah. All right.

In case, you prefer real games, as I do. Here is a physical instantiation-- it could be many of the games. The one they chose to highlight was G4. So this is a game called Peek. And that name has been reappropriated for many other games, which we'll talk about in a moment.

But the idea is you have a bunch of trays. Each tray has a bunch of holes in it. Everybody knows where all the holes are. Each tray has exactly two positions in all the way and out all the way. It's like here.

And there are white trays and black trays. So G4, white player can take any of the trays of their variable, and if they want to, toggle its state. They can either push it in or pull it out or leave it as it is. And then, you're going to win if, viewed from above, there's a hole that sees all the way down to the ground.

So you could put something red here. And if you see red at the top, then you've satisfied the DNF formula. It's common between the two. So the first player to make a move where you could see a hole the way through, that is expressing DNF formula. So that's kind of a fun way, geometric way, to think about these games. And you could adapt it to all of these games, just with slightly different geometric setups. We'll see one in a little bit.

Cool. I have here the next thing to mention is why are all these games in EXPTIME. I think that's worth talking about.

The idea is pretty simple. All these games-- I mean, you can represent a state in a polynomial number of bits. It's just the current setting of the variables and maybe whose turn it is. Therefore, there are an exponential number of possible states. And the approach is basically to build the entire state graph and then rank for each position whether it's a win for one player or a win for the other player.

And I think the easiest way to think about this is you start with all the configurations that satisfy the win condition. You call those mate-in-0 positions. And then, from there in exponential time you can compute, find all of the moves, where all the places where you can make a move in any response lead you to a win position.

And those are the mate-in-1 positions. Or in two ply, one move, one pair of moves. And so on and so forth. And so then, at most you are going to be mate in exponential, because they're only exponentially many states. So the longest mate path is going to be exponential.

So you can do like 4k equals 0 to the total number of states, which is an exponential compute mate-in-k. And then, you've characterized the whole state space in nearly exponential time. And now you know which ones are winning states. And then, you check whether yours is one of them.

So there's probably other ways to do it. You could probably start from your initial configuration. But this makes it clear that it's at most exponential time that you'll be spending.

Not polynomial space, because we're building the entire graph here for this to work. We have to remember for all the states that we visited, so we can check for duplicates. So that's intuitively why we're an exponential time bigger than PSPACE, assuming PSPACE does not equal EXPTIME, which we believe.

Cool. So let's do some graph games next. I have two games to mention and a proof for one of them.

First game is based on Hamiltonian cycle. So this is a funny kind of game. And the claim is that you can do this for a lot of different NP complete problems.

But in general, we have a graph. Every edge of the graph is either black or white. And black player can only play black edges. White player can only play black edges. In addition, every edge either in or out, in the graph or currently out of the graph. And that's what these variables are going to correspond to.

I mean, we're going to convert into a corresponding game. In this case, it's going to be a G6 style game. So the winning condition is that the set of in edges form a Hamiltonian cycle in the graph. So you could think in means in the cycle, out means not in the cycle. If we ever get to a state where the in edges form a Hamiltonian cycle, player 1 wins. Player 2's goal is to try to avoid that.

Player 2 can only control the black edges. Player 1 can only control the white edges. At the end of each turn, you can change, if you want to, one of the edges between in and out and one of the edges of your color. So I'm just going to leave it at that, because the proof is not terribly interesting.

No gadgets, but this next game is I think a lot more fun, more surprisingly hard. And it has a nice clean reduction also from G3.

So what's the game? We are given three graphs on the same vertices. So really you can think of this as a three-edge colored graph. There's the red graph, the blue graph, the green graph, same set of vertices.

And we are going to have tokens on vertices. And we're going to have at most one per vertex. There are white tokens and black tokens.

A move by a player is going to, call it, slide a token, exactly one of them, of your color along path in one of the graphs. There's three of them.

And that slide has to satisfy a few conditions. One condition which is that the target vertex you get to and all intermediate vertices that you follow along the path must be empty. They should have no tokens.

So you pick up any token of your color-- you pick a color red, green, or blue-- and you follow a red path or you follow a green path or you follow a blue path. As long as that path doesn't hit any other tokens, then you drop your token, erase it from where it was originally, and you keep making moves like that.

What about the win condition? We look at a player-- I'm going to come the players 1 and 2. Player i will win if they get one of their tokens to a node in the win set WI. So also we're going to color the nodes as winning for white or winning for black. And if you ever get a token to a winning spot for your color, games over. You win. So this is a pretty natural token-sliding game. As EXPTIME complete, very hard.

And this one we're going to proof. So this is a variable gadget. So we're going to reduce from G3. I think I said that. This one.

So G3, a move corresponds to you're forced to flip a variable of your color. Although we could switch that to the other version. And you're going to lose the game if you satisfy your DNF formula.

So there's lots of edges here. But in particular, there's solid edges, dashed edges, and the dotted edges. That's red, green, and blue. This is before color journals. Question.

AUDIENCE: I didn't catch what the purpose of coloring the edges is.

PROFESSOR: Oh. It's basically to decompose the graph into a bunch of different types of paths. So when you follow a path, it must be a path of a single color.

AUDIENCE: Oh. OK.

PROFESSOR: Yes. So you can only follow the solid path or follow a dashed path or follow a dotted path. You can't mix edges of different colors. So that's the three graphs.

So what's going on? We've got a white token here, black token here, white token here. This is a win position for white. These are win positions for black. So there's a lot of things going on here.

But the main idea is that white can move the token to here. That corresponds to EXPTIME being false. Or it can move it to here. That corresponds to EXPTIME being true.

And that's essentially all you can do in this picture. Why? Because if this guy moves to here, for example, then black can win. Black can go along this path and get to a win state. And symmetrically, if white also goes off the track, then black can go to the win state.

Also, we had to put this vertex here, put this token here, to make that true. You might worry, maybe black tries to move out of that position to there or to there or to there. But then white will win by going like that. So under optimal play, if you don't make silly losing moves, all you can do is move the white token between here and here.

And I didn't say it, but the path here is supposed to be a non-zero length path. So you have to move your token. And therefore, that represents this constraint that you must put the variable. You must flip exactly one variable of your color.

So that's the variable gadget. And for a while, that's just going to happen. But now we have to implement the win condition or the lose condition.

So that's this thing. This is a clause. This is a DNF clause. Is it x3 is false and y5 is true. In that case, we want to lose or win. It doesn't particularly matter here. We want to lose, I guess.

Obviously, black can't move first in this configuration, because then white could just win. So black can't move first, so what can happen in this gadget is that white must move here because you're not allowed to collide. Once you move there, the threat is that white will win in the next turn. So black must move up.

And then, if this is empty, white can move here. It could actually move to here, because that's a path. And then, the threat is white will win. So black must move up.

Also, there's a black win there, but we'll get to that. Actually, I'll do it now. So if this was blocked by x3, because x3 is currently set to true, then you won't be able to move this way. So you'll basically be stuck here. And then, black can win like that. So you don't want to activate this gadget unless this is clear. Similarly, we don't want to activate it unless this is clear.

So if we're up to here, then white can move up, threatening to win. Therefore, black must move up. And then, if this is clear, white can move up to there. And then, black has to go there to prevent this win. But white can win that way.

So white is going to win if this clause was satisfied or unsatisfied. And I'm a little confused about how many times we negated. But I think you get the spirit. And if this DNF clause is satisfied, then white wins. And you can have a bunch of these clauses just sitting around.

These are all vertices unique to the clause, except for these guys, which are shared with the variable gadgets here. So those are the same as these vertices. They're all glued together. No planarity here.

Cool. So that's EXPTIME completeness of this game called Block.

All right. This is a little ambitious. But the next one I'd like to prove is a real-life classic game, namely checkers. This is most of the proof in one slide. It's a little bit complicated. But let me give you the high level idea. It's actually a really cool proof. It's also from my birth years. So it's cool.

So don't worry about this too much, but our construction is going to be this little square inside this little square. And then there's this giant spiral outside that will cause everything to self-destruct when the game ends. I'll talk about that later. Let's focus in first on what's happening within this little square. But that will be the context, and we'll get to that again.

So inside the square is this thing. So I mean, I think the perspective is from white. So the goal is for white to win. And in checkers, there's two types of pieces. And the circles are going to be pieces that are going this way for white and that way for black. And the squares are going to be kings. They can move up or down.

So white and has some variables that it can set between true and false up here. Black has some variables that can be set between true or false.

I didn't mention, but we are going to be playing G3. We are going to be simulating G3. So a move is going to be forcing to flip something. And we're going to lose if the formula ends up being satisfied.

So then there's this computation done in the middle. Basically, each of these boxes is going to correspond to a DNF clause. What makes it tricky is each clause involves some black variables and some white variables. And those are going to behave quite differently. But this construction is going to make that happen.

And overall, let's see. Where to go? Let's start with the variable gadget. So this is a white variable gadget. Black variable gadget is symmetric with all the colors flipped I believe. Yeah. That should be fine.

And the key actor is this king right here. I've drawn it at the intersection. But it could be on the t position or the f position. t corresponds to variables being said true. f corresponds to the variables being set false.

And so most of the game is going to be all the players moving their kings back and forth between these t and f positions. Those will correspond to flipping a variable of your color.

Now, constantly you're worrying about the threats. These attack zones and defend paths are the threats. And if ever a threat is satisfied, somebody's going to win. And the game will end.

So when that happens, players in the variable position can choose to activate a variable of their color. And the way you would do that, a key rule which you may have forgotten about checkers, is that if you can capture you must capture. So that's how we're going to force the opponent to do lots of things we want them to do.

If we move this white token up and to the right, then suddenly this black token can capture. And it must capture along the unique path boom, boom, boom, boom. Assuming that we're in the true position, he must jump there, jump there, jump there, and activate some stuff down here.

Conversely, if we're in the false position, we're going to move this B guy up and to the left. And then, black token will have to go off into this position. So far so good.

Now, so this variable may occur in several different clauses. There's one clause that matters, because this is a DNF thing. We're just trying to prove that one clause has been satisfied.

So white's going to want to send things that satisfy it. Black is going to want to send its variables that cause it to be not satisfied. So there's going to be some white variables and some black variables involved in this one clause.

But we need the ability to send the copy of this variable to the clause that matters. So the threatening player, white, is going to be sending their thing to a particular cause.

They do that using this fork gadget. So if you're coming from up here, you jump here, and jump here. Black jumps here. And that's the end of black's turn.

Now, white is going to move this position to either H or I, forcing black to go follow the next path. So white's always in control for this path. There's going to be a symmetric thing where black is in control. And black and send their token to where they want it to go, one of these squares.

Cool. Now, maybe at this point I should mention that there are these letters here, MWD and MBD. And this is relevant to the overall architecture of the game.

What we're essentially trying to do is get free moves. Free moves will win the game, because of the spiral on the outside. But before I get to the spiral, here's a way to generate free moves.

Suppose right here I'm going to throw in a whole bunch-- think of infinity-- copies of this gadget, so-called white delay. So black is forced to jump here and just keeps going through. There's no options for black. It just has to keep going.

But after that move, white has a penalty, because white is forced to capture that token. So white essentially lost a move sometime in the future. As long as there are still capture moves it's OK. But that's kind of bad for white.

And there's a symmetric version for black. I didn't bring the picture, but it's pretty similar. And black will be forced afterwards to jump.

So the point is if we essentially stopped here, black would win actually. Because there's so many moves here that white is forced to waste, they'll end up being able to win the whole game in a way that I will get to. For that reason, you better not trigger this thing unless the king is in the right spot. Otherwise, black will stop there and then black will win.

But there's a matching MBD down here for black. And so we're forced to use both of them. Then, they cancel each other out. Because if black has n free moves and white as n free moves, they'll just make all those captures and then get back to the game. So, in general, the gadgets use these things to force you to not stop in the middle of the construction.

The heart of the matter is this gadget, the clause. So there's numbers here. If you look closely, these are 1's and then these are 2's and then these are 3's. So the idea is let's suppose that there are three white variables and two black variables in this clause that you can generalize. That's what the figure shows.

So the idea is that first white's trying to win. So it's going to send its variables of the appropriate truth assignment into the one clause. It's going to activate whichever one happens to be glued to the one clause first or the one position. And what will happen is the white token will go here.

I forgot to mention we've switched colors from white to black. That's because there's a color change gadget, where you can convert white wires into black wires. So don't worry about it.

So now we have a white token that captures all these pieces. Notice it captures this one, which will enable the two path next time. So it is first you send in this. It just stops, nothing to do there.

Then, we send in the 2's. And if this has been done already, so this is basically a conjunction of 1 and 2. 2 will be able to get here if and only if both 1 and 2 are set.

Also capture this position and this position, enabling both of these 3's. Basically, we have two copies of the 3 wire just because the opponent could mess things up for us by moving one of these. Black can move this over here. And then, we're kind of annoyed.

But we have two copies for redundancy. So even if black messes up one of them, we'll be OK. The worry is after you activate this thing black has a turn. And so maybe instead of doing something else, black could just move this. But we have enough redundancy. That won't hurt us.

Finally, if we can send 1, 2, and 3 together, then black will waste n moves. And so white is in the lead. And then, white will win the game in a way that I haven't shown you.

Conversely, black has a blocking strategy, a defense strategy, which is after 2 has activated, if black has either of these variable set to the right value, they can trigger them with a corresponding black variable gadget. And then, the white token will come up here and hit this gadget MWD, which is a whole bunch of these.

And then, white is behind. And then, black will immediately win. Before white gets to do this, black will win.

So overall structure make sense? Basically, this gadget is active as soon as the white player sends down the one thing and then sends down the two thing. And then, if black happens to have a response which blocks it, what that corresponds to is that one of the black variables is in the wrong setting for this clause. Then, black will have a defense strategy and win the game.

And so you only want to activate a thing if all the black variables are in the correct setting currently and the white variables in the correct setting, so you could do the 1, 2, 3, and then win. And symmetrically, there are things like this for black. But that is the game.

Now, let me tell you how once you have n free moves, how you could win the game. This is the outer spiral. So everything we're talking about was in here very far away from the spiral, like n squared away or something.

And the spiral in more detail looks like this. We have a whole bunch of kings, and white king, black king. And if you have a bunch of free moves-- you don't actually need very many free moves. It's like five or something. But they didn't feel like computing it, so they just called it x and then figured out for x.

You can make a few moves and get from this position to this position. So far so good. Still have a free move. Iam going to move this A king to the B position. At that point, black has a forced jump. Now, they may have other forced jumps. We'll just just sit there and wait for them to do all their forced jumps.

And then, black will have to make this jump, capturing one of these kings. So this guy's basically to prevent you from jumping that way. So this guy moves down, which forces this guy to jump here. And then, from here, you can jump here, here, here, here, here, so satisfying. Take all the black kings on the outside.

Now, you own the entire spiral, which is a huge number of things. The number of rings of the spiral is greater than the number of pieces in there. And also, these things are very far away. So you have plenty of moves.

So whatever's happening in here-- chaos could be happening-- you can rearrange the white spiral into a bunch of white rectangles. And it's already been shown conveniently that if you have n white rectangles in this sense, less or equal to n pieces, then you win.

So it doesn't matter what's in here. You can come in and use each ring to destroy one black token at a time. And game over.

[LAUGHTER]

PROFESSOR: Isn't that fun? So that's checkers EXPTIME complete. A very cool proof.

There are details I didn't show. There's a parity shift, a few other things. But that's pretty much everything. And that's the power of G3.

I'll just show pictures for chess. It's even more complicated. But chess is also EXPTIME complete.

Now, I mean, both checkers and chess in the real game sometimes in some versions of the rules it says you can make at most 200 moves or 500 moves or something. Those games are of course PSPACE complete, if you add an artificial move bound of n or something.

But if you don't, the natural game I think is when you don't bound the number of moves, then they become EXPTIME complete. This is also from G3. That's I'll say about chess.

AUDIENCE: Do you know in chess what pieces are used?

PROFESSOR: I think it's just pawns and bishops, which is pretty cool. One of these figures--

AUDIENCE: Oh. It's on the left.

PROFESSOR: Yeah. White pawn, black pawn, white bishop, black bishop. I'm pretty sure that's all of them. But it's a little hard to tell him from this figure. I think you need a better scan to know which are which. And

This paper's a little hard to find, but I got a scanned copy. Go is EXPTIME hard. So how many people know Go? Good. Almost everyone.

So Go is a little funny, because you might think it's a bounded game. Every time you place a token, you consume area. But captures remove area, and furthermore kos are the action. So kos are these threats that when I capture and someone else can capture me back. And you can flip back and forth from those types of states.

There are actually different rules for Go. The Japanese rule set says that you are not allowed to repeat the position you were just at. And this is the subtlety of a ko.

So this is the canonical simplest ko. So if you move white here, you capture this stone.

And under Japanese rules, you're not allowed to repeat the previous position, which means you're not allowed to immediately play back here and capture this stone.

You can see why. It'd be kind of a boring game if you just went back and forth capturing the same guy. But as long as you play anywhere else, then you can come back and capture. That's the ko threat.

That game is in EXPTIME. And Go is EXPTIME complete under Japanese rules. In some sense, because the number of states is still only exponential, you just need to remember the current state and the very previous state. Because the ko rule is that you're not allowed to repeat the previous state.

In the USA and China, there is something called a super ko rule, which is that you're not allowed to repeat any previous state of the game ever seen. That means that you are not necessarily still in EXPTIME, because you might need to remember all past states, which could be exponentially long. So it's like double the exponential size.

In general, you are in EXPSPACE. And it's an open problem whether Go with super ko is EXPSPACE complete. Currently, best known lower bound is EXPTIME hardness. I won't cover the gadgets here. I mean, it's even more subtle, because you can't really be capturing. It's all about ko threats.

Or you can't really be adding pieces. You have to constantly be capturing, because you must conserve volume. You're running for exponential time. It would be quite an interesting game to watch. But it's a reduction I think from G3 again. That's Go.

Let me tell you a little bit about constraint logic. So we got a question.

AUDIENCE: Sorry. I was just wondering if you're allowed to use superpolynomial reductions for EXPTIME purpose.

PROFESSOR: So in fact, all the proofs I've seen in this world use LOGSPACE, which is even stronger than polynomial time. I don't know why. Probably just because they can. And it's slightly better.

It makes sense to use LOGSPACE reductions for PSPACE hardness. But they ended up still doing LOGSPACE reductions for EXPTIME hardness.

For my purposes, I think polynomial time would be fine. You could imagine using a little bit more time. Definitely you don't want to use exponential time, because that wouldn't preserve a lower bound. But you could use quasi-polynomial or polynomial spaces or some crazy thing.

But all the claims I've made don't even need polynomial time reductions. It's pretty strong. Yeah.

AUDIENCE: That rule confuses me a lot. So that means I can't ever go in the erased area?

PROFESSOR: Well, no. You could play here as long as something else changed. You cannot repeat the entire board configuration. So the local configuration can repeat. So as long as something else has changed, then you can play here.

AUDIENCE: But then it's the same rule, right?

PROFESSOR: It's the same rule. But if you have five of these, so in the ko rule, as long as you're basically playing them in a different order, you can keep playing around and ko, ko, ko.

But in the super-ko rule, at some point, you're going to run out of ko patterns. And at some point, you won't be able to repeat something. And that changes how the game behaves. Of course, we don't have a hardness proof for that case, or a stronger hardness proof. It's subtle.

Two-player constraint logic unbounded is also EXPTIME complete. This is a reduction from G6. You can set one of your variables however you want. And player 1 is going to win if we satisfy a CNF formula.

So the formula is out here or here or here. This is a white variable. This is a black variable. The only difference is this edge is white here. This edge is black there.

So the idea is this can be set left to right. If it's left, that corresponds to false. If it's to the right, it corresponds to true. And so most of the time, players are just going to be flipping their edges back and forth. It'd be a pretty boring game.

But at some point, there's this constant threat network, which is if the formula is satisfied, then the white player is going to be able to win. How do they win? First thing they do is lock a variable. So this is pointing to the true state-- sorry, also left and right pictures are symmetric, so symmetric that we didn't even draw it in the book. So I reflected the image and all the letters are backwards.

So let's say the variable's pointing to the right. So white is going to lock the variable, prevent it from flipping anymore, by flipping this edge.

Now, we enter attack face. And lots of things happen. So this is a SPLIT gadget. So immediately, for example, black could flip here and start winning the game. There's a long path here. It's slow, so it's a long path.

But eventually black will win the game. The timer starts ticking now potentially, if black chooses to follow that path. Simultaneously, white is going to flip this guy, then flip this guy, and then flip this guy and activate the formula.

Now, it's a little awkward, because we needed 1, 2, 3, 4 moves to trigger the formula. We basically want to consume four moves of black so that things remain balanced. And that's this 1, 2, 3, 4 flips.

If black does not flip all these edges, then white can flip this and then flip this and then flip this and then instantly win. That's the fast win track. So to prevent that, black must flip this edge. It can't flip this edge until it's flipped this one. It can't flip this one until it's flipped this one.

So as soon as you lock a variable, black must respond with A. At that point, white can afford to flip here. And black must respond with B. And then flip here, black must respond with C. And then flip here, black must respond-- I guess, flip here first and then flip B. And then you can activate the formula.

So then white and black are balanced again. What's cool about that is it means white cannot only lock this variable but lock all of the variables. Every variable behaves the same way.

And the goal for white is to get all the variables locked so it can start triggering the formula. So every time you lock a variable, black is forced to respond in a particular way. And so after that, white can lock the next variable, both the white ones and the black ones, locks all of them.

And now, the formula's basically primed, ready to go. The variables can't change anymore. And now, we're in full attack mode. Cool. And then, the formula happens.

Meanwhile, this slow win is probably happening, fusing. And we set up the formula so that if it's satisfied, you end up flipping the win edge in a certain amount of time. And this is slightly longer. So if you don't end up satisfying the formula, black wins. And then, it was not a valid threat. You shouldn't have triggered this attack mode.

There's another threat here, the slower win. So I assume that people were flipping these edges. But it could be actually black flips A early. If it does that, then white will eventually win through that path. So this win path is just to prevent A from flipping before it's locked. And the timing is to setup right.

Obviously, lots of timing issues here, so we have to build an equalizer gadget that ends up slowing down the traversal of an edge basically in a particular way. And these are all of the vertices need. It's a fair number, including some white and black edges, which is a little bit annoying.

It'd be nice to simplify instead of gadgets. But that's the best we have so far. And you can use the regular NCL crossover appropriately, making everything else slower so that crossover doesn't slow things down too much.

Cool. A couple more minutes. I want to mention a couple of games that do not fit into the table. In some sense, they are higher than the table. So we have polynomial bounded number of moves. And then, above that, we have exponential number of moves.

But if you take those exponential number of moves and add an extra rule, so this is entirely motivated by Go with super-ko rule. And you take any of these games like G1 through G6. So you have the same win rule. But in addition you will lose if you ever repeat a past game configuration. So it's forbidden to do that.

Then, Robson, same guy who proved Go is EXPTIME hard, proved that G1, G2, and G3, and chess and checkers become EXPSPACE complete, which is cool. That's a little harder. And it makes sense they're in EXPSPACE because it can sort of walk the graph in exponential space. But they actually become that hard.

And even crazier to me is this rule. It seems almost the same. So we're going to have two extra variables x and y, which can appear in the formula. And instead of this lose rule, we add this lose rule which is if you ever repeat a past game configuration-- and between now and that past configuration at most one of x and y have changed and that in turn-- then you lose.

So if both of them get changed, you're allowed to repeat it again. I don't have a great intuition why this makes the game even harder, but it pushes it up to the next level which is double the exponential time.

AUDIENCE: This is change and changed back [INAUDIBLE] configuration?

PROFESSOR: Well, right. One of them will have changed, x or y. Hm. You're repeating the configuration. So you changed something-- it be some other variable z-- and then change it back. As long as you didn't also change both x and y, that's considered invalid. Once you change x and y, you are free to repeat everything again, until the next time something gets repeated.

So I guess you have to keep track of when the x and y changes are. So it's even harder to know the history of the game, which repeats are valid and which are invalid. In this case, all that he could prove is the G1 becomes double the exponential time.

Cool. I could say a little more. In this world, it's probably good to know this fun fact. Normally, when we think of non-determinism, we think of an NP. We have existential quantifiers on the guesses. If there's at least one branch that works, then we're happy. Then, we return, yes.

You could also think of universal quantifiers, where if all of the paths going out of here end up with yes, then we're happy. You can also think of an alternating guess machine, or alternating lucky machine, where every other time it's an existential guess branch and the opposite times it's a universal guess branch.

That corresponds to two-player games in a natural way. Alternating polynomial time thing corresponds to queue set. So that would give you PSPACE.

Alternating polynomial time gives you polynomial space. Alternating polynomial space gives you exponential time. Alternating exponential time gives you exponential space. Alternating exponential space gives you double the exponential time.

So in fact, this result is stated as alternating exponential space complete, which is the same as double exponential time. So time and space are interleaved in the natural way. And if you add alternation to one layer, you get to the next layer.

So it's just fun facts. And it's useful for reasoning about these games.

One more class. We've so far talked about perfect information games, where everybody knows all the states of all the variables. You can define a private information game, where you can see some but not all of your opponent's state. You know all of your variable settings.

In this setting, a slight variation on G1 and G2. We have to generalize the formula's a little bit, become double the exponential time complete, just as hard as the weird repeat rule. Private information gives you a very hard thing.

A more restricted form is when player 1 can't see anything about player 2. But player 2 can see a little bit of player 1. This is sort of a semi-blind game. In this case, G2 at least is exponential space complete. That's weaker than here.

So lots of weird separations here. There are no well-known games in these categories that have been proofed hard. But here is the version of Peek. You can redesign the Peek structure.

So you have some walls here. So you can't see whether this guy's in or this guy's out. Player 0 can only play on these guys. Player 1 can only plan on these guys. Player 1 can see all of this state, a little bit of the state. In the blind version, you can't see anything to the right.

And because they couldn't do G4, this is the G2 version, which is there's also a wall here. And you can only see red through these holes. And your opponent can see red through these holes. So you've got two different win conditions. That's G2. And so that's a version of Peek. This one is double the exponential time hard and complete. And this one is exponential space complete.

So these are like way in the deep end. But next class we'll talk about this category of team imperfect information games. And then, we even get undecidability, which is the ultimate hardness. But that's all for today.