.slide 1

Hello, everyone. My name is Peter Guba, I'm a Ph.D. student here at the faculty and I'm going to be teaching a part of this course, as David hopefully explained to you during the introductory lecture. Today's lab is titled "Game AI Design Crash Course 1" – the first in a series of labs which are going to be sprinkled in between the more regular ones and where we are going to be taking a look at some interesting problems that designers faced when coming up with AIs for various games. And, since this is a lab, you are also going to get a chance to think about these problems yourselves and discuss your ideas. I therefore hope that you come reasonably well-rested and eager to talk about games, otherwise, this will be a very painful experience for everyone involved.

First, let me explain the motivation behind these labs. During the lectures and most of the labs, you are going to be focusing on various algorithms from the field of game AI. And these are very useful tools to have in your toolkit, but there are very few design problems where you can just go 'yeah, I'll just slap on this algorithm' and that magically takes care of everything. When designing AI – or anything, really – you have to be able to think about problems and come up with novel solutions that may be completely unrelated to any algorithms you already know. And these labs are one of the aspects of this course through which I hope to give you a chance to hone this skill, at least a little.

The games I prepared for these labs all contain some AI problem that I found interesting and that is different from the stuff you'll learn during the rest of the course. So, this won't be about applying specific algorithms – although, if you see a good opportunity to apply an algorithm you know, don't hesitate to do so – but about coming up with your own solutions. It also means that there isn't any overarching theme, any connection between the presented examples, it's just a collection of – hopefully interesting – problems to think about.

.slide 2

One more thing before we get into our first example; let me just briefly mention the resources I used when coming up with this lab as well as other parts of this course, as they may be useful to you. Game AI Pro is a website that contains free articles on various AI topics from professionals, including explanations and analyses of AI techniques used in various games written by people who actually worked on them. AI and Games is a YouTube channel that has a ton of videos on how AI in various games works. These videos usually aren't too in-depth, but they do give you the general idea. And, last but not least, there's another YouTube channel called Game Maker's Toolkit, which has videos on various game-related topics, not just AI. I would specifically recommend you watch their video "What Makes Good AI?", which I've linked here.

.slide 3

I don't think it contains any groundbreaking insights, but it names a lot of aspects of AI that aren't immediately obvious.

Ok, with that out of the way, let's jump into the first game, which is one you should all familiar with by now –

.slide 4

Pac-man. Classic game – you play for a yellow dot with a mouth and try to collect smaller dots while avoiding four ghosts. Totally relatable stuff, we've all done it at one point. Now, what interests us here are the aforementioned ghosts – how are their movements programmed?

To keep things simple, let's forget about the fact that one can make the ghosts edible by eating one of the larger dots, in which case their movements change. With that assumption, the problem we're facing here is as follows.

.slide 5

We have a 2D map with obstacles, where the player tries to collect all the little white dots. We have four ghosts, all of which can move across the map but not through obstacles. If they touch the player's avatar, the player loses a life. Their purpose is to provide a challenge. How should these ghosts move?

I said in the beginning that I would give you some time to think about this problem and try to come up with your own solution – well, that time is now.

Before we get into the discussion, let me just mention that this is going to be the general format of these exercises – I show you a short description of a game AI problem, we discuss it for a while and then I show you the original solution. If at any point you find something unclear or would like more info on something, don't hesitate to ask. I can't guarantee that I'll know the answer since I haven't played most of these games, but this is just a thought experiment, so we can just make stuff up.

So, let's discuss this – does anyone have any ideas?

…

Alright, let's take a look at the original solution now.

.slide 6

The most straightforward approach would be to just have all the four ghosts chase Pac-man all the time. However, that would probably mean that they would either follow Pac-man in a line, which would be pointless, or come at him from multiple sides, which would potentially be impossible to escape. So, the designers had to come up with something else.

The ghosts have two movement states – chase and scatter. They alternate between these with 20 seconds of chase followed by 7 seconds of scatter.

.slide 7

In scatter mode, they move towards the four corners of the map, with each ghost being assigned a specific corner, while in chase mode, they try to move towards different target tiles. Each of the four ghosts – who have names by the way, but I'm going to omit them and just refer to them using their colours – has different rules for picking their target tile.

.slide 8

The red ghost simply chases after Pac-man, so its target tile is whichever tile Pac-man is on at the moment.

.slide 9

The pink ghost targets the tile that is four tiles in front of Pac-man (in front relative to the direction he is facing). There is an exception to this – when Pac-man is facing up, the target tile is four tiles up and four tiles to the left. However, as this is due to overflow of values during the computations, I suspect this was not the original intent.

.slide 10

The cyan ghost computes the tile that is one tile in front of Pac-man – or, again one tile up and one to the left, if Pac-man is facing up, as is the case in this picutre – then computes the vector from that position to the position of the red ghost and rotates it by 180° around the first tile. This results in this ghost sort of working in tandem with the red ghost to catch Pac-man from two sides.

.slide 11

Finally, the orange ghost has the same target tile as the red ghost – Pac-man's current position – however, only if Pac-man is eight or more tiles away. If the two are closer-

.slide 12

Its target tile is in the lower left corner of the map.

There are several things about all this that I find interesting. First of all, it seems like such a simple problem at first glance, but the solution is in fact quite nuanced. Second, there's nothing about this that would make you think 'oh yeah, that's the obvious right solution'. That's one of the lessons that I hope to impart through this lab and the group project that you'll be assigned later on, because, at matfyz, we are all very focused on solving problems in the one correct way, right? You get a math problem and you have to come up with just the correct steps to solve it, or you get a programming assignment and you hack away at it until you have a program that works exactly as intended in every case. But there is no such thing when it comes to AI design – or design in general, I suppose. And third, all the rules I've just described just seem so random, yet they come together to create such a good player experience. I wonder if they went through a lot of iterations when designing this and whether there was a lot of thought put into designing these specific rules, or whether they just tried random stuff until something worked.

Ok, so, that's Pac-man. Moving on, the next game on today's menu is called –

.slide 13

The zombie apocalypse cooperative FPS Left 4 Dead. In this game, you play as one of four survivors of a zombie apocalypse who are trying to survive a little longer. As you try to make it from one safe place to the next, you are assaulted by hordes of zombies of the totally mindless variety.

This game obviously has quite a few AI agents – the zombies, for one thing, although they probably aren't among the most complex creations, but potentially also the other survivors, depending on whether you are playing with friends or not. However, what we are going to be talking about today is something else – its director AI.

So, the problem is as follows:

.slide 14

You have a first-person shooter. The enemies are mindless zombies that have to come up to a player to attack them, plus some special enemies with various abilities. The players' goal is to get from one place to another. The task is – spawn the enemies in such a way as to make this challenging, while also making sure to not overwhelm the players and give them time to breathe. How would you approach this problem? Any ideas?

…

Alright, if there are no more ideas, let's take a look at the original solution.

.slide 15

The authors created a unique agent, called the director, which is in charge of spawning and despawning enemies. It works in four phases – Build up, Sustain Peak, Peak Fade and Relax.

.slide 16

In the first phase, the director spawns a population of zombies to attack you. Then, it observes the players and waits until the fighting reaches sufficient intensity.

.slide 17

Once that happens, it sustains the population of enemies for 3 to 5 seconds – this is the Sustain Peak phase. Then

.slide 18

The director goes back to spawning a minimum of enemies and waits until all the players have had a chance to calm down – that's the Peak Fade phase –

.slide 19

– followed by 30 to 45 seconds of spawning only a minimum number of enemies in the Relax phase (unless the players get to the next safe room before then, in which case the Build Up phase resumes).

Now, the critical thing here is – how does the AI know when the fighting is intense and when it isn't?

.slide 20

To ascertain this, the authors created a couple of hand-crafted metrics which, together, are meant to give an idea of the current emotional intensity for a player. These are things such as the distance at which enemies are being killed, damage taken, whether the player has been incapacitated and a special case is when one of the special enemies manages to attack a player and pin them down – in that case, their "emotional intensity score" will reach maximum. This score is tracked for every player and once the maximum is reached for any one of them, the Build Up phase finishes. Then, the intensity is sustained and then, when the director transitions into the Peak Fade phase, it starts decreasing the players' scores until they get sufficiently low – if it jumped straight to the Relax phase, then that would start while there still were zombies around to fight, which wouldn't be all that relaxing. So, once the scores for all players are sufficiently low, the Relax phase starts and then we're back to Build Up. And that's the gist.

What I like about this and why I thought it would be a good idea to include this in this lab is that I find this to be very far removed from what we normally think of when we think of game AI, right? The closest you'll probably come to discussing something like this during this course will be during the lecture on the AI of F.E.A.R., where a squad coordinator assigns goals to individual agents. So that's just to point out that this is a whole different branch of game AI and interesting in its own right.

Alright, moving on the next item on today's menu is another zombie game –

.slide 21

Days Gone – a post-apocalyptic, open-world RPG. Although it does feature hordes of zombies, we'll again omit those and instead take a look at its human enemies.

.slide 22

These enemies are split into factions which, while they don't like the player, generally don't have to like one another either, so the player can come across squads of NPCs fighting one another – and can, of course, join in on the fight. And the question is – what should the squad behaviour of the NPCs look like and how should it be implemented?

You may notice that the problem statement here is quite a bit more vague than the other two and therefore also larger in scope – that is intentional as I'd like to spend more time discussing this problem than the other two. So, we are trying to implement realistic combat between groups of NPCs and between NPCs and the player – you could of course just root the NPCs in place and have them try and shoot at one another, but that wouldn't be very entertaining.

To start with, let's ignore the player and just focus on fights between NPCs. So, picture this, you have two groups of NPCs. One is here and let's say there's a road and the other group is coming this way. At some point, the NPCs are going to spot one another. What should happen next?

…

Alright, I think we've discussed this enough, now let me show you how the designers handled this problem.

.slide 23

At the start of our discussion, we talked about whether or not to group the NPCs into squads and came to the conclusion that it would probably be a good idea – the authors of Days Gone felt the same. These squads are created dynamically based on proximity – the rule is that every member of a squad has to be within a given distance of at least one other member of that squad.

You can see that illustrated here – in picture a, all units are within the desired distance of one another, so they form a squad; in picture b, they units form a sort of chain, with the two furthest one being out of each other's desired distance, but they still form a single squad and in picture c, they form two squads. These squads are adjusted dynamically, so if two groups of units get too far apart, they split into separate squads and vice versa – they get too close, they join up.

Now, there are two important concepts that govern the behaviour of each squad. The first is-

.slide 24

A confidence value, which is computed using the formula you can see here. It may seem complicated at first glance, but it really isn't. F stands for friend, E for enemy, K for kills, L for losses, s for strength and c for confidence. x is the NPC for whom we are computing the confidence value. As you can see, this formula tries to approximate the mental computations going on in the human mind when in battle. If you are mowing down your enemy, then it makes sense that you'll be confident you can win and will be likely to push forward. If, on the other hand, the enemy is tearing you to bits then you probably won't be very confident and will want to retreat.

This value is computed for every member of the squad and is then averaged to give the confidence of the entire squad and this then determines the squad-level behaviour.

Here is a quote from the authors about the effect that this formula has on their behaviour.

.slide 25

"(The formula) has a self-reinforcing effect allowing a few confident characters to rally their Squad, while a few characters losing Confidence has the opposite effect, causing their allies to lose Confidence as well. Conversely, seeing your enemy falter will boost your own Confidence, while enemies that are confident in their imminent victory will make you think twice."

So, as I said, this is meant to approximate what goes on in the human mind when in battle. Minus all the PTSD.

It's also important to note that-

. <- slide 24

the weight of the kills and losses goes down over time, so that the squad can recover.

.slide 25

.slide 26

The second core concept is the frontline. This is a game object that partitions the battle area into three regions. The region controlled by our squad, a neutral area that usually has a fixed width, though there are exceptions, and the area controlled by the enemy. These position and direction of the frontline are computed based on the so-called "centres of gravity" of the squads, which just means the average positions of all their members, and are used to determine the positions and movements of squad members.

.slide 27

.slide 28

The area controlled by our squad is further divided into lanes, one for each squad member. The members then all get assigned lanes and try to find good positions within them to both take cover and attack the enemy.

.slide 29

Combining these two concepts, we can produce sensible behaviours for the squads – if the squad members are confident, they will try to push forward, if they aren't they will retreat. And these manoeuvres are performed in waves. So, if the squad is retreating, first, the members closest to the frontline move to the back while the rest provide cover fire. Then the members that are now closest move, etc.

.slide 30

Regarding the special behaviours we discussed, such as flanking, these are also governed by the squad-level AI. The NPCs themselves are implemented using behaviour trees – or I at least assume they are, though it wasn't explicitly mentioned in the article, but it makes the most sense to me – and the squad manager can plug specific behaviours into their trees.

An interesting design choice to note here is that the AI actually lets itself be flanked by its opponent in order for the opponent to appear more competent and dangerous to the player.

.slide 31

We also discussed the case when some NPCs are supposed to guard a certain area. Under this system, this can be easily achieved by manipulating the confidence value – when a unit strays too far, its confidence begins to drop, until it retreats.

The frontline also needs to be adjusted to work with this – the NPCs should naturally want to try to keep the area they are protecting at their backs. It would be weird if the frontline intersected it, for example. To achieve that, the authors put a limit on the degree to which the frontline for the protecting squad – remember each squad computes its own frontline – can rotate. The closer the squad is to the location they are guarding, the stricter that limit becomes.

.slide 32

Finally, we are getting back to the player. They can join in on the fight as well, of course, and they also affect the frontline. In the specific case when the player is the only entity that a squad is fighting, the player's position is used as the "centre of gravity" based on which the frontline is constructed. However, if the frontline kept changing every time the player moved, it would change constantly, which could lead to some really chaotic behaviour. So, instead, the frontline only changes when the player slows down or stops altogether. This is because human players usually can't shoot very accurately when running around, so they have to slow down or stop at some point.

Alright, that's all I have for Days Gone. The source article also goes into a bunch of special cases and ways the system had to be adjusted to work with other aspects of the game. For example, they mention that snipers couldn't be included in squads, since they are stationary and can be quite far away, so that would mess with the formation of the frontline, or that, when an NPC starts panicking, their strength actually gets added to the enemy's in the confidence formula, making panic highly contagious. Also, they mention how, when flanking-

. <- slide 30

They had to treat the flanking units as if they hadn't moved, creating these ghost positions, in order to not disrupt the frontline. And a lot of other interesting nuances. I think it's a probably a good

introduction to what developing such a system looks like, so feel free to give it a read if you're interested in this sort of thing.

.slide 31, 32

.slide 33

And that's it for Days Gone and for today's lab. There will be another one of these in two weeks – see you then.