.slide 1

Hello everyone, welcome to another AI and games lab. Today, we are going to have another game AI design crash course. If you weren't here the last time, the way this works is that I show you some AI problems encountered during the production of actual games, you will then have some time to think about them and present your ideas and then I will tell how the designers went about solving them.

And the first game we'll be looking at today is-

.slide 2

Hitman Absolution – an open-world RPG where you play as a hitman tasked with getting rid of some bad people. And this game is quite a marvel of engineering, because-

.slide 3

It contains very large crowds of NPCs, potentially more than a 1000 of them at one time. Actually, I'm not sure whether this number is from Hitman Absolution or some subsequent release, but there definitely is a Hitman game where you can have that many NPCs in a crowd. And these crowd NPCs have to walk around, of course, as long as nothing's happening, but they also have to react to some of the player's actions. For example, let's say that the player pulls out a gun in public, the bystanders should definitely show some reaction to that. And, at times, they also need to exhibit some more complex behaviour. For example, if the player picks a fight with one of them, they should be able to fight back. And the question is, how to simulate all this without melting your computer. So, any ideas, thoughts, observations?

Questions:

Forget about the player and suppose we have a crowd of NPCs moving around – how to make this efficient?

How to implement panic or other simple reactions?

How to implement fighting or other complex reactions?

Ok, let's take a look at how the original authors achieved this functionality.

.slide 4

An obvious bottleneck when it comes to simulating thousands of NPCs milling about is pathfinding. If all your NPCs keep generating pathfinding queries all the time, you're going to have a bad time. So, to help with this, the designers created a 2D grid which they overlayed on top of the navmesh to help with this process. Each cell of that grid has a flag that indicates whether it is walkable or not and also various annotations, such as whether it belongs to an exclusion zone, which is a zone that only some NPCs can enter. It also keeps track of all the agents currently within its borders.

Now, the resources I consulted for this didn't go into detail about how this actually helped with the pathfinding process. I assume that, with this grid in place, the NPCs can just find paths across cells

marked as walkable instead of querying the navmesh and can also coordinate with all the other agents present in those cells to avoid crashing into one another.

Also, it can maybe help with finding good locations. For example, if an NPC wants to get to a hot dog stand, then they can find a cell that's closest to it and head there.

.slide 5

The navigation process entails a lot of comparisons of NPC attributes, so the designers used some neat memory tricks to make it more efficient. One thing they did was, they split the NPCs' attributes into core attributes and all the rest and put all of the core attributes of all the NPCs into one chunk of memory. This allowed for nice memory alignment and also resulted in less cache misses, making the process more efficient both in terms of time and memory.

Another trick they used was that, instead of having all the cells store their attributes separately, they created arrays, one for each attribute, and had the attributes for a given cell be stored on the same index in all the arrays. So, if you wanted to fetch an attribute for, say, cell number 4, you would get the array in which the values for that attribute are stored and look at the value at index 4.

.slide 6

The actual AI of the NPCs is a finite state machine. When nothing much is happening, they have 3 navigation states – Idle, Pending walk and Walk. In reaction to the player's actions, they can also enter other states such as Alert, Dead, Prone or Panicked – more on that on the next slide – and every NPC's decides whether it should change its state or not at every game frame.

.slide 7

Now, how does the player influencing the NPCs actually work? When the player does something to which the NPCs should react, behaviour zones are created around the player which notify all the agents within them that something is going on.

For example, when the player pulls out a gun, three behaviour zones are created, as you can see on the slide. The NPCs that are in the red and blue zones go prone – which means that they will get on their knees and cower before the player. The NPCs in the green zone will go alert, so they will freeze and stare at the player.

And I imagine that, if the player starts shooting, that's when the characters will panic and start trying to flee.

.slide 8

Processing 1000 navigation queries at ones would be a problem, of course, as well as quite redundant, since all the NPCs will be heading to just a couple of exits. Instead, the NPCs try to follow directions stored in the cells of that 2D grid we talked about, which lead to the nearest exit – these are called panic flows and they are precomputed for every cell and every exit.

As I mentioned at the beginning, the NPCs sometimes need to exhibit more complex behaviour. However, running a complex behaviour tree for every NPC probably isn't feasible when you have hundreds of them. So, instead, a number of these behaviour trees are instantiated at the start of the

game and, depending on the circumstances, any NPC can become temporarily “possessed” by one of them, effectively being upgraded into a more complex AI agent.

And that’s it for Hitman. Game number 2 on today’s menu is-

.slide 9

The Last of Us. Yet another post-apocalyptic zombie-themed RPG, this time taking place in a world where a fungus has infected much of the population. The player must traverse the ruins of cities while defending themselves against both zombies and humans. They are accompanied on this journey by an AI companion named Ellie – or rather, storywise, they are accompanying her – and she is who we’re going to be taking a look at today.

Ellie can do quite a few things, such as actively helping in combat and gifting the player ammo or health packs. We’re not going to be taking a look at all of that, instead, we are going to focus on arguably her most foundational behaviour. So, here is the problem specification.

.slide 10

You have an open-world RPG. The player has to use stealth to avoid and ambush enemies. Ellie, the AI companion, constantly follows them around. The questions are – what following behaviour should the companion exhibit? How should the companion behave when the player is in stealth mode? And how should these behaviours be implemented?

Questions:

How should following the player when nothing much is happening be implemented in order for the behaviour to look natural? (We’re interested in how the AI picks its next position, not in how it pathfinds.)

How should it be implemented when the player is trying to be stealthy?

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

AI companions can be quite a bother – from walking at half the player’s speed to actively getting in the player’s way, there’s a lot that can go wrong when implementing them. The developers of The Last of Us were aware of this and were determined to create a believable character that the players would genuinely care about – which, by the way, led them to scrapping their entire system 5 months before shipping the game and starting from scratch.

I would like to start by sharing two quotes from one of the authors.

.slide 11

“Characters give the illusion of intelligence when they are placed in well thought-out setups, are responsive to the player, play convincing animations and sounds, and behave in interesting ways. Yet all of this is easily undermined when they mindlessly run into walls or do any of the endless variety of things

that plague AI characters. Not only does eliminating these glitches provide a more polished experience, but it is amazing how much intelligence is attributed to characters that simply don't do stupid things."

.slide 12

"As a general rule, characters don't need complex high-level decision-making logic in order to be believable and compelling and to give the illusion of intelligence. What they need is to appear grounded by reacting to and interacting with the world around them in believable ways."

In accordance with this line of thinking, the people behind The Last of Us put great emphasis on perfecting all the little behaviours of their AI agents, and especially Ellie, so that they would come off as believable. The most basic of these behaviours, for Ellie, was following the player.

.slide 13

To ensure that she was always in a reasonable position, the authors started by defining a follow region around the player. Then, they did three steps of raycasting. In the first step, they cast rays from the player to the follow region to make sure there was a clear line of movement there. Each of these rays generated a candidate position, if it reached the follow region. Then, from each of these positions, they cast a ray forward to make sure that the character wasn't going to walk into a wall. This generated a set of forward positions. Finally, from each of *these* positions, the last set of rays was cast back to the player to make sure that Ellie wouldn't wind up putting an obstacle between herself and the player. Without this, she could walk behind fences or the other sides of doors.

So, that takes care of normal walking. Now, what about stealth mode?

.slide 14

To find a good cover position for Ellie, the authors started by reusing a tool they were already using for the human enemies, who also take cover when fighting with the player. This tool analysed the collision mesh and pre-calculated potential cover spots. This was good enough for enemies, but not for Ellie. The authors therefore added a system where rays would be cast from the player's position in a circle and then an analysis would be performed on their intersections with the scene to find more potential covers. Specifically, each ray generated an intersection point and the normals and locations of the points were then compared and nearby points with a similar orientation were combined into a single cover.

The two sets of potential cover places were combined and compared using various metrics to pick the best one. These metrics included current visibility to enemies, predicted future visibility, and proximity to the player.

So that's basically their solution to the problem that I outlined. However, this doesn't really do the source article justice, because it's filled with these little insights about what the designers did to make Ellie's character feel more human. For example, when implementing Ellie's follow behaviour, they had to decide what she should do when the player would try to walk over the place where she was standing. Their first thought was to just have her move out of the way so as to not impede the player's movement. However, they decided that this seemed unnatural and, in playtests, they found that players usually didn't try to run into her if she was standing somewhere. They therefore decided that she wouldn't get out of the way until the last second and then would reprimand the player for their behaviour.

Or, another decision they made was to almost never have Ellie cheat in any way, such as teleporting or dealing more or less damage than the player, as that would make her feel less natural and break the immersion.

There are a few exceptions to this though and one of them is what happens when the player is trying to be stealthy and the enemies catch sight of Ellie. This is a rare occurrence, thanks to the system outlined here, but it does happen once in a while and the designers had to make a decision about what should happen in such a case. At first, they tried to stick to their philosophy and have the enemies react by attacking, but they found this to be just frustrating for the players, so, instead, they decided to make Ellie invisible for the enemies in such circumstances.

There are many more of these nuggets of insight into other behaviours in the article about Ellie and then also the articles about human enemies and infected enemies, so I definitely recommend you check them out if you're interested in these sorts of things. Moving on, we have one more game to look at today and that is-

.slide 15

Sims 3.

.slide 16

This is a social simulation game where the player gets to create a bunch of custom characters and then navigate them through their lives.

This means that the Sims – as the NPCs are called – have to respond to the player's commands, but also be able to act autonomously. It also means that, since this is supposed to be a simulation of human society, the Sims need to exhibit a wide range of different behaviours and personalities – it is therefore out of the question to just control the Sims using some simple scripts. So, the question is, how to implement all this? How to make sure that the Sims are varied and believable, that they can behave autonomously when the player isn't controlling them *and* that they act in line with the player's directions, not just momentarily but throughout the game – for example, it would be quite odd if the player told some Sims to get married and they immediately broke up. So, basically, how to simulate a human society, that is the question. That, obviously, is quite a complex task, so let's try to break it down into smaller pieces.

Questions:

What needs should a Sim have?

How should a single Sim decide how to satisfy them without the player's control?

How to make sure they aren't completely deterministic and therefore robotic and predictable?

How to make sure that the Sim assigns a higher importance to some needs, such as hunger, compared to boredom, for example?

How to make it easily extendable?

How to add personality to the Sims?

How to construct believable social scenarios using the system we've discussed thus far?

How to make sure the sure the system isn't too computationally intensive?

How should the player's input be taken into account?

Ok, if there are no more ideas, let me tell you what the designers did.

.slide 17

First of all, the Sims all have needs, or motives, as the designers called them – I'm going to call them needs because I think that's a more fitting term. These are modelled on human needs. So, they're things like 'huger' – that's pretty self-explanatory – social – that corresponds to our need for socializing – bladder – that corresponds to our need to urinate – etc.

All of these needs have metres associated with them which are constantly ticking down. They can be refilled by interacting with the world in various ways. However, the Sims themselves don't actually possess the knowledge about how to do that – instead, all the know-how is stored in the objects themselves. These advertise themselves to the Sims, providing information about which needs they can satisfy and by how much.

So, when deciding what to do, a Sim queries its surroundings – this usually means the house that it's in – and the objects, which includes other Sims, all tell it which needs they can satisfy. The Sim then scores these possible actions based on the current state of its needs and picks one. It doesn't always pick the best one, instead, it takes the top 3, I think, and constructs a probability distribution for them based on their scores and picks at random using that distribution. That makes the Sims less robotic and predictable and also means that they won't be perfect at satisfying their needs, which gives the player something to do.

Now, it would be pretty weird if the Sim was both starving to death and extremely bored and they chose to go to play a game instead of getting something to eat – I'm aware that may be relatable to some of you, but it isn't regular human behaviour. To prevent this, the needs have different weights associated with them.

.slide 18

These weights are dictated by designer-defined curves. You can see some examples here. So, if you take a look at the hunger curve, this means that when the hunger metre is full, meaning that the Sim is satiated, the weight for this need is very low. In fact, I'd expect it to be zero, but it's hard to tell from this graph. And, as the Sim gets hungrier, the associated weight grows exponentially.

.slide 19

Other needs, like social and fun, have more of a u-shaped curve associated with them. This is once again meant to simulate human behaviour. If you're out with friends and having fun, it doesn't happen that you suddenly go 'my fun metre is satiated, time to quit'. Analogously, this means that, if a Sim is having

fun, it will be motivated to keep having fun. You can also notice however, that these curves never go as high as the hunger curve.

.slide 18

.slide 19

So, if the Sim is really hungry, that need will always override these.

And, besides weights computed from these needs, other weights can also be taken into account when computing scores, for example, ones based on distance or the Sims' personalities – we'll get into those in a sec.

.slide 20

Now, the Sims aren't the only entities in the game that have needs, the town also has them and some venues do too. The town's needs are things like employment rate and gender balance. It can fulfil these needs by generating life events for the Sims. So, some Sims get a job, some Sims die, etc.

As for venues, such as restaurants for example, their needs are things like 'there should be X number of people here at such and such time of day'. To fulfil these, they can give nearby Sims the need to, for example, eat outside, in the case of the restaurant. This strategy of giving Sims additional needs to alter their behaviour can actually be pretty useful for various things, as we'll see.

.slide 21

Besides needs, Sims also have their own personalities. In the first two games, this were defined using only a couple of traits that were common for every Sim, just dialled up or down. However, for Sims 3, the designers really upped their game and introduced a new system of character traits. There are around 60 of these in the base game, I think, and each Sim can have up to five, creating loads of possibilities.

These can do a few things. First of all, remember how I said that objects advertise to the Sims what needs they can help satisfy? Well, these advertisements include a score – that is, the number by which the associated need's metre would be increased. And these personality traits can apply multipliers to these scores, which can create Sims with different preferences for leisure activities or food.

Besides that, the traits can also unlock specific actions for the Sim, give them additional needs and also adjust their animations, so the way they walk, idle, etc.

Now, the strategy of adding needs can also be used for adjusting a Sim's behaviour in specific situations, especially social ones. For example, when visiting someone's house, a Sim may be given additional needs in order to be incentivised to act in a socially appropriate way.

.slide 22

However, social situations are quite varied and complex and this just isn't enough to produce halfway believable results. To remedy this, the designers introduced so-called production rules. These are rules that have some conditions on one side and the outcome of an interaction on the other. For example, they can be used to define an outcome of an interaction when a Sim is telling a joke to another Sim that is angry with them, or that likes them, or that's already heard that Sim tell the joke ten times. For every

interaction, these rules are sorted by specificity and then the most specific one is applied. Supposedly, there are thousands of these handcrafted rules in Sims 3.

.slide 23

The system that I just described to you probably isn't all that computation-heavy, but if you run it for a lot of Sims at the same time, you may very well end up with some performance issues. So, to prevent this, the lives of Sims that the player can't see are simulated only at a very low level of detail. For example, when the player sees a Sim again, the values of their needs metres will be decided based on predefined curves. For example, hunger has a curve like this (draw this). The start is when the Sim wakes up, at which point they are moderately hungry, then these three spikes correspond to the times they have breakfast, lunch and dinner. And, if the player sees a Sim at midafternoon, let's say, their hunger value will be this (draw).

Also, important life events are generated for these Sims once in a while, at random, I think.

.slide 24

And, of course, we can't forget about the player's role in all this. As I said at the beginning, the Sims should take into account the player's actions, so they shouldn't break the story that the player is trying to tell. The achieve this, the designers implemented a bunch of so-called 'autonomous feedback loops'.

Let me give you an example. The Sims' relationships with other Sims are defined using metres that reflect how much a Sim likes another Sim. I think each Sim has one such metre for every other Sim, but I'm not sure. The point is, the value of this metre is increased when they have a positive interaction with another Sim, independent of whether it was commanded by the player. So, if the player instructs two Sims to hang out and they have a good time, they will be incentivised to do so more in the future.

Now, there is lots more to the AI of the Sims – I didn't even get into their wants and fears and the hand-crafted trees that govern their life trajectories, but we could probably have a whole lecture just about the AI of the Sims, so I think this is enough.

.slide 25

So, that is it for today, thank you for your attention and see you in another two weeks for what will be the last instalment in this series.