RIK EBERHARDT: So welcome to Creating Video Games, CMS 611 and 6073. Is everyone in the right class? Yeah, all right. Cool.

So this class is taught by-- it's a Comparative Media Studies course. So we'll be talking a lot about the kinds of things that they talk about in Comparative Media Studies. We'll study research and design practices, but as it relates to software engineering, and in particular to developing video games-- so digital video games for, basically, screens.

The CMS department has a lot of different, really cool research labs that are a part of it. And we are one of them. So we are the MIT Game Lab.

Basically the whole MIT Game Lab staff teaches this class together. Because we really like each other, and it's really fun. But also because there's a lot of you. And it's a really big, complicated class to teach. So we need all the extra support we can get.

We teach classes in game design and game research. Those are all the course numbers that we teach other than this one. Maybe you've taken one of these before. Maybe you're thinking about some other game courses you want to take in the future. So take a look at that.

We also have UROP opportunities. So we'll advertise those classes as they come up. We do mentor Course 6 students for the UAPs. What we usually ask is for you to bring to us an interesting research topic that you want to research. And if we are also interested, then cool. A match is made.

Same thing when it comes to masters. So if you're going to get your master's of engineering here in Course 6. And you want to do it in game design or game development, and you need a mentor, we can be your mentor.

We also host a lot of events and outreach events and guest lectures. A lot of the guest lectures will be part of this course-- that we'll be opening to the public, as well, for a few of them. We've got Riot coming up in September. We've got Japanese developer Swery65.

Does anybody play *Deadly Premonition*? Yes! It's awesome. And he's coming to talk about it.

And some other cool stuff-- we also do a lot of events. So like game jams-- we'll have an alumni panel. And the next big thing is the Boston Festival of Indie Games is going to be on campus in 11 days.

So 200 developers are going to be showing off their games over in the Johnson Activity Center. And we'll have a conference track going on here. So take a look at that. You'll get all that information as the course goes on.

So next up, we just wanted to introduce ourselves and our individual-- basically what we do, and why we're teaching the course.

PHILLIP TAN: Hi. I'm Phillip Tan. I've been teaching various game courses here at MIT for about 10 years now. I'm a research scientist over at the MIT Game Lab. And I've worked with all of these fine folk for many years.

Up here are a couple of projects that I've worked on. On the right is a game called *A Slower Speed of Light.* If any of you are interested in relativistic speeds and simulating them on a computer, you are welcome to check it out. We also have an open source version of our code that lives on GitHub. So you can download that version using the same tools that you might end up using in this class. But basically it's, you travel near the speed of light, and you see what this might look like.

I play a lot of *StarCraft.* Anybody here play *StarCraft*? A few people. All right.

I've been working with Blizzard this past year to work on their spectator interface. The idea is that there are a lot of people who watch e-sports right now, especially on *Twitch.tv*. Who watches *Twitch.tv*? OK. All right. Lots of people.

E-sports specifically on *Twitch.tv*, or just like Let's Play? OK. A few.

Those are spectators, just as if you're watching a football or hockey game on TV. And they need infographics to come up in real time when something interesting is happening in the game. So I write code together with UROPs to be able to bring up cool infographics bring the hype, basically. So that's me.

SARA VERRILLI: Hi. I'm Sara Verrilli. I'm the development director at the MIT Game Lab. I came to the Game

Lab out of industry, where I've worked as a producer and a game designer and originally, how I entered it was as a QA tester. So I've held a lot of the jobs there.

I've worked with a lot of students on games when we were still running the summer program, the one we did with Singapore-MIT GAMBIT Game Lab. And now I have focused to mostly seeming to work on classes and teaching game design. So that's a lot of what I've been doing and what I've done.

RIK EBERHARDT: So I'm Rick. I'm the studio manager for the Game Lab. I teach this course. I co-teach with Sara 617, our advanced game studio course where you spend the entire semester making one game.

These are the games I've worked on at the Game Lab. I worked on all sorts of different games from anything to games about depression to games about identity, games that are demonstrating advanced 3D rendering techniques, and conference games-- PAX POX, a game that was played live action at PAX East in 2009 at our booth there. So I do digital and non-digital game design. But for the most part I'm studying and working on project management.

ANDREW GRANT: Hi. My name's Andrew Grant. I'm the technical director at the MIT Game Lab and help out with the classes too. I'll probably be helping you out a little bit with technical issues or what have you.

I was in the games industry. I started with Sara at Looking Glass and moved to Dreamworks Interactive. And after that, I kind of got burned out, to be honest. It's a pretty rough industry sometimes.

I spent about 10 years consulting. And I kept coming back to games and coming back to games. And so when I got the opportunity to come to the Game Lab, I said, well, yes.

So anyway, here I am. I've been playing and making games all my life. And I think we'll have fun doing it this semester.

RIK EBERHARDT: And if you can stand up-- our TA is Paulina. Wave and say hi. She'll be helping with all the administrative stuff. She'll be helping us with grading. And she'll also be helping with any questions you might have about some of the technology that we're using, too. She took the course this past last year.

Great. So get down your pencils. This is the cheat sheet, the one-line version of the class. If you learn anything, you learn this.

When you're working in this class, the best way to succeed is working face to face, testing your games often, prioritize, integrate, and cutting features early. Sleeping is awesome. Avoid doing 3D. Avoid doing network code. And please use version control.

We're actually going to require all of this. But every year, a team or two doesn't do one of these things. And it's OK. But they struggle. So if you do all these things, through the semester, you'll do really, really well.

Yeah, please. What was the question?

AUDIENCE: Slides. Oh, the

RIK EBERHARDT: Slides? Yeah. All the slides will be available on Stellar. We're posting all of our slides to Stellar.

I don't actually expect you to take notes on all these things. But it's awesome if you do. I remember it that way.

PHILLIP TAN: This is 140 characters.

RIK EBERHARDT: This was actually Phillip's tweet from last night, in case anybody was following him already.

SARA VERRILLI: Then they were probably not sleeping.

RICK EBERHARDT: Actually, yeah. So the next bit of administrivia, we're going to go over the syllabus a little bit. It's in front of you right there. If you have any questions, feel free to interrupt. Fellow instructors, if I say something wrong, or if you want to say, hey, but also this, please do.

So first thing-- the course is structured into four game projects. So you're going to be making four projects over the course of the semester. Our first three projects are about two weeks each.

The third project is a little bit longer, if you decide to work over the student holiday. And then project four is our final project, our main project. And that project's going to be done for a client who will be coming in to talk about what he works on later today.

So your first project is a non-digital project. It's a prototype. Your second and third are small, short, digital prototypes-- so making a digital game very quickly.

Your fourth is a longer time period project. But it should be about the same amount of features, the same amount of work, the same amount of game play as the first three. It's just you had a lot more time to actually do it right, and to do it well, and to polish on it, and iterate on it.

At the end of today, we'll talk about what we mean by these common themes. This common theme-- what did I say-- meaningful decision making, or meaningful decisions in games. And each time we start a project-- our first project starts on Monday-- we'll introduce the theme in more detail and talk a little bit more about it. But you can see all of the details about the projects that we have available for you at Stellar right now.

Our grading rubric for each project is largely team-focused. So all of your projects are worked on in teams. They start in small teams and get a bit larger.

You'll see each of these lines is 20% of the grade. So the actual working of the game is only 20% of your grade.

Following the iterative design process that we'll be talking about, using the project management practices that we'll be talking about, doing a good presentation-- a group post-mortem presentation about your game-- is another 20%. And then every student will write an individual post-mortem, their own one to two page essay, at the end of-- turned in at the same time as every project.

And what we're asking for all these things is we want to know about your process. We want to know how you made the game, what challenges you had while you were making the game, what difficulties you faced. How did you surpass them?

And even better, what are you going to do in the future? When you either go into the industry or go into another software engineering project or another class, what would you do differently, after the experience you just had? So 80% of your grade is team-delivered. 20% percent of your grade is your own work.

So speaking of in-class expectations, the class is a three-hour class. Holy crap. And it runs twice a week. Class time is actually allotted for your work in teams. We don't expect you to get your games actually built entirely within class. But important team meetings, test sessions, code review-- you're going to generally have a lot of time in class to do a lot of that stuff.

For projects one and two, you'll probably have about 60 minutes at the most per session to work on your team work outside of an activity. But we're also going to have a lot of activities where you're doing some of the work that's required in each project as part of a guided activity that we're going to run. So it's actually a little bit more than 60 minutes.

Products three and four, we're probably going to have about 90 minutes per session for working in your teams. And that's going to generally be at the end of class. At the beginning of class, we're going to have, usually, a guest lecture for those last two projects. And then we'll have a playtest session. And then it'll just be open work.

Our original classroom had flat tables. This classroom-- that moved. This classroom does not. So it'll be interesting. I don't know if it's got power on any of the stations.

But the short version of it is, bring your development station to class, if you can. If it's a laptop, bring it to class. You'll be using it a lot starting Monday-- well actually, starting two Mondays from now.

And the other thing-- teamwork is the heart of this course. Actually, participation is the heart of this course. So even though we're in this tiered thing, and I hate giving lectures to people this way, we're going to ask you to talk back to us a little bit. So fill in the gaps when you come in.

Participation is actually very important. Class participation-- so showing up to class on time, showing up to class at all, and participating in class is 25% of the grade.

So speaking of that, attendance-- you will be penalized if you miss more than three classes without justification. So what is that justification? It's like if you're sick.

If you're sick, we don't want you here. Stay home. Don't make us sick. Don't make your team sick.

He's a real stickler about that. I'm invincible. I won't get sick. But everybody else will.

Do not be late. Again, we're recording. We have very important guest lecturers who are in our network that we'd really like them to enjoy being here.

So I really hate it when people show up in the middle of a guest lecture. It makes me feel like a jerk for asking them to talk to MIT. So don't show up late.

But do show up. So if for whatever reason, the doors are locked, it really just means, hey,

there's a guest lecturer here. And coming in is going to disrupt things. The doors will get unlocked.

I don't expect to have to use that. But if I will, I will. Ha ha.

And usually, guest lectures end about 2:10. So you can come back. Come into class later to work, and actually get counted for the rest of the day. So I'd rather you come in later than at 1:15 while somebody's talking up here.

Class is generally going to start-- lectures are generally going to start at 1:10. We'll do announcements at 1:05, if that makes a difference. Because I know a lot of people are coming in from recitations and whatnot.

The other thing-- MIT has an anti-harassment policy. We've also got an anti-harassment policy. If anybody's on Twitter these days, or anybody's engaged in game culture these days, it's toxic. And it sucks. And it makes me mad I don't want to see it in this class, please.

So this is on the syllabus. It's the last page of the syllabus. Basically, don't make offensive verbal comments or any other kind of comments about all these different things, when it comes to gender identity, sexual orientation, disability, physical appearance, body size, race, religion. Just don't be a jerk. Don't be an ass.

More importantly, though, if you see it happening, let us know. If you are a target of this, let us know. There's ways to get around this. There's ways to report this.

In the syllabus, it actually lists the website for the official MIT way to report it, if you'd like to go that way, if you'd rather not come to either the instructors or your TA. And our email address-- videogames-bosses@mit.edu. Email that.

Don't email us individually. That makes sure that one of us reads it.

SARA VERRILLI: Can I add just one more thing?

RIK EBERHARDT: Yes, please.

SARA VERRILLI: Rick put up a giant list of things that you should [? worry about ?], and that you should not be harassed about. But I will go ahead and say that you should not be harassed about anything. Right? We have a list. It's not meant to be exclusive. So if there is for some reason some other things coming up and bothering you, come and talk to us.

RIK EBERHARDT: We are-- team development and game development is stressful. Yeah, question.

AUDIENCE: One phrase that I've heard recently that I love is, "Rights work up. Rules work down." You have a right to be respected. The rules that are listed will be paper mached on to make that work. But in general, be respectful.

RIK EBERHARDT: Yeah. That's really what we're asking for. Just be respectful for each other.

You're going to be working on large teams. There are going to be deadlines. It's going to hurt. It's going to be painful. Things are going to slip. Don't let that be the way that you express it. We'll actually give you some better tools to express your pain and suffering.

So the class is called Creating Video Games. How do we create video games? We have a whole bunch of people involved in creating video games.

Was this the slide you were coming in on? Or is it the next one?

SARA VERRILLI: This is the slide I was going to chime in on. But you were chiming so nicely.

RIK EBERHARDT: Come in whenever you like. So basically, all sorts of people are required to make games. We have all those people in the room.

SARA VERRILLI: Actually, a whole lot of skills are required to make games. You should not come in and think, I'm going to the programmer on this team. You should not come in and think, I'm going to be the game designer on this team. Or I'm going to be the artist on this team, or the sound designer on this team.

The teams are small enough, even when you hit the eight-person team for the large project. And the projects are always challenging enough. Even if you think you have a short list of things to do, you have a challenging project. That's one thing we've noticed every single year. Everybody manages to come up with a challenging project.

The team's going to need you doing everything you can do. And that means that if you can program, they probably want you to program. If they need programmers, chime up.

If you're primarily a programmer, or you think yourself of, they need you to chime up on the game design and comment on places where it will work better. And you understand how it will work better, because of your understanding of the code. If you're the game designer, you've

got to be ready to pitch in and help make all those things you see in your head happen, whether that's creating some sketched doodled art, so your team can go with the little pixel figures, or whatever.

We don't-- usually these teams aren't big enough to support someone who does nothing but draw art. We're not actually grading you on the beauty of your games. I mean, yes, we want good, well-functioning, rounded games. We want games that are relatively easy to use.

But you don't need super beautiful art for that. So if what you'd like to do is primarily art for the game, think about what else you can be contributing to the game as well, so that your game gets more functionality in it.

And let's see. For community managers, business analysts, and marketers-- the people who are pitching your game and talking about it-- don't mean on the person who ends up stepping up to organize your team and steps into the producer role. They're already doing an awful lot of job just producing. They're going to need your help figuring out presentations and talking about your game, and doing the reflection, and keeping the team running.

So don't think of these as a whole bunch of individuals. Think of it as a whole bunch of skills, one of which may be your primary. But choose at least two or three as your backup. That's my shtick.

RIK EBERHARDT: Cool. And that's basically how indie game development is run. When you see a game made by three people, they're doing all of that. And so that's basically what we're expecting from y'all.

So if we're not teaching a lot of the skills, the main skills that we are teaching in this class is giving you hands-on experience working on complex projects with large teams. So what does that mean? It's basically project management, team management, and communication skills.

You're coming to us through your prerequisites with some design skills. You're coming to us with some programming skills. You have support for some of the game engines you haven't seen before.

But like any other engineering course at MIT, you pick up a tool. You learn how to use the tool in the course, if you need to use it. So the tools that we've chosen should be short enough that you can be able to pick them up pretty quickly. And actually, we're going to go into that today as well.

But the remainder of the course-- particularly project two and project three-- are going to be focused on communication. If you're on a large team, how do you talk to each other? How do you communicate with each other? Is it going to be through email, instant message, chat, post-it notes, mash notes, hate notes, karaoke?

Bottlenecks are going to form in your work process. You're going to find out that oh, all the assets are going to one person. And they're sick today.

The quality of your code and assets are going to quickly nosedive if you don't have a plan.

SARA VERRILLI: Yeah. This is another one of those, don't assume that one person on your team is going to do all the art or all the sound. And that's all they're going to do. Because if they get hit by the flu truck-- as frequently happens around here-- you'll discover you've got nothing to put in the beautiful game that you've made.

RIK EBERHARDT: So to get around that, we're going to introduce to you tools and methods through the course that are going to help you get through these problems. How to do code workflow, how to do asset workflow-- we'll talk about that a little bit. That's actually the first assignment.

We'll talk about code review. We're not necessarily teaching a lot of this. We'll be talking about a lot of this. I know some of you have done some of this in other experiences. We actually want you to help other people out in the course.

Talk about your previous experiences working on a project. If you've done an internship before, if you've made an app for a company before-- because everybody has apps these days-- you've probably run into some of this. So we want to hear about your experiences, too.

We're going to talk a lot about product backlogs and task lists-- paper!-- using paper and spreadsheets to tell each other what you've been working on and how you're going to work on it. I'll be talking a lot about retrospectives-- about how, at the end of a project, you can talk about what you've just done, and how you can do it quickly.

But ultimately, we can going to give you all the tools, all the methods. The best way to learn all this, to know all this, is really just through prior experience. So this class is going to be that prior experience for you when you go to your next challenge, either working in the industry, working doing any kind of software engineering, or going to another class.

So again, the cheat sheet-- work face to face. Test often. Prioritize, integrate, and cut features

early. Sleep. Avoid 3D. Don't do networking. And use version control, please.

SARA VERRILLI: And this is where the cut I have from the last three years of this course, and the final presentations from pretty much every group that's given a final presentation for their final project-- in which they look at us sheepishly and say, you said we should test early. And you know what? We really should have tested earlier.

I can't actually be 15 different MIT students who have taken this course saying this sheepishly up on the screen. But I've heard it from all of them, pretty much every single group. So I'm going to give you guys the really early heads-up warning. Test earlier. We say it. We mean it.

RIK EBERHARDT: I'd like to introduce Pablo Suarez. He and his group will be our client for project four. So we're asking him to give an introduction of the kind of work that he does, and give a feeling for what the kind of work we're going to do in class. The first three projects are going to build slowly towards that final project in the theme and strength of those games. And in that fourth project, he and his group will come back in again and tell us more detail about what they're really looking for, and what kind of games they're really looking at from us. So take it away.

PABLO SUAREZ: Thank you. Thanks Rick. Thanks Game Lab team. I'm very grateful for this. And you-- I am grateful to you, even though you don't have much of a choice that I show up and you have to suffer through me.

My name is Pablo, Pablo Suarez. My strange accent comes from Argentina. I live in the Boston area, where a fraction of my time I'm a researcher. I used to be a researcher affiliated with MIT. I just came from the Lincoln Labs, where I'm collaborating with MIT folks there as well.

But most of my time, I work with a humanitarian team-- the Red Cross/Red Crescent Climate Centre. It's like a think tank within the humanitarian sector. It's based in the Netherlands.

We cover the planet-- 187 countries. I am in charge of research, in charge of innovation, mostly in charge of Africa. And unfortunately for the world, we have way, way too much work.

The workload is so ridiculously huge that in my opinion, especially coming from science and technology, we need to change the way we think about humanitarian work. And if it is about changing how we think, traditional ways are not working too well. And that's why I'm here.

I love this slide. We have a comfort zone, which is doing things the way we've been doing

them. Someone's hungry? Go get food. Someone's homeless? Go give shelter. Someone's sick? Help them heal.

But we no longer have the money, the brain power, the equipment to deal with all that's happening. Too much is happening. The climate is changing. Environmental degradation, urbanization, political instability, weird diseases-- you name it.

How can we learn more about what we can do? We need as humanitarian workers to get into a new place where we can do more with what we have. And that place is outside our comfort zone.

Now magic can happen. But very often, we step out of our comfort zone and we fall into the abyss. It's not easy to find the right place. This is a risky endeavor for me.

I'm here hoping that you guys will accomplish one of two things. One is maybe you come up with a game concept that has a life after the end of your course. You get your credit. You graduate. And the Red Cross or others can play a game that helps someone understand something, helps someone do something, helps someone collect data about something. And we'll talk more about what are those things that we want when we get to project four.

The other one is maybe we can lure you-- even if one of you, even if one of the Game Lab team-- to consider humanitarian work as part of what you may be doing in your future life, either as your job, or as your hobby. It's great fun. It's a great intellectual challenge.

The head of my team, Martin [? Van Ahls-- ?] he's an astrophysicist by training. OK? So you don't have much awareness of the extent to which humanitarian workers can come from unusual places. And we hope that MIT stops being an unusual place for humanitarian stuff.

So let me illustrate the kinds of challenges we get. This is an actual fax that actually arrived in May, 2008 to the office of my colleague Yusef [? ayd-Shaloush, ?] who at the time was based in Dakar.

Let me see if this works. OK. For those of you who know Dakar, it's near those green pixels.

That's where, at the time, the regional office was for West and Central Africa. 28 countries going from Mauritania-- where there have been a military coup, and there was bloodshed in the streets and Red Cross volunteers needed to learn first aid to prevent someone bleeding to death-- all the way to Congo Brazzaville in the southernmost part of this area, where there was

flooding-- urban flooding. And as Red Cross, we don't fully know how to operate in rapidly changing urban contexts. We're used to a planet where poor, suffering people were either in conflict or in rural areas.

And Darfur-- people going to Chad-- needing shelter, aid, all sorts of things. Yusef has to deal with those three and everything in between-- chronic food insecurity, Burkina Faso, you name it-- insufficient training for his staff. And he gets this fax.

Now I want you to think you're a decision maker. You're someone who has to make decisions and gets this fax. It was issued in May, referring to the three coming months-- June, July, August.

And it says if we define an extreme precipitation event-- extreme rainfall-- as being in the top 15% of the historical record, then this year-- there was a lot of text. I won't bother you with that. This year, because of sea surface temperature anomalies and other things that maybe you know if you're an MIT student, but it's not what we're used to-- this year, because of that stuff, the probability of extreme rains is enhanced from 15% to between 40% and 50%.

So this fax is in the office. And Yusef looks at it. And what does he do? What do you think he does? What do you think the humanitarian sector does when we get information?

First of all, to understand it fully you need to stop doing everything else you're doing. You have to not answer an email. You have to not sign the check for the fuel for the driver for delivering the something or other.

And once you start thinking about it, what do I do with the place with the green pixel? Do I send-- I don't know, one tent? A thousand tents?

Where do I send them, and to do what? What are the chances? When will it happen? So my job is to help the Red Cross family and the humanitarian sector at large to understand science that can help us make smarter decisions.

Now if you want to represent, to model, with an experience of forecast, this is how it normally works. There's information that allegedly can support action. That is communicated with an audience in a passive mode, like a fax or an email or a TV or radio announcement. And then you hope that people will get it and do the right thing. Of course, I leave a question mark, because-- not always?

Now when I talked to people about climate science-- about changing climate conditions; sea levels rise, so we need new locations of shelters; about El Nino and changing risks of drought and food insecurity; about rainfall upstream, therefore flooding downstream-- I would be talking science. And the people I was talking to-- they had something else to do.

They had to go save lives. And they were forced into a room just like you now. And I was talking science. And I could feel the extent to which I was successfully shrinking their brains.

This is a metaphor. There are very talented and smart people in the humanitarian sector. But my putting them into a passive mode, talking jargon to them, showing maps and graphs and so on, was really not accomplishing much. So I had to try something you.

Oh, by the way, I remember hearing the snoring while I was talking science. So it was shocking. Because they were wasting their time listening to me unsuccessfully try to communicate information that can help save lives.

So I started doing other things. And we're going to get a five minute flavor of the kind of things we're doing. So let's play.

You're a team. Split in half. Each one of you choose, are you on this half or this half? You're a team. You're a team. You're a team-- four teams.

There will be one winning team. The winning team is the team that has the most people standing by the end of the game. We'll have a few practice rounds, and then we'll play for real. OK?

You are not who you think you are. You're not some MIT student or something. You are magically, in the magic circle of this game, a Red Cross disaster manager. OK?

Good. Someone is celebrating. You're hired. We have no money, but you're hired.

So there are three kinds of things you can do. And everyone will be standing when the game begins. One is to go like this. This means I hope there's good rains. I'm going to take advantage of this opportunity to do good things when there's good rains.

Train people. Change the tires of the vehicle. Build a shelter. Write a proposal for donors. Write a report to the donor that gave me money last year. All those things, you can do when it's a nice day.

Now maybe there's too much rain coming. Then you can invest in preparedness against too much rain. You make this gesture like an umbrella. And it means you're preparing against the risk of too much rain.

This could be prepositioning relief items, like tents, for those who have to be displaced with the flood. But it's better to do it before the flood. Because if you wait for the flood, then the bridge has been washed away. And you have no way to take tents to the other side.

The other option you have is to do a bucket-- something to hold water from the sky. If there's a risk of a drought, you want to collect water before the drought so people can have water to drink.

So each one of you can do three things. Good rains, you go like this. Too much rain, you go like that. Too little rain, you go like this.

What determines rains? The probability distribution function of precipitation based on the past record. So you guys are affected by this PDF, this die. You guys are affected by this one.

A six is what? Too much rain-- it's a good idea to be like this. If you're not like this, you sit down. You got it wrong.

A one is what? Too little rain-- it's a good idea to be like that. Otherwise, you got it wrong. You sit down.

Everything in between is what? Good rains. You better go like this, or you got it wrong.

Practice round-- everyone stand up. Because the winner is going to be-- well, let me show this. Losing players [? all ?] get it wrong. You sit down. You're basically-- not fired, but people suffer because of your bad decisions.

And the winning team is the team that, at the end of the few rounds-- I will arbitrarily declare the end, because in the real world, there is no end. We'll have three practice rounds, and then for real. Most people standing-- so if you guys have two people standing and you have 10 people standing and they have less than 10, you are the winners.

And the winning team will win a publication. I will explain how you'll have to fight over it. So let's go back to-- actually, here. So that you remember the three things you can do.

Each one of you has to make a decision. Two, three, four, or five, it's a good idea to do this.

Six-- good idea to do that. One-- good idea to do that.

You can have a collective conversation. But each one of you has to be making one of those three. If you're making none of those three, you sit down. You're fired.

You have 30 seconds to consult with your teams.

[SIDE CONVERSATIONS]

PABLO SUAREZ: The rains are coming. Make your decisions. 10, nine, eight, seven, six, five, four, three. When I say stop, you have to be in the position. Three, two, one, and-- be in the position or you're fired-- stop!

OK. Here we see a lot of thumbs up, two buckets, two umbrellas. Roughly the same kind of configuration-- all right. Let's see what happens to this half.

Camera, I wish you good luck. This is for you guys. It is a five. So if you're like that, stay standing. Otherwise, sit down. You didn't invest in the good things you could have done.

For you guys, let's see what happens. And it is a four. If you're like that, stay standing. Otherwise, sit down. Moving on to your two-- practice round. What are you going to do?

Those who are sitting, stay sitting, but can give advice. OK? Make your decisions. Five, four, three, two, one, and, stop. Lots of thumbs up.

Are you with these guys? All right. So you're the only umbrella on this side. Let's see how it happens. There we go. There's

A four. Thumbs up, stay standing. Otherwise, sit down. This team not surprisingly is weak. They started with few.

Over here, you guys-- there we go. A two. So if you're like that or like that, sit down. Now, the last practice round. You receive this fax.

That's all you know. You got these fax.

AUDIENCE: Are we using the same dice?

PABLO SUAREZ: I am answering your question by saying, all you got is this. There's a deliberate scarcity of

information. You have 30 seconds to make your decisions.

AUDIENCE: Wait, where are we?

PABLO SUAREZ: You are in the place with the green pixel.

[SIDE CONVERSATIONS]

PABLO SUAREZ: 10 seconds left. Five, four, three, two, one, and, stop! OK. So in this team, there's only three left. One third are investing in too much rain.

Here we have about eight or so. Half or more are investing in this. No one-- oh, only one person is going for drought.

Over here-- all right. I see over here we have three. So about half, also, investing in flood protection. Over here we have more than before, two out of six.

Well, of course, what this says is that on a normal year the chances of this is 15%, or about one in six. One in six is 16%. But this year, because of unusual conditions, it's a flip of the coin.

So for you guys, I'm going to flip this one. Of course, you cannot now change your decision, right? It's too late now. You either sent the tents or bought the cement for the shelter.

So I'm going to throw this up in the air. If it falls like this, no problem. It's like a three. If it falls like this, too much rain-- 50% chance or so.

It's a good idea to have the umbrella. Otherwise you sit down. If it falls like this, meaning it stops moving and it stays standing, it's a good idea to have a bucket. You guys will get the yellow one.

Ready, set, and go.

AUDIENCE: Yes!

PABLO SUAREZ: Flood. Someone's very happy. We love it. So if you're not like this, you have to sit down.

Now notice this interesting feature of reality. If you take the right action, an extreme event is not a disaster. He was prepared. He's happy that a disaster happened. Because he was prepared.

Which is a good thing. A disaster is only a disaster if it leads to negative outcomes. If someone tells you it's going to be 110 degrees today, and you have some health condition, you turn on the air conditioning. There's no disaster. Unless there's no energy for your air conditioner.

OK. Let's see about you guys. You also got a flood. So if you're like this, stay standing. Otherwise, sit down. If this were the end of the real game, we'd have three standing here. No other team has three. So you would be declared the winning team.

Oh, you got four! So my apologies. You would be declared the winning team. Congratulations. Now.

[APPLAUSE]

PABLO SUAREZ: Thank you. Everyone sit down. And now we're going to see what you would do for real in this scenario where we have climate change. Have you heard of climate change? Things are changing.

This is based on the past record of precipitation. But the past no longer explains the present. Things are changing.

So I hereby introduce you to climate change. This one is going to be for you guys. I'm going to throw this up in the air with some spin, just like the dice.

Look at you. I love this face. This is a very productive silence.

I'm going to throw it up in the air. if it falls like this, it means normal rains. No problem. It's like a three-- a good idea to go like that.

If it falls like this, too much rain. It's a good idea to invest in your umbrella.

AUDIENCE: Oh I get it.

PABLO SUAREZ: If it falls like that, what is it representing? It's too little rain-- bucket-- good idea to go like this. This is for you guys. We're going to imagine we play three consecutive years. Can you hold this for me? No spinning, please. No throwing.

For the other half, we have this other one, which is also climate change. But the climate doesn't change in the same way in every place. So I'm going to throw this up in the air.

If it falls like this, it means no problem. Good idea to go for thumbs up. If it falls like this, too

much rain. If it falls like this, too little rain. Go for that.

As a team, you have two minutes to come up with a decision-- or more likely, one minute. Go.

[SIDE CONVERSATIONS]

PABLO SUAREZ: Less than a minute.

30 seconds.

10 seconds. Everybody stand up. Make your decisions. Five, four, three, two, one, and, stop!

All right. Compared to the first round with the die, there's much more proportion of people investing either in too much rain or in too little rain. Can you notice any difference between this half and that half?

Not really. Very substantial, phenomenally different climate change projections coming from science-- this could be like the IPCC report, right? And yet, you don't know how to interpret those differences. Thank you for incarnating what happens in the real world. OK? We just are very confused to understand that the risks are higher.

So I will not roll it. Because I would rather keep sharing with you some of the things we were doing. But I can share with you that you already have-- no, I want to know! Let's play. I want to see what the climate will bring.

And that's exactly the point. We are creating your appetite for learning. And that's what games can help me and my colleagues do. We have a bunch of [? reckless ?] people, and suddenly they want to know the real climate projection for their project site.

All right. Everyone can sit down. So we went through this-- oops, sorry. This is where I wanted to go.

So that's the fax that Yusef received. You played, and you had many more people going like that-- which is the right thing to do. Now it's 50-50. It could have been that it doesn't flood.

But if you understand the risks, you can say to the face of your staff, to the face of your donor, or to the face of the communities you're helping, look. I had reasons to act that way. Because it went from one in six to one in two.

And games can help process that. What games can do is to help embed information into a

system where decisions have consequences, and shape of the information that becomes available for the next turn. So you can have more or less resources and so on. Yes.

AUDIENCE: Yeah. I was wondering. If you have a 50-50 chances, right, and let's say it turns out you made the right choice this year, why not invest more into better measurement [? increments? ?]

PABLO SUAREZ: Exactly! Why not invest more in better everything? We can. But we don't have the money.

Right? So we're talking to Lincoln Lab and donors and so on to help us improve our ability to sense what is going on in terms of hazards-- including too much rain, too little rain, and other things-- including vulnerabilities-- it's not the same for me to be exposed to 100 degrees of heat wave versus to an elderly person with asthma. Right?

And it's the same with capacities. What do we have, in terms of skills and knowledge, in terms of equipment, in terms of experience? The disease chikungunya-- has anyone heard of it? So what are you going to do if an outbreak of chikungunya comes out to MIT tomorrow?

Right? Knowing can help. Sensing can help you make better decisions. In short, in my experience, only games can do this. Only games can get a bunch of people together, and in a very short time understand the role of information in the context of decisions that will have consequences-- including lifesaving or life-losing consequences.

This is an example-- I'm going to skip it, so I can get to the end. But basically, games can also help you collect data and compare one scenario versus another, and see which are the things, the aspects of the real world, where people are failing to get something. And so you can target your efforts in capacity-building training, et cetera.

This is one of the things I will talk to you about. I will want you want to help us design games or playful activities that can help people make their decisions based on forecast, as opposed to always doing as if it were the normal die. And then if there's a flood, as the forecast said was more likely, you go, eh, I don't know. I wasn't prepared. Which is what happens quite frequently.

There are many advantages. One is that we get better performance in terms of outcomes-- fewer sick people. Better efficiency-- so you save more per unit money or equipment or et cetera. Flexibility-- people can do more if they understand the forecast.

If you know that Katrina is about to hit your village or your city, there's more you can do in

anticipation, as opposed to waiting. And once it slams, there's very few things you can do at that point. Because if you step out of your door, you're in trouble.

Games can create the appetite, through confusion. With the cone, you were like, huh? We deliberately create systems that push people to the tolerable edge of confusion. So that they want to figure out, to make sense of what's going on.

And then participants themselves-- individually or in conversation with peers, they go like, huh. So if we do this and this happens, I'll be better off than if I do this other thing.

The aha moment-- the epiphany, the revelation-- is constructed by the players. And that has much more lasting effect than if I give you the answer. No one remembers the answer if no one was asking the question.

Very few examples-- these cones are the result of a collaboration with the World Bank Chief Economist for Sustainable Development. He was trying to explain a confusing concept called deep uncertainty. [? Jana ?] over there-- can you wave, [? Jana? ?] She's my partner in game design, and facilitation, and life.

She had the brilliant idea of taking-- our cone had surgery. Sorry. Our cat had surgery. And she thought, how about using the cone of shame-- you know, for the creature not to lick its surgery-- she said, how about using that for depicting this uncertainty? Now we had students tossing each one of those cones 100 times to see the probability solution, with and without spin, and augmenting the gap between what you think will happen and what actually happens.

A few other cases-- this is from Hanoi University of Science and Technology. These was with 350 students playing a game on dengue. It's like a rock paper scissors, where three players are mosquitoes. And they can either bite, or with your blood, lay eggs.

And the other players are humans. And they can either protect from biting or attack the breeding grounds. This was designed by students at game design students at [? Parsons ?] and professors collaborating with the other students.

This one is a publication from NASA. It doesn't get more NASA than a rocket launch. But if you look here, it says, Erin Coughlan-- former intern and now staff in my team. And the Red Cross/Red Crescent Climate Centre project. You go to page 47, and you see [? Munu. ?] [? Munu ?] is a volunteer of the Zambia Red Cross, from the village of Kazungula in Western

Zambia on the Zambezi River.

I mean, there's not much electricity or NASA-ness in Kazungula. But he was trained as a facilitator for a game where, with cards and dice, there's rain that happens over there. So her cup gets some water.

And then from her cup, she passes the water to the person next to her, representing the upstream going downstream. She can get more water or not. And it keeps going until maybe he gets water to overflow his cup. He's in trouble, unless he takes some action ahead of time.

And the game we're designing is not only with cups and to mimic the upstream/downstream. But we're also making a digital game where people actually monitor river levels. And they make predictions. They make bets as to what the river level will be in two days or so.

And then the winner wins points. But they can use points to buy information from upstream. And then with that data we're developing a hydrological model that can help us better know what is going down the stream, and are we going to have more flood trouble?

As you can imagine, it's awesome to have NASA report on some little project in Western Zambia, in part because we know that we can partner these on-the-field game-based data collection with satellite image for ground truthing and so on.

This one-- because we are being recorded, I will not disclose the full story. But in short, we play the game that had the same story. You play with a 6-sided die. You know your probabilities.

It was being played with the top-level climate scientists in this planet. And as you can see on the left, there's all sorts of calculations with probability distributions and the cost benefits and so on.

Then we change the probabilities on them. And then we gave them that. Just like you were saying, what is that? What are you going to use? We just gave them that graph and the caption.

And some of them were saying, but wait a second. Why are you giving me this information? It's completely useless in the context of my decision situation.

And then we told them, this is the information that you yourselves produced in the summary

for policymakers of climate science stuff. And they became fully aware of the need for them to improve the way they communicate climate science. This-- you may recognize that dude. I think it's the last time I wore a necktie-- at the White House.

I was given five minutes. I begged for seven. Seven minutes given, preceded by extensive PowerPoint presentations or other forms of uni-directional, so I had an audience that had brain wave activity declining.

And I said hello. My name is Pablo. And this is not a Frisbee. It's a hurricane. Everybody stand up.

And people were somewhat confused. Some were standing. Others were distracted, and they saw people standing, so they stood.

And I said, OK. I'm going to throw this frisbee in some direction. It's a hurricane. If it hits you in the chest, it's trouble. You're suffering.

Now if you sit down, it will fly above your head, no problem. It's like evacuation. But if you sit down and evacuate, and the hurricane goes in some other direction, someone may still your refrigerator. Right? You're doing the wrong thing.

So where it will go will be determined by these two dice. So you either say standing or sit down to avoid it. And I throw the things. And I look. And I throw in this direction and in this direction.

People were being hit. There was huge confusion. No one understood what was going on.

And then someone got it. Double one, I throw over here. Double six, I throw over here. Interpolate. So someone figured it out. They said, oh, a nine! It will go in that direction.

Everybody sat down. And I threw the frisbee. And it flew elegantly until it hit the wall of the White House.

And then I said, well, you know, climate science can give us this. Think Katrina. Think all that we could do. We're here to help. Thank you.

And that game was memorable. It opened doors. And it created awareness of the value of linking climate science with decisions of the humanitarian sector, of the government sector, and beyond.

So game play beats PowerPoint in a million ways. For me, as someone who has to help others take actions that save lives, I'd say first of all, there's active learning.

When you were playing, and you were confronted with that fax, you were saying, what is that? What does it mean? And what should I do about it? It's about action.

And there's plenty of peer to peer-- no, but look! If it's 50%-- and people learn from each other. There are plenty of "aha" moments.

It's fun, but it's serious. And the fact that it's fun is important. Neuroscientists, including some in this building-- I think Satra works here-- have evidence that if you are emotionally engaged, your learning goes deeper and lasts longer.

There is also the ability to collect data. I didn't go into it. But that's one of the things I'd be keen on using with you, if you are able to make a digital game that can collect data and help us understand, what do people know? What do people think? What do people want?

And then we can use it as an optimization platform. How much should we do? How many people should stand up next time we have that forecast? It's like a Monte Carlo simulation, right? You do things many times. And you figure out, what are the things that you call optimal, based on objective or subjective metrics?

That's the website for my team-- climatecentre.org. Note the British spelling.

This is at the core-- why we love games. There's an experiential learning. That's from our conversation with your colleagues at the Lincoln Lab.

You want to do things, reflect what would happen, form abstract concepts-- maybe I should do this or that-- test it, and do it again in a better way. And it's much safer to do this in game play than to do it in the real world, where if you fail, you can fail soundly.

These are some of the partners we have worked with, including UN system, academic organizations, Rockefeller Foundation, and many other partners like Oxfam America and so on. It's been delightful.

We have severe shortage of human capacity to make this happen. We have Jana, who will likely come as a resource person to interact with you when I'm traveling, which will be a lot over the coming months. There may be other colleagues who live in New York, who may

come.

We also have Willow. Willow, would you like to wave your hands? Willow is very likely to become a consultant for my team going to Nairobi to help people use digital devices to map risks at the community level. Maybe there's a way to infuse it with games. She already developed games for Tanzanian communities.

So in short, I wish you all the best-- learning a lot from these guys, doing a lot based on what you learn, and your own motivations. And whether you like it or not, project four will involve doing something for us. Thank you in advance for accepting those rules. And there shall be more. Thank you very much.

Oh and by the way, I will leave this with you guys. It's our publication-- a shared endeavor of my Red Cross team and Boston University Pardee Center. It's called, "Games for a New Climate-- Experiencing the Complexity of Future Risks." I promise you it's fun to read. And it's available online for free as a pdf.

ANDREW GRANT: I'm going to talk about game engines. And before I do, if you want to think a little bit about one of the things that Pablo was saying. It's about, one of the things that a game does is it forces you to look at what happened, and sort of say, huh? You figure out why did that happen. And then you try to fix it.

And that's actually one of the things I think is very much like programming. If you're trying to write code, and something goes wrong, often that's what the process of debugging is like. You go, why did the screen flip upside down? That just makes no sense.

And then you have the aha moment. And you can go and fix it. Turns out that making software is a lot like playing a game.

So anyway, we're going to talk about game engine selection. Because one of your early tasks will be to sort of try out a game engine. We're not going to be using a game engine for the first game project. But we want you to be ready for it when it comes to project two.

Now the first thing to realize is that digital games are basically a software project. They share a lot of things in common with software in general. You're going to have a UI that your user has to interact with. There's going to be something going on in the background. For a commercial or business offer, it might be a database or something.

There's probably going to be either a design you're working from. Or you're making it up. Or maybe you've got a customer who cares about what's going on.

You've got features. You've got bugs. We've got task lists. All of the same sort of managementy things that happen, happen on software projects and game projects the same way.

There's a little bit of a difference, though. Games are a little bit harder in some ways, and easier in others. But we're going to talk about the ways that they're harder, because that's more interesting.

The thing that happens with most business software-- or even something like Photoshop. Have any of you used Photoshop? Excellent.

Photoshop is one of those software packages that-- a lot of people like to take classes in Photoshop before they get started. If you sat down to Photoshop, you probably didn't think to yourself, wow, this is easy. And if you did, wow.

But basically, Photoshop has hundreds, or perhaps even thousands, of features in there that you don't know about until you've taken the classes. And if you want to figure out something about Photoshop, your best bet, really, is to go to a search prompt. And go to your favorite search engine to find out what to do. Or ask someone who knows. Because actually trying to figure out Photoshop from first principles is really, really, really hard.

The thing is that with a game, you can't do that. You can't get away with that. If your game is complex, no one's going to take a training class to learn how to play your game.

Now it might be the case that someone will look at a walkthrough or a cheat sheet or something to play your game-- after they've played for three hours and decided they liked it. But they're not going to do that first.

That's a general, broad, sweeping generalization. There are some people that like nothing more than to read the manual for a game. They mostly died out about 10 years ago. But there are still some hanging on.

Generally speaking, nowadays, you're going to play the game, and then you're going to decide if you're going to read the instructions or not. And this is going to be true for your games as well.

So your UI task-- your user interaction piece for your game-- is going to be hard. And the only way to make sure that works properly is to test it over and over again, as early as possible. So that's one way in which games are going to be a little bit harder than normal software.

Also, games have to be fun. And we use the word fun kind of loosely. And we're always scared of using the word fun. Because there are serious games. There are games that are supposed to make you think instead of get a big smile on your face.

But what they have to be is engaging, or interesting. Photoshop's not a great example. If you're playing with Photoshop as an undergrad, you're probably thinking to yourself, this is kind of fun.

But if you're using a fancy piece of software for work, you're not doing that because it's fun. You're probably doing it because someone's paying you to do it. The games-- not so much.

People are going to play it because they want to play it, and for really no other reason. So your game has to be fun. And what you think is fun they may not think is fun. So the only way to make sure that your game is fun is to find some users really quick, and try out your game on them as soon as you possibly can.

The same thing happens with your game play difficulty. By the time you're done with your game, you're going to think the game was too easy. I guarantee it. And everyone who plays it for the first time will think it's too hard. I guarantee that, too.

So you have to test it with other users, not yourselves. Because you will think your game is too easy. And you'll be wrong.

And then, of course, finally, your game may have an impact you want to have. And I'm going to keep saying this over again. You're going to have to test your game. Remember, there's a theme that's going to keep going on here.

And we're going to use this theme with absolutely everything we do. When you're playing a game, you try something. It doesn't work. You think about it. You try something new.

When you're developing a game, you're going to try something. It's not going to work. You try something new.

When you're trying to make your team work smoothly together, you're going to try something.

And that's not going to work. And you'll try something new. We're going to talk about that circle over and over and over again in the class. Because it works.

So moving on, we're going to talk about game engines. And if I talk to quickly, as I have a habit of doing-- even more so when I'm talking to people-- let me know. Tell me to slow down.

So this is the game you want to make. It's beautiful. This is the tools you have to make it.

[LAUGHTER]

ANDREW GRANT: So obviously, what you're really going to do is going to be somewhere in between these two extremes. And you have to think really hard about how you're going to accomplish it. The vast majority of game developers, especially new ones, are going to come into it thinking, this is the awesome, perfect thing I want to create.

And then they're going to come to their tools. And they're going to discover that it's really hard to do. And so I encourage you-- rather than thinking of the big vision of what you want to make, and then figuring out how to get there, I recommend thinking about what you've got. And then figure out, what can you make?

Now these are a couple things you can do here. One, reduce in your mind the scope of the thing you're going to make. You're not going to make your favorite RPG. You're not going to make an MMO. If you try, we will laugh at you.

You will occasionally think about this top row. How do I get the awesome thing? But I really want to encourage you instead to think, as much as you possibly can, this is what I know how to do. These are the tools I have. What awesome thing can I do with these tools? Rather than, how do I get over there when I'm way over here right now?

This is a quote from a discussion I had with a game team once where we were talking about a particular technical problem. And I said, how hard is that? And the developer says, well, it's really hard because of this, that, and the other thing.

I said, wait. Stop. Actually I didn't mean that. I don't care how hard it is. What I want to know is, how long is it going to take?

And we're going to say that a lot in this class, too. We're going to say, how hard is something? And often what we really mean is, how long is it going to take? Because we don't care if it

takes an hour of really hard thinking, to some extent. What we care is that it takes an hour.

If it's going to take you an hour to do an easy thing-- like sort a bunch of yellow stickers and a bunch of orange stickers because that's part of your game prototype. If it takes an hour to do that, that's an hour. It doesn't matter that that's an easy task.

That's time. And you have to be careful how you spend your time in this class-- actually, through all your classes at MIT, I suspect. But in this class, we know that you don't have enough time to get done every thing that you want to get done. And so you have to think very carefully about how you spend your time.

It turns out that game development is already hard. You don't need to add hard stuff to it.

So a lot of what we're going to focus on with choosing a game engine and all the decisions you make is you want to write as little code as possible. And that's not because we're afraid of code, but because every line of code you write is time. Every 10 lines of code you write is a bug.

[LAUGHTER]

ANDREW GRANT: Right? So what we want to do is minimize the number of lines of code you write. We want to minimize the number of bugs you write. Because when it gets to be the bug stage, that takes even more time-- to find it, to fix it, to write new code to fix it. So we want to reduce as much as possible all of that time.

So we want to use all the tools we can to reduce the amount of code we're going to write. One of our big ones is we use what we call paper prototyping. We don't actually mean paper. We mean whatever materials you can use-- chalk, pieces of plastic, dice, frisbees.

Anything you can use to make your game without writing a line of code, you should try it. Test it that way. Because it's so much faster to test it that way.

When else do we have up here? Iterative design-- I've talked about this before. We're going to use the phrase iterative design to talk about that whole cycle where you try something, see how it worked or didn't work, change it, try it again. That's iteration. And we're going to talk about that a huge, huge amount.

And one of the big parts of this, of course, is game engines. Which is not to say that you're

going to build your game engine. But rather, you're going to try stuff. And the game engines will let you iterate faster, do more with less code, and get more done.

All right. So again, you're going to use a game engine to save yourself some time. You don't want to reinvent the wheel. I'm confident that you could write a game engine if you wanted to. But I don't think it's worth your time.

Someone else has already done it. They may or may not have done a better job than you can do. But you can bet that the thing that you're going to get from your game engine will be faster than if you stopped and wrote it from scratch.

And then finally, the game engine that you write from scratch-- for every 10 lines of code, you've got a bug. I'm making that number up. But it sounds good.

That means that a game engine you write has bugs. Now I'm not going to claim that the game engines that we're going to suggest you use don't have bugs. They totally have bugs.

But what we have is 10,000 monkeys on the internet, using that game engine, reporting the bugs. And there have been a lot of cycles of people fixing bugs. And so hopefully, the worst ones are out of your way.

The other thing you can do with a game engine is you can use it as you tinker toys, as we talked about before. Your game engine has a list of things it does well. Great.

Use those things as tools. Take advantage of that. Do everything you can with those tools.

And if it can't do something, don't fight it. Don't say, game engine, darn you, if only did this thing. Think, how can we change our design to not need that?

That's another advantage you can use, another trick you can play. If you limit your design, sometimes it increases your creativity. And it can actually help you get your game done faster.

So throughout this class, I use this phrase. I'm not sure anyone else does. But I consider certain things to be nooses. There are some things we will tell you you cannot do, or that we tell you not to do, flat out. And those things-- don't do them. OK?

But there are some things that we're going to say, we recommend that you don't do that. Or, you know, no team has tried that and succeeded yet. Those I consider nooses.

You can try them. And you can put your head in there and see how comfortable it is. But probably by the end of the class, you'll say, wow, you know what? When you said not to do multiplayer networked play, you were right. We have heard that every year. We will hear it this year, I suspect.

But again, we're not telling you to never try crazy things. But merely be aware of what the crazy things are. If we're suggesting that you not try something, but not telling you you can't do it, and we're looking kind of pained when we say it, be aware that this is a huge risk for your project. Be aware that this is a place where you should be putting a lot of early attention to make sure that you have a back-up plan if it fails.

All right. On we go. Now we're actually going to talk a little bit about the thing we're supposed to talk about. Picking a game engine is super important. Everything involved in your game relies on it.

But again, contradictorily, don't worry about it. Because you have four projects. And the first one doesn't count for game engines, because you're not using a game engine.

But for two and three, I'd say experiment. Try some stuff out. By the time you hit project four, you should have some opinions about what game engine works well for you.

And try to get there. Try to use the one you're comfortable with by then. But experiment on the first two.

No game engine's going to be perfect. They all have flaws. They all have advantages. And your job is to find the game engine whose advantages you like and whose disadvantages you can live with. And that will be different for different people.

All right. Here we go. So the most important things we're talking about for game engines-- this is not an issue for this class. We're only recommending free game engines. But in general, if you're trying to pick a game engine or a software of any kind to help you, these are the three categories of cost.

I call things free if you don't care about the cost. If it's $20, it's practically free. But if it's $1,000 and you're a college student, that's not free. That's painful. And if you're corporation, however, $1,000 is free. So these categories change.

For us, we're talking about free. Impossible is CryENGINE or something brutal where you get

the source license and do everything. But we're not going to do that.

So this is the first one that actually matters. It's does your game have requirements that the game engine has to meet? And for most part, in this class, this also won't be true. Because you can choose your requirements based on the game engine you're choosing.

And our games are pretty small scope. But basically, if you need a special input, if you want to publish to iOS, if you need to support network play-- which you don't, remember-- then that might affect which game engine you choose.

This one is actually my biggest important thing. Whatever you choose for your game engine-- and we're actually going to give you a short list. So it's not quite the open free-for-all you might be worried about.

It should be easy to learn. And there's a couple reasons for that. But the main reason is simply the amount of time you've got.

We don't have a lot of time. You don't want to spend time mastering a game engine. You want to spend time mastering the art of creating games.

So you want to pick something that's well-documented, that's got a good online community that's going to help you out. Even if you don't ask the questions, in a big enough online community, someone's already asked the question you have. I suspect you're all familiar with this point. We can do a search for the question you have on something. And some forum post pops up where someone else has already asked the question you have.

And you just read the answers and think, yay! There was an answer. Or probably, there wasn't an answer. I'll look at a couple more of those. And so you can get an idea for your game engine as to how robust the community is, how good the support is, by sort of looking and seeing how much support there is online for it.

One thing on in-house experts-- it's very common for a team to come together and say, well, we all know this game engine a little bit. But there's this one person on our team who's just an expert in this other one. So we're going to rely on that person to really teach us how to do it.

That only, only works if that person is really willing to be a teacher. And it's not a bad thing if they're not. But don't choose their engine based on their expertise if they're not willing to really be a teacher. And what I mean by that is that that person will have to be willing to not write

code a lot of the time. Because their time is better spent teaching everyone else how to write code.

And that really means that if you find yourself in that situation, where you think that you are the expert in a particular engine, and you're trying to get your team to use it, really think about yourself and how you're feeling about that. Make sure that you are willing to not dive in and write the code. Make sure that you're willing to help everyone else do it instead. Because that is going to leverage more of your team.

On a small enough team, that's not true. So on your two- or three-person team, you can probably get away with that expert. On an eight-person team where five of you are coding something, you really can't lean on that one person that heavily. You're going to need to spread out the knowledge as best you can.

And so some game engines are easier to learn than others. It's not just a matter of the online support. But we'll get to that in a little bit.

On a team programming project-- and this is a personal bias of mine. And I admit that some people disagree. But I personally think that a strongly typed programming language is a must.

If you're talking about five to eight programmers, there's an additional problem. Whenever I write a function, and you're going to call my function, you have to know what the arguments are to know what's going to happen. And a strongly typed language will enforce that contract. It will make sure that we're on the same page, at least a little bit, as to what that function is doing.

A lot of you are probably used to-- MIT teaches, in a lot of the programming classes, it uses Python, which is awesome for quick work. And it's awesome for a single person doing work. But it does kind of start to fall apart on a larger team, when you have to read someone else's code to know what their function is doing, what the arguments are supposed to be.

If instead, you have a strongly typed from the beginning, the compiler will enforce this communication. And we're going to talk a lot about teams and communication. It turns out your code is one of your methods of communication. The actual written text in your code is a way you talk to your team members. And a strongly typed language sort of forces you to keep that communication line clearer than a weakly typed language will.

And again, this is one of those things where programmers might have battles over whether or

not it's true. But I'm pretty confident of this one. I would say it's a noose to use a weakly typed language.

A lot of our game engines are going to be talking about C#, or ActionScript or whatever. And I'm going to tell you that if you know one of those languages-- if you already know Java, for example-- it's really easy to transition to C#. It's pretty easy to transition to ActionScript. ActionScript and Haxe are very similar.

So I would say any of those top four, you can transfer back and fourth pretty easily. JavaScript, kind of. But don't try to transition to C++ from those quickly.

I mean obviously, you should someday learn C++ if you are considering yourself a Course 6 major or whatever. But don't assume you could pick up C++ in a weekend because you already know Java. They're just gotchas in C++ that don't exist in those languages. And other important note, if you know Java--

[LAUGHTER]

ANDREW GRANT: It really looks like you know JavaScript. And you kinda do. But you don't.

And that's not to say that you shouldn't make that transition. But be aware that it's going to take a little bit longer than you think. There's a lot of synthetic sharing. But JavaScript is a weakly typed language that has a lot of weird, very web-specific features. And just be aware that that's not as easy a transition.

I talked about ease of understanding and learning. Actually using the engine is also very important. How are you going to get stuff done in it? If the engine has an interactive debugger, that is wonderful. If it doesn't, then you're relying mostly on print statements. And that's hard.

If you've never really had the joy of using a debugger before, I recommend you try to learn. Because it's really an extremely powerful tool. We're not going to get into that in detail right now.

And source-control-friendly is another important one. If you're on a team with eight people, you're all going to be making changes. We're going to talk more about source control in the future. We're not going to go into details on that now.

But basically, it's a way for your team the team to communicate and make sure you're always

in the same, up-to-date source code for your project. And if your engine doesn't support that, then you can be in trouble. In the past, for example, people have talked about some of the cool, sort of hobbyist game engines, like GameMaker, or back in the day, just bare Flash.

And those are really cool tools. But they would save the entire project as one monolithic binary file. And so if one person makes a change, no one else can change the code while they're doing that. Because there's no way to merge our changes.

Whereas with code or text files, or a good game engine that separates things out a little bit, one person can change an art asset while someone changes a code file, while someone else changes yet another code file. And it all merges together pretty seamlessly. And that's what you're looking for, if possible, on a team project.

I talked about this before, but you really want a product that's been around a little bit. It's cool as a hobbyist thing or a solo product to take on something really brand new. But I think for a class where you're trying to get stuff done really, really fast, and you're working with a team, you really don't want to go too far in the direction of the new, latest, and greatest thing.

Because you want to have a game engine that people have put some miles on. They've tested it and shaken out some of the bugs before you get to it.

All right. Those are things that I consider-- you've got to have them for a game engine. Here's some stuff that's kind of nice.

Yeah, it's nice to be able to easily import images and sounds and stuff like that. It's nice to be able to get the source code to the engine. But it's not essential if the engine's well documented. Let's see.

It's nice if you have an integrated development environment that can help you with auto-completion of various functions and really help you along with your development. Definitely nice, but not necessary. And then finally, an editor of some kind-- if I want to lay out my levels in some way, is there a graphical editor for me to do that? That's certainly nice.

And a profiler that would tell me where my game is slow-- that's also nice. But without these things, you could probably get by.

Bells and whistles-- this is actually relatively unimportant, especially for a small project, prototype-style stuff that we're going to be talking about. For the most part, you don't need all

the fancy stuff like particle systems and shaders and physics and all that stuff. You might use physics in this class. But be careful.

So that's one of the things. You don't need that stuff in your game engine for most small games, at least in the scope of this class.

These are things that I really don't care about-- scripting languages that your designer might use. Why is your designer not writing code in your strongly typed language? I never understood that one. Some designers are afraid of writing code. But hopefully in this class, you're at least willing to try writing some code.

And then finally, just because someone has made a beautiful game in a game engine, that has no bearing on whether or not your game will be beautiful in that game engine. The vast majority of the time, a game built in a game engine is beautiful because they have a really good art team.

And we don't have a really good art team here. We don't have the time to do that. And we don't have the resources to do it. So you shouldn't measure your game up against the beautiful AAA games. You should be measuring your games up against some indie, quick, prototype type of games.

All right, so Flixel is the thing we've been recommending for many, many years. This is the game engine that is easiest, has the quickest learning curve I've seen. It gets everything done. It's very quick to learn, to use. And it runs on the web. Great. That's pretty good easy stuff.

It was developed by Adam Atomic for a game called *Canabalt,* that you can see up there. The game that, as far as I know, sort of popularized the infinite runner genre, which we've seen clone after clone after clone of. But I believe this is at least where it entered my consciousness. And therefore, it must be where it came into the world, right?

But anyway. The cool thing about it is it's got a very, very, very simple object-oriented state-based system. It's really, really good for 2D sprite stuff.

Last year, we had a bunch of game engines for people to try. By the end, pretty much everyone just used Flixel. Because it was easier to use.

And this kind of says it all to me. Even if you haven't used ActionScript, the learning curve on Flixel is awesome. You'll spend a day or two staring at it. And then you'll know most of

everything there is to know.

And you'll just be able to do stuff. You won't be able to do all the cool, awesome stuff other game engines will let you do. But you'll be able to do pretty much everything you'll need to do on the scale and scope of games you're looking at in this class.

Let's see. It doesn't so good at GUI. If you wanted a fancy GUI-- lists and lots of text and buttons and sliders and stuff-- it's kind of a pain to code that up.

Right. So Flixel is also unfortunately, in its current form, kind of in an end of life cycle. And I've been saying that for two or three years now. And it's still holding on. Because Flash is still holding on.

Adobe is trying to sort of get out of the business of maintaining Flash as a web development place. And I'm not really sure why. Apparently, Adobe decided a while back that HTML5 was the future. But it's been four or five years now, and HTML5 is still not the future.

In the meantime, we're sort of hanging on with Flash and the occasional JavaScript HTML5 thing. I suspect it was actually more of a financial thing than any actual HTML5 preference thing. But anyway, we're trying to get out of Flixel a little bit in the long term, because Flash is going to be a dying platform pretty soon.

PHILLIP TAN: Can I just add something to that?

ANDREW GRANT: Please. Stand near me. Because I have a microphone.

PHILLIP TAN: All right. I'll stand right next to him. So the mic can pick up. One thing to keep in mind-- that are free versions of the Adobe compiler for Flash, called Flex, that you can download for Linux, Mac, and PC.

However, there is also this licensed version, which you can download for a 30-day trial, called Flash Builder. Unfortunately for Mac users in particular, if you want to get those, I use it. I like writing code in it. But as soon as that 30-day free trial is over, you are on an annual subscription paying Adobe money every year. So "free" with an asterisk.

If you want to do free development on a Mac and use Flixel, be very, very careful. And use this upcoming exercise to actually ensure that the things that you're downloading and the things that you're signing up for are actually free, and not free for the first 30 days.

For PC users, you have the advantage of using FlashDevelop, I believe. That's what it's called. That is actually a tool that you get without having to pay for it.

For Linux, you are pretty much stuck to using command-line compilers. And if you really want to do free development on a Mac, you're also using command-line But there's all kinds of clever things that you can do with text editors to sort of make that a little bit more comfortable.

But just keep that in mind. That's another reason why Flixel isn't our number one choice, nowadays, is because Adobe's really trying to squeeze money out of everybody, including students. Adobe's insane.

ANDREW GRANT: One can't blame them for trying to make a little bit of money. However, it's no longer as attractive as it used to be. That's certainly true.

Unity is the next one on our list that we've had good success with. It's a free, full-featured game engine. It's got all the bells and whistles you could want.

It's a little bit dangerous to use it, because it has all the bells and whistles you could want. And you should probably only use some bells, and not the whistles, and whatever.

You should be very careful and judicious in your use of the features. Because the tinker toy model of, only do what the engine lets you do is a little bit dangerous. Because it lets you do pretty much everything. So you have to be very careful about which features you choose to use.

However, it's great. It's free. It does a whole bunch of neat stuff. There are paid versions. But you won't need them for the class, certainly.

I call these disads. 3D is harder than 2D. And 3D is at least three times as hard as 2D, really. It's rough. If you've never done 3D before, it takes a while to get your head into it. Even if you're doing a 2D game in Unity, sometimes you have to think in 3D a little bit, which makes it a little bit harder to use.

On the other hand, Unity does have this nice editor. And it's possible for a non-programmer to get some stuff done in Unity in a way that's not possible in others. You spend more time learning the GUI. But you can actually get some more things done. So it's an interesting little trade-off there.

It is also, again, not good at GUI work. They're starting to address that a little bit. But it's hard. You have to kind of hand code your buttons a little bit sometimes. That is improving, as I say. But it's not quite there yet.

It suffers a little bit from the problem of merging. Unity uses particular objects that you really wish you could merge-- just so they're text files or code-- but you have a whole scene of objects, for example. And only one person can really work with it at a time, unless you're really very careful. So when you're using Unity, you have to be extra careful with your team to make sure that people aren't messing with the same files at the same time.

The code, however, works the way you want it to. And source control works fine. So you've got to be careful. But you can work around it.

Moving on to the ones that-- you know, we want some of you to practice with these. But-- oops, that's the wrong one. I left a slide in I shouldn't have. I will ignore that.

Let me see if I can find it here. All right. Apparently I did not include the slides I thought I included about the two other game engines. So I'll talk about those instead.

So there's another engine call Haxe Flixel, which is sort of a port of Flixel to a language called Haxe. This is an engine that we have not used yet. We've looked over it a little bit. And it certainly looks very promising.

It's using the very same Flixel style. It is, however, using an interesting language named Haxe, which is mostly based on ActionScript. But it has some Java and C#-like features to it. It does some crazy stuff, though. So it might be a little bit interesting to use.

Haxe compiles down to a language of your choice, which it then compiles. So for example, if you want to export your Haxe to Flash, you'll compile your Haxe to ActionScript, which will then get compiled to a Flash app. Or if you want to make an iOS app, you'll probably compile it to the [? XCode ?] Objective-C, and then compile that.

But basically, it tries to be all things to all people. Which is really kind of dangerous. But hopefully, for the kind of smallish project we're talking about, that'll be find. I call that one a noose, because we haven't used it much. But it has all the simplicity of Flixel for the Haxe Flixel game engine. So that is a good thing.

The other one we're looking at using this year is called Phaser, which we know not too much

about in this lab. The Education Arcade is using it. And they seemed to like it.

And it is an ActionScript, HTML5 style of engine. That is not the most mature platform to build on, because of all the browser differences and stuff like that. But it is certainly possible. I recommend if you end up using Phaser that rather than coding in pure JavaScript, you code in TrueScript, which is basically a strongly typed version of JavaScript.

We'll get on to the rest of it.

PHILLIP TAN: TrueScript? TypeScript?

ANDREW GRANT: Sorry TypeScript. Yes, TypeScript.

So when you're learning an engine, you start with tutorials. That's the easiest thing to do. Once you've learned three or four engines, you might not use tutorials. But until then, I would say you should actually go through something that someone's written and thought about how to progress.

Do something extremely small. Don't start with the be-all and end-all game of awesomeness. Start with something very, very easy. And then after you've done that, you're going to discover that you've done some things. Then read the documentation a little bit more. Because you're going to be better at picking up stuff after you have questions in your head. Even if you don't realize what questions you've got.

And we're warming up to an assignment where we're going to hand you a game engine and you're going to take a look at it over the weekend, or over the week. But the thing you need to remember is that all software really does suck. And game engines are no exception.

At this point, invariably, somebody will claim that Linux doesn't suck. To which I'll have to claim that it does. But it's also awesome. And that's the thing about software and game engines. You have to really understand and appreciate the ins and outs of Linux to understand the ways in which it really does suck, but there are ways in which it's really also really awesome.

Operating systems all, really, are pretty bad, I suspect. Windows and Mac-- you can make huge arguments about all of them. Photoshop we talked about earlier. It's an awesomely powerful tool. But to sit down and actually start using it as a beginner, as a newbie, is a miserable experience.

I think you've probably all had very similar experiences with software. It's buggy. It's painful. This is true for game engines, too.

You're going to try out a game engine. And you're going to probably come back to class thinking, I tried Unity. It was terrible. I didn't understand what was going on. It was really hard. And then you'll hear someone else talk about Flixel. And you'll think, wow, I didn't have any of those problems.

And so this is the trick. All software sucks. And your job is to find the software that sucks in ways that you can handle. Right? You don't mind so much that you have to think in 3D for Unity, because you're willing to put up with that because you get these cool, nice, rendered look.

Or maybe you have to deal with the fact that, yeah, Flixel's written in ActionScript 3. And ugh, I hate that language. But boy, it sure is easy to put stuff up on the screen, and have spaceships moving around shooting each other in, like, a day, with no problem.

And these are the kinds of trade-offs you're going to have to look at. So when you're trying to evaluate a game engine, or any piece of software, really, but game engines for this class, you're going to think about the ways in which it's terrible. But also think about the ways in which it's great.

And when you get together and talk with a team for projects two through four about what game engine to use, you're going to want to be able to say intelligently, I don't like Flixel because of these things. But actually I can live with those things. Or, I don't like Unity because of this thing, and I really just can't get past it.

And if your team's insisting on using that engine, and you really know you really don't like it, consider another team. And that's not bad. Different people are going to have different preferences for different styles.

And that's part of what we're going to be doing. Any time you're going to be forming a team or trying to work with a team, you're going to have to figure out what works for you, what's hard for you, what's easy for you. And match yourself up with tasks or software that matches your set of skills.

Or, if you want to challenge yourself, OK. Be ready to challenge yourself and try something that you thought was too hard or too painful. But be aware that you're doing that. Because you

need to let your team know, hey, I'm going to be on a learning curve here. You're going to have to help me out.

So as you look at a game engine, be aware that it is going to disappoint you. That's just the way it is. But try to figure out where the disappointments are.

To that end, one of our assignments-- yes.

AUDIENCE: Which 2D game engine would you suggest for those who are comfortable with Java and C#?

ANDREW GRANT: With Java and C#? It kind of depends on the situation. I would actually still suggest, for most simple-- which game engine should I use for 2D? And the answer is, if you have no 3D experience whatsoever, I would still recommend Flixel today. If you are comfortable with 3D and some of the barriers there, I would go with Unity. I personally use Unity for my own development.

That said, what we're going to do is everybody's going to take a look at a game engine. So you'll have at least one of the four that we're talking about for the class that you're familiar with. And hopefully in the next couple of weeks, you'll get your own preference.

Because I can easily say, sure. Flixel's great for 2D games. And it is, for the right kind of 2D games. It's not good for all the 2D games. An isometric thing with lots of farms and stuff-- you might be able to do it. But you might be more comfortable using a 3D approach or whatever.

So in other words, it really depends on the game, too. I wish I could give you an easy answer. But I can't, which is why the whole lecture.

So we have an assignment, which we'll hand out to you now. And we're going to talk about it one week from now. And do stay tuned. Because we're not done talking at you, or giving assignments.

PHILLIP TAN: Do we hand these out, too?

ANDREW GRANT: Oh, the game engine tutorial? Yes. What I want you to do is pick a game engine out of this hat. And pass it on.

If you do not like the game engine that you get, trade. Um, sure. Feel free. Take a handful.

If I run out, I have more tokens. But this is what I start with, just to make sure there's a good

distribution. Just go ahead and hand this back. I don't need to be with it.

So one of the things we're going to do next Wednesday is try to do a source control exercise, which will introduce you to source control, and hopefully refresh some of you. So ideally, if you can, we want you to bring in a development machine, a laptop if that's what you're using, on that Wednesday. If you only have a desktop, don't. If you've got no computer, don't.

But you know, try to bring in a development machine of some kind that you can actually use on Wednesday. Ideally, it's the machine you used to do the tutorials, so you have your code on it.

Let's see. What else do we want to say there? We don't want you to spend more than four hours on this exercise. There is no "turn in" for this exercise.

What we want is, on Wednesday, we're going to be able to talk about the game engine that you looked at. We want to be able to talk about its pluses and its minuses, and what was frustrating. If it took you all four hours just to install the stupid thing in the first place, that's information. We want to know that. Because that will help people choose to not use that game engine, for example.

But basically, we want to be able to talk intelligently in the groups, amongst yourselves, about the game engine pros and cons. That's the main goal. OK. Any questions or comments so far?

AUDIENCE: [INAUDIBLE] can't just pick just any one of those [INAUDIBLE].

ANDREW GRANT: Pardon?

AUDIENCE: So we can only trade. But once you have the [? others ?] you can't [INAUDIBLE]

ANDREW GRANT: Right. Correct. Do not pick a game engine at random. You may trade the one you've got in your hand with someone else who has a different one in their hand.

This is not just-- it's partially arbitrary. But mostly it's to make sure that all the engines are covered about equally. Also, there's two entries for Unity. One is sort of thinking about it in 3D, and one is thinking about it 2D.

And because Unity has recently added sort of a 2D layer on top of it that you can sort of use. It's a sprite-based engine. So we want you to put that through its paces, too, to compare to the other 2D engines, to Java.

[INTERPOSING VOICES]

ANDREW GRANT: So I've never had much of a problem transitioning between the two.

[SIDE CONVERSATIONS]

ANDREW GRANT: Let's see. Yeah, I would trade if you could.

RIK EBERHARDT: So, like we said, this is a video game development or video game production class. It's about team management, project management. We're spending a lot of time on those kind of topics.

We aren't spending a ton of time talking about design in class. But we were jerks. And we're giving you a really, really hard design constraint to work with for product four.

So we're going to have some optional assignments, optional reading-- basically things that we're going to list on Stellar, and a couple lectures we're going to give-- basically just kind of opening your minds towards design. So you can think through some of the big, hairy problems when it comes to designing some of these things.

The first thing is talking about the theme for the course for the semester. So we're talking about meaningful decisions in games. So we're asking you, when you're creating these four projects, to think about what decisions your players are making during the game at all times, and to make sure that all those decisions they make are meaningful. And this is going to be the foundation for the four projects that you're working on. All the constraints are going to be based on this kind of thing.

So first off, what do we mean by meaningful decisions? This is what I mean when I say it. There's other definitions out there. But this is the bare minimum. It should do this. It should have this.

So three things-- any decision a player makes in a game has an effect on the game's overall state. So I make a decision. Something changed.

Not only did something change, but that change ripples through the systems within the game. It affects the entire game state, the entire game world-- not just what the player might have, not just the player's performance, not just the player's abilities.

So a jump in these games-- it can be meaningful in some games. But for this particular game and this particular use, if the jump affects everything else in the game in some form, that was

really meaningful. And I'd be surprised to see a jump do that.

But it probably isn't. It's probably just a way to get over an obstacle. So there's something else. There's some other kind of decision a player has to make, rather than just that one small decision.

Third-- there should be sufficient feedback-- and we'll talk about feedback later on and what exactly we mean by that-- that lets the player be aware that they made that decision. Not only did I make a decision, but I knew that I made a decision. There was something that happened in the game that prompted me to make that decision.

And I actively made the decision. I made the choice to make the decision. I know I made the decision. And I actually see all those effects happen in some form.

At the bare minimum, I make a decision. I see some short-term effects. There might be long-term effects that I don't see right now, but I will have the opportunity to see later. That's a little bit more advanced.

So basically this lecture is really just going to be talking about some example games that, when we're talking about the games we would like you to make in class, these are kind of the exemplar games for this. These are the games that we think of when we think of this kind of thing.

And the problem that I think is going to come up is we're going to sometimes call them strategy games, which is a really loaded term. What is a strategy game? Does anybody have a definition for what a strategy game is?

Somebody who took-- yeah.

AUDIENCE: [INAUDIBLE]

RIK EBERHARDT: Yeah, a game that's going to require a lot of thinking. A game that's going to require-- and by thinking, there's a lot of actions that happen when we talk about player thinking. Like all right, so I'm thinking. So if I'm thinking, I might not be taking action. So I might be thinking to take an action. I might be planning. That's going to be part of it.

I might be thinking. But there's something that's distracting me for thinking. Maybe I have to make actions really, really fast. Or I can make actions and I can think a long period of time. All

that is the big wide spectrum of what we're talking about with strategy games.

Can anybody list some strategy games they've played or have heard about?

AUDIENCE: *Age of Empires.*

RIK EBERHARDT: *Age of Empires*-- which one?

AUDIENCE: The second one.

RIK EBERHARDT: Yes it is.

[LAUGHTER]

Civilization. Yes. Which one?

AUDIENCE: *V*.

RIK EBERHARDT: OK. I like that one, too.

[LAUGHTER]

More. What's that? No?

AUDIENCE: *StarCraft.*

RIK EBERHARDT: *StarCraft*. Which one?

AUDIENCE: I've only played *II*.

RIK EBERHARDT: I played *I*. But I haven't played *II*. But he plays *II*. A lot. More.

AUDIENCE: *Fire Emblem*.

RIK EBERHARDT: *Fire Emblem*-- ooh, that's a really good one. Which one? All of them, yeah. Yeah.

AUDIENCE: Go.

RIK EBERHARDT: Go. Ooh. You took the board game class.

AUDIENCE: I play MOBAs.

RIK EBERHARDT: MOBAs-- so which ones? What's a MOBA, really?

AUDIENCE: Like, [? *Noldor* ?] or *League of Legends.*

RIK EBERHARDT: So what does MOBA stand for, for people who don't play it?

AUDIENCE: Massive Online Battle Arena.

RIK EBERHARDT: Massive Online Battle Arena. And you were talking about *League of Legends.* How many people here-- raise your hand if you played *League of Legends*. OK. Riot's coming September 24 to talk about to us about that.

How many people have played *DotA, Defense of the Ancients*? The same people, for the most part. Cool. So that's a really good spread.

And actually, that's exactly what I expected-- except for the Go person-- to hear, are games about war, and games about conflict, and games about fighting against another player, or against a computer-controlled opponent. That's not exactly what we mean when we talk about strategy games for the broader aspect of the class. But it's a great place to start. And that's where we're going to start.

So here's *Civilization V,* and here's *StarCraft II*. I'm going to use these for a couple examples. And then we're going to split off. And each of us are going to talk about some of our favorite games that we think of when we say, this was a really cool mechanic. This is a really cool thing I've seen in a strategy game, or an aspect of strategy or an aspect of tactics that I've seen, and I'd love to see happen in some of the game we make over the next semester.

So there's two types of decision-making that I see. And again, this is all just me in my head a couple days ago putting a lecture together. It's not an academic taxonomy. It's to help us out when we're thinking about the kinds of decisions we're making in these games.

There's two kinds of decisions that happen in these games. There's a micro-level decision-- short-term decisions I make that help me out right now. There's macro decisions or long-term decisions.

And I'm using micro and macro because I'm obsessed with economics, even though I don't understand it. But also because you see the games represented that way sometimes. There's the micro, where I'm looking really tight at this one aspect of the game, and the macro, where I can see a bigger, expanded version of the world. I can see the systems and the gears that are happening behind the world. And they're represented in many ways. And I'll talk about a

couple examples there.

ANDREW GRANT: People often use "tactics" to describe micro, and "strategy" to describe macro. So there's actually a difference between strategy and tactics. That's one way of thinking about it.

RIK EBERHARDT: Yeah. So for the micro-- exactly, tactical. The way this is often represented in these kind of war games or strategy games-- there's usually some kind of spatial component going on. There's some kind of placement going on.

There might be some direct conflict happening. So I've got an army. And it's facing off another army. And we're throwing spears or shooting each other with lasers-- basically the same thing.

But there's also firefighting. There's another kind of conflict that happens. And by firefighting, I don't mean fires.

But I've got a problem that just happened. And I need to solve it. I need to react to it. So it's reactive.

I might have this really long long-term strategy. But unfortunately I've got to make some decisions right now that may or may not hurt my long-term strategy. There are some tactical decisions I need to make.

It's not always reactive, though. Sometimes what we're talking about is you're executing a plan. I've set up a long-term strategy. And now here's the final dramatic moment where I get to actually implement the thing and see if it works-- throw the dudes on the map, and they're going to die or not.

And when we're talking about tactical, there's always a short-term resolution. So I do a thing-- feedback. I know it happened. There could be long-term ramifications. But there's definitely a short-term resolution to the decision I just made at that point.

So my example, because I am a huge *Civilization* nerd-- *Civilization V* brought some very tactical game play to the series that wasn't really there in previous installments. So in the past, you could stack units on a tile. So *Civilization* is basically a game made of tiles where you're placing items on tiles. And the tiles give you bonuses or whatnot. That's the very, very basic version of it.

In this case, what we have-- and it's kind of blurry-- but you have a city in the center, where

that arrow is pointing to. You have these blue units surrounding the city, and a couple small yellow units going on.

Can anybody describe what's going on in this scene?

AUDIENCE: A raid.

RIK EBERHARDT: What's that?

AUDIENCE: A raid.

RIK EBERHARDT: A raid. So we're fighting. What kind of decisions is a player being asked to do right now? And the player is the blue player that has the red arrow there.

AUDIENCE: [INAUDIBLE]

RIK EBERHARDT: What's that?

AUDIENCE: [INAUDIBLE]

RIK EBERHARDT: So yeah. You're trying to implement the strategy. Can you kind of see-- apologies if you can't make out the quality of the slide. Can you see what is exactly going on? Yeah.

AUDIENCE: They're about to make a ranged attack with the cannon against the city.

RIK EBERHARDT: Yeah. So first thing is, all right, so I've got a range attack that is really diagonally away from the city. It's two hexes away. I can do a range attack on it.

The cool thing about this is that the city can't attack me right now. Well, actually it can. But I'm making a mistake there. But it's not the knight's turn. So the knight can't attack the ranged attack right now.

The other thing that's happening is the player had a long-term strategy, which meant that they brought some foot soldiers in to surround the city, protecting the ranged attacker. They've also got this thing called Zone of Control going on in this.

All right, here's an enemy person. Here's an enemy person. Here's an enemy person. If any of these people try to attack any of these blue people, there's some kind of negative modifier going on. Because the blue player has set it up such that they're being flanked.

So there's some spatial placement going on that's been decided. Not only did they have to get

all the units over here, but once they were here, they had to put those units in the right places for this tactical maneuver to happen.

So really, in *Civilization V*, we're talking about tactical in one aspect of it. In the combat aspect of it, we're talking about flanking, army positioning, Zone of Control, preventing other players from entering into tiles, controlling areas, and making this area basically more effective for the blue player to attack the city than for the yellow player to defend it.

So how did the player get there? How did blue get into that situation? They had an operational or strategic plan that they set forth. They were able to develop those weapons that they had. They were able to make them in the right location. And then they were able to transport them over to the right location.

So if we're talking about military strategy, tactics, that might be closer to logistics, supply chain, getting supply into the area. The strategy that might actually be happening is even bigger than that. So this is where it gets into the realm that I really nerd out in is spreadsheets of numbers and modifiers.

There's resources you need to manage. I need to anticipate what's going to be there and have the right things built in time for them to be useful in that situation. There's an economic system I need to manipulate and do well with in order to afford the units that I want over there.

If I'm playing well, I need to be proactive in my strategy. All strategy really should be proactive. And someone can argue with me if there is such a thing as reactive strategy.

But really, it could be proactive and wrong. But you did it beforehand. You set up the gears in motion for it to play out the way it did. And you're always planning for some long-term future.

So in *Civilization V*, one way that this plays out non-combat is city placement. Where I put my city is incredibly important. Using the real world map of Africa, what they've done is the designers put these resources-- those circular items-- spices, bananas, silk, diamonds, oil.

Some of the resources are luxury resources. So they help my cities be happy. They help my citizens work well and be productive. There's agricultural resources that help my cities grow and be healthy.

And then there's strategic resources-- and I could be mixing these terms up. They're things that are going to help me build things. So oil or steel or iron will help me build better units.

So my strategy here-- there's a little bit of tactics going on, where when I actually get my settler in the right place, I need to then find a very specific place to go. But long-term, if I'm thinking about the long term, I want to place my cities in a way where they're near resources. They're near my friendly cities. They have room to grow. Because if I group my cities too close to each other, they're not going to be as efficient.

This is really apparent if anybody's played *Civilization II*, where there's one real strategy to the game. And that's just build a city every three hexes. That's just how it works. And in the later games, they tried to introduce some nuance to that.

And again, I'm mixing strategy and tactics as we're talking about this. They all go hand in hand. A lot of the games that we've played, and that are these strategy games that you talked about, are actually more tactical games. There's much more emphasis placed on the short-term thing I do now than the long term.

That's not to say there isn't long term in these games. If anybody's played *League of Legends*, the metagame about the pick process, so what champ to pick or not. That's a strategic maneuver. You're planning for a long-term strategy that you're going to play out within the game.

And not only are you planning for it. You picked it, and then you're stuck with it. So those decisions last.

With *StarCraft II* it's build order, which can be a little bit more flexible. There are build orders that you can-- that's build order in that you are building units and buildings and upgrades in the right order, such that you're prepared for multiple different strategies coming into them. And I could be butchering that. I'm not a

PHILLIP TAN: That's definitely a good [INAUDIBLE]

RIK EBERHARDT: Cool. But that's more flexible. Because I actually have time within the game to make some tactical changes to that long-term strategy that I decided on early. Unless you're playing professionally. And then I've just seem that they just GG after five minutes.

So anybody recognize this quote? Sid Meier, the developer for *Civilization*, famously said, "a good game is a series of interesting choices." It's very much the case for the games that we're making in this class.

The thing I want to note is strive for interesting choices in your games. Be prepared for your choices not to be interesting. Try to iterate through them and make them more interesting. But we don't have a ton of time to really help you make the greatest games in the world. Right?

So we want your choices to be meaningful. And if one or two of them are interesting, awesome. You did a really, really good job with the game development process of it.

I'm going to do a really, really quick talk about how you can make things interesting. And we're going to come back to it in later projects. But I just want to kind of get these two terms out there.

We're going to talk about conflict and tension. So what is conflict? If I say the words conflict and tension, what's the definitions you're more familiar with? Anybody heard it in drama or writing?

Like the narrative arc-- there's a conflict. There are two people who want the same thing. Or there's an internal conflict.

And that's really what I'm thinking of when I'm saying conflict in this case. It's, what is the conflict in the player as they're making these decisions? What are the trade-offs they're making?

Do I send an army this way? Or do I send an army that way? Or do I not fight at all?

If that is a really hard decision for the player to make-- if they can't calculate it beforehand, if they have enough information to know that one may be better than the other, but they're not quite sure-- that's what we're talking about with conflict.

With what Pablo was playing with us, the conflict in the individual teams of, we know the probabilities. We can calculate that. But are we sure? Are we sure it's going to play out that way? I really wish he had used to cones. Because I really want to know how often the cone's going to fall on its end.

And then tension are ways we can modify that conflict. Maybe there's no conflict existing. And we need to use tension to create some conflict. Maybe there's a little bit of conflict going on, and tension's going to amplify that and make it more dramatic.

And there's just some basic things you can change. And when we talk about the example

games, and if you play these example games over the weekend, think about these terms. If I were to change the timing in a game, or the speed with which you can make decisions, or the precision that's required to make those decisions-- would that increase the conflict? And by increasing the conflict there, are the choices more interesting? Is the game more engaging, or the game more fun?

Limitations are awesome in games. Limitations are really how you can design and control where and what a player can do. So how many options do you give the player? Do you give them two, or three, or five, or infinite? I can tell you right now. Having infinite choices is really, really difficult, but not that interesting.

Space, costs, economics, information, fog of war-- we've seen in some of these strategy games where you can only see things that are around where your units. And you can't see things outside of that. These are all methods we use as designers to create conflict, to make those choices that we're asking our players to make interesting.

So example games-- what I'd love for you to do-- and this is totally optional. But if you're interested in design, I totally recommend it. Over the weekend, play some games. We're going to talk about these games. And think about these things. And the slides are on Stellar.

Think about the types of decisions you can make, the timing of the decisions, whether the game is single-player or multiplayer, if there's competitive or co-op play, if there's teams or alliances, and what kind of conflict and tension exists in the game. All of these are things that we're going to be asking you as designers over the semester to decide on, to design, to craft, to create a very particular experience that you're trying to create. So Drew's going to kick us off with this game.

ANDREW GRANT: Who's played backgammon? Wow, really? No one plays backgammon anymore. All right.

Backgammon is a pretty classic game. It was invented a good, oh, 5,000 years ago or so, about the time that we learned how to write. It's got a lot of planning, oddly enough.

There's a lot of probability and randomness to it. But you can plan for that randomness by setting your pieces in the right spots. You can choose different strategies.

Are you going to be aggressive to take the other person's pieces? Or are you going to be defensive, and just try to protect yourself? Are you going to leave something in the opponent's

backfield, so if you're losing at the end, you can maybe mess them up somehow? Or when do you change from trying to mess them up to shooting for the end goal?

There's a couple different ways to play backgammon, oddly enough. And it actually is a fairly deep game. There's a lot of probability involved, but some tactics and some strategy going on in there, too.

If you haven't played it and you like board games, it's well worth playing. Because I think a lot of the things that you're going to see in modern board games really owe a lot to this game right here, especially when you add randomness as a mechanic.

Go and chess are obviously really, really awesome games, too. But they're a very different style of game. Because they don't have the randomness in there. It's a very, very different way of thinking about it.

This is another one where I think there's more planning than people give it credit for. Who's played *Twister?* That's more like it. OK.

So in *Twister* you actually can plan ahead. You probably don't. But if you played *Twister* a lot, you might start. You might realize, if I put my right foot there I'm really screwed if they call red next time. So I guess I won't. I'll put it over there.

It kind of depends on how many people are playing. If there's enough people playing, you can't do that. You just have to be fast. So *Twister* is an interesting game where sometimes you have to make your decision extremely quickly.

Is there strategy in *Twister*? Anyone? I see some nods. What decisions would you make?

AUDIENCE: Maybe to go under somebody, if you have to reach and lift them off the board.

ANDREW GRANT: All right. Yes. So there's a strategy to *Twister* if you want to play *Twister* to win, darn it. And maybe you're thinking that I've got better balance. Or maybe I'm a little bit bigger than they are. I bet I can nudge them a little bit.

That would work. Right? That does work.

Or maybe you want to play off in your own little corner? That works, too, unless no one else lets you do it. Or if you get yourself too tangled up, you can't move even when you're by yourself.

Have you done that or seen that? Someone's off by themselves doing *Twister* and they fall over with no one near them? It totally happens.

There's other strategic decisions for *Twister*. Are you trying to win? Are you trying to stay near someone, or away from someone? There are totally other game goals to *Twister.* And they're up to the people who are playing it.

What is next?

RIK EBERHARDT: Sara.

SARA VERRILLI: OK. That would be me. So this is not what everybody thinks of when they think of strategic games. But I want to point out that decisions come in a lot of different styles.

You're not going to ever ruin your game by any decisions you make in *FarmVille*, particularly. But it is a game that makes people make a lot of choices about what they want to do, relatively freely.

The strategy-- where strategy and resource management comes into *FarmVille* is you have a limited number of actions you can take. Which means you can only play the game for so long. Assuming you do not want to become one of those people who spends a big pile of money playing *FarmVille*, which I'm going to assume you are.

So you need to manage your actions, and think about how many actions you're taking, and think about when you want to come back to the game. And the way they do that, and the way that you can game that system, is by looking at how long it takes things to grow, what you want to grow, what you want to make.

FarmVille is actually at this point kind of outdated, because they've moved to *FarmVille 2*. But I did not actually want to go there and play it to have the expertise to tell you about it. I did spend enough time playing *FarmVille* to know that there's actually a whole lot of interesting decisions you get to make there.

And it's not a game people think of when they think of meaningful decisions. Because they've heard a whole lot of, oh my god, it's that click game. Just because it's a game that doesn't have a lot of deep consequences doesn't mean it can't have a lot of deep play.

Am I hitting the right button? Let's find out.

So *Plants vs. Zombies* is essentially a really popular tower defense game. And tower defense was not very popular until *Plants vs. Zombies* came out. Because most people hadn't done it very well or presented it very well.

But in *Plants vs. Zombies*, you've got tons of choices. Who here has played *Plants vs. Zombies*? Let me ask the first question. OK. So yeah, you know what I'm talking about.

They got the UI right, absolutely. So it's easy to play. And every time you go in to play a level, you get to choose. Which strategy are you going to use?

You've got sort of the long-term strategy choice for each level, where you get to choose which creatures you're taking or which plants you're taking. And you can go back and play each level with a completely different set of plants, and still succeed, or fail at it. In my case, it's often more failing.

They give you just enough information so you can think about what you're going to bring. You can see what's coming in. You don't know how many of those are going to come. You don't know when they're going to come.

So you have to both plan for, I know there's going to be a zombie need. But I don't know how much of them it is. How much resources do I need to devote to worrying about the zom-- zom-- zombies. I'm not getting that right at all, am I? How much resources do I have to worry about devoting to that versus all the other threats I've got coming in?

Now I have to admit something terrible. My daughters play this game. And I have watched them play this game. But I haven't actually played this game.

And I was going to play it last night so I could talk about it. And then my daughters hid my iPad on me. It's very hard to download a game when you don't have the device to download it onto.

So has anybody else played this game? OK. A few people have.

Your goal in this game is you're a virus. You would like to evolve and hopefully take out all of humanity. And as you're playing the game as the player guiding the virus's development, you get to make a lot of choices.

How lethal is it? What kind of symptoms do you have? How easy is it to spread?

My daughters are working on the infect the whole planet with a completely symptomless disease tactic, and then evolve quickly up to lethality and take everybody out. They haven't succeeded at this. It sounds like the perfect strategy.

But they haven't succeeded at this. Because you kind of need symptoms to cause damage. And they can't evolve the symptoms fast enough. Once they've gotten everyone infected and they're going from completely harmless to deadly, they can't manage to evolve it fast enough. So it's got a lot of interesting play there, where you've got a lot of different tactics. I think they can get it someday, but they haven't yet. They'll need to try a different tactic.

And finally, a game I have played an awful lot of is *Pandemic*. Who here has played *Pandemic*? OK.

So what is the main tactic and resource you have in this game, if you've played it?

AUDIENCE: Turns.

SARA VERRILLI: Turns-- well, time. Time. Yeah. In a lot of ways, in this game, what you're playing against is time.

Because when you run out of the deck, if you haven't managed to clear things, the game is over. And the resource you've got, and the choices you've got, is where people are going and which area they're fighting.

The overall plan of the game, of course, is there are four epidemics raging. You're the CDC. You need to find cures for all the plagues before they take out the rest of humanity. And so you have to balance both your long-term strategy of finding a cure for each of diseases with the short-term strategy of keeping the diseases under control, so that you don't end up with everything being destroyed.

PHILLIP TAN: All right, my turn. [INAUDIBLE] Who's played *Dominion*? OK, pretty popular.

I certainly recommend it as a good introduction to the basic idea of deck-building games. Actually, how many people have played *Magic*, *Yu-Gi-Oh!, Pokemon* card game? OK. All right, a lot of that.

In many of those games, and in *Dominion*, a lot of the fun is just building your own machine, your own little engine, that is going to execute out your strategy. Now your engine happens to

be a deck of cards. You get shuffled up, so it's in some sort of random order. But you get to choose what cards go in there.

In a collectible card game like *Magic: the Gathering,* you are buying those cards, and then assembling the cards, or there's a draft [? mechanic ?]. In a game like *Dominion*, there is a way where you buy more cards while you're playing the game.

But in the end, what it all comes down to is as you play this game, you get cards that go into your deck. They are going to end up being shuffled in your deck. And at some random time in the future, it's going to get played out. And so the cards that you choose to put into your deck turns out to be your main means of motivating you towards a [INAUDIBLE] strategy.

There are a range of different ways to play this game, and different possible success criteria. And the really interesting thing about the *Dominion* deck is I think it gets something like over 20 different kinds of cards in a box of *Dominion*. But you only play with 10 of them-- 10 plus victory points.

So every time you play the game, the strategy starts by just seeing, what are the 10 cards I've got in front of me for this session of play? And then you choose, out of these 10 cards, what are the cards that I'm going to select and in what proportion, in order to build this engine that is going to motivate me towards victory? So it's an interesting game. I would strongly recommend it, if you know someone who's got a copy of this game or any of the expansions. They're all interchangeable.

Drop7-- mostly played on mobile, although I think there is also a browser version of this game. How many of you have played this? Yeah. Perfect for a T ride, I find. You can finish a session in a single trip on the T.

The idea is that you're just dropping one number from the top of the screen. And that number ranges anywhere between one to seven. And if you manage to make a row of numbers that add up to-- let's say I have a row of sixes. And I have six sixes in a row. Then all of the sixes are going to disappear.

And it's going to also clear out some of those gray circles where you don't know the numbers yet. It's going to review the numbers behind these gray tiles. You can think of them as tiles that were turned upside down.

So what you really, really want to make is chains. Chains are where, if I have a six, five, four,

three, two, one, then what happens is that six is going to burst. Then that five is going to burst. Then that four is going to burst. Then a three, and so on and so forth.

You set up these massive chains. And basically, you're just assigning how you're going to place this. And if you played a game like *Puyo Puyo*-- I'm trying to think-- certain kinds of Match 3 games like *Bejeweled,* you'll get this sort of chain reaction feeling.

And some games more than others encourage you to set up these chains before you drop the triggering piece. *Drop7* is very much about that. So I believe, if you can find it on a browser, it should be free. I'm not so sure whether the iOS versions are free. They are? Great.

I keep forgetting the computer is on this side. The *StarCraft II* visual novel.

AUDIENCE: Wait, is it done?

PHILLIP TAN: No, no. It's still in development. Full disclosure, I am a Kickstarter backer of this game. As has been mentioned many, many times, I am a *StarCraft II* player.

But I am not encouraging you to go and learn that over the weekend. Because *StarCraft II* is more of a lifestyle kind of thing.

[LAUGHTER]

But if you're just kind of wanting to see what the strategy is about, this is an interesting game where-- if you've ever played a visual novel, they're kind of like *Choose Your Own Adventure* in digital form. And they are usually very strong narrative- and character-driven games.

You play a teenager who has moved to Korea in the hopes to start his or her pro gaming career. And you go into a LAN center. And you play random games against people.

Only these are the kinds of decisions you get to make. Which strategy are you going to go for? I usually find this to be a really, really good way of conveying the macro-level decisions, the really, really high-level long-term strategic decisions that you're going to make.

Because you're not actually playing *StarCraft*. You are choosing between two or three choices. And then it plays out.

But it gives you a good idea of the kind of high-level decision making. Whereas most people see someone play StarCraft as like, the [KEY-CLACKING NOISE]. And you're not seeing that

long term strategy.

So we do have a Let's Play up. And the URL is there. But if you just Google MIT Game Lab, *StarCraft II* visual novel, you'll see the YouTube video and get an idea of that. Because I'm not so sure if you can still download the demo. But the demo does have some of these features.

ANDREW GRANT: I am last.. *Mini Metro*. All right. So my games that I chose-- what's up? Oh, thank you. Cool. So this one I really like as an example. Eww, it looks awful on the screen, though.

So I really like this one as an example because it has not yet been published. On their website they have their alpha build. And if you were to buy their game, what you would see is the beta build. And then later on, when it actually does publish-- you guys can even see this on YouTube. You can see what it's going to look like when you actually buy the final game.

It plays exactly the same. It looks polished. It's a really good example of here's a game that works and is functionally perfect. And they could have just shipped it.

And I'm really glad that did, because I play it a lot. And when they finally ship it, it's still going to play the exact same way. It's just going to be a little bit prettier.

And it really doesn't matter. I don't really care about that. That's something that we're going to stress in this game. We're more concerned about that, is it playable? Does it work? Can I understand how to play it-- more so than the pretty, more so than the aesthetic and visual polish.

So in this game, it is a game about managing a subway. All of the sudden a subway station appears in the middle of space. And you have to build a line to it. So you build your line.

And then passengers randomly appear on the subway trains. But you don't have to actually get them from place to place. You don't control the trains. You just build the lines.

The trains move on their own. And they're moving at a constant rate. So you can kind of guess when the train is going to get to the station.

There are strategies to-- you can move your lines around a little bit. But once your lines are in place, for the most part, they're static. You can't really change them all that much. So there's some limitations going on between how much you can move a line or not.

At the end, if you last a week, it's going to give you one of three options to do a power-up,

basically. Either you can build a new line, add a new train, increase the capacity of a train. So you're making a permanent choice there.

And that's going to affect the later game. So you're making all sorts of these decisions at both a macro and micro level. And there's a really good write-up about the game right there, if you just want to read that.

FTL-- I lost a ton of productivity a few months ago because of this game. It is a game about fighting in spaceships without actually controlling all the fighting so much. You're actually upgrading your ship, increasing your capacity, buying crew members-- and hoping your crew members survive is a lot of it.

You could-- and I do this-- manually control all your weapons. And actually to succeed, you really do have to. Because the interesting thing with this game is it's a perma-death game.

As soon as you die, it's dead. You have to start over. You have to restart the game. Luckily the game only takes about three hours to play. But you're lucky if you get through those three hours.

And when you finally get to the last boss, and you're there-- the thing about this game is in order to get to that boss, there's all sorts of different challenges you're facing. And you're making these decisions about, do I buy this weapon or that weapon to help me with the thing I know that I'm going to face in the next sector?

Because I'm losing health. I'm really dying here. But is this weapon really going to help me when I get to the boss? You're constantly going back and forth.

So I'll often find, after three or four, that I'm at the boss and I'm completely ill-equipped to fight this thing. And I just die. But I do play out to the end, because it's fun.

Last game I have-- this one's free online. It's called *Gridland*. If anybody's played the game, *A Dark Room,* a text-based supply chain game, this is the Match 3 version of that game, a little bit.

There's two stages in the game-- day and night. It's Match 3, so you just collect three gems. And you get whatever the gems represent-- resources like clay, wood, and stone, food, paper, gems.

You can upgrade your little factories to make better gems. Every time you upgrade one of these, though-- so if you upgrade your wood, you'll get a better sword. If you upgrade your clay, which you need to make the better wood, you're going to get in the nighttime a stronger monster.

So in this case, this is immediately daytime and then shifted over to night. The exact same layout is there. It's just the logs have become shields. The little blue things that are supposed to be iron become swords. And all the other things become enemies.

So a lot of the strategy for this game is, what's going to help me now? What's going to help me in the future? What's going to hurt me in the future? And what's going to hurt me immediately as I switch over to night?

Once I get into the night mode, I'll have known exactly what I'm going to be facing. Because I set that stage up. So unlike *Bejeweled* where you're just constantly just clicking things, or *Candy Crush* where you're just constantly just clicking things, there's actually no time limit on this.

And it took me-- it's embarrassing how long it took me-- a day or two to realize that I didn't have to play that way. I could just wait, pause, and play. There wasn't going to be a time limit pushing me to continue moving forward. And that's an interesting presentation thing.

When you're presenting your games, especially if you're using a previous genre, there's going to be genre conventions that the players are going to understand. That is going to shape their user experience. And we'll be talking about that about project three.

So that's it. If you want to play these games or watch these games on YouTube, please do. And if you want to know more about choice and conflict, there's a nine-minute video at Extra Credits. We use Extra Credits a lot, because they're short. They're fun videos on game design. I highly recommend it for this class to give you a leg up on some of the design challenges that we're facing here.