PROFESSOR: All right. Today we go back to fun with hardness proofs in games. It's been a while since we've done many games and puzzles. So today we're going to talk mostly about this one paper, which has a ton of results in it, by Giovanni Viglietta. "Gaming's a hard job, but someone has to do it!" What's particularly cool about this paper is it shows a bunch of metatheorems, which you can think of as general techniques for proving hardness. And I'm going to talk about essentially two main metatheorems, although there'll be a few different versions-- one for NP-hardness and one for PSPACE-hardness. And this'll be our first real PSPACE-hardness proof other than lecture one.

And the proofs here aren't very hard, and the fun part is applying them to various games. And they apply to a lot of different games. So metatheorem we're using here in a somewhat vague sense. This won't be a formal theorem, because it's hard to state all of the assumptions you need about your game. But it's sort of-- think of it as a roughly true theorem. We're not going to state all the assumptions you need.

But the general setup for metatheorem one is that you have a player-- or the paper calls it an avatar, you know, a little guy walking around. You're traversing a 2D environment. So you're not allowed to have crossovers. From a given start location. And given that setup, if you have two features in your game, one is called location traversal, and the other is called single-use paths. Then your game is NP-hard. That's the metatheorem.

So location traversal means that there are some locations on the board that you have to visit in order to win the level. So there's no target location. Your goal is to visit all of the locations. Once you do that, potentially you win. Or maybe there is also a target location you have to get to at the end. Question?

AUDIENCE: Does planar mean you're on a plane, or like a planar graph kind of environment?

PROFESSOR: Either way. You could be sort of in a grid. You could be in a planar graph. It's vague. But the goal is two-dimensional, so the point is we don't need crossovers for this reduction. So location traversal, you have to visit certain locations. Single-use paths, you can only use them once. They're traversable, but once you traverse them once, they're no longer traversable. So from this, you can get NP-hardness. Any guesses how? What should I reduce from?

AUDIENCE: Hamiltonian.

PROFESSOR: Hamiltonicity, yup.

AUDIENCE: All paths are single-use?

PROFESSOR: No. Not all paths are single-use, although it will work here. The idea is that you mark some paths as single-use, others not. So reduction from planar, let's say, max degree three. Hamiltonicity. And so the idea is whenever I have a vertex in my Hamiltonian graph, I'm going to turn that into a location that must be traversed, because we want to visit all the vertices. So there you go. That's where to put them. And whenever I have an edge, I'm going to convert that into a single-use path.

And because I'm maximum degree three and each of these edges is single-use, once I use one of the edges and it disappears, and then I use another edge, and I get the treat that was in that node, and then I use another edge and that disappears, it's a really bad idea for me to use the third edge, because then I'm trapped there forever. So I would have had to have already solved the puzzle. I would have had to already get all of the vertices before going here. So there was no reason to traverse that edge.

Therefore, you're effectively doing a Hamiltonian path from the given start position, which we know how to reduce from Hamiltonian cycle. And because we can do planar graphs, this is planar setup, max degree three is critical for not being able to revisit a vertex. OK?

We've seen a very similar proof to this back in the Hamiltonicity lecture, but there, the assumption was that there was a time limit, plus collectibles, plus location traversal. This is another way to do the same kind of proof. And it makes it a little easier to be convinced that your proofs work perfectly, because with time limits, you're very sensitive to timing. Every edge has to be exactly the same length. Here we're adding single-use paths instead of that to force you to finish the level. The claim is also it makes the puzzles more fun. So that's, of course, up to interpretation. But at the very least, using this proof, we can get a whole bunch more games that we haven't seen.

So one of them is *Boulder Dash.* How many people have played *Boulder Dash,* let's say? Yeah. Only-- not very many. I used to play this in C64 days, Commodore 64. But you are walking around these boulders. You can hold them up, but as soon as you let go, they fall. And if they fall on you, you die. So you don't want that to happen. And the goal is to collect all the diamonds, roughly, and then to get it to a target location. There's some weird physics about how boulders fall.

But basically, here are the two gadgets. Location traversal, you just put a diamond there, and then you have to get it. And then this is a single-use path. The idea is you can come from either side, push this out of the way. The boulder is then in the way. So if you then tried to traverse it, you can't turn around the corner, so you get stuck. OK. So once you have those two gadgets, you get an NP-hardness proof. So *Boulder Dash* is NP-hard.

Lode Runner. How many people have played *Lode Runner*? A few more. I used to play these in, I think, Apple IIe days or something. Well, originally C64 again. So you have this guy who walks around, has to collect all the parts again, these things. And the fun thing about *Lode Runner* is you can dig a hole, and then the monsters fall in. They'll drop their diamond if they have one. And they might be able to climb back out. Eventually, that square will refill. So it's a timed thing. And so based on that kind of physics, we can build the two gadgets again.

You put gold wherever you want to make the thing. And this is a single-traversal gadget. This is after it's been traversed once. You get stuck in the hole. But if there's a monster there, you can actually stand on it, dig the hole, let the guy fall. This will regenerate. This guy's stuck, but now you cannot traverse this thing. There's no jumping in this game. You can't jump over there. You have to get to the other side by falling. And that's only possible in the initial configuration. So *Lode Runner's* NP-hard. Questions? Cool.

Also, *Legend of Zelda II, Adventures of Link* is NP-hard. On the one hand, we have single-use paths illustrated on the left. And location traversal we're going to do by keys. If you get enough keys, you can open enough doors. At the end of the puzzle, there's going to be a whole bunch of doors. Let me get this one going again.

For single-traversal paths, there are actually these bridges that disappear after you traverse them. So you can do some amount of jumping, but if you make it really long, you will be prevented from re-traversing that path. So we're going to put a key at each vertex. And then at some finishing location, we'll have end doors, one for each vertex. And that should do *Zelda II.* OK. So that is metatheorem one.

I want to tell you about a slight variation on this metatheorem, and then another result based on it. So metatheorem two is same kind of setup. So slightly different. Yeah. I'll keep a little bit of suspense about why we care about this. Instead of single-use paths, suppose that you could build tokens, which are things you can pick up. And for this reduction, let's say you can only pick up one at a time, though it won't matter.

And toll roads, where you have to pay with a token in order to traverse the edge, this essentially can be used to simulate a single-use path, but it's a slightly different perspective in that you can visit an edge if you-- well, what we're going to imagine doing is putting tokens at the vertices, but just one at each vertex. When you pick up that token, you can then traverse one of the incident edges. So again, every edge will become a toll road. And every vertex will become a location traversal point that has to be traversed. And it will have one token. And so you can use this same kind of proof. It's just a different perspective on the same kind of thing.

And we can use this to prove that *Pac-Man* is NP-hard. This is the motivation. So again, you have to get all the dots in *Pac-Man.* So you, in particular, have to get that one. When you get that one, this edge becomes traversable. So this is a somewhat generalized version of *Tetris.* You have lots of houses for the ghosts, lots of ghosts, obviously. The algorithms that the ghosts follow doesn't matter too much.

The only constraint is that ghosts change direction if and only if the state changes, like you turn them into-- I forget the names of all the modes-- but you turn them into the blue mode or they turn back, then they will switch direction in their tunnel. And that is the only setting in which ghosts will reverse direction, double-back on themselves. They never double-back otherwise.

And given that setup, if we make these edges really long, we can make sure that the ghosts always stay within their edge, because you're basically alternating between switching the state, turning the directions of the ghosts, traversing an edge, and then doing it again and again, so that the ghosts always sort of go back and forth. If you happen to capture one, it doesn't really change its location much, because it just goes back to the house.

But you obviously need one of these in your possession. You end up using the token right away to traverse one of the incident edges, so you can actually get across. But it's long enough, so that by the time you cross over, the ghosts have reset. So that is metatheorem two applied to *Pac-Man*, in case you were wondering. This is, I think, quite clever, to get these things to work out. OK.

NP-hardness, we're used to that. The next metatheorems are about PSPACE-hardness. And before I get there, I need to tell you a little bit about some basic PSPACE-hard problems. Like, we have SAT and Hamiltonicity for NP-hardness. For PSPACE-hardness, there are a couple of natural problems. First, let me remind you, because it was lecture one that I defined PSPACE, these are all problems solvable in polynomial space. And so we're measuring space instead of time.

This is contained in exponential time, because there are, at most, exponentially many states of a polynomial space machine. And it contains NP. In particular, every

NP problem can be simulated on a PSPACE computer just by trying all possible options for the guests, but backtracking and throwing away the information from the other guests' paths, and just keeping one bit of state, which is, did I find a yes solution yet? So you can turn any NP machine into a PSPACE machine. So it's somewhere in between.

The other fun fact is that it equals non-deterministic polynomial space. So non-deterministic in the sense of NP. If you add guessing to a PSPACE machine, that doesn't help you. You can always simulate it with a deterministic machine, at most, blowing up the space by a square. So if you had SSPACE before, you have S squared space after. Our polynomials are closed under squaring.

So this is especially useful for showing that a lot of games are in PSPACE. It's a lot easier, usually, to give a non-deterministic polynomial space algorithm. You have the freedom to guess whenever you like, just like with proving containment in NP. And if you prove containment in NP space, you are also proving containment in PSPACE. This is called Savage's theorem. OK. So that was just some recall from lecture one.

And now let's talk about some sort of base problems, PSPACE-complete problems that we can reduce from. I would say the most obvious one is to simulate a polynomial space algorithm. You can relax it slightly to say, simulate a linear space algorithm or a Turing machine, if you prefer that. Of course, you are PSPACE-complete if you can simulate all algorithms that run in polynomial space.

And there are some proofs that work that way. It takes some simple model of a computer, like a Turing machine, typically. Like your reading and writing cells on a tape. And you're doing whatever you can do in that linear amount of space. And just simulate those transitions one at a time. But it's usually pretty tedious to do that. I don't think we will see any proofs directly of simulating a Turing machine.

Usually simpler way to go is a problem called Q SAT. It's actually usually called QBF, and sometimes called TQBF, and occasionally called Q SAT. But I'm growing on the name Q SAT, because we can specify things. It's consistent with our naming terminology. We can say Q 3SAT, and we can say Q 1-in-3SAT, and we can say planar Q SAT. Whereas QBF, usually people mean Q 3SAT, I guess. But this is a little more explicit what kind of SAT we want. So we're adding the letter Q, which is Quantified.

So the general problem is, given a fully quantified Boolean formula, you want to decide, is it true? That's the T in TQBF, if you prefer that term.

So for example, here is a quantified formula. It took me a little thinking to come up with one that is actually true and in not a quite trivial way. This is a 2SAT formula. And so in particular, you can read this as x implies y, and y implies x. And we're saying for every x, there is a y, such that this is satisfied. And that's true. Namely, y equals x would be the correct choice. But it's not totally obvious, given that formula, that it's true. And in general, given such a formula-- probably not with 2SAT in here, but with 3SAT formula-- it's going to be PSPACE-complete.

To contrast with SAT, essentially, there are a bunch of There Exists at the beginning. With Q SAT, you can have For Alls in addition to Exists. That's what we're changing.

And we can simplify things a little bit. We can assume a prenex form, which just means that all the quantifiers are in the beginning, and then we have a 3SAT formula, or whatever. And we can assume that the quantifiers alternate between For All and There Exists just by adding in extra dummies if we happen to use a bunch of For Alls in a row or a bunch of There Exists in a row.

But really, the hard case is when you have lots of both, and they are interwoven. So it's not enough to look at formulas that have a bunch of For Alls and then a bunch of Exists. You have to think about the case where there are lots of alternations between the two. We'll talk more about that when we get to two-player games, because that relates to two-player games.

But for now, we're going to stick to one-player games. And this is just something we need to simulate by our single player. So now the question is, what forms can this formula take? Because we have lots of forms of SAT that we know are hard. And pretty much all of them remain hard in this setting as well, when you add in the quantifiers.

And in fact, there is a Schaefer-style dichotomy theorem, like we had with SAT and NP. And basically, this quantified SAT is polynomial if and only if the types of clauses you allow are all Horn clauses, or all dual-Horn clauses, or all 2SAT clauses, or a system of linear equalities mod 2 X or and x nor quality constraints. Those could all be solved before. They can still be solved in this setting in polynomial time, even with alternating quantifiers. So that's cool.

However, it is no longer polynomial time if your clauses are satisfied when you set the variables all true or all false. Because we have these For All quantifiers now, being all true is not the panacea that it was before. So that's one case.

And otherwise, you are PSPACE-complete. All other versions of SAT are PSPACE-complete-- sorry, of Q SAT. So it's almost the same as the Schaefer theorem, but there's one case that differs. It's hard where it wasn't before. So that's cool for general, from a formula perspective. But we, of course, typically care about planar bipartite graphs expressing those clauses and their variables. And good news is everything works as before. So planar Q SAT and planar 1-in-3 Q SAT are hard. Planar not all equal. Q SAT is also going to be easy like it was before.

But planar Q SAT, we can use the usual crossover gadget. What that crossover gadget does, it forces some variables to be the same, but it also creates new variables. And those variables need to be quantified. All you need to do to simulate our old proof is put a whole bunch of existential quantifiers at the end of the quantifier list. So for every new variable you create, you put There Exists at the end for those variables.

And we know those variables are forced to be equal to other copies of variables that are given to us. And so it stimulates the old planar SAT proof, but in a quantified setting. And that was actually done by Lichtenstein in the original planar SAT paper. It also talked about planar Q SAT. And then once you have this problem is hard, the same reaction we had for 1-in-3 Q SAT also just works. So with the same idea.

So there are some base problems. Now we can do so reductions. First we'll prove a theorem. And then we'll apply that metatheorem to a bunch of different games. One-player games.

So for next metatheorem, we're going to have a similar setup. We're traversing a planar environment. But this time we are given a start location and a destination. Because we're no longer going to have the location traversal goal, we need to add some kind of goal. Otherwise, you've solved the game by starting.

[LAUGHTER]

So it's a path traversal. Get from A to B. And what we're going to need is a door plus a pressure plate. So the idea is that there is an obstacle, a door, which is either open or closed. And there's a pressure plate, where if you step on a pressure plate marked open-- so you're walking along here, you step on this square, that will open this door and make it traversable. As you go across it, you're forced to open the door.

And there's going to be another pressure plate marked closed, which, as you traverse it, you step on that plate, it forces the door to close. So in order to get to the other side here, usually this is in some kind of tunnel. So in order to get from this side to this side, you must close this door. And we have more than one. So there are many such doors. I'll say A, B, C. And this is going to say Open A. This is going to say Close B. And we can furthermore assume that there's exactly one open and exactly one close for each door. OK? So maybe I should have written A here.

So this is the complete story for A. And there's a similar open and close for B, an open and close for C. And otherwise, you can just walk around. And your goal is to get to destinations. Sounds pretty simple. And this is a very powerful metatheorem. This is PSPACE-complete. This applies to a lot of games.

But before I get to what games it applies to, let's prove that this is indeed PSPACE-complete by a reduction from Q 3SAT. We won't even need planar here. We won't need planar, because we can open doors kind of remotely. When I step on this button, which could be far, far away, I open this door. So effectively, the connections between variables and clauses can be crossing. There's no restriction on that here.

So you're going to see a few different proofs like this. We'll see others in future lectures. But this is a typical prototype for how to reduce from Q3SAT. You have your start location, you have your finish location. Your goal is to traverse this thing. It's sort of like a circuit. And then you have these blocks which represent quantifiers. We'll talk about those in a moment. In the end, you exit here. And you have to satisfy this formula in order to get through.

If you get through, then you are able to enter this quantifier gadget. And these are kind of in a nested structure. You can think of this quantifier gadget as calling this one by traversing this path. You're recursively calling the rest of the formula. And then this is sort of the return value. Yes, I did it. I succeeded, I got a yes. So for the existential quantifier, what this is essentially doing is, say, OK, I don't know what x is going to be, I'm going to let the player decide between two different paths.

Set x to true or set x to false. So there'll be two ways to go here. And then you'll exit along the same path. But x will be set one way or the other. Recursively, you want that formula to be true. And if that happens, then the player will be able to come back here. And if you successfully set the rest of the formula to be true, then it means the overall formula is true, because you just wanted to check that there was at least one setting for x that made it true. And so this is just a direct connection.

On the other hand, for the universal quantifier, this a little bit trickier. And it's drawn at a very high level here. You need to check both settings of y. So what you're going to end up doing is if you enter this gadget, you're first going to set y to true and exit out here. And then recursively, the rest better be true. If it is, you come back here. When you come back here, you mark the fact that you got one setting correct.

Then you set y to false, or the other setting. And then you go out here again. And if the formula is again true, you'd be able to come back. And if you've accumulated two correct answers internal to this gadget, you know that the overall formula is true, with a universal quantifier here. And then you'll be able to return out to the caller. So that's sort of the recursive algorithm view of what's going on here.

Let's do that with pressure plates. So it looks a little bit messy, but the idea is very simple. It's to implement exactly what I said. First, these clause gadgets, these are 3SAT clauses. And what we're going to do is just have three doors, and you have to-- this is drawn in the reverse order of what it should be. It's going right to left usually, but it also works in the other direction.

If any of these doors is open, you can get to the other side. And in order to get through all of them, all the clauses must be satisfied. So that's the 3SAT part. Really easy in this setting. Where door open means that the variable was set true. In fact, we will have both an xi and an xi bar door. And so you use the corresponding literals there. Cool.

So now this is an existential quantifier, an easier case to think about. So let's say we come in from the left, as in here. And that's coming into this part of the gadget. And we have to give the player a choice. They can either take the top path or the bottom path. Now, there's a worry that maybe you go a little bit this way and then double-back. But that's not going to be possible. We need to argue that. And negative means close, and positive means open. So if I come in here, the first thing I do is close the A door. That's going to prevent me from later coming back through this way. I can only have one door for each button.

But I want a literal to be able to appear in many classes. So in fact, I'm going to have several copies of x here. So this is the first occurrence of x1 bar, and the first occurrence of x2 bar, and so on. I want to open all of those doors, so I can use individual copies of the variable up there. And then I'm going to close all of the positive versions of x. This corresponds to setting x to false. And then at the very end, I open the B door, which lets me go through here.

The reason you can't mess up this gadget is if I went part way and then doubled back, I would immediately close the B door, which means I can't really exit this way, which means I'm going to have to go all the way through here and reset all of the variables. And this is exactly the complement of this. And so exactly one of these will be fully traversed. The other one will be nullified. And then that corresponds to existential quantifier. Question?

AUDIENCE: The doors start in a closed state?

PROFESSOR: Let's say all the doors start in a closed state. Although, I don't think--

AUDIENCE: Can you close things more than once?

PROFESSOR: Yeah. Sorry. One important thing is if you click on Open when it's already open, nothing happens. If you click on Close when it's already closed, nothing happened. So you can close multiple times. It doesn't make it more closed. It's just closed.

AUDIENCE: Stepping on the Close and then going back and opening B seems to work?

PROFESSOR: So you could do some of this and then double-back. You're right. B will open up again. But by traversing all of these, you undo all of the things you did up there, because these are complementary. So you've fully traversed one as the last thing you did. That's going to be the one that sticks. Yeah. So that's pretty nice.

And on the other hand, here is a universal quantifier. Not that much harder. But this is essentially-- there's one door inside here which is keeping track of, am I going through this for the second time? So in this case, there's no choice. We just have to try both settings. The first thing we do is set x to be true. That's this setting. And we also open door-- or sorry, we close door D. Oh, right. Sorry. Your other question is, do the doors start open or closed? I don't think it matters, because we have these close things in the beginning, essentially, everything gets closed at the beginning.

So here we're going to close D, which is our final exit. That's where we want to get. So that's going to be hard. We have to get to the open D spot. Then we set x to our setting. Then we open A. And now we're allowed to go here. You could try to go this way, but it's just going to close a couple doors. So it doesn't really help you. So we recursively call the thing with x true. If it comes back with the thing being satisfied, we can't go through here, because D is closed. So we're going to have to go through here. We open B. We immediately go through B. We immediately close B. So it's like that's a one-way gadget.

Now we set x to false, the opposite of this thing. And we can't go backwards, because B is closed. Now we open D. So if we can get through again, we'll be able to exit. We can't immediately turn back, because again, B is closed. We open C, visit C, close C. So another one-way gadget. And we also close A to prevent other weird things from happening. So now we go through again. If we get back with D open, then we can return and we're happy.

And this gadget in the actual construction, keep in mind, what's happening here is we are making an exponential-length puzzle. The length of the solution is going to be exponential, because you have to try both settings of all the universal quantifiers. So any gadget is going to be visited many, many, many times. But each time, we're completely resetting x, resetting x again, and then returning. So we'll revisit, will reset x to true, and then later to false, and so on. But we are checking all the possible assignments for here. And we're letting the player non-deterministically guess assignment for the existential quantifiers. So that is pressure plates in the abstract PSPACE-completeness.

AUDIENCE: Sorry for another question. In our usual sense of reduction, you usually think of a polynomial size thing that we make.

PROFESSOR: Good. Yeah. I didn't mention it, but there are different notions of reductions for PSPACE-completeness. But for the most part, we will just use the same notion of reduction we did before, which is the running time of the reduction should be polynomial time. So here we're given a formula which is expressed in this kind of succinct form. And we're producing a level whose size is polynomial in that.

Now, the actual execution of the game, the trace of what happens, or the certificate that is solvable, that has exponential length. But we're not talking about MP here, so we don't need polynomial certificates. But the reduction itself is polynomial. And again, of course, you need to preserve the yes/no answer. So that's our notion of reduction. Question?

AUDIENCE: Is there a way to make this reduction work where you just have toggle plates rather than-- so that they're not marked with Open or Closed, but just change the state of the door?

PROFESSOR: Let's come back to your question. I have a proof that does essentially that. I was wondering the same thing, but I'm not totally sure. So let's come back to that and try to figure out the answer.

So this applies to many, many, many games. Here is an example of an RPG, one of the earliest-- *Eye of the Beholder.* I think it's officially DND-licensed. It has a small number of pressure plates. Here you are. You touch the pressure plate, and this door opens. Ta-da! We don't know to what extent these things exist, but reasonable to assume you have this kind of infrastructure. And then PSPACE-complete. Even back in the early '90s.

A lot of first-person shooters have pressure plates. *Doom* doesn't have literal pressure plates, but it has triggers. There's like walls that if you walk through them, something happens. A block can appear, or a monster can appear. If you make a block appear, that will essentially close a door. If you make a block disappear, which is also possible on the general script editor, then you can do all this in *Doom.*

Quake is the first one I know that has actual physical representations of a pressure plate. You touch this-- it's a little hard to see, but this drawbridge is opening.

And a little more far. Adventure games. It's hard to define the generalization of an adventure game, but one definition is that there's an older engine for adventure games called SCUMM for all the old Sierra games. This, I think, stands for *Maniac Mansion.* That was the original game they read it for. But if you ever played *Monkey Island* or *Space Quest XII,* which that doesn't exist. But there was *Space Quest IV* where they time-traveled to the future and played *Space Quest XII.* It's a very good game, by the way. A lot of these are actually freely available now.

And anyway, it's an engine. It has a very restricted set of scripting languages. And in particular, it lets you do, effectively, the style of pressure plate things, where if you talk to someone, then they make you go to a new place, and they open up some other traversal options, or vice versa. So you have to take this a little bit liberally, but you can essentially implement these kinds of things, because you have rooms. Here you don't even need planarity, and you can implement pressure plates. So that's cool.

A little more literally pressure plates. *Prince of Persia*, one of the first platform PC video games, I guess. I don't know exactly the right qualifiers. This, I think, they even did motion capture. It's one of the first instances of motion capture for video games in the very early days of graphics. And you see in particular, this was a pressure plate literally that opens up that door. So that's pretty cool.

One catch is that pressure plates-- in this game, you can jump, so maybe you skip a pressure plate. But if you put a pressure plate on the top of a ledge that you have to climb up, that will force you to touch it. So that's *Prince of Persia*, PSPACE-complete.

All of these games pretty much are in PSPACE, because the state of the game is some polynomial space thing-- which things have been pressed, or opened, or closed. The total space of the game is usually polynomial. Each thing has some, at most, linear number of bits of states. So the overall thing is polynomial. Or I should say constant number of bits. That doesn't matter. I get to take a log.

And so you're just traversing this giant state machine. To store the current state of the game only requires polynomial space. And at best, you're making a non-deterministic choice at each step, of whether I push this button, or that button, or wait. As long as you discretize time, you get an easy NPSPACE algorithm for pretty much all single-player video games. Not quite all, but all the ones I've talked about here. Cool. So that is metatheorem three.

But this issue of having to force someone to pressure plate is a little bit restrictive. And that you can definitely get rid of. And this is what Giovanni calls buttons in metatheorem four. If instead of doors and pressure plates you have doors and buttons, you get PSPACE-completeness. Now, this is a little bit more restrictive in that we're going to allow a button to, let's say, open or close three doors. So pressing one button has a bulk effect. It affects three things at once.

This has since been improved down to two. But that hasn't been published yet. So I am going to cover the published version, which is this one, where you manipulate three doors for every button that you press. And here we're drawing it with the buttons on the sides of the walls. There are many, of course, physical manifestations of this. And the proof is a reduction from pressure plates. Question?

AUDIENCE: The buttons are allowed to close one door and open another?

PROFESSOR: Yes. So you can see that here. One button will close some doors, open others. This is one gadget. Here's the gadget on the top-left, and this is a sample execution going left to right. This simulates a pressure plate. And in this case, you can enter from the left and go to the right, or vice versa. Although, in the proof, we actually only need one of those traversals. So you could just stick with the top half of the gadget.

And the white means open, the black means closed here. So the initial state is supposed to be A is open, B is closed, C is closed. So you come in here. If you want to traverse, you have to hit one of these buttons. Let's say you hit the top one. And we're going to do this plus or minus x, which is the actual thing you wanted this pressure plate to do-- open or close door x. If this wasn't open for x, there would be a plus x here.

But then also, we will close the door we just came through and open the door to go to the next spot. Then you go to the next spot. Let's say you press the bottom button. Because that opens C. And you want to go forward. So it's the only one that opens C. This one would have closed C. And we also closed D and opened B, which gets us to the next stage.

And let's say we press this button, because this is the only one that opens D. Then we close C and open A. The point of that is to reset the gadget to its original state of open, closed, closed, open. And then along the way, we also activated x in the way that we wanted it to be activated. So that was pretty easy. Now we have buttons which you can press or not. But when you press them, it activates three things at once.

And this also exists in many video games. Here were a few that I found. We have the original *Tomb Raider*, and I think all the *Tomb Raiders* probably have some notion of switch, which opens, in this case, one door. But you could imagine generalizing to more doors.

We have *Sonic the Hedgehog* here. When you press that button, it opens that door. You can use that to simulate the same effect.

And this game, *Lost Vikings*, which I barely remember playing, but I do remember in particular that the main character here is called Erik the Swift.

[LAUGHTER]

And you see him pressing a button which, in this case, actually opens two doors. So that's pretty compelling.

And in general, tons of video-- I mean, almost every video game with some puzzle element has buttons in it. So now you know a ton of video games that are PSPACE-complete. Questions? All right.

So the next metatheorem is going to be a specific kind of implementation of that same construction. It's more of a physical realization of a door. It's a door that can be opened or closed. And it has various ways to interact with that door. If it's open, you can traverse through it. If you go through the clause connection, you are forced to close the door, in which case you can't traverse it. If you go through the open connection, you can either just go through, or you can open the door if you want. That doesn't really-- you don't need this connection, but a lot of gadgets will end up doing that. So this is just one bit of state. Door is open or closed.

And we can simulate this kind of thing, or simulate the previous type of pressure plates if you have a door, and an open pressure plate, and a closed pressure plate. We're just sort of remapping that into this gadget. When you go through the open traversal, that's just visiting here. And when you go through the traverse connection, that's going through the door. And when you go through the close connection, that's visiting that pressure plate. And vice versa. So you can take the previous reduction, translate it into this view.

Of course, you're going to end up with crossings, and it will be kind of ugly. But you can do this. So as long as you have a crossover gadget, doors-- this is called a door gadget-- by themselves are PSPACE-complete. So we'll call this metatheorem five. I mean, this is very similar to metatheorem three, just like two and one were very similar. But just give it another name.

Doors plus crossover are PSPACE-complete. So this is sort of a reinterpretation, if you will, of the original Viglietta paper. Viglietta's a co-author, but this is the Nintendo Games paper. So this is-- you can think of as a simplification of that previous metatheorem with pressure plates.

And then that has been implemented in even more games than you've seen so far. So one of the first ones we did was *Legend of Zelda-- A Link to the Past.* So a few things going on here. But this is basically the door gadget. And then you're done. There are lots of ways to do crossovers in *Zelda* that's not too hard. We showed a hook-shot version some time ago. But anyway. So there are these notion of doors in *Zelda.* These are currently in the closed state. And when you walk through this in the open state, you'd come out on the other side. And so in particular, the to traversal is going to be whether this door is currently open.

Now, getting back to the question, in *Zelda-- A Link to the Past,* you only have toggles. If I press this button, all the doors labeled one switch state. But you can't jump and then land on the thing again. In this case, you are falling down. The drawing isn't clear. This is the 2D drawing of falling down a tunnel. And then the only thing you can do is walk down. And then you hit a teleport, and you immediately go here. So this is one way to get traversals to work by preventing backtracking. Sorry-

- getting toggles to work by preventing backtracking.

So the intent of this gadget is there are doors labeled one and two. And it should always be exactly one of those is closed. Either the one is closed or the two is closed, but not both. And in *Zelda,* all the games are initially closed. So this is annoying. So we have to build this initialize gadget through all of these doors, where you come in here. This is the end.

These are one-way teleporters. You can't go up here. So you have to go here, toggle two. Then go to this teleporter, go here. This is a block, so you have to go over this way. We're going to chain all these gadgets together through these initialize paths, so that all of the doors are initially in the closed state, with one closed and two open.

And then there's a crystal-something that you break. And it flips. All of these blue blocks become these, and all of these become blue blocks. And that only happens once. So now, basically, the initialize paths are destroyed. So now think of this as being untraversable, and this is traversable. OK? So now, if the doors open, door number one is open, then we can traverse it. That's cool.

Let's look at the closed paths. The closed path-- exactly one of these is going to be open. So if the door was already closed, that means two will be open. We go down here. We teleport to here. And then again, because two is the only thing open, we go out through the close exit.

On the other hand, if the door was open and we're trying to close it, we have to be forced to close it. And indeed, we'll go through door one. Can't go through door two. We'll go down here. We'll toggle door one. Now door one is closed. And so is two. We'll teleport to here. Remember this is blocked now? Now we're forced to switch two as well. So we're effectively simulating the effect of doing two doors with one choice of a thing. First we flip one, then we flip two. Then we teleport here. One was open. We flipped everything. Now two is open, and we go out through the closed. So in both cases, we exit through close. Yeah?

AUDIENCE: How do you ensure that the blocks only get flipped once? Because the crystals can be hit as many times as you want, right?

PROFESSOR: OK. I had forgotten that detail. Essentially, we want to just put a one-way gadget. So when you hit the crystal-- I mean, either you hit it or you don't but if you hit it, then we have a cliff that you fall off of and visit the rest of the game. So you can't go back and flip the crystal. If you don't flip the crystal, then the gadget will break in other ways. If you didn't flip everything, you won't be able to traverse again. Yeah. I should have that drawn, but I don't. OK. So there's just one crystal, and it's hidden in between the initializer and the rest of the traversal. OK.

So what haven't we done? The open case. So maybe we come in the open door. Well, we're not going to have a choice. Either it's already open. Then we'll go through here, teleport to here. And we can go through the one, and we're out. Or it's closed, in which case two is open. Then we can't go here, so we go down here. We toggle one, we toggle two. And then we go out here. And now one is open. So it's kind of cool.

When we're going through open, we know that we'll end up opened. And so we'll go out the open. When we go through the closed traversal, we know we will end up closed. And so we will always go out the closed exit. And traverse is just completely separate. Cool? So this is a door in *Link to the Past.* And so that's a good example of toggles.

If you, again, have sufficient crossovers and so on forced, I think you could argue this is more generally letting you deal with toggles as long as you can, right next to a toggle, put a-- after traversing it, you essentially have-- say you have a one-way gadget before and after it, in a certain sense. Here in *Zelda,* there's only two levels. So we can't make you fall off another cliff. But it's the same thing with the teleporter. We have a one-way right before it, a one-way right after it. You could only hit it once, then you're essentially doing toggles. Cool.

Next one is *Donkey Kong Country*. This is a recall of the physics of *Donkey Kong.* You can jump around and kill enemies and stuff. But the main goal is to get to a destination other than points. There are these bees which kill you if you touch them. There are ropes and things you can climb. And there are also barrels that you can throw at things.

And there are three main *Donkey Kong Countries, 1, 2,* and *3*. And they each add an additional feature. But none of them share that feature. So there are three different PSPACE-hardness proofs, each with an additional feature.

This is *Donkey Kong Country 1.* Let me tell you the notation. We have these bees, which are stationary. So those serve as obstacles you can't touch. Then there are these bees highlighted in red. They're not red bees. That's different. But they are moving, according to this arrow. So you imagine this whole thing shifting over to here, and then shifting all the way over to here, and back and forth. And this is naturally occurring in *Donkey Kong Country*. You have bees which have some horizontal path that they traverse. But we're going to generalize a little bit and allow the bees to move different amounts. This one only moves a little bit back and forth-- I guess between here and here. All right.

Then there's this additional feature, which is this tire. And the tire, if you land on it, you basically bounce back up very high. And that's going to be annoying. From the traversal standpoint, we're going to be in this barrel. It's going to shoot us-- we're forced to shoot straight down. And if we hit this thing, we are basically stuck in this connected component. There's bees everywhere. We can't get out. OK?

On the other hand, if the tire were up here, then if we shoot down, we can time it right, wait for the bees to be going to the right here. And then we can run down here and fall down, and go through the traverse out. Because of these bees, which are going to go most over to here, we can't go out through the open setting from traverse.

But if we come from the open setting, because these bees are going to go way over here, we can run over and push the tire up to this position. So if you land on it, you bounce. But if you come from the side, you can push the tire however much you want. So that will be pushing it into the open state. And these bees will go right up to here. And that's as far as we can push it, right on that ledge. Because you don't want it to fall down the ledge, which also happens in *Donkey Kong* physics. OK. So that's open. You don't have to open, but you can.

Now, close. You have to close the door. And this corresponds to coming through here. If there was a tire here, if we were in the open state, we have to push the tire off of the edge here. And then it will roll down to that position. And if we time it right, the bees will be right over here. And then we can jump around. You can climb the rope. And if you're really fast, you're climbing the rope. And these bees are really slow, rather. Then you go over here. This might not be to scale. Maybe you want to widen the section here, so that you have more time to climb the rope. But that's just constant factors.

Either you allow me to slow down bees, or you stretch this diagram a little bit. OK? So that's close. In order to get from here to here, you must push the tire off the edge. And then it will roll to exactly there, preventing you from traversal. So that's a door. *Donkey Kong Country 1* is PSPACE-complete.

What about *Donkey Kong Country 2?* It has no tires. But it has the balloon. The balloon-- normally, balloons go down, as you might know, from gravity. But if you have these air currents, the balloon will rise. And when you're on the balloon, you can use it as sort of a ledge. It may still go down, but you can control whether it falls to the left or to the right, according to whether you press left or right on the controller. So this is actually the closed state.

Again, there are these bees. They're moving a little bit. But if you time it right, you can get down here. But if you land on the balloon, there's no way to get beyond this barrel. And so you're stuck. If the balloon is out of the way, it will end up being here at position A. Then when you shoot from this barrel, you end up in this barrel, and you get to traverse out. So that's good. Again, these barrels is a forced shot. So you can't make any decision at those types of barrels. OK. Good.

So the idea with the open clause is that there's a little bit of rope here. So you would climb this rope, jump. When the bees are over to the right a little bit, they get over to here. Then go on this balloon, push it over to the left here, and then jump back out, and grab the rope, and leave. And if you get the balloon to here, it will settle to this position, so that you've successfully opened the gadget even though you couldn't get here from there.

On the other hand, the close is going to force you to reset that. So we get shot over this way, shot down here. Now, it could be we just get shot, shot, shot, and we're out. But if there was a balloon here, we will land on it. In order to get to this barrel or the next one, we have to push the balloon over. Then we can go here or go here. And we'll end up getting shot to this position and out. But because of these air currents, the balloon will go up to here in *Donkey Kong* physics. And so when you do close, you are forced to put the balloon back here, which is annoying for traversal. So that's a door. Question?

AUDIENCE: Is there any reason you need the [INAUDIBLE] at the top?

PROFESSOR: Why do we need this exact? I don't remember. I was wondering that as well. Yeah. What it does do is force a particular sort of timing maybe, about whether it's to the left or to the right. But not sure. I'll bet there's a reason. But this figure was drawn by Giovanni. Cool.

What about *Donkey Kong Country 3?* Well, *Donkey Kong Country 3* has neither balloons nor tires, but it has tracking barrels, with a T on it. So tracking barrels, if you land in one, you can then slide it left or right. And when you leave it, you always get shot up. And even if you jump out and then, say, go over here and fall, the tracking barrel tracks you. So if you're coming from traversal and you get shot down here, if the tracking barrel's out of the way, then I traverse. Then I'm forced to go down here and leave via the close exit, which is not what I want. I want to leave according to the traversal exit.

Now, this has some leakage, but you can show this leakage is OK. You can go from traverse in to close out. But that won't end up helping you, because you were already close, so you knew you could already get to close out. So this is the closed state when the tracker barrel's over here. This is the open state. When we're in the open state, we come through traverse. We can jump out and leave here. So that's good.

On the other hand, if we come via the open position, we can go get shot down here. And if the barrel was there, we're going to have to move it over here. I guess we don't have to. We could leave it there. But we are able to bring the barrel over here and leave through open. And now we're in the open state. That's good.

On the other hand, if we come from close, if it's currently open, we will be shot into the barrel. And then to get over here, we have to bring the barrel over here, putting it into the close state. Now, this is the relevant part, where the tracking hurts you. You go in here. You'd like to just jump out and get out here, but because it's a tracking barrel, if you jump out here, the barrel will come over anyway even though you don't go back into it. So if you leave through the close exit, you have to bring the barrel over to the right. So that forces the clothes, preventing traversal. That's *Donkey Kong Country 3.*

But there's one Nintendo game that rules them all, and that is *Super Mario Bros.* And for the first time, we are announcing a PSPACE-completeness result for *Super Mario Bros.* So the Nintendo paper and another paper by Giovanni, which I will talk about next, was prevented at Fun with Algorithms earlier this summer. Two people were there. And then we started talking, oh, we should really prove *Mario Bros.* is also a PSPACE-complete. So this is our current reduction.

You may recall another crossover gadget that involved invincibility stars, or mushrooms, or something. Those are consumables. We can't afford consumables in a PSPACE reduction, because we have to visit the same thing many times. But using *Mario* physics, we designed this crossover. You're going to be small Mario throughout this reduction.

If you come up here and you run, you will fall here. And if you come over there, you run, you'll fall here. But you can't jump around and land here. You'll end up down here. And that's larger than your jump distance. OK? So as long as you can't glitch through walls, this is a valid crossover and it's reusable. It's a one-way crossover.

You can only go from up to down. OK.

Now the door. The door's a little bit weird. But it has actually been built by modifying the *Super Mario Bros.* ROM, so it does exist in the physics of the real Mario. This doesn't have to be length, but this is a fire bar, which is normally rotating. But if it has size one, then it doesn't really do much rotating. So these are things you're not allowed to touch. The version we implemented has fire bars. Those do exist in the ROM. But even if you have fire bars that are much bigger, you can still do all the things you need to do in this gadget. It's just a lot harder to do correctly. I was trying it earlier today.

[LAUGHTER]

And then we have this guy. I forget what they're called, but they're dropped by the cloud thing in the sky. You can also create them at the beginning of the level even though that doesn't appear in the true levels. You can make a new level that has these guys initially. It'll just be the case that there's also a cloud floating around at the top. But if you shield from that, you're OK.

So there's one of these guys, and he can basically be walking back and forth here or over here. And you can flip that by if you hit this block with the right timing, this guy will end up bouncing and walking over here. And practice. You have to hit it a few times. Sometimes he'll walk back over. But if you get the direction that he's facing right, he'll jump up and then keep going. And then he'll be over in the other side. OK.

But you can't just reach everything here. If you are on the open side, you can hit him if you want and put him over there. That's going to be the open setting. The traversal requires you to go through this way without the ability to hit this thing. And so if the spiny guy is over on the left, you will die, guaranteed. We checked. And otherwise, you jump-- if he's not there, if it's on the right side, then you can traverse. So this is the open state that's drawn. So this is the tricky part. We need something that's untraversable by the player but traversable by the monster. And this could be a fire bar that's rotating. If you time it right, you could still rush by it.

And of course, close. Close, we need to force this guy to be on the other side. And so we do the same thing that we did on the left to go through to close. Just like on the left, this guy has to be on the other side. So first, you can knock him over to the other side. And then you can go back and jump through there. So *Super Mario Bros.* is PSPACE-complete. Yeah?

AUDIENCE: What was the advantage of the rotating fire bar?

PROFESSOR: Oh, it's more naturally occurring. In real *Mario,* there are rotating fire bars. There's never this single fire just floating in midair. Fire bars can float in midair. In all the levels that exist in *Super Mario Bros.,* fire bars are a fixed length. We have six fire things or whatever. But in the ROM, it has support for making fire bars of arbitrary length or up to some size. So you can make this in the ROM, or you can have it rotating. And then it's a more intense experience.

AUDIENCE: Why are there fire bars [INAUDIBLE] that they're hitting?

PROFESSOR: Yeah. That's dealing with glitches. Yeah?

AUDIENCE: What is the shell guy dropped by the cloud thing? Or I don't know the game.

PROFESSOR: Oh, in the game, the cloud is up there. And every few seconds, he throws one, and it falls in some kind of random spot. And then he traverses left and right like a regular turtle. But in this, we're assuming that you can have him initially appear there. So he's not thrown by the cloud. There would also be a cloud throwing spiny guys. We're assuming that this guy can just be there from the beginning.

AUDIENCE: So it drops more than one?

PROFESSOR: Oh, yeah. It drops one, and then drops another, and drops another. I think it's unlimited. I mean, there might be a practical limit given by the memory of an NES, which is pretty small. But I think it limits to like eight creatures on screen at once or something. But we're generalizing that as well. Are there questions? OK.

I have one more proof I want to cover. And this is the other Giovanni paper, and

Fun with Algorithms is here. And it's about *Lemmings.* If you haven't played *Lemmings,* it's sort of the first real-time strategy game.

So you have these characters which appear periodically. Then you can click on characters and make them do different things. You can make them build bridges. You can make them dig horizontally. You can make them dig vertically. And so going through that again. Here we're making this guy climb, and then digging down. And then digging left. And then digging down. And then they can dig left to switch directions. And then make them dig down again. And then build bridge and dig through. This is, I think, level two or something. It's quite challenging. And your goal is to get as many lemmings as possible to the exit.

And the way you do it is you select the ability you want, and then you click on the lemming that you want to do it on. And otherwise, they are walking around. So how many people have played *Lemmings?* Yeah, this is more popular one. If you haven't played, you should. It's really hard, though.

I was reading Wikipedia. Sarah, one of your favorite authors, Terry Pratchett was playing *Lemmings.* He wrote a book that is clearly inspired by *Lemmings.* And then he said he had to delete *Lemmings* from his computer and override it in order to stop playing. It was so addictive. There are a bunch of levels. It's fun.

There are many free versions as well that have duplicated the *Lemmings* physics and so on, with new levels, and there are level editors, and all that good stuff. So you might expect this game is pretty hard.

Here we're going to prove PSPACE-hardness. There was the first paper by a Graham Cormode, which showed NP-hardness and conjectured PSPACE-hardness. So this was the results before. So there are all these abilities. I haven't defined all of them. I'll show you just a few of them. If they're unlimited, normally you're given a limit to how many you have. But even when they're unlimited and you have arbitrary time, the game is still in PSPACE, because again, the state is polynomial number of bits. OK.

There's another thing I didn't mention, which is hazards. These are traps. If your lemming goes there, you die. So those are also important for some of the proofs. And so Graham showed that if you have diggers and a time-bound, then you get NP equal to time-bound, you actually get NP-completeness. And you can convert that into other abilities.

But the [INAUDIBLE] new results are this one. We're going to focus on this column, which is given builders and bashers, either unlimited or given an exponential number of them-- because we need to run for exponential time-- and hazards, and just one lemming, the game is PSPACE-complete.

Another result in the same paper that might interest you-- but we're no longer in the Approximation section-- is APX-hardness for any ability you want. And a polynomial number of lemmings, and time, and hazards. And it's based on this weird thing about hazards that if you have many lemmings go through a trap at once, then only one of them dies.

[LAUGHTER]

Because the trap has to activate, and it takes a certain amount of time. So if they're really bunched up, then most of them can get through. So you can use that to reduce the number of lemmings. And you're trying to approximate how many lemmings get through. OK. So that's fun. But I don't have those pictures here, because the PSPACE-hardness is kind of the more impressive technical thing. And it's already quite complicated.

So in order to really get this proof right, Giovanni went through an incredible effort of documenting *Lemming* physics at a pixel-by-pixel level, because it is pretty weird, especially when you start using the bridge building. If you've ever played this game, it will stop at kind of weird moments.

So here is the physics of what happens. In general, the lemming is represented by a pin right under its left foot. I don't know. Maybe that's the right foot. And that pin should always be on the ground. If it's not on the ground, the lemming will fall. If it falls too far, it dies. But that's not relevant here.

So the general walking-- all lemmings start as walkers. And what they do is basically move left to right. There's an animation. But mostly, this pin is moving one pixel to the right at a time. And so it's visiting these things. Once it sees, oh, there's no longer an obstacle beneath me, I'm going to fall this amount, and then I will fall one pixel at a time until I hit the ground. So there's a gap here. That won't be important here, but this is the real physics.

If I hit a step-- and this is, at most, eight pixels up-- then I will rise up and go over. But if I have two big a step, more than eight pixels, then when I hit the wall, I will switch directions. So in this case, he will hit here and then climb back up. That is the regular walker physics.

Now, we're going to use two abilities. One is called basher. And there's some animations here of what happens with bashers. In general, it's horizontal digging. But it's kind of weird horizontal digging. There's this particular pattern that you dig, which ends up tunneling a big horizontal rectangle more or less. But the way it tells whether it's finished is it looks at these four positions right in front of the dig mask. And if at least one of them is solid, you will keep digging. If they're all empty, you will stop digging. That's important, because we need to see exactly when these things stop happening. Then you return to being a walker.

So if everything is normal, you just dig through. Here, that position was when all four were empty. One was empty, but then all four were empty. And so then you stop digging and keep walking. OK. But there's also this notion of steel. Any pixel can be marked as steel. It's a separate, independent bit.

Usually there's some graphics to indicate your steel. But here we're going to use blue. And there's this one steel check. And if that position is steel, you will immediately stop digging. Or you will stop digging when that start position hits a steel pixel. So we're going to use that in the proof. But that's how bashers work.

Next we have builders. These are building the staircases. And there's this subtlety that the solidity check are these three points. If any of them is solid, it will stop digging. And if this one's solid, you'll also switch directions. There's also a limited number of stairs, because you can only fit so many in this little bag.

[LAUGHTER]

That won't matter for the proof. We're always going to stop before we hit the 15 stairs or so, 10 stairs, something like that. So you can see it in action here. You place the step. The initial step overlaps your feet by two pixels. And then you do this offset of two to the right, one up until you hit something like that. And then in that case, you turn around, because you hit the wall, so to speak. But notice there's this big gap here. OK.

So now we want to use metatheorem five. We want to build a door and crossover, I guess. I don't know if I have the-- I do have the crossover here. It's not so hard. First we just need to be able to move lemmings around. And they move in a very simple way. So yeah.

AUDIENCE: How did he check all these pixel things?

PROFESSOR: [LAUGHS] That's a good question. Definitely one thing he did is look at one of the clones of lemmings and read the source code. How that clone was done to mimic exactly the original lemmings I'm not sure. So I don't know whether this has been compared against disassembled, original *Lemmings* code or what. I'll follow up with him. I was talking to him earlier today. OK.

So we are only going to give the player the ability to do builders or bashers. And we want to prevent the player from doing builders in random places. So these red things are traps. If you ever at any point decide to become a builder, your dot will not be here. Your dot will be above. And then it will be touching the trap, and then you die. So these red things are to prevent you from building anywhere. You also can't bash anywhere, because let's say this is all steel. OK. Or actually, I think the whole background here is steel except in specific places in the gadget.

So this is a small thing. This basically let's us ascend slightly. Because lemmings only like to go left and right and there's no jump ability, we want to be able to build a graph. So this will let us go up a little bit. This will let us go down a little bit. This will let us change directions into going up, change directions in going down. This is a crossover, because lemmings will just proceed to the right. There's no way to make them change direction unless they hit something. So that's basic lemming driving.

These are not explicitly stated in the metatheorem. You see here the vagueness of the metatheorem. We need the ability for the player to choose whether lemmings go one way or another way. We need that, for example, in the existential quantifier. But in general, we were kind of assuming that the player has this kind of agency to choose directions of lemmings. So for that, we need a kind of fork gadget.

And you can either do a build immediately followed by a bash, or you can do a bash in order to get down. So if I do a build and then a bash, I will stop building and then just make one step. Because if I made a second step, I would die from the traps. That will let me choose to go to the top path. Or even if this is already there, I could bash it away. Or if it isn't there, just wait. And then I will go down here. OK?

So no matter what the state of this gadget is, I can either take the top path or go down to the bottom path. Because here I have unlimited bashes and builds. So those are the two states. This is the non-steel part. We need that for the bash to succeed. So it ends up bashing, but because it's all empty here, you immediately stop bashing. OK. That's the fork.

And finally, the door. So if the door was open, you go through it. If it's closed, you die, even if you try to build a bridge. If I go this way, I can either fall down, or I could bash it away and then open the door. But I still fall to the bottom. If I come from here with a closed door, fine, I can exit. If it's an open door, I better build this thing. If I build too far, I'm in trouble. But if I build just once and then immediately bash, then I will get to the right.

What did I just do in rapid fire? That was the closed path, where I must put this back in in order to survive. The open, I had a choice. I could bash it. In either case, I end up to the open exit. And for traversal, you will only survive if this is not there. And you get out. So lots of crossing paths. But you can see why you need this pixel-by-pixel analysis to really be sure that this gadget does exactly what you want. And that is a door for *Lemmings.* And therefore, *Lemmings* is PSPACE-complete. Pretty epic. And that's all for today. Yay, video games.