Shipping State of Decay 2

Anecdotes and ramblings from Jørgen Tjernø,
a programmer at Undead Labs

Thank you all for showing up today! Slides will be available online, last slide has the link.

About me

- Norwegian born & raised
- Entered games industry in 2013
- Worked on NVIDIA graphics driver, Steam, Dota 2, Planetary Annihilation, and State of Decay 2, among others
- Currently an engine, tools, and gameplay programmer at Undead Labs
- Contributor to Unreal Engine 4

<1> I'm not from around here -- I've only lived in the US for 9 years. You might not know a lot about Norway, but for one -- there's not a lot of game industry there (though it's getting better).

<2> I started out outside of the games industry, because everyone I knew told me that the games industry was a rough place to work.

<3> My first employment in the US was for NVIDIA, because the work was interesting, and it *wasn't* the games industry. I took another job after that, but started working on some games projects in my spare time, got an amazing opportunity in Seattle, and then moved up here -- and I've lived here since.

<4> I do a lot of different things at Undead Labs -- I like doing things that are valuable, rather than things that belong to a specific problem space.

<5> .. And as a part of that, I've gotten a chance to contribute to Unreal Engine 4, submitting some of our engine changes back to Epic, which then gets included in future engine releases.

State of Decay 2

- Developed by Undead Labs, a wholly owned subsidiary of Microsoft
- Shipped in May of 2018
- Over 4 million unique players & 6 billion poor zombies murdered
- More than 4 years in development
- Team size grew from ~30 to ~70

Some stats about State of Decay 2

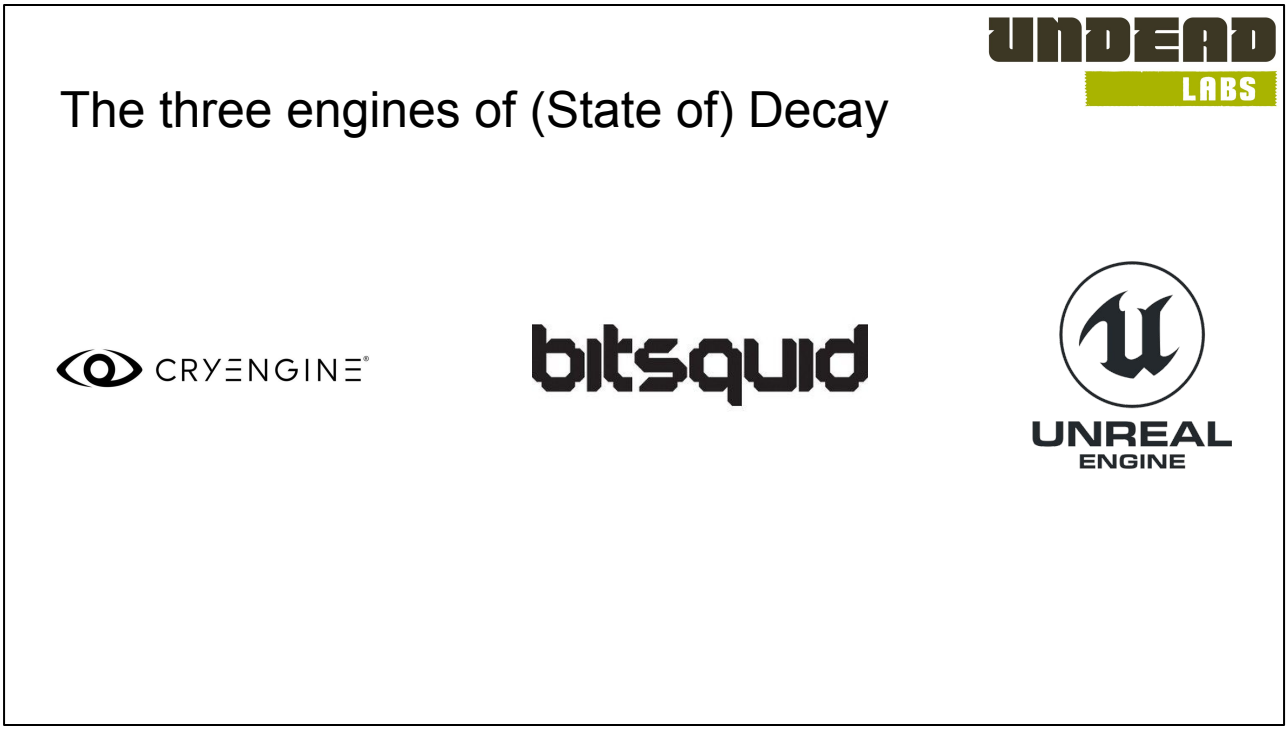
<1> We shipped the game as an independent developer -- Microsoft was our publisher -- but a couple of months after launch, we were acquired by Microsoft

<2> The game came out a bit over half a year ago, and we're still actively expanding it and updating it

<3> More than four million people have played the game on the two platforms we support

<4> Shipping the game took a while, though I was only involved for the last ~2.5 years of development

<5> When State of Decay 1 shipped, I believe the studio was around 30 people. Now we're over 70, and we're likely to grow some more in the coming years.



<1> SoD1 was developed on the CryEngine

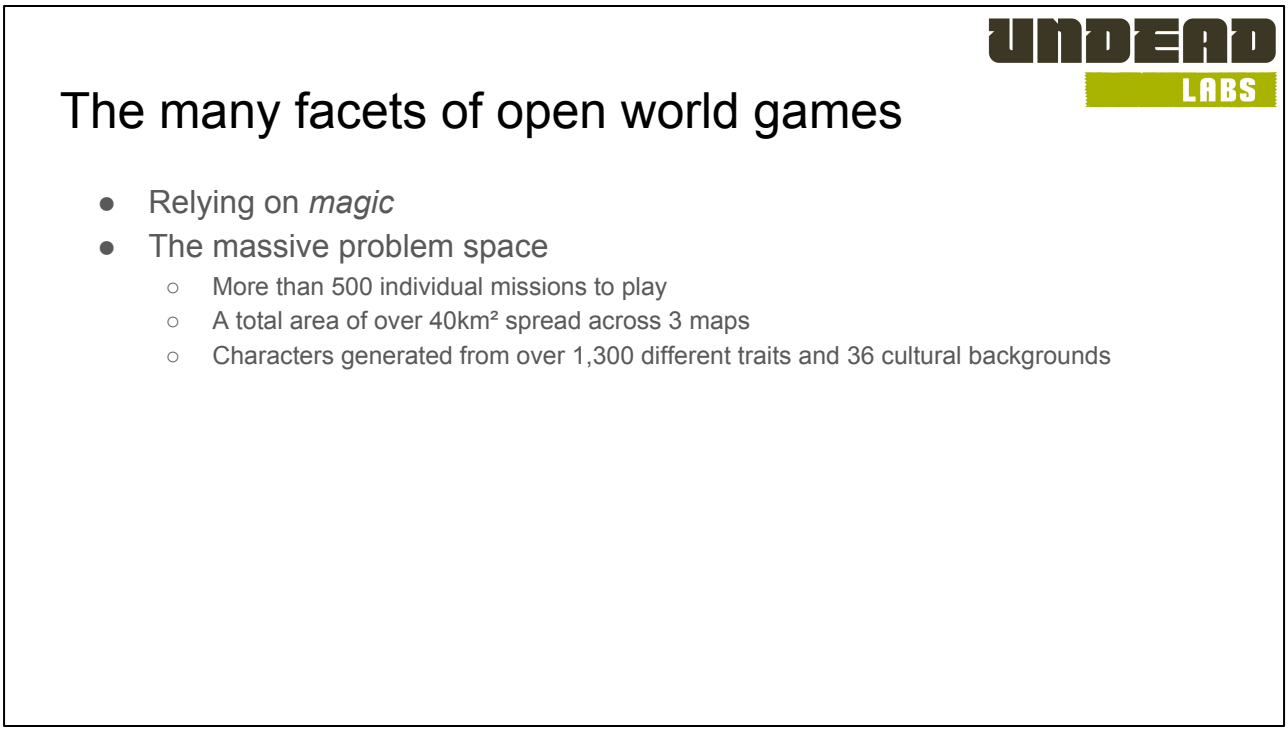
<2> SoD2 shipped on Unreal Engine 4

<3> While shipping the first game, it became apparent that the team would not be able to ship a bigger scoped game on the same engine. At first, SoD2 was developed on an engine known as bitsquid (later Stingray, one AutoDesk acquisition later). About two years of development went into the game on that engine, with many aspects implemented or prototyped. The main problem was that bitsquid did not have a great content pipeline, and so the programming team was the bottleneck for a lot of work that needed doing.

It was a difficult decision to make, but after a company-wide vote, the conclusion was to re-do everything in UE4, with its robust content pipeline. I joined Undead Labs shortly after this decision was made.

It obviously put us in a bad spot, with a lot of work needing to be re-done in a different engine. Very little (if any) of the code could be ported straight over. While the programmers were busy trying to get the game back to where it was on bitsquid, the designers were (of course) very bored, and did their best to keep busy. Sadly, this led to a situation where as soon as a programmer finished their work, there was a fully designed system already documented for them to implement -- which is not the normal workflow at UL. We prefer to involve all the different disciplines in reaching decisions, so that technical constraints and different perspectives can be considered.

Over the next ~2.5 years, we got the game back online, many more systems implemented, and the game shipped.



Now that I've set up a little bit of back story, let's talk about some of the concrete challenges we encountered.

Open world games often struggle with having a large number of systems that interact and influence each other. This can cause it to be hard to verify your ideas early in the development process, because it's easy to get in to situations where they "depend" on each other.

<1> It can often be easy to assume that some other aspect of the game will solve problems in your domain, and that other aspect might come in late, be cut, or not live up to the initial expectations. In our case, we had an early version of a system we called the story director that was supposed to handle a lot of pacing needs for the game, and funnel the player into certain paths. It was a complex system that looked at the state of the world, at the state of your community, and how you were playing, spawning missions to encourage you to explore the world and to drive dramatic peaks.

This system turned out to come in late, and perpetually "just needed tuning" to solve the problems we were having. Frustratingly, the system we had created was hard to tune, because the concepts it worked with were so abstract.

This meant that late in the process we switched to a simple system that had a number of "decks of cards", where each card was a mission, and rules about when to draw, play, and re-use cards. While it was much easier to tune and reason about, this

system came in pretty late and so our development tooling for tweaking and investigating it is still not what we would've liked.

In game development there's a constant struggle to not be overwhelmed by complexity, and even at a studio full of veterans we sometimes make mistakes.

Another problem is the size of what you're working on <2> which affects your QA, your development, and how your decisions to cut functionality propagates.

One of the systems I've done a lot of work on is the mission system and with over 500 missions <3> every change to the system has consequences that can be quite difficult to test and verify. To further complicate things, all of our missions are "parameterized", meaning that they can occur in a number of different locations, involve different characters, and contain different goals

Another issue is that the map is very large <4>, and so there's a lot of potential geometry problems and metadata issues. As an example, there was a spot when we shipped where the AI could just fall through the ground. It also puts a lot of pressure on the team to address how we store and load that amount of data.

Similarly to how our missions are parameterized, so are the characters in the world <5>. We generate characters based on a wide range of settings, and end up with different skills, names, ages, pronouns, outfits, and backgrounds. A lot of the heavy lifting is done by a system that picks their cultural background and their traits. The system has tools to help us generate balanced characters that are free of contradictions, but with such a large set of data, it can be easy to miss a case. Examples of problems we've encountered are traits that individually are innocuous, but that add up to e.g. creating a character who has cataracts *and* is a professional hunter, or a character that uses a negative number of beds.

The many facets of open world games

- "*Onion development*" -- start with a core and layer around it
- Invest in automation -- tests, verifiers, validators, et cetera
- The right way to do something has to be the easy (or ideally, only) way to do it
- Actionable error reporting, and **act on it**
- Hard limits

So obviously I have some opinions on how to address these things, otherwise why would I be here? Due to the tight nature of our deadline for SoD2, many of these things are things I'd love to have seen more of as well.

<1> This is a phrase that our founder loves, but it's something we're really hoping to be better at the next time around. If you start with a narrowly focused experience, keep it playable (and ideally, fun), and layer on functionality on top of it, your feature cuts will be less dangerous and "relying on something that'll come down the road" will hopefully be less prevalent. It's better to cauterize a feature and implement it on its own, and then later go back and integrate it with something new if it ends up happening, than having an open wound where other functionality should be when you ship the game.

<2> The larger the scope of your content, the scarier it becomes to make changes to content or code. One important way to fight that is to invest in tools -- tools that help QA do their job, tools that can tell you that things are consistent, tools that tell content creators that their data is invalid, and tools that tell programmers that they're trying to ship bad code. In SoD2 we have a large swathe of tools, from things like tools that tell us that a particular location that is requested by a mission is never used by it, to tools that load the maps and verify that the geometry isn't set up in a bad state. Some of these run on every check-in, some of them run every day, but the important part is that they report their errors in a way that is acted on.

<3> Another problem is that if there are many ways to do something, usually only one

of them is the *right one* for 99% of cases. It's surprisingly common that errors are introduced simply because the editor **allows** the error to be introduced. Ways we address that in our game is things like using a dropdown instead of a text field if there're only a certain set of values that are actually "good" values, automatically running validation whenever you save your content, and running content validation on every check-in.

<4> A lot of these problems are combinatorial, so it can be infeasible to enumerate all of the options. If you have an error reporting infrastructure set up where you can collect runtime errors and make sure they're addressed, you can have the game report "bad" states -- e.g. if the content is set up so that a character can be generated with "Has Cataracts" (which limits their ability to shoot) and "Professional Hunter" (which boosts their ability to shoot), having an error report tell us how we got into that situation has saved our bacon many times.

<5> Finally, programmers and designers both tend to want things to be "infinitely growable", to support as large numbers as possible, but the truth is that you, your performance, your UI, and your QA department will all be happier if you strive to set hard limits. Pick an arbitrary number of skills your character can have and stick with it, enforcing it in code. Don't allow an individual's number of "consumed beds" to be negative -- what does that even mean? When you have these hard limits, you can always relax them if you have a concrete use case, but in the mean time, you can have error reporting when something attempts to violate those limits, or make your tools error out if someone tries to enter them.

(More) common challenges

- Multiplayer
  - Testing
  - Agency vs isolation
- Cross-platform
  - Input & UI
  - Performance

<1> State of Decay 2 is a multiplayer game with up to four concurrent players. We had some challenges around that, and here are some examples of them.

<2> Testing multiplayer can often times be tedious and complicated, and multiplayer programming is complicated. Testing needs to be a regular part of development and of your company culture -- at Undead Labs we have regular playtests, and will test both single player and multiplayer. In addition, when you're testing multiplayer, you're probably on a **much** better network connection than your customers. In Unreal Engine 4 you can set up "network simulation" parameters to pretend like you're playing from a cabin in Iceland with packet loss and latency like nobody's business, and for State of Decay 2 our internal test builds always have "average network conditions" simulated. This helps you catch race conditions and issues that don't show up if your internet is fast enough. In addition, testing multiplayer needs to be fast and easy. You need to prioritize tooling and performance for those situations, because friction causes people to steer away. Making it easy is one of the things we did not do great with for SoD2 and are very aware of needing to do better in the future.

<3> One thing that was a constant struggle with State of Decay 2 was deciding where to draw the line between **agency** and **isolation**. When I say **agency**, I mean how much say someone has when they join your game and your community -- how much impact can they have on the world. When I say **isolation**, I mean how well do we isolate you from negative effects stemming from other players in multiplayer. We used to have the ability to open fire on NPCs, which would turn them hostile towards you,

and at one point this was something someone who joined your game could do. This meant that they could turn enemy enclaves permanently hostile and effectively fail some of your missions, or cause you to be in a bad situation. One of the reasons this is an important consideration for us is that we have a matchmaking system where anyone can call for help, and people who have volunteered will be joined to your game to help you out -- but people don't always have honest intentions.

<4> State of Decay 2 came out on PC and Xbox One, and was always a cross-platform title. Everyone at the office has at least one Xbox One development kit on their desk. Even so, our cross platform development had some rough edges that had to overcome.

<5> The most obvious user-facing problem with a cross platform game is when the most direct interaction you have with the game -- input and UI -- do not match your expectations. UI is a deceptively complex problem to solve, and our game has a relatively significant amount of it. Even simple things like "how do you back out of a choice you've made" needs intentional choices -- on Xbox One you'd likely just say "press B", but on PC you probably need two ways, e.g. ESC and a mouse button.

Other challenges can be things like assuming that "ESC" and the "Menu" button on Xbox One should behave the same -- except that user expectations differ. If you have a menu open, users tend to expect that "ESC" means "back out of this menu", but on Xbox, "Menu" often means "close everything and open the menu".

Which platform people tend to test on will likely decide which platform "feels better", simply because there will be more feedback on it. Left to their own devices, people will test on the platform that's easiest to use. This is why **everyone** at Undead Labs has an Xbox on their desk -- it doesn't matter if you're a concept artist, a designer, or a programmer. In addition, we started doing specific platform tests, where we'd ask everyone to play the game on a specific platform. That's also a good way to figure out where the pain points are -- why people **aren't** playing on the platform you're talking about.

<6> Performance characteristics are different on PC and Xbox One, especially when it comes to graphics pipeline. In addition, on an Xbox One you know **exactly** what the system specifications are. When you're developing, it's important to think about what the "worst case" scenario for your performance is -- what's the worst situation a player could get into for your systems? For State of Decay 2, that's being the host of a 4 player game on a low-end system, with a lot of zombies around. It turns out that the network code in UE4 is responsible for a lot of work in that case, as well as handling movement and AI for all the zombies. If you have hard limits on e.g. how many zombies you spawn, it's a lot easier to create test scenarios where you can measure how bad (and why) your performance can get. Like I mentioned, we didn't do the best job of making it easy to test multiplayer as a part of your day-to-day process, and programmers and designers mostly tended to test on PC (because that's where the

UE4 editor runs) rather than Xbox, so it was relatively late in the development process that we realized how poor performance was in that case. Due to heroic efforts on the parts of some of my coworkers, we managed to get that back under control, but if we had been aware of it earlier it would've been a **lot** less stressful.

UNDEAD LABS

Questions?

jorgenpt

https://jorgen.tjer.no

jorgenpt@gmail.com

DMs open, e-mails welcome -- please reach out with any questions you have about anything. Career growth, technical challenges, et cetera. I will do my best to answer.