PROFESSOR: All right, so that's a good start to Battlecode Lecture 2. Welcome. Today, we'll be talking about how to write your first player and then at 6:00 PM we're going to be having some Indian food. I'm pretty excited. And it looks like there will be enough Indian food for all of you. I correctly estimated how many of you would show up today.

Isn't that a curious aspect of statistics, that you cannot tell what one person is going to do tomorrow, but you can tell, very accurately, what everyone, in aggregate, is going to do tomorrow? Quite interesting. I believe in 2006, the number of automobile accident deaths was correctly estimated to the last man, or something absolutely crazy like that. Because it's a number like 40,000 and it was unbelievable.

So with that heartwarming opener about automobile homicide, I'm going to play some slightly less-- some slightly more calming images. So you'll see here on the screen is the evolution of a very simple rule. This rule is something like sum up the number of nearest neighbors and if that number is even then the square will be white in the next turn, and if the number is odd then the square will be black. It's something like that, in essence. And it ends up making this repeating pattern that is quite mesmerizing to see and is an excellent example of emergent complexity.

So, what we'll be doing today is we're going to write code that's pretty darn simple. A rule like mod two = zero, something like that. And what we're going to get is some really awesome looking behavior that comes out of that very simple rule. So there you go, that's my tie-in for that.

Administration stuff, so here I've brought up the Battlecode website. You should all be set up with Eclipse right now, already. If you haven't then you're falling behind, but then again, at the same time you've saved yourself some effort because we've just put out another release. So, what you want to do is download the installer and install it to the same directory where you currently have Battlecode. And that should update everything without deleting the progress that you've made on your player.

So there's that. There's also some exciting things I'm supposed to be announcing. That we'll be giving away prizes for things other than winning. So, if you're going to see my amazing code and you're going to go, man, I didn't think I could do that well. Although, I find it much more likely that you could watch what I'm going to do and then go, pfft, I could have done that in my sleep, I can't believe this guy. But yeah, we'll be giving away prizes for doing things other than winning this year. And those things I'll be announcing throughout the year so you can keep them in the back of your mind and be like, oh I have code that does that I should think about doing this.

There will be beating the reference player with the fewest total bytecodes, beating the reference player in the shortest possible time. By the way, we haven't released the reference player. We will be releasing one within the next couple of weeks. And, I don't know if I've said it yet, but to get credit for the course you have to beat the reference player that we'll provide on a map set that we provide. So it'll be called reference player and there will be more information about that later. You're also able to get credit by writing a short report on how your code works if you weren't able to beat the reference player. So don't fret too much. So, those two ways are ways of winning money. You can also have the most interesting use of our source code, like if you want to make it so that Nyan Cats are part of Battlecode, or some other silly thing, that might get you something. And there's also most impressive non-winning strategy, which may very well be the thing that we see today.

All right, so here I'm going to go to Eclipse, this is our supported development environment, and I'm going to start setting up a new player. So, I've got all these extra Battlecode installs because of the development nonsense that went on. So what I'm going to do is I'm going to just start from scratch and I'm going to make sure that I talk about the functions that I'm using in ways that are useful to both beginners and intermediate people. So, beginners have to like hurry to catch up and intermediate people are more hearing about the Battlecode API and hearing about how things are put together, specifically, about Battlecode that they wouldn't already know from their job experience, for example.

So, let's start this way. I'm going to make a new team that I can fight against my other teams that are here. Because I want to keep trying this. A big thing about programming is making it iteritable-- Iterable? Iterable. Interval implements runnable, anyway these kinds of things. And so, you'll want to have a bunch of different robots that can play against one another. And so, you can be like, all right, I'm going to right-click this team's folder and I'm going to go to Create Package and I'm going to call it-- What's a good name for this awesome robot that we're going to make? Yeah, Awesome Robot.

OK. Play Awesome Robot player, all right, I guess that's decent. Yeah. And then, in this one, I go New File, and I call it robotplayer.java. It has to be called that because that's the thing that our game engine is looking for. So then, at the very top, we're going to write down package Awesome Robot player, and then we're going to put a semi-colon. Yeah, because you've got to do that.

Then we'll import everything in Battlecode. So, we can use the Battlecode, the Battlecode API which is here in battlecode.common.star. So that'll give us everything that's in the Java Docs. So, what I'll want to do is have open the Java Docs, maybe in one window or on the side, and have open this in another window so that I can bounce back between them. I'm not going to do that at the moment because that would make things even harder to see.

So, what I'm going to do here is I'm going to increase the text size because I know some of you at the back are not going to be able to see 100% of what I'm doing. If you want to follow along on your laptops, go ahead. I believe we'll be able to post this code after lecture, so don't worry too much about making sure that you can type as slowly as I can. Awesome. Is that too big? It might be.

Here we're going to-- we're just going to run straight in. So, public class

RobotPlayer. You've got to write that. So now, so you've written that. The Java IDE, you notice, will put these brackets in for us real nice. And you need a run method, public static void run. And that's going to take the argument, RobotController, and then you can name it whatever you want. I generally call it myRC.

So, this RobotController object is the thing that is a robot. So, when you run this code every robot has is its own myRC. And so, the methods of myRC, which you'll see later, will apply to that particular robot. You'll see some examples. So, what I like to do is I like to make-- I like to define the variable for this RobotController object. I like to define a variable so it can be used throughout this class, throughout this package. So I'm going to type private static RobotController RC. And RC is going to be the thing that I define to myRC. I'm just going to say, there you are, now you've got it and I can use all of the methods in RobotController by typing rc. and then, bingo, right there, you see Eclipse is telling me all of the methods that are in the Battlecode.common directory that are all these methods that you can use.

So, right here, it's like this is a really convenient way of doing things. And so, you're going to see that throughout today's lecture. So, what I want to do is I want to make sure that I've got everything organized in a way that I can use it. Because I'm sure everybody here has written code in some language or other that just goes on and on and on and it's absolutely massive. And so, every time you want to make a change down here, and up, and down here, you just get a headache and you have to fall over and it's not very good. Because you could be at elevation and falling over at elevation is highly damaging.

So, what we want is this method will run, but if we only run it like this let's see what happens. Can any of you guess what's going to happen if we run Awesome Robot player?

AUDIENCE: It's going to die.

PROFESSOR: That's right. It's going to die. Can you tell me why it's going to die?

AUDIENCE: Because run will reach the end of the method.

PROFESSOR: That's right. That's right, because run will reach the end of a method. So, both of the teams explode. I'm kind of curious which team won that match. Let's see here. It says they both died. Ah, Team A won. Good thing to know, Team A will win if both teams die. So, let's make a while true.

Yeah, yeah, all right, the game's unbalanced. I recognize that. We'll patch it in, I don't know, in two seconds or ten minutes, or whatever. We're up to-- did I mention that we're up to 1.1.0. So, definitely go install that. 1.1.1, that's so fast. Our development cycle is amazing.

AUDIENCE: [LAUGHTER]

PROFESSOR: So, while true will just keep looping and so let's try this, let's do while true. What's going to happen? Can anybody tell me what's going to happen when I do while true? Something that's undesirable.

AUDIENCE: It's going to stall?

PROFESSOR: These guys are going to use 10,000 bytecodes because they just keep looping, while true, while true, while true, while true. And I guess, this one comparison, this going to the end of the loop and coming to the top, I guess, costs some non-zero amount. So, they end up reaching their bytecode limit, that's no good. So if we do RC.yield that ends the turn. So, whatever happens here above rc.yield will happen and then it will go to the next round. So now the bytecode usage for each robot, you'll see-- and right now all we've got are these headquarters robots-- the bytecodes used is three. I'll just have to read it out because it's so small you can't see. Yeah. Yeah, it's three so that's fantastic. That's the minimum that you can get.

All right, so now we're going to put real stuff in. We want the headquarters to behave differently from the robots. For example, because the headquarters can never move, it can't walk around. So let's use if rc. Now, what we want is we want to figure out what type of robot it is, so let's look at the different options here. Oh look, there is an rc.getType. Yeah. All right, so now we can just say if that = RobotType.SOLDIER. That's not what I wanted. Soldie-- Yeah, if it's a soldier then we can put soldier code here and, otherwise, we can put headquarters code here.

So, let's make a very simple, a very simple thing. Let's say we want to just kill the enemy. Kill them. Kill them dead. So, let's have, let's have us do this. We will define a new variable, right here now, in this space. Direction dir = rc.getLocation.directionTo. And now, we'll just-- we want to go straight to the enemy. We want to make, we want to find the direction to the enemy and then we want to go that direction. So it's telling me there's an error because the directionTo needs a direction to what. So let's give it a location. We're going to give it the location rc.senseEnemyHQLocation.

So, there you go. Now we've already got the direction. I'm going to put a semicolon at the end of the line. And now, I can see I'm going to try to go that direction. To move in a direction you've got to be active. Because there are certain things you can do, like laying mines and defusing them, that make you inactive. So, I'm going to check if I'm active. If rc. is active which returns a Boolean true if I'm active. Well then, let's go ahead and try to move. And we'll see what happens when we try this, rc.move in direction dir. And so, now we're just going to see, we're going to see what happens. Oh, what's this? It's telling me that an unhandled exception is here. Move can throw exceptions. And so what we do is we put in add throws declaration. So then, up here RobotController is going to throw an error if this thing has a problem. And this is a way you can use-- this is Java's built-in form of debugging. And you're going to find it very useful because when you have a problem, it'll tell you what the problem is.

So let's go ahead and do that. Hm, well, the problem is we don't have any soldiers, we can't produce any units, so there's nobody who can run this code. So, let's go ahead and add in some unit spawning abilities right over here. So I'm going to go to example folks player, open up robotplayer.Java, and I'm going to copy this part which is for if the robot is a headquarters. And I'm just going to go ahead and paste it right in there. And now, all of a sudden, I'm spawning units at random directions around myself as the headquarters.

So now, we should have units in our Awesome Robot player and they start to run at one another. Oh, but something strange is happening, when they run into one another it seems like they are immediately colliding and exploding, boom. That one exploded, and then that one exploded, and that one did. That's weird looking, isn't it? It almost looks like this guy is changing color, but no, they're running into each other and exploding.

And the problem, it's not just that they're going too fast, because that isn't the thing, it's that there's a game action exception. And you can see here that there's tons of exceptions. And they all showed up here. And I can be like, OK, I don't understand anything at all, I can't read any of this language, oh, but look, it says robotplayer.Java:15.

I can just click there and it shows me what's throwing the exception. Oh, it's this line. Oh, I see, I see, it's having trouble moving there because, maybe, there's something in the way. So let's also stipulate that rc. move in the direction. This is another method that's provided to you that lets you check whether the direction is movable.

So now, when we run the code, they're going to just go to the middle, kiss and kill each other nice and slow. Just like that. Wow, that's beautiful.

AUDIENCE: [LAUGHTER]

PROFESSOR: And I think that's just going to happen all day long until the end of, see when you get past a certain number of rounds, the headquarters begin to take end of round damage. So you can see, the headquarters are just going to die. And that happens. And they both die at once. And I believe player A will have won. Oh no, player B won.

AUDIENCE: [LAUGHTER]

PROFESSOR: Now, let me tell you, that's balanced.

AUDIENCE: [LAUGHTER]

PROFESSOR: Yeah, so we know what we're doing. All right, this is starting to get cluttered, right, I'm not happy with this. I'm not a neat freak, my wife kind of is. No, I shouldn't say anything about my wife on this because it's going to be recorded. No, she's not a neat freak, she's a very wonderful woman. OK, so, god I'm in so much trouble.

All right, all right, so here's what we're going to do. We're going to make a new method that will clean the place up. We'll call it public static void, we'll call it hqCode. This is the headquarters code and it's the stuff that I want the headquarters to do. And I'm just going to copy this in here and then I'm going to put hqCode there, bing. So now, I'm calling this method which I've defined below. And that'll just insert that code right in there, really convenient. Now, can anybody tell me why there's a red underline under spawn?

AUDIENCE: Game action exception.

PROFESSOR: Yeah, a game action exception. So what'll happen is I'll just click on this button and now this code will have-- this guy will throw the exception. So, exceptions from here will throw up and they'll throw to here and then they'll throw even more. Now, let's say that we don't want to throw them all the way up to the top. Because you don't want run to ever throw an exception because it causes your robot to explode. So, how do you prevent thrown things from getting out? If Billy throws me a football, how do I keep the football from hitting my house, which is behind me?

AUDIENCE: Catch it.

PROFESSOR: That's right, I catch it. OK, and in Java, the way that that's done-- I know a lot of you this already, but some of you don't and you're going to think that was a really clever analogy-- is you put try on the outside. All right, and then you put, I think I want to put it here, I want to put catch here. Yeah. And so, specifically, I want to make sure that I catch the exception. So, I'll go, I'll catch exception e, which will be whatever exception comes out of try will go into e.

Now, you see that the tabbing is messed up because try should have a tab after it. So I just highlight this and I do control I and that automatically tabs it out. 95% of you knew that already, but the 1%, the 5%-- I can add-- the 5% that didn't know are like, oh, that saved me so much time, I'm so happy. Yeah, so you we're going to catch the exception and then we're going to do System.out.printline and we're just going to say "caught exception before it killed us." Yeah, and we're going to just print it. So we're going to go e.printStack.Trace, and we'll pick the third one because it looks good. Yeah, so now when there's an error it'll just print down to here, rather than killing the robot. At the same time, I should mention, that any time a try finds an error and moves to catch you get 500 bytecode penalty right there. So, that's not a great way to write code. You want to avoid, you want to avoid having the exceptions at all.

All right, all right, but we're getting stuck because we haven't had as much killing as we really wanted to have. So, I've got an idea for how to make code that kills the enemy. So, let's go with that idea that I have, that I definitely have, and I'm not coming up with on the spot. So that, let's-- that was a joke, I actually am not coming up with it on the spot. I'm so offended that you didn't find that it was joke. I just have to find out where it is in the notes. It's in, it's in, capture encampments, oh, yeah, let's not capture an encampment yet, because that's not as exciting as-- because what we'd really like to do is just defeat the enemy and then we'll worry about shaming them into a greater defeat, subsequently.

So, let's see here, if you're a soldier, maybe the right thing to do is start by massing up your troops in a place. So, let's get a place to mass up the troops. Now, maybe you only want to do this once. It's like a rally point for everybody. So, let's get this rally point. I'm going to make it here because I want everybody to know where this rally point is. So, a rally point is a map location because you want everybody to localize around a place. Now, a map location is yet another object, like RobotController, and it has its own methods and properties associated with it that you can find in the Java Docs and that you can call-- and that you can find your own way.

So, let's do this, MapLocation rallyPoint. Yeah, we're just going to rally somewhere. This is going to be a good strategy, trust me you're going to be excited. So, let's do, let's find the rally point, start out. So, let's make a function because it's already getting far too fluffy, I think. So, let's find the rally point here. We'll say rallyPoint = findRallyPoint. Yeah. Now, where should we rally? I think a decent place-- oh, look at this, it's even going to help me out here. It noticed that there's no method, so I can just click here and it'll make me a nice method. Look at that, it's just like this hqCode with the private static and then it has the type that it returns. See, this time we're making a function that returns a type. This didn't return anything, it just did code here. But now, we want it to spit out an answer. So, we're going to have it return a map location when it's done.

So, what we'll do is we're going to get some location between us and the enemy, seems like a great place to rally our troops. So, between us and the enemy. Let's see, the enemy location-- so, I'm just going to make this nice and simple-- MapLocation enemyLoc. You can define variables anyway, this'll be local to this function. The enemy location is rc.senseEnemy. Yeah, there. OK, MapLocation, our location is rc.senseHQLocation. Yeah. And now, I want to find the average. I'm not going to do it in the smartest possible way, I'm going to do it in a way that's easy to understand. Is I'm going to say there's an integer x which is enemyLoc.x + ourLoc.x. All right, now, I guess if you took the average of the two that would be in between, right, that would be in between the two. But we don't want it completely in between the two because we want to just move a little bit out. So let's make it more of ourLoc and less of theirs, so three of ours. And now we put a multiply and that should get us one quarter of the way to the enemy, because it's mostly us and somewhat them. Yeah. We'll do that for y as well. There's probably smarter ways of doing it than just writing out two lines, but it's pretty clear.

And now, what we'll do is we'll use the constructor for map location. So, we'll say mapLocation rallyPoint = new MapLocation and we give it an x and a y. Yeah. Yeah, all right, now we can just say return rallyPoint. OK, so now, we're going to try to go to the rally point. RallyPoint has been defined here, that's useful. So now, later we don't have to give it a direction. We can just say Direction dir = getLocation.DirectionTo. Now we'll go to the rallyPoint instead. Yes, yes, to the rallyPoint. Let's see if our robots are going to the rallyPoint.

I recommend that you try running your code every time you make a significant change. And look, they did, they went to the rallyPoint, but they went there in a dumb way. Because when one of them was in front of the other, they couldn't go around. Because now, this guy can't spawn anymore robots. I'm pretty sure he's trying, but it's not working. In fact, what error am I getting here? I'm getting you cannot move in the direction non or Omni. That's an interesting problem.

That's the problem that when a robot is on top of the spot he shouldn't try to move to the spot that he's on top of. So, we should also specify that there's a distance and he should only try moving if there's a distance. So, let's just put dist = rc.getLocation.-- do you always do that too, I always put i before o-- .distanceSquaredTo, and distanceSquaredTo rallyPoint. Yes, that's a useful method. There are a lot of useful methods for getLocation.

And you might say, all right, how was I supposed to know that distanceSquaredTo was a method from getLocation. I was going to write my own manual function that goes looking for x's and y's, and squaring them, and subtracting them, and so on, not that subtraction is really even relevant here. But, how was I supposed to know? Well I'll tell you how you were supposed to know. You just go to where you installed it. I put it in Battlecode 2013. You go to doc and you go to this thing, all classes frame, and this shows you all the classes. And one of those classes here is Map Location. And under Map Location, you see that you can add map locations to directions, you can add them to x's and y's, you can use directionTo, distanceSquared, you can check whether two map locations are equal to one another, if they're adjacent to one another. That's pretty useful. So, having a working understanding or memorization of these things will help you along the way.

So, now we've got distanceSquaredTo. So now, we only want to move toward the enemy. So, if our distance is greater than 0, then we'll try moving there. Otherwise, let's not, because it's just going to end up giving us trouble. And once again I'm going to do control i to do this indenting. But this is getting somewhat cluttered, so why don't I just make this soldierCode, especially here when everything is a giant font 16. So, this is soldierCode and I'll just go ahead and have it automatically create me that method. I'm going to go ahead and put this in the method. And now rc.move has got a throw declaration. Do you see this? Pretty soon I'm not even going to need to be here because it's doing everything for me, my goodness.

So now, they should go to the rally point and they shouldn't be throwing quite as many exceptions as they were before. Oh, look at that, in the game output window you see no exceptions from round 0 to 5,000, excellent. But, again, they're just stuck behind one another. How can we make it so that they're not stuck? Well, here's a good way to do It. So, here you're only trying one direction, but now let's make a way to try multiple directions. The way that I'm about to show has its own emergent complexity built in. You're going to like this a lot, I believe, so. Yes, let's do it. Let's define a list.

Now, lists are defined this way, you put an open square bracket and I'm going to make a list of integers. Because I'm also going to use some tricks for direction because a lot of you are probably thinking, oh gosh, I noticed in the Java Docs that direction-- oh here, I'm going to go to Java Docs again-- direction is here and there's all these directions, east, north, northeast, blah. There are eight directions. I can rotate them left and right, but am I really going to enjoy rotating them again and again to find which direction I'm going to. OK, I could go forward, if forward is blocked, then rotate the direction left, if the left is blocked, then rotate-- no that would be actually really annoying to write. And it's not the most elegant way to do it.

The way we're going to do is we're going to get the direction values and then we're going to use the direction list. So, the direction list consists of these eight things. I believe it starts at north and then it goes around to northwest. And so, direction list at 0 is north. And so, I'm going to show you how we're going to do that and what amazing result it has, because a lot of you already know how to code, but you don't know what amazing result it's going to have, or maybe you do. So, this integer I'm going to call the directionOffsets. And you can define an integer list by setting it equal to this construct. So, I'm just going to put curly brackets and now I'm going to do 0, 1, -1, 2, -2, yes. So, that now gets me some direction offsets and I need to start checking directions. So I'll use for integer d among the list directionOffsets. So now, this for loop will loop again and again and it'll put one of these integers in each time it runs. So, the first times it runs, d will be 0, the second time d will be 1, and so on.

So this is for d in directionOffsets and there needs to be an s. And what I'll check is if I can move in that direction. So, let's go ahead and check that. Oh, by the way, let's simplify matters a little, because we've got is active here but there's no reason to look at directions if you're not active. So, let's go ahead and put this is active up here. Oh gosh, the guys of you who are following on your laptops must be just frantic at this point, yes, because it's like that, it's very emotional, very personal. So then, what we'll do is we'll check if you can move. If you can move-- Oh, but wait this isn't right, what direction is now important? Direction lookingatCurrently-- you can name these whatever you want-- the direction you're currently looking at is the following.

First, you want to look at direction. And so, see dir up here is the direction to the rally point. So, we'll look at direction we'll say, OK, what is, oh, let's see here, ordinal? Ordinal gives me which number this is in the list of eight directions. So, if I'm currently looking at north, that is to say if the direction of the rally point is north, then dir.ordinal will return 0. Because the list of directions starts with north and in Java the index of a list starts at 0, so the first element is at 0. So dir.ordinal gets me 0.

And then I'll add the direction offset number and now I will index into directions again. So, I'll say Direction.values is a list of directions. And the part of that that I take, indicated by the square brackets, will be dir.ordinal + d. I'll add 8 to make sure that's it's not negative, and then I'll do mod 8, the %8 is a mod 8. And you're going to find, this may be alien to you to begin with, but you'll find that this kind of construction helps you out immensely. So now, you're looking at a direction currently. If you can move in that direction, what you want to do is leave this loop. You don't want to stay in the loop anymore. It's giving me this, oh yes, semicolon? No, that's not right. Oh right, right, if I can move. I forgot to put if I can. Yeah, thank you for catching that me, yes. If I can move the direction I'm currently looking at I want to leave this loop, so I'm throwing a lot at you, but here you go.

We'll name this thing the lookAround loop. And now, we can do break lookAround. And now, that will get us out of this piece. Yeah. Everybody likes to be splashed in. Nobody likes the first 27 pages of that intro to C++ book that talks about declaring variables and what the hell private static means. It doesn't matter what private static means, you don't need to know, you don't need to know. Don't you think? I hate that. I tried to learn languages so many times, here I'm thumping the desk. Anyway, I have nothing but animosity for their correct ways of doing things. That's why I don't study number theory, my goodness.

AUDIENCE: Nonsense.

PROFESSOR: Yeah, because they prove that addition works before they do any addition. I can't imagine how they get anything done under those circumstances. So, you can see here that nothing works at all.

AUDIENCE: [LAUGHTER]

PROFESSOR: Let's see, the robot is spawned, but it never moves. OK, so it can move, but it never actually did move. That's right, so what we need to do is make it move. So, within this loop, we've just exited the lookAround loop, that's this one. So, we're just going to put rc.move-- please move, please move-- atCurrently. Yes, when this guy moves-- Did I-- Oh, scope. Oh man, these guys are smart. So, you see here that this is inside this block so it can't be accessed from outside. So, what I'll do is I'll define it outside and I'll just equal, I'll pre-initialize it at dir and then when I use it in here it'll be accessible to that out there. Look, you think I made a mistake, but it was actually deliberate so that everybody would know about this scope problem, as you refer to it. Although, to be honest, Scope to me just means Listerine.

All right, so now, what they're doing, it's hard to see, but they're moving around one another and they're oscillating. See that? Look at yellow. He's going around and that's because he's looking at the best direction that he can go in. Yeah, that's pretty cool. What is going on here? Cannot move in the given direction west, south, blah, blah, blah. Now why can't they move in that direction is an interesting question to ask. And it may be because some of them are entirely surrounded by other robots. This guy, for example, may end up being entirely surrounded by the robots.

One of the problems that you will encounter is that every round robots move and they don't move synchronously. So it'll be really instructive, if I had the time I would show you that the number of adjacent robots is not the number that you might expect. Because, for example, let's say this guy wants to move to the right. Let's just say he does. He can't move to the right this turn if he executes before this guy. Let's say they both want to move to the right. But if this guy executes first then he'll move to the right, that'll take place, and then when this guy looks surround himself he's going to see an empty spot there. It can be very confusing for debugging. So, that's going to happen.

And you might ask, OK, OK, I recognize that nothing can run synchronously because then two robots can try to move onto the same tile and who knows what would happen. It could be like Pauli Exclusion Principle is violated, two electrons occupy the same orbital and the world comes to an end, a screeching halt, as it were. So, here's how we do it. We use the robot ID. Now this is going to come back later, so don't think that it's just an obscure little thing, this is important. So, you can see here that when I click on a robot it tells me-- Oh, is this bothering you? I can tell it's bothering half of you. You've got like a twitch developing, you're starting to shake, the sweats are coming out. No sorry, that's me.

So, the robot, you can see here, this is robot 38, the headquarters. This is robot 115, the lower the robot number, the sooner it executes, because they execute in that order. And I think when a robot dies it's number becomes available and another one can spawn into that number, in case you wanted to know. That's all too nitty gritty. So anyway, there we've got it. Let's start owning some noobs as it were. Let's destroy the enemy. It's not going to be as exciting if we're all the same person. So, let's make sure that we know who we are. Now, one problem here is that this only goes to rallyPoint. Let's put in an argument to soldierCode and change the name so it's more reasonable. So, that it's goToLocation. That's kind of nice. And we'll give it MapLocation whereToGo. Yeah, now we'll just put this in here and now this goToLocation function can be reused.

So first, we'll just goToLocation rallyPoint and then after a certain number of turns maybe then we'll just go to the enemy base. Because we'll have all these clumped up guys, we'll be ready to own some noobs. So let's do that. Let's go to the enemy base, let's kill them dead. So let's say here, how are we going to figure out what round it is? This is sort of an obscure function, Clock.getRoundNum. I don't think that there's really, is there anything else under clock that's worth looking at? Clock, get bytecode number, oh, you can get the number of bytecodes you have left, that's pretty useful. So you can say, oh, is it worth doing this computation, I don't know if I need that many extra digits of pi. So here, if my clock, which returns an integer, if that clock number is less than 500, yeah maybe, let's just make it 200, we don't have all day. Yeah, we'll make it 200. If it's less than 200, then go to the rallyPoint. If it is more than 200, or equal to 200, let's go to rc.senseEnemy-- Yeah, let's go to the enemy, let's just go ahead and kill them.

So now, we've got these two pieces in here. We're going to take our Awesome Robot player. We're going to play against Hardbot. Yeah, a bot that's supposed to be hard on my map tiny. Yeah. Cannot move and get-- Well, let's just hope that that's not going to be a problem. OK, so I'm getting ready, getting ready, get ready for the blitz. It's round, it's round 150, it's round 150, go, go, yeah.

AUDIENCE: [LAUGHTER]

PROFESSOR: Yes, knocking them out. Wa-bam. Wa-bam.

One thing that's important is you've got give yourself a little pat on the back from time to time. That's what teammates are for, to pat each other on the back. My brother always used to do that. He'd give me a giant five star, when you slap someone so hard on the back that it leaves a handprint. Yeah, but that let me know that he cared, that was important.

AUDIENCE: [LAUGHTER]

PROFESSOR: I think every younger brother needs that and if you're the oldest brother in your family then just hit the guy twice as hard. So yeah, you see here that that worked. I might just go ahead and send myself an email saying Max, you sly dog, you made a player that beats the hard bot. I mean, the hard bot's not actually very hard. All right, all right, so we've got that. That was good, that was good, but maybe what we'd rather do is be more hunter like about, we'll be stealthy. What I think will be really cool is to go after the closest enemy unit. Now, you'll find in the documentation a lot of ways of getting enemy data. You can sense where their encampments are and go after those, you can sense where your mines are, you can sense where its headquarters are, you can sense where the enemy robots are. And you'll probably find yourself, at some point, going through a list and trying to find the closest enemy, or maybe the closest allied unit.

So, we're going to do an example of that. And what we're going to use it for is-- See, for example, this case where they're trying to build this encampment and they're shooting at all of our dudes, and our dudes are just, they're getting hurt. They should just go gang up on that guy. Why aren't they ganging up on him? So, let's make a function for ganging up on the enemy. And it'll require that we locate the closest enemy. I mean, we could probably just go to the first one that's in the list.

So, I don't know which one should we do? In fact, how about this, we could, we could make code-- Yeah, let's try this. OK, this is interesting. I don't always plan, but when I do I drink-- No, nevermind, that doesn't work. So, let's see if you can copy a package and then we'll be able to play Awesome Robot player against Awesomer Robot player. Yeah, paste. OK, enter a new name. Oh, that was helpful. I mean, it's not very smart, AwesomeRobotplayer.copy? No, it's Awesomer. Yeah. Look at that, and it even changed it for me. I tell you, I'm going to be out of a job before the end of the week.

So, what we want to do now, in the remaining moments of lecture, is we want to find the closest enemy robot. Now, it may be that there are no enemy robots. This is the thing that I was talking about before. Let's do it, let's do it. OK, try. So here, I'm a soldier. OK, I don't need to do it if I'm a headquarters. So, let's just do it if I'm a soldier. We'll say MapLocation of the enemy robot because, after all, all we currently do is go to places, we go there and it's implied that we kill all the things that we find. Yes, so the map location of the closest enemy. Now, let's see here. This is only going to be valid when there are enemies. You've got to think about this. So let's instead do, let's get all the enemy robots. So, this is a robot list, a list of enemy robots, we'll call it enemyRobots, if we like. Doesn't matter what we call it. And we'll go rc.sense-- now this is the hardest thing to remember, this is the hardest ever possible thing to remember-- senseNearbyGameObjects.

AUDIENCE: [LAUGHTER]

PROFESSOR: It's hard to remember because it's so accurate. Robot.class, so we want robots, we want them anywhere. This is the range, the squared range that we're interested in picking up enemies. Now note, they won't show up unless they are within one of my allied units' sight range. I put in this huge number just because I'm willing to take those. I put in a million and then I'm going to make sure that they're enemy by doing rc.getTeam, so that's my team, and .opponent is a team method that gets me the opposing team. Now the two teams are Team A and Team B. So, that's literally written out as Team.A. So I could have written Team.A or Team.B, but then I would have had to check if I was Team A or Team B. So, you're better off doing it this way. So, I'm sensing the nearby game objects. Now, I need to know if that gave me any game objects. So, I don't know how big this list is. If the list has a length, if enemyRobots.length = 0, well, then there are no enemy robots. So, I should just do the same thing I was doing before. So, there you go.

Now, we've got code that should be working. But, if there are robots-- Now, what have I done here? I haven't commented a single line of code. OK, enemies, no enemies nearby. Yeah. I heard somebody tell a real zinger over there. No, I'm just kidding. So, no enemies-- OK, someone spotted. OK, enemy is spotted. So, yeah, no that's not right, I put this in the wrong spot. You guys are not helping me, not helping me nearly as much as I was expecting.

So, now what I'll do is I'll loop through each of the robots. Now, I could do the same looping thing that I did before, where I could say, Robot arobot among enemyRobots. And then, it would put arobot into each one, but instead I'm going to do it a different way just to demonstrate it. I'm going to define an integer i = 0. And then, there you go. Oh, thank you, I saw you we're going to help me and I really appreciate it. And we'll make sure that i-- And so, this is the format for a loop that goes with this type.

Some of you are going to be like that's not the way that the words work that make it explained. And I'll tell you that you are right and you get a gold medal, as soon as we mint them. So yeah, enemyRobots.length and we'll do i++. So this'll make, i will go up by one every turn. So now, I can type Robot arobot, it's an enemy robot, = enemyRobots at i.

And now, I'm going to look through and find the closest one. Now, I'm sure somebody's going to show me a way that I'm doing it wrong, but here's how I'm going to do it. I'm going to keep track of which one was closest and how close it was. So int closestDist, and I'm going to initialize it at a large number. And I'm going to make it a location. And I'll do MapLocation closestEnemy. Yeah, closestEnemy. OK, so now, we'll have-- Now this is very tricky. This little bit of sequence is very tricky in the Battlecode API. When I first did Battlecode, they didn't give an example of this and it took me 10 years to figure it out. This is how to find out where an enemy is. You type rc, that's like you're starting from you, senseRobotInfo. All right, and then you pass it the robot. You tell it to sense the robot info.

Now what does that get you? Does that get you the location? No, that would be too easy. , So, that gets you a robot info object. So, now you've got RobotInfo arobotInfo = that. So now, I've got some robot info. Robot Info is yet another one of these classes. So here in the Java docs. Look at this, I keep opening more and more of them. That's another one, there's another one, they're multiplying.

AUDIENCE: [LAUGHTER]

PROFESSOR: So here, you could see Robot Info, there it was. So, you can see that information about the enemy robots can be things like the amount of shields they have, the type of robot, its location. One of the Robot Info things is Robot. So, what you can do is you can sense the robot, you can get the robot, sense its info, and then do info.Robot, and then sense that things info, and then you'll have the info. So, you go back to the robot and it'll be a really nice little loop. It's wonderful, it's just, [KISSING SOUND] mmm, just great. I highly recommend that kind of code. It just warms my heart.

So, like what a little kid does, is that computationally impressive? No, but it's cute, so it warms it anyway. So, that was the reasoning behind that crazy stream of logic. So, here we go, we got Robot Info. Let's check the distance, we'll be done with it. So we'll say that the distance, which is an integer, equals arobotInfo.location.distanceSquaredTo. Of course, you can write your own distanceSquaredTo metric or something or other, but we're just going to do it this way. And we're going to make it the distance from, I guess, it could be myself, as the robot. So, distanceSquaredTo, hmm, it's sort of hard to decide. It could be the one that's closest to me, or let's just do me for the moment.

Maybe later we'll make it more complicated. Actually, we're going to run out of time and we're just going to end up eating Indian food. But that's wonderful. How many of you have eaten at India Samraat? That place is fantastic. Wow, oh my goodness, you're in for such a treat. I don't know if it's going to be considered the same level of treat as-- Oh anway, I'm just going on and on. So if the distance that we're currently looking at is less than the closestDist, then we'll say that is the new closestDist, and we will assign closestEnemy=. Well, I didn't record it anywhere. It's this one. Yes, so there it is. That's the closest enemy.

So, once again, if there were no enemies nearby we would either go to the rallyPoint or goToLocation. But now, what we want to do-- Oh, goodness, hey quit sending me things on the-- Anyway. So now, what we'll do is we'll just go straight to the enemy. That's what we'll do. If we see one, we'll go kill them. So now, what we'll do is we'll pit this code against the previous version of this code and we'll see which one wins. It's an interesting question. Which code is better, to go straight for the enemy base, or to go for the closest enemy robot? Because maybe on their way to the base, I mean everybody shoots on the way to everywhere, but what's the best thing? I'm think it'll be quite interesting.

So let's see, where do we shoot? It's here. So, now we can go to the location. And we can go to the closest enemy. Yeah, because there's guaranteed to be an enemy to go to. Oh bother. It's another one of these things. No, wait that's there, goToLocation, closestEnemy. Oh, right I've got to initialize it. Yes, so I put null. It'll never see null, but I've got to put it there anyway so that it's comforted. I believe that's the technical explanation for what's going on in the background of Eclipse. I'm pretty sure.

So now let's run. Oh man, this is getting exciting, it's 5:55. I guess I'll have some closing remarks. But this'll be the thing that is supposed to be exciting and amazing. We'll have awesome versus awesomer. All right, these guys are so, they have a lot of fun. That's so important. I didn't say that again. All right so, raise your hand if you think that Awesome Robot player, the original awesome is going to win. Three hands. How many people think that Awesomer Robot is going to win? Four hands. OK, well, I recognize that premature arthritis can be caused by coding. Raising an arm may be beyond some of us, in terms of, oh, ohh. OK, we're going to find out. OK, I actually don't know the answer. OK, so they're just grouping. It's round 70, round 80, it's round 100. At round 200, exciting stuff's going to happen. I wish we could slow this down. It's going to happen all at once. Boom. Ah. Oh, but they stopped them. They stopped them. The Awesomer player has won, hurray.

Did you see that? It was all too fast. It just went by before, I mean before I even noticed it was, ah goodness gracious. So, here it goes. These guys will run back to the base, but these guys will be following them the whole way through. OK, so let's back it up a little bit. This is interesting. Now, I think if you push like f, or b, or c, you can turn off the circles, maybe not. OK, so they're forming a concave. Oh man, a lot of them are blowing up. It's hard to tell what's going on. OK, so here it looks pretty darn even, 1,2,3, 4,5,6, 7,8,9 versus 1, I can't tell if he's dying, 2,3,4, 5,6,7, 8,9,10.

OK, so it's 10 versus 9. The blue guys have done well, pretty well already. Oh, but then they just ping, they just ping and turn around and kill everything. I've got to step forward more slowly. This is a fast-paced game, let me tell you. Let me tell you, you're not going to be here for an hour watching somebody--

Oh man. OK, so these guys are all working over there. Oh, look at that. Meanwhile, these guys are ganging up on him and these three guys are getting totally double-teamed, or quadruple-teamed, or whatever the word is, by these fellows. Oh, I'm pointing with my finger, how can you see that?

AUDIENCE: [LAUGHTER]

PROFESSOR: Oh, my goodness. Look I'm enjoying myself OK. Yeah, and then they turn around and they kill the enemy, and then they turn around and they kill the enemy base. There you have it, boom, done.

All right so, final closing remarks. I'm looking at the wrong page. Yeah, so improvements to this robot, there's lots of things that you could do, that you would do, that are worth doing. For example, everybody's making decisions independently, you could be using messaging. I think this is a big point because somebody asked me the other day, what is the point of messaging. Because all of my robots have access to all of my other robots' data. And you saw that happen here. Because would these guys have been able to respond intelligently? Would this guy have come-- Gosh, I'm using my finger again-- would these fellows be able to come all the way back? I agree Steven, I should close GChat when presenting. Would these guys have been able to come all the way back to here if they didn't have shared data? No, they wouldn't. So, they have shared data, why would you ever use messaging? Because you could simplify it by having your headquarters do the processing, and say, all right, I want everybody to attack this guy. And then they all just go ahead and follow that command, rather than having to do the processing themselves. Because if everybody has the same data, why should everybody process on the same data?

So, there you go that is a reason, it doesn't prove that it's always better, because it may be much better to have a decentralized approach, and so on. There are very many reasons, but there you go, that's an answer to that question. You saw Awesome, you saw Awesomer, and you saw it right here. Thank you for coming. The next-- Just remain calm in your seats and there will be food. The food guy probably arrive already, I'm sorry, yesterday I thought the food guy hadn't arrived. They don't mind being called food guys, don't worry it's not insulting. I thought he hadn't arrived, but I think he was poking is head in and out like a drinking bird, and I just didn't see it because of various myopia and such, no, what do they call it, tunnel vision.

So, yes, the next lecture. I'm sure you enjoyed this one a lot, even if you know Java already. Because you could see the warmth and the smiles on the faces of the people who didn't know Java and now do because of this amazing whirlwind tour. So, yes, the next lecture will be on navigation, as if this one wasn't. Yeah, and you're going to hear all kinds of amazing things that you never would've expected. And you're going to see this like awesome visualizer that's a little bit like the fractals that I was showing before. It's made in Mathematica and it's made in the USA. And yeah, hurray. So, thank you for coming. And this is the end of lecture. And I'll just be here answering questions. Again, there's lab after lecture. My god, this is like one of those symphonies that you think keeps ending, but it doesn't end. And that will be all. Thank you.

AUDIENCE: [APPLAUSE]