

DAVID CANDLAND

UI DESIGN LEAD

Tenacious Design and the Interface of Destiny

Good afternoon.

I'm David Candland, User Interface design lead at Bungie.

I've been at Bungie for over 15 years. This means that I was instrumental in creating and designing the UI For Halo, Halo 2, Halo 3, Halo 3 ODST, Reach, and now Destiny.



Right now, you're probably wondering why I have a slide showing a piece of spaghetti on a wall.

Well, let me tell you.

Years ago, I was boiling some spaghetti and forgot to set a timer. I asked my wife, "how do I know when it's done?"

"Did you set a timer?"

(Sheepishly) "No."

"Alright, take a piece out and throw it at the wall."

I did as she said, and it hit the wall and it bounced right onto the counter. She told me to try it again in a minute. (NOT THE SAME PIECE) When it sticks to the wall, it's done.

This is a lot like the design process. You research, you calculate, you plan, you document and then you take your best shot and throw your design into the game and see if it sticks. Sometimes it does, sometimes you use the last iteration as a learning opportunity. The key is to keep trying. To be tenacious until the design has sticking power.

FEATURES DISCUSSED IN-DEPTH

- The Free Cursor
- Creating Gear icons
- Optimizing Localization
- Designing The Director

Today, I'm going to discuss 4 of these "noodles", if you will;

As I go over each of these aspects of our game, hopefully I'll give you a few takeaways on our design process, some technical notes, and some of the things we learned in creating the destiny UI.

These features are:

- The free cursor
- Creating Gear icons
- Localization Practices and Release Valves
- The Director

THE FREE CURSOR

BUNGIE

DESTINY 

Let's start with the free cursor...



(Video)

This is one of my characters. Let's invoke UI to get a better look at all the armor I have equipped.

Selection is controlled with this cursor that moves freely about the UI. Most console games use D-pad or sticks to navigate, so a cursor on a console game is pretty much unheard of.

The background counter-scrolls in the opposite direction. The left stick controls the cursor, leaving your right thumb for button presses or rotating your character preview around for a better look.

Your inventory can fill with things like guns, armor, materials...You can have a lot in your inventory.

Trying to accomplish simple functions like equipping a helmet and then going all the way down to equip your ghost would've taken much longer, and many, many clicks of the d-pad just to pull this off.

WHY A FREE CURSOR?

- A free cursor and tooltips allow for high information density on a single screen.
- Free scrolling screens with animating interactive elements are sexy.
- Free cursor screens work in both 4:3 and 16:9 aspect ratios.

So, Why a free cursor?

I just covered this first point: saving you all those clicks saves you time and tedious effort.

Free scrolling screens with animating interactive elements are sexy
Pro tip: Best way to make someone think your UI is easy to use: make it sexy. Right? It becomes a little easier to overlook some of the small things that may get on your nerves that way.

Free cursor screens work in both Standard Def and Wide screen aspect ratios without layout changes or large dead areas on HDTVs.



(Video)

In this example, you can see that the cursor gives us freedom in 4:3 aspect ratio without altering our primary layout due to the background counter-scrolling the content around.

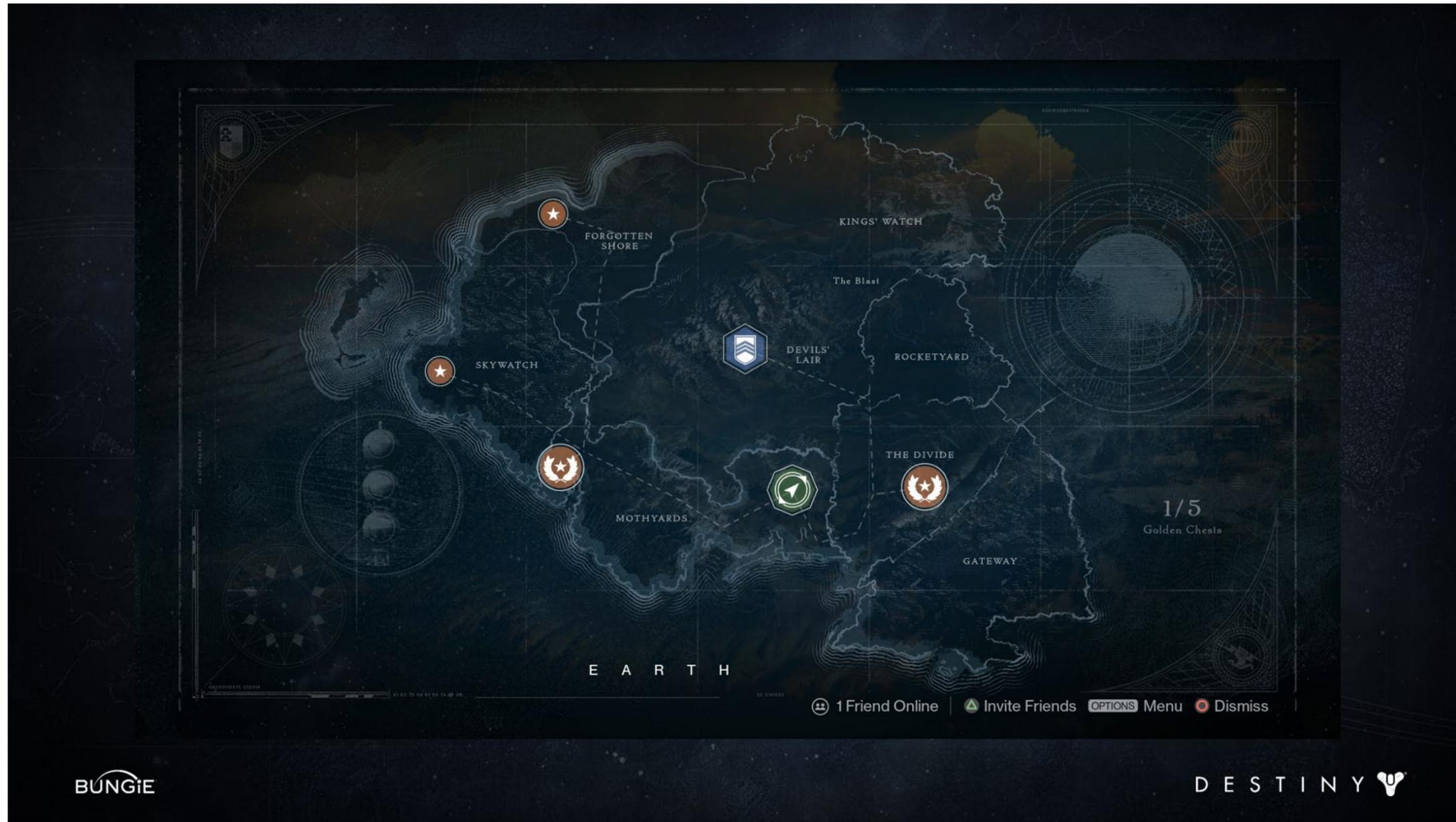
If you move the cursor from one end of the screen to the other, you will always have access to all content on the screen.

WHY A FREE CURSOR?

- A free cursor and tooltips allow for high information density on a single screen.
- Free scrolling screens with animating interactive elements are sexy.
- Free cursor screens work in both 4:3 and 16:9 aspect ratios.
- A free cursor enables interactive elements of a screen to not be grid aligned.

A free cursor enables interactive elements of a screen to not be grid aligned.

DPAD navigation requires predictable, linear alignment of interactive elements which limits visual grouping and graphic design.



This is a shot of one of our maps in the game. If you wanted to navigate a selection from Skywatch to Mothyards using a d-pad, it would be guesswork... is it mostly right or mostly down?

Navigating a map with a d-pad would get frustrating, you'd guess wrong 50% of the time.

FREE CURSOR GOALS

- Spend “autoaim” time to make it feel right
- Embrace it completely
- Counterscrolling required

We'd previously prototyped using a cursor, and because of all these things I just explained, we came to the conclusion that we needed to move forward with this decision. Jason Jones (Creative Chief) agreed, but gave me the following goals to work from:

It's important to work from goals. When you tell designers what to do, you stifle exploration.

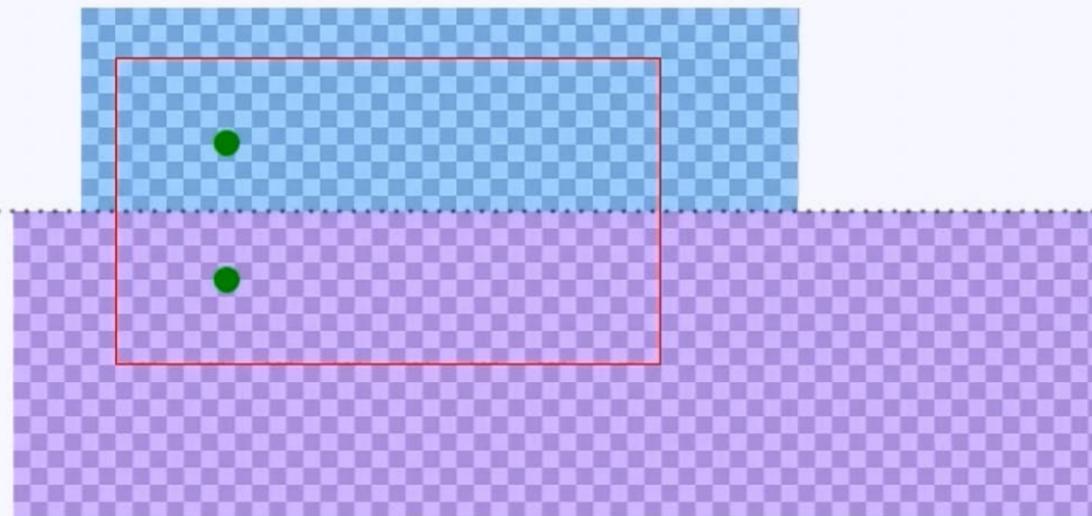
These were the goals:

- Make it feel right. “Autoaim time” Harkens back to Halo 1. We had moved from a PC platform to a console. Using your thumbs on a stubby stick to aim without any assistance is really hard. Our sandbox design team spent weeks and weeks of time to figure out target magnetism, bullet spread, aim assist, and all the values associated with those.
- Embrace it completely; no scroll bars, d-padding, required zooming or other modifier buttons. Point and click only. Use it everywhere, not just when it's necessary.
- Counter-scrolling is required. Early prototypes showed that felt best. Static elements when you move the cursor feel wrong.

Cursor Speed Constant to Screen Space

Always takes the same amount of time to cross from one side of the physical screen to the other.

Small Screen Space



Large Screen Space

(Video)

When counter-scrolling, the content moves from one end of the screen to the other. The content of your screen essentially becomes a big scrollbar that operates on 2 different axis.

As you can see, that even though the cursors are moving at the same speed, the one on the bottom feels faster because the counter-scrolling has to move at a faster rate to be able to shift that much content around.

Big question 1 – do you vary the speed of the cursor or vary the speed of the background?

Question 2 – how do you handle screens that are disproportionately long? Fast on one axis, then slow on the other?

We prototyped several explorations in after Effects, and then in engine to try and help us find solutions to these.



The best solution, we realized, was to make both a non-issue; measure the most screen real estate we will need, and make all screens roughly that size.

Smaller screens like this would have an extra large buffer, a margin of space on all ends to ensure they counter-scrolled at about the same speed as larger screens. But what about all the extra space? Pushing your cursor all the way to one side would scroll all your content off the screen. After getting it into the game, we realized this wasn't a big deal. Nobody chooses to move their cursor all the way to the side because there's nothing to see there.

Selecting items:

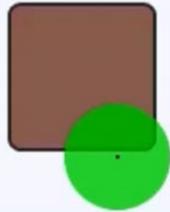
Players' thumbs are not too dexterous, especially on stubbly little controller sticks. Players need assistance to help you land the cursor on an object. We looked into two main options:

- Gravity well (cursor snaps to the button)
- Friction (cursor slows over the button)

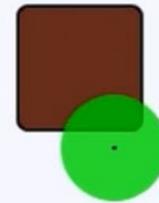
Offset Cursor Behaviors

Speed values picked arbitrarily, only to show differences between normal, fast and slowed cursor movement. All examples are as if joystick is pegged all the way.

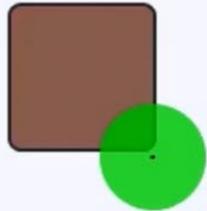
No Effect (200 px/sec)



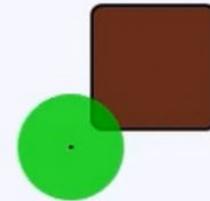
No Effect, Selection Based on Center Point (200 px/sec)



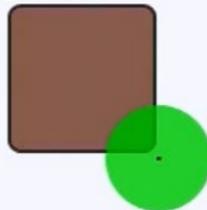
Full Friction (100 px/sec)



Full Friction, Selection Based on Center Point (200 px/sec)



Friction On Enter Not Exit (100 px/sec)



Gravity Well (5 px/sec)



Gravity Well No Acceleration (100 px/sec)



We researched all the variables we could think of... what if there's less gravity if you pegged your sticks? How much friction? What if friction is only applied on the way in? Etc. We eventually prototyped each.

After seeing it in action, we immediately ruled out the gravity well approach. A screen dense with icons means erratic cursor movement and lack of control if your cursor is constantly snapping to each button.

Friction felt good. We got global settings from engineering to tweak the values. If there was too much friction, people would circumvent all the icons. Too little, and you cant hit targets without frustration.



We eventually found the sweet spot, though there was concern about the varied experiences with the extreme cases of screens; those with densely packed icons and open screens with only a few hot spots.

We asked our engineers for screen overrides to tweak down friction on dense screens, so even though cursor speed and friction varied, the experience felt consistent.

The thing about design, is even though on paper and mathematically things may be correct, they may not feel that way, and sometimes you need to fudge things differently to make them feel consistent.

THE FLYOUT MENU PROBLEM



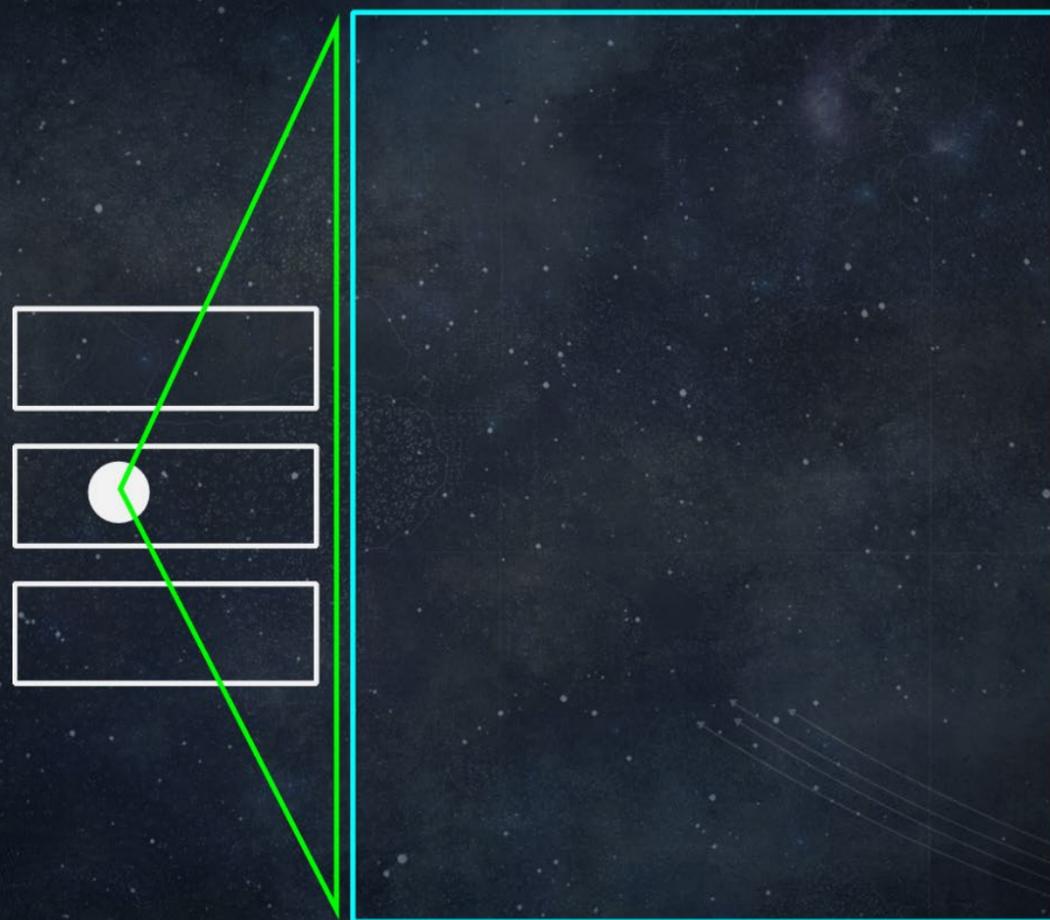
It's human nature to take the path of least resistance. As humans, we're always cutting corners to get from one place to another,

This created a problem with our menus. When players took the path of least resistance, they would find themselves leaving the menu and crossing over an adjacent button with the intent to get to the setting they want, only to have the menu disengage as soon as they left it.

The only way around this was to move the cursor at a 90 degree angles -which feels unnatural. We needed some way to gauge player intent so when they moved the cursor off the menu in a trajectory to select an item at the top of the menu, we didn't pull the rug out from under them.

2 key variables:

- Velocity
- Trajectory



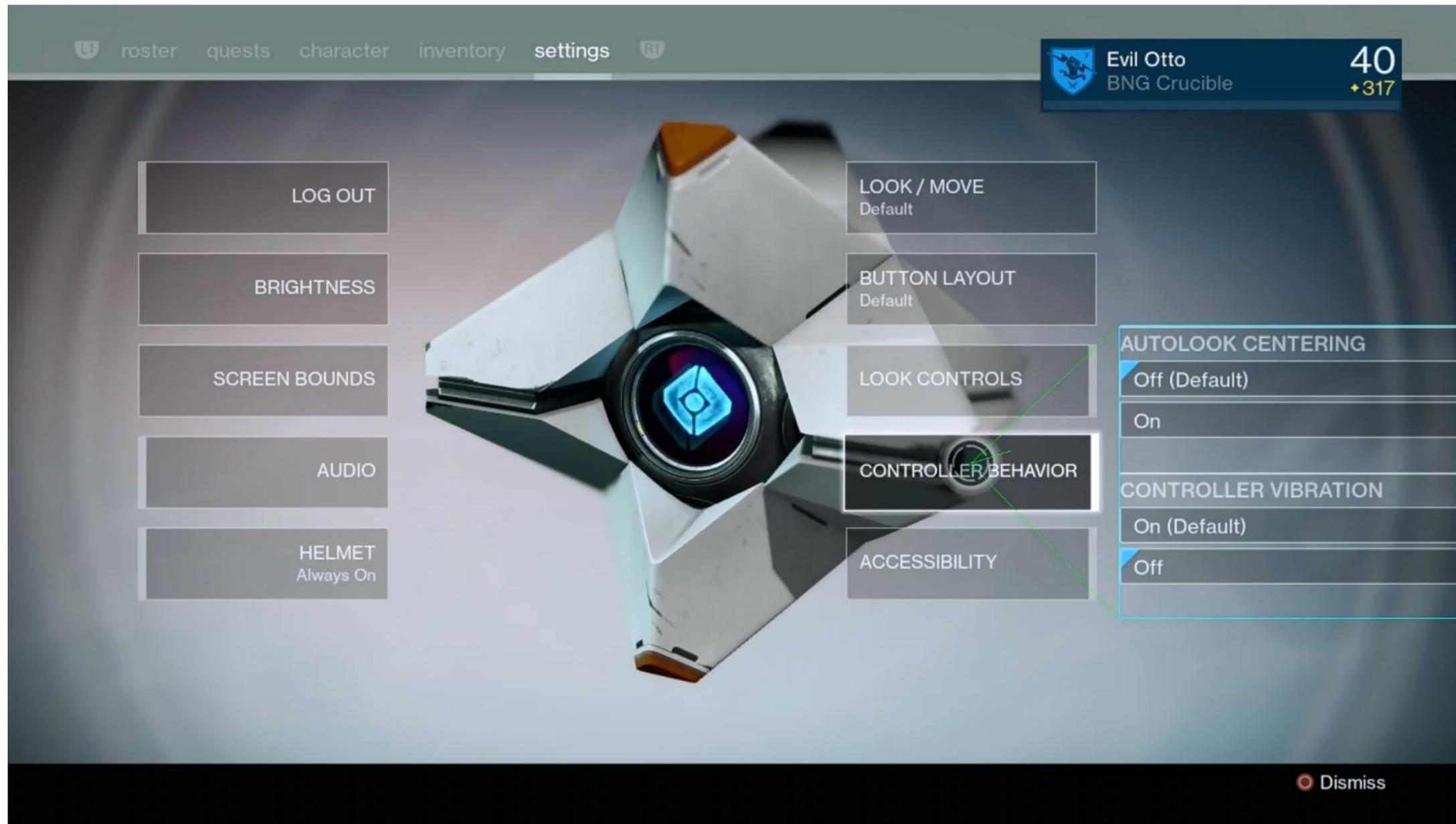
We had to calculate player intent.

If the player is pegging the stick quickly in one direction, we can assume their intent is not to stop on the button directly above or below their current position.

Calculating velocity is easy, you get that analog data from the sticks. Trajectory was a bit trickier.

We basically did this by computing a rectangle around the Submenu Container bounds. The top and bottom of the rect calculated the 2 side points of a triangle, with the cursor location making the third point. Any trajectory within that triangle gave us a positive result.

Our engineers set up debug geometry for us to test our settings with.



(Video)

This is what it looked like.

When the triangle is red, that means the cursor is moving at too low of a velocity to trigger our solution.

Green meant that we hit the speed threshold to ignore the cursor passing over other buttons in the menu.



(Video)

The free cursor was a lot of work, we spent weeks and weeks on this feature.

In the end, we were very happy with result, though we're always looking at ways we can improve the feature.

CREATING GEAR ICONS

BUNGIE

DESTINY 

Next, I'll talk about the gear icons

GEAR ICONS

- Represent the item visually so you can recognize your item at a glance
- Make it feel tangible, collectible
- Players must be able to have a lot of these
- Make Thousands of unique icons

Gear icons are an intrinsic part of the UI.

These were our goals for this feature.

Early on we started to explore...

EARLY EXPLORATIONS

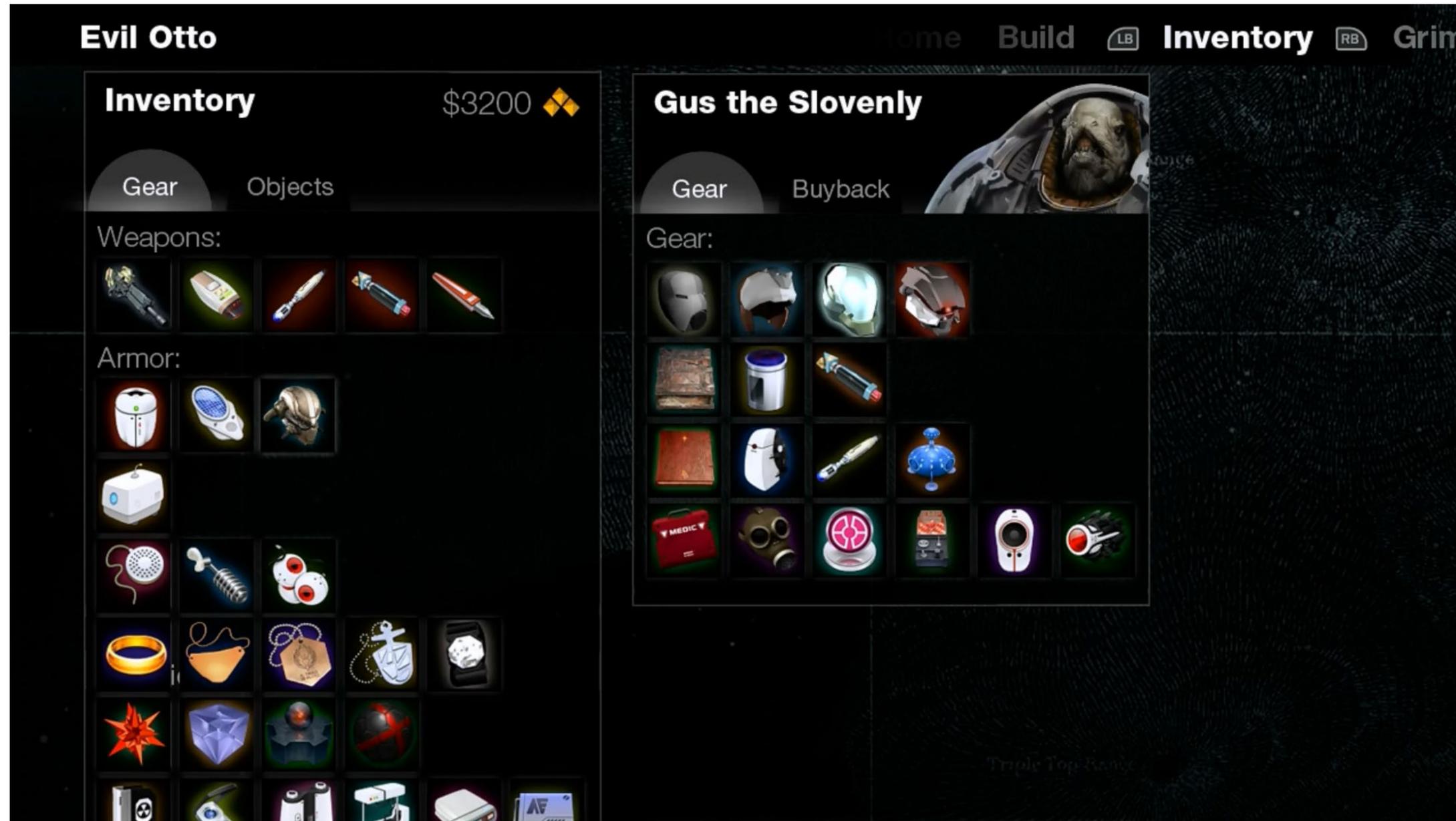


Are there any UI designers in here? If so, raise your hands.

So, everyone else, here's something you should know. When a game is in early development and we have no real content to put in our mock-ups, we make stuff up and slap in placeholder images that we find online.

So with that caveat, please try not to read too much into these. If you're watching this later online, and picking things out from freeze-frames, like a sonic screwdriver in my inventory, or a mission on Jupiter, it's likely you're just seeing something of an artist's musings in Photoshop.

With that said – here's an early exploration of the player inventory. We wanted to keep the icons isolated in a panel on the side, so we could bring it up while the game was running.



This way we could see both our inventory and the vendor inventory simultaneously - which is pretty traditional in RPGs.

While this was better in some respects, it covered up our ability to do any vendor performances as you interacted with them. It also filled the screen with icons - the screens became visually dense and hard to read at a glance. We also couldn't display very many icons at a time without making them fairly small.

Gus the Slovenly

Gear

Buyback



Boots

Chestplates



gloves

To increase the item count, we tried going with overlapping icons...

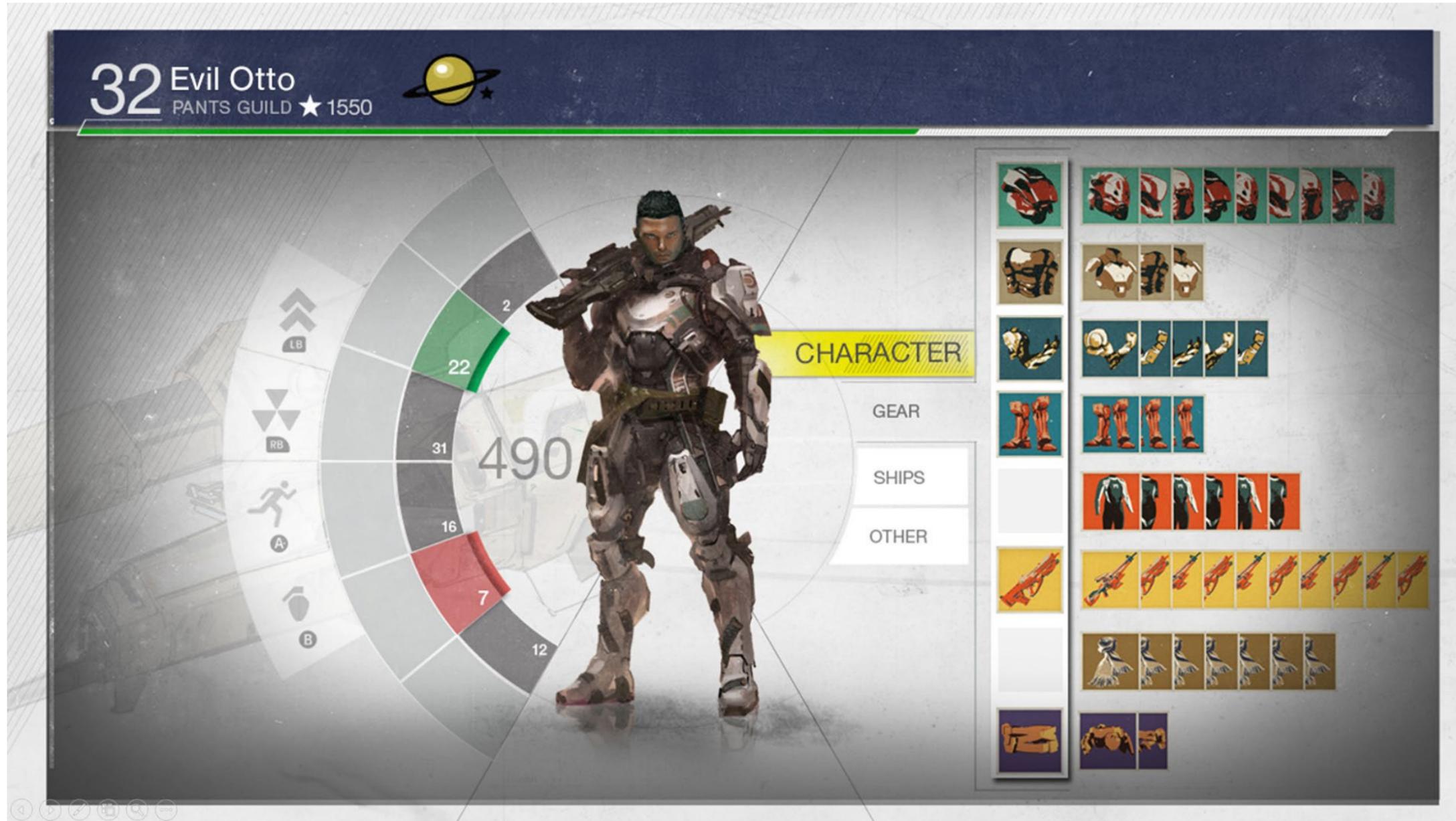


...but that just made things harder to read and parse.

Seeing all your gear at once Was something we were hoping to do, but we started to realize this wasn't probably going to work without really cluttering the screen.

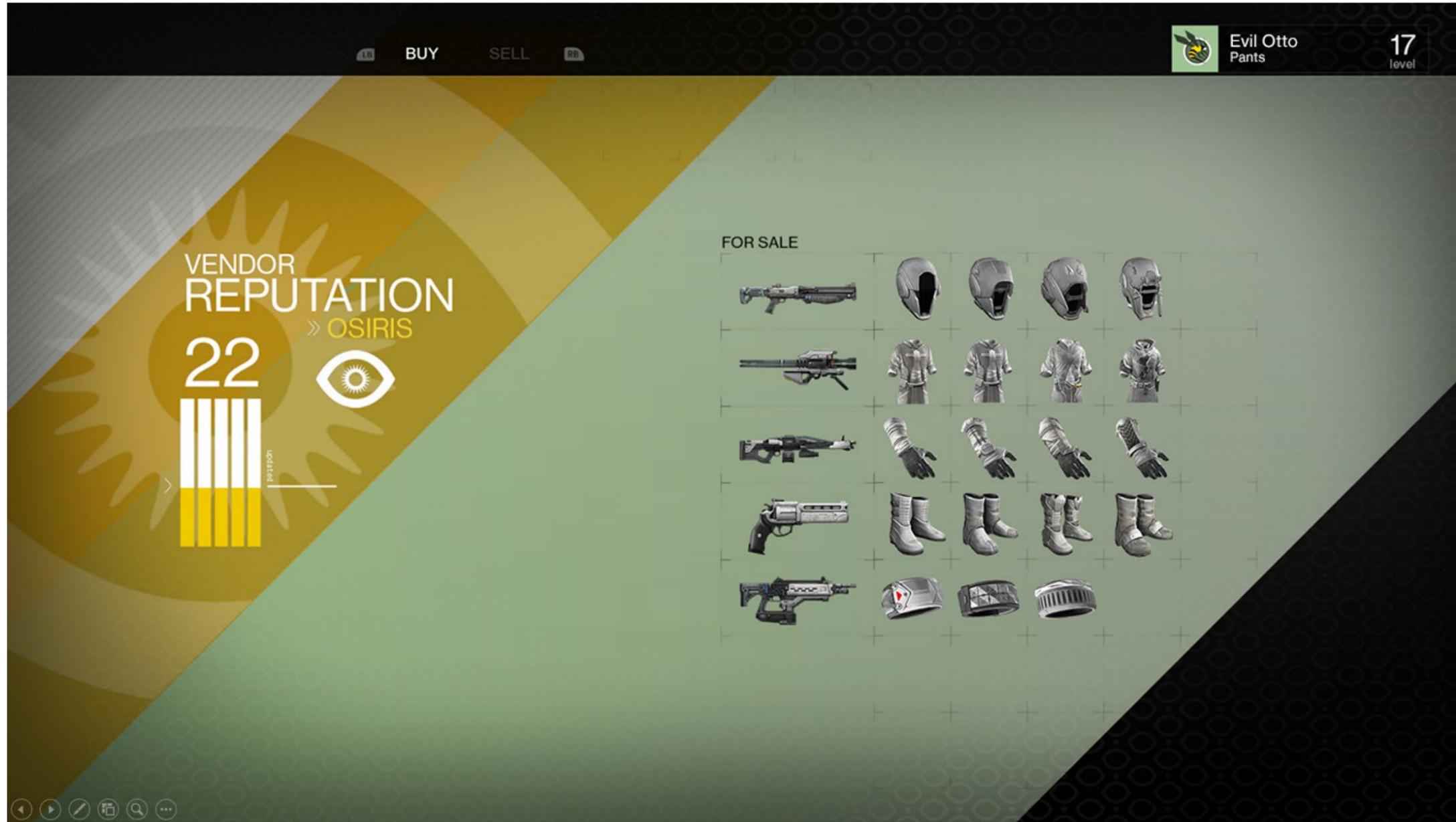


We started looking at ways to nest the content. Representing each slot with your equipped item seemed to be a good conservation of space.

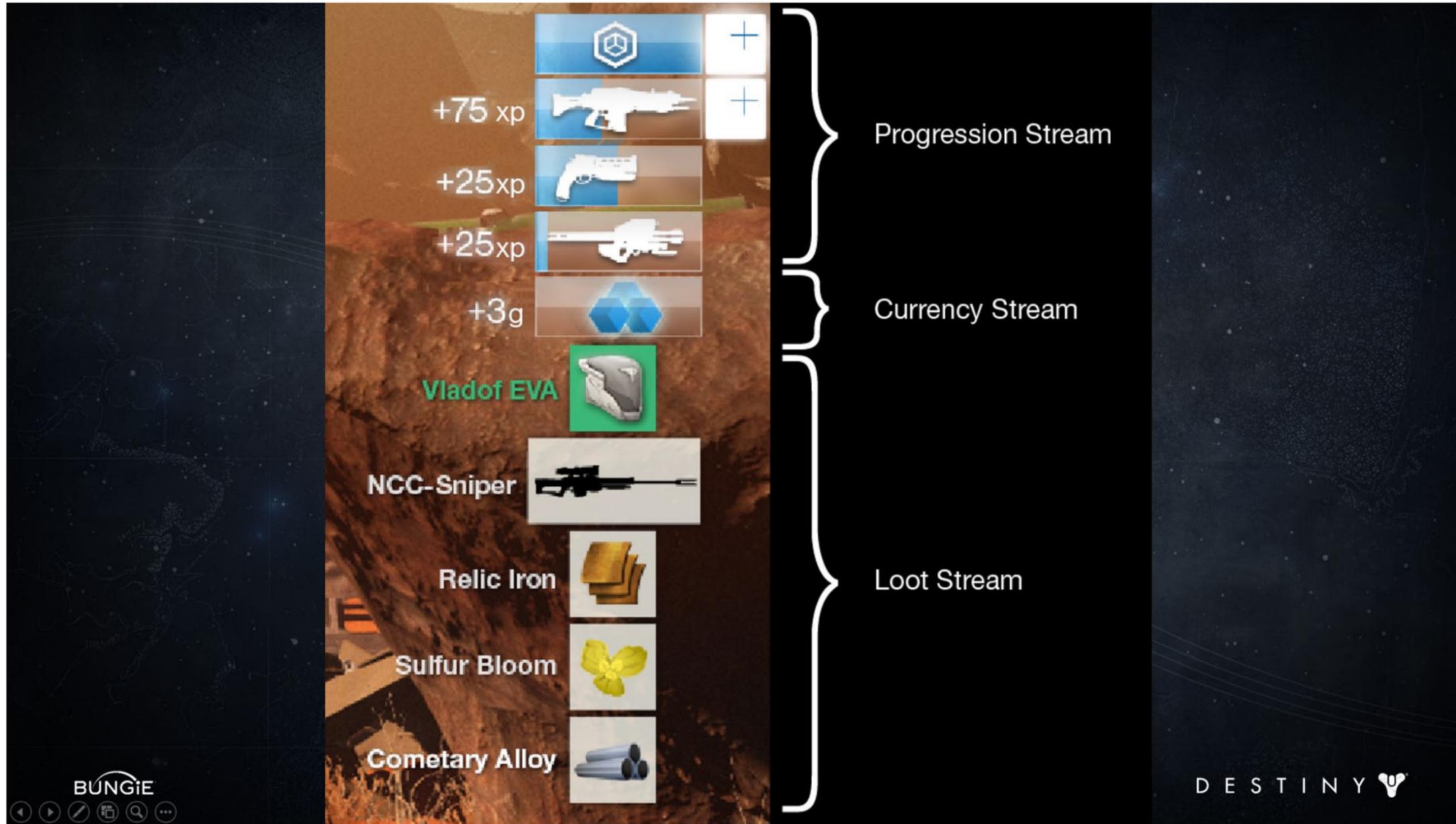


As we were exploring inventory layout, we were also exploring icon appearance.

We looked at using unique tritone palettes for each of the item types. While it looked more cohesive and organized, it wasn't very representative of the items



At one point we tried icons with no background at all, putting progress on the tooltip, but it felt weird for items that had to get cropped, like a long sniper rifle. It also had the effect of making these feel less like tangible, collectible objects.



We experimented with icons that were longer for weapons so we could show the entire silhouette. This caused too many layout as well as technical problems



By this point, we found that we could quickly indicate the contents of each bucket with some graphical representation to the side of the icon rather than adding even more numbers to this screen.

Progress towards leveling the items up was indicated by the blue meter in the background. We abandoned this after a while because it was difficult to see progress when the icons took up much of the background, like the rocket launcher.

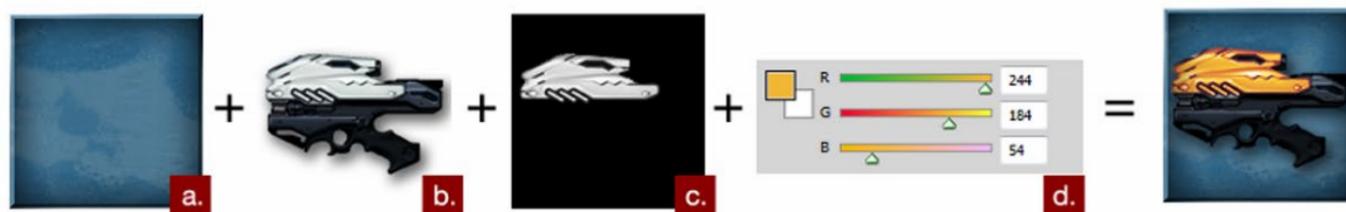


By 2013, we had things moving in the right direction. Right before E3, our investment team introduced the concept of rarity, hence the background colors to symbolize them

Item icon support

As the investment game gets more fleshed out, it is becoming more apparent that people will need to be able to tell their gear apart. I propose that we add the following visual components to our ui_icon_library:

1. Icon name
2. Icon category
 - a. Background
 - b. Foreground
 - c. Changecolor alpha
 - d. Changecolor value



Currently in the ui_icon_library tag, we have an icon name with a bitmap reference. While a single, flat list may work for now with just a few icons, once we start getting icons in the thousands, it will become quite unwieldy to work with. We should allow the investment designers to categorize and sub-categorize these to their hearts content. The icon will still have its unique name so we only have to pull icons from a single gear library.

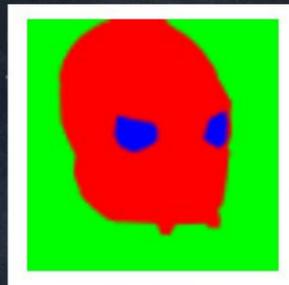
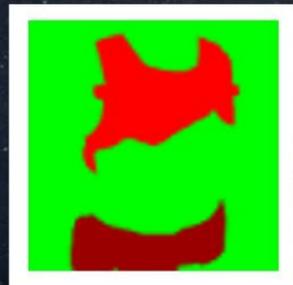
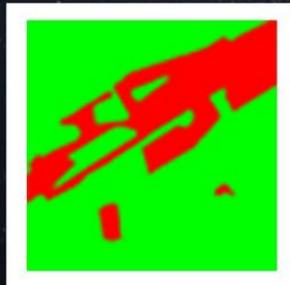
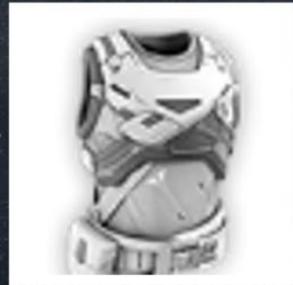
Having a bitmap preview would be great, but not required for M4. The tag structure should be:

- Category (weapons)
 - Subcategory (personal weapons)
 - Name (personal_weapon_base_hunter_seraph)
 - Bitmap reference (background_blue4b.ui_bitmap.tft)
 - Bitmap reference (personal_weapon_base_hunter.ui_bitmap.tft)
 - Bitmap reference (personal_weapon_base_hunter_changecolor_alpha.ui_bitmap.tft)
 - Rgb value (244,184,54)

During our time of exploration, we tried to figure out how we were going to author all of these icons.

In an attempt to reduce artist workload, the plan was to create representative icons for each archetype, add a tint and swap out the background to give things variety.

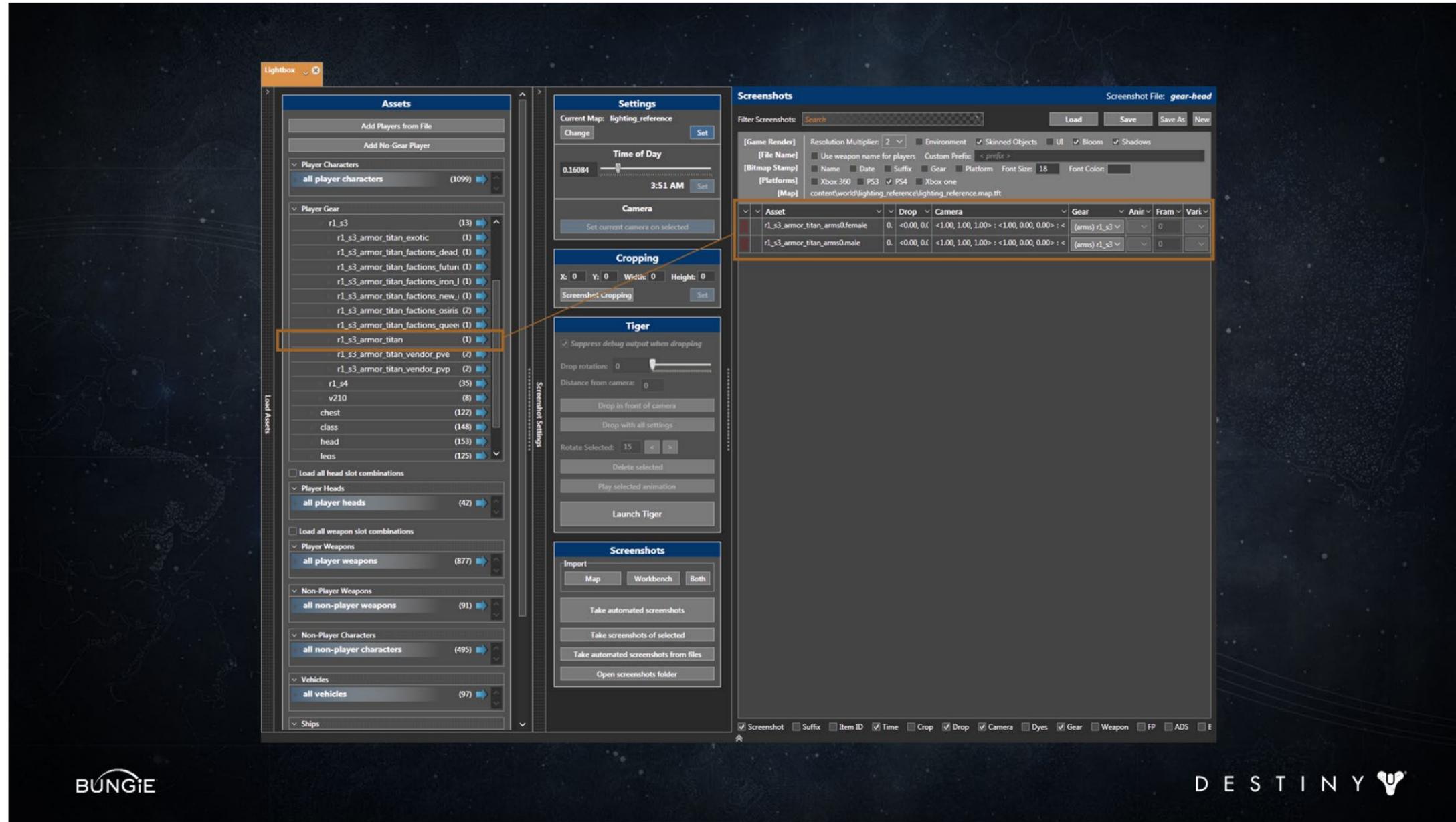
EARLY WORK



We set them up by hand, masked them by hand, painted the change colors by hand, just to get a few iterations out of one small piece of art. Not only was this time consuming, but it was not 100% accurate.

We quickly realized we were making passingly acceptable work and putting a lot of time into it

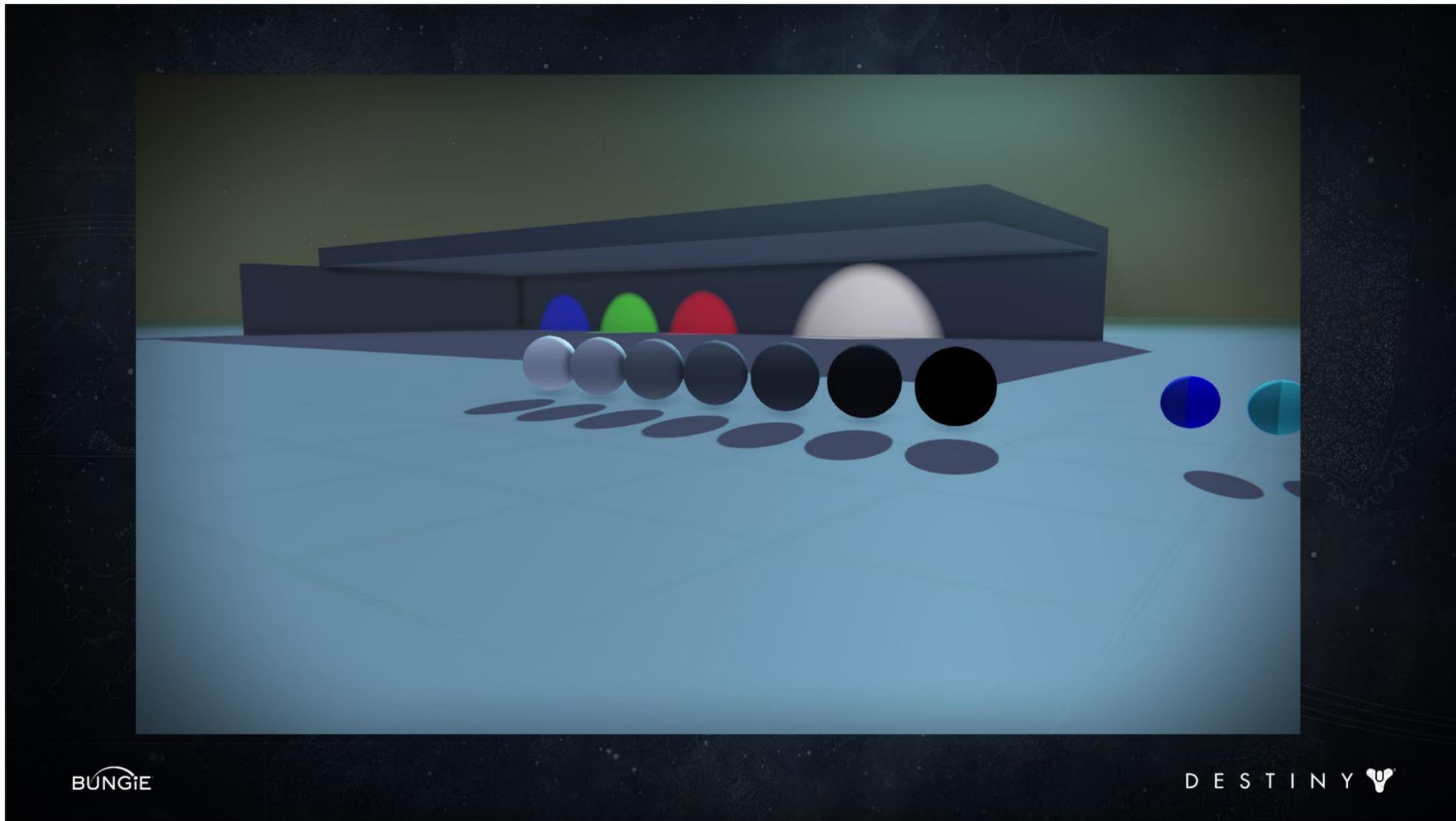
So we costed an automated system.



This is lightbox, our internal asset viewer. It has many uses, but in UI we use it for shooting icons.

The 3D surface team authors the content. The investment team then assigns and names the content, adding them to a registry.

An artist selects the items by name, usually grouped by release and adds them to the render queue.



The assets are then loaded into a lighting test environment. The artist has full control over the lighting.

We have preset lighting models based off of the locations and times of day in the game. Each has a very distinct palette.

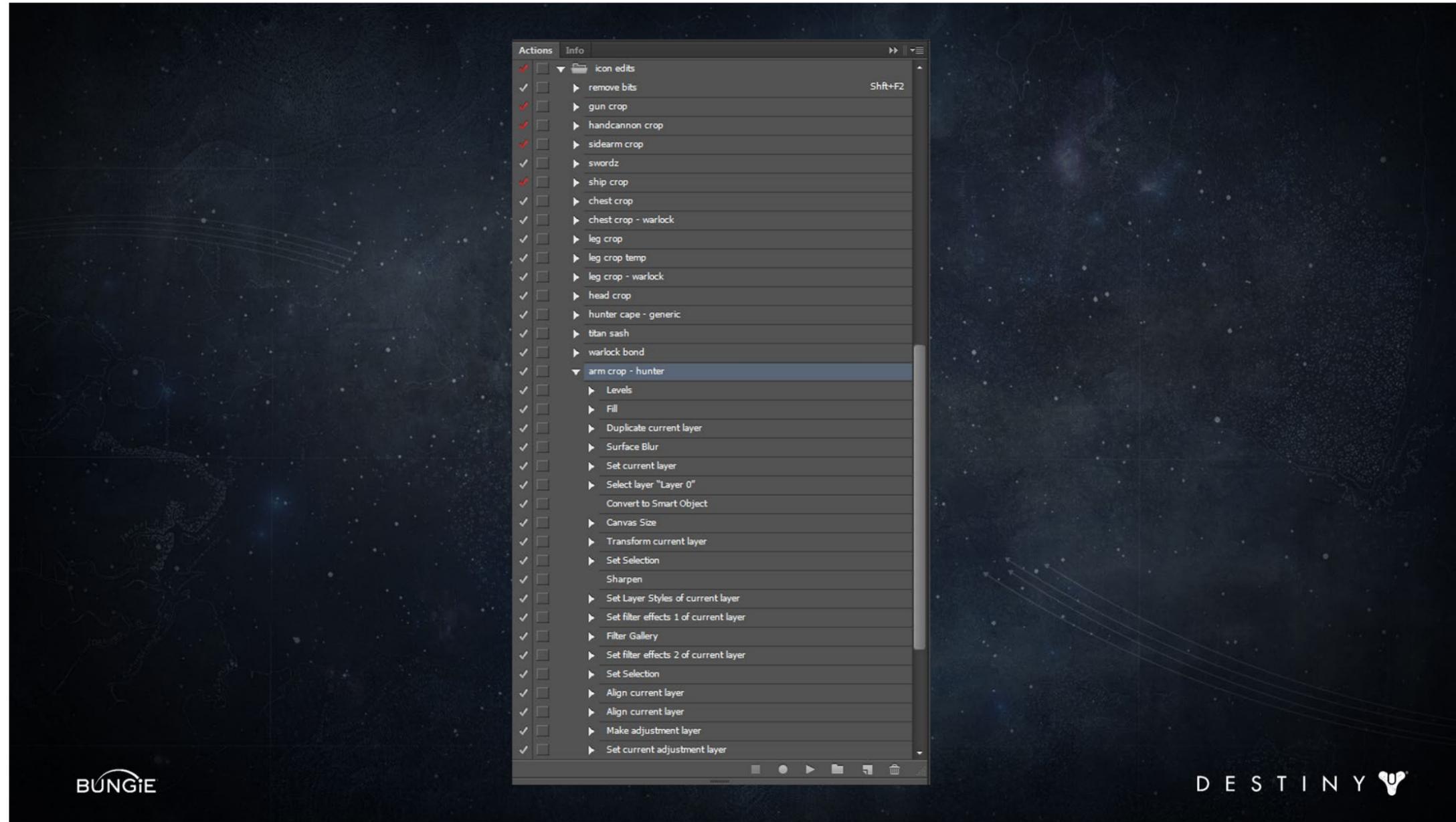


This is Venus at night, which is the most neutral, diffuse lighting preset. It's also perfect for icon shots.

The models can be loaded in on their own, or equipped to a biped. Here it is equipped to a biped that has been render toggled off, so all you see are the pieces of arm gear



Then various rendering processes are turned off; the environment, anti-aliasing, bloom, reflection maps, self-shadowing, etc. Each class and armor type has a predefined camera and pose. The snapshots are taken in quick succession.



The files are reviewed by an artist and then brought into Photoshop where the photos are post-processed with automation scripts.



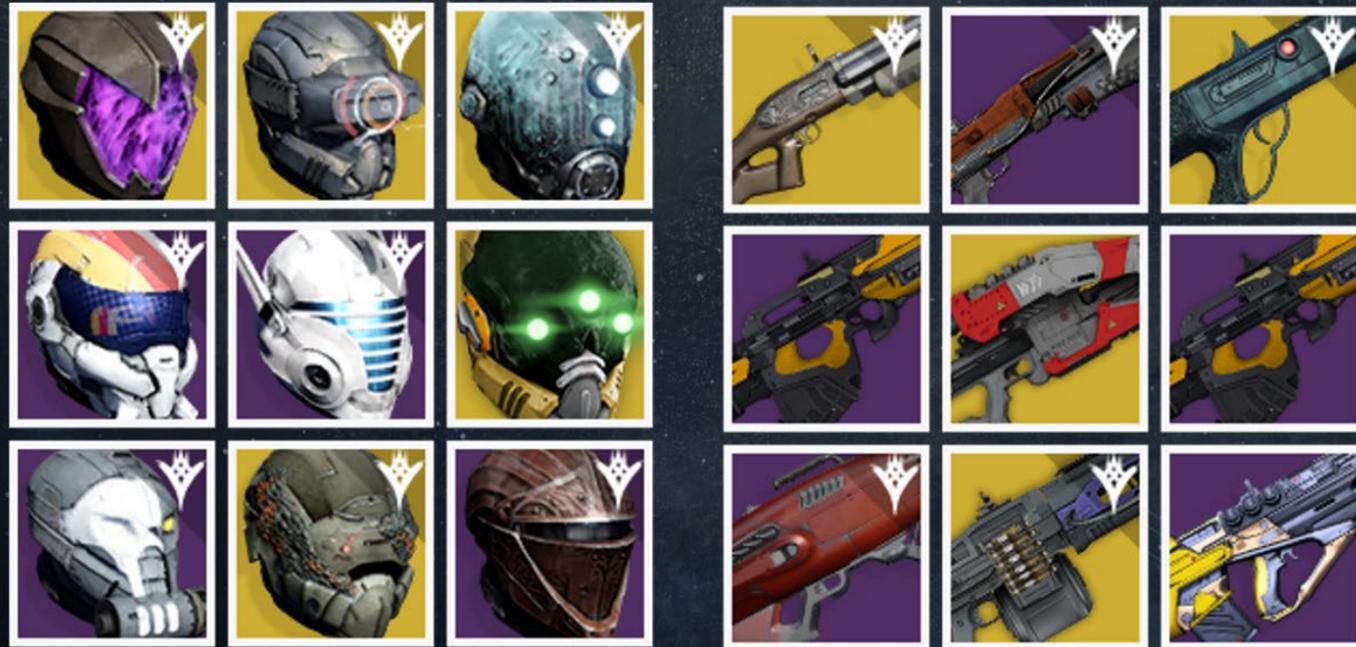
The artist then reviews the shots.

Sometimes if the geometry is unusual, like there are big protrusions or bad tangents, the artist will retake the shot, or re-crop it in Photoshop.

Now we have much more accurate coloring – we aren't using changecolors anymore. The representation of uncommon to exotic items is much clearer.

Notice there are 2 of each. One for the PS3 and 360, the larger ones for the current generation.

Source files are maintained just in case we end up having to reprocess them in the future to support something like large shots on our web site, or 4k resolutions.



I love how well these are working. I Like the nice clean alignments of a uniform height and width, and unlike many RPGs that have tinted approximations, we have actual representations.

There's probably even someone in this audience that could rattle off each of these items by name.

Let's talk Loc

OPTIMIZING LOCALIZATION

BUNGIE

DESTINY 

OPTIMIZING LOCALIZATION

- 40% buffer

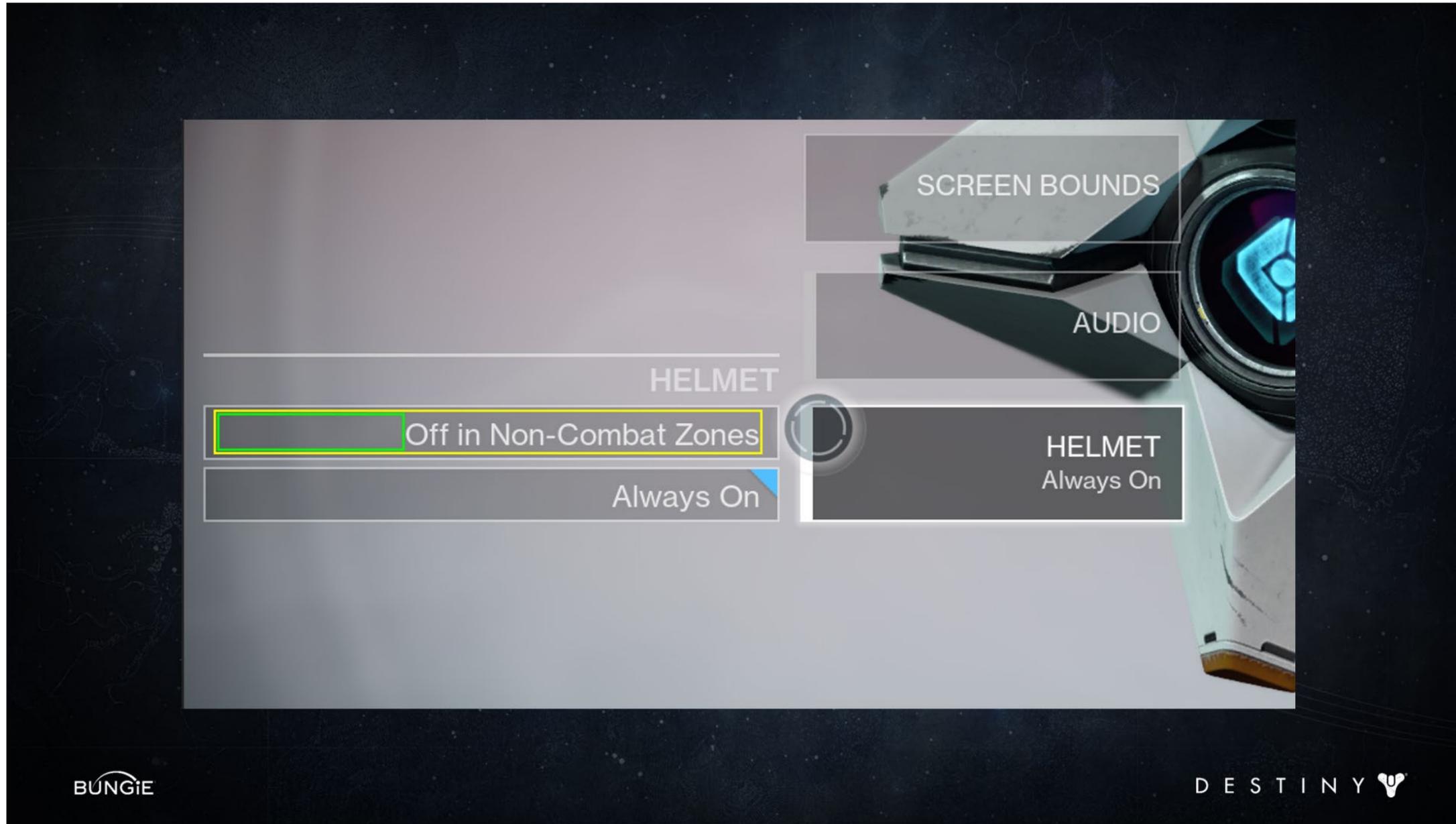
BUNGIE

DESTINY 

Destiny is translated into EFIGS (English, French, Italian, German, Spanish) as well as Portuguese and Japanese.

We have been refining the loc process over the years at Bungie. Here are some things we are doing to streamline the process. Some of these concepts may be familiar to you, but there's probably a few practices unique to our studio.

40% buffer



Just a common practice. Leave about 40% extra space for your text strings.



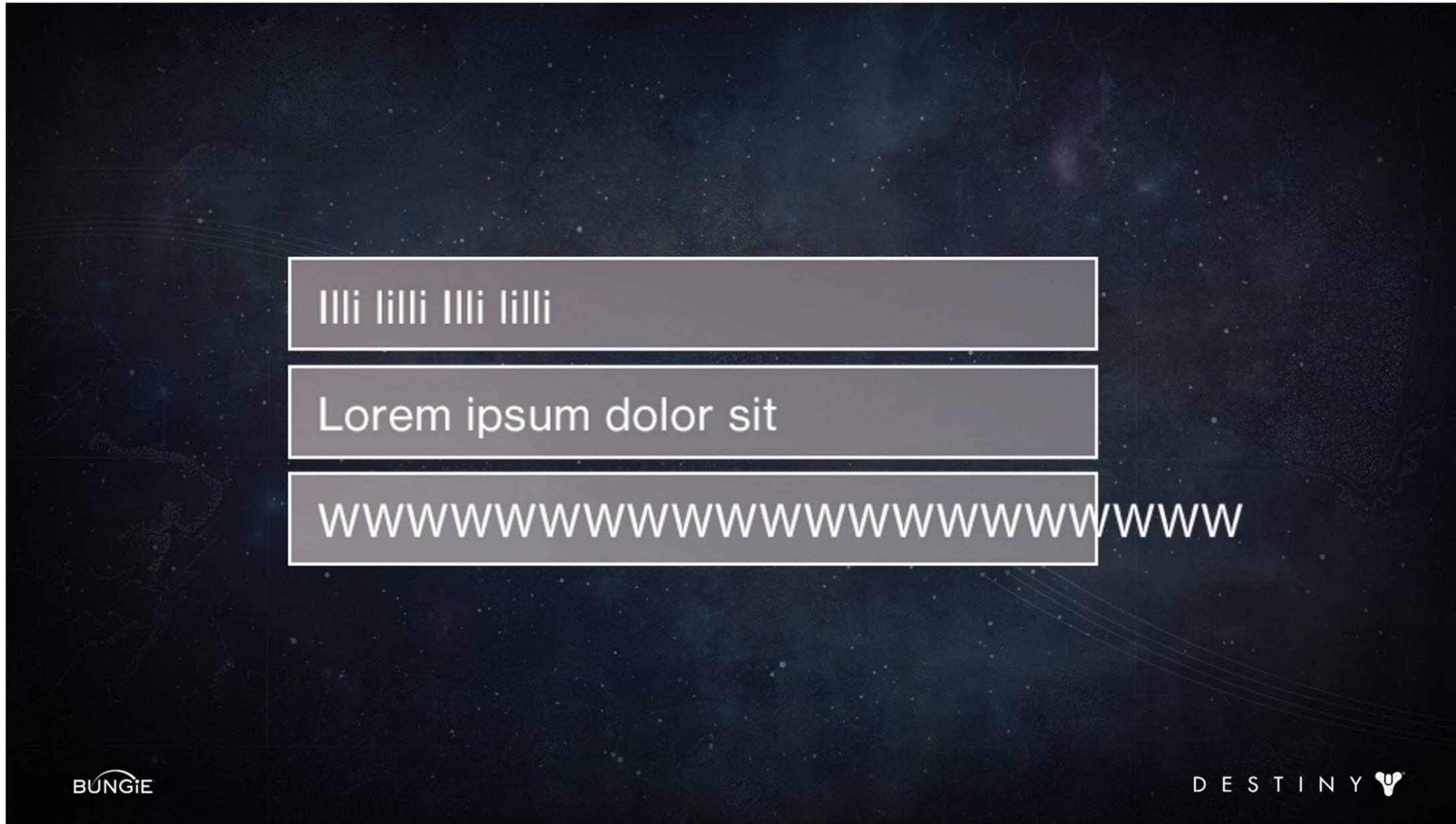
Reason being, different languages tend to fluctuate in length.

There are some languages that tend to do this more often than others, but I'm not going to name any names.

LOCALIZATION BEST PRACTICES

- 40% buffer
- Avg. Character counts

Often times writers ask for character counts.
Unless your font is monospace, you need to give an avg expected count,
give or take.



Each button has the exact same number of characters (21, in case you were counting), but if you tell the writers they can use up to 21 characters, they could potentially go over their allotted space depending how many capital Ms and Ws and how few spaces, ls, and ls there are.

It's best to give your translators a ballpark figure, leaning on the smaller side – roughly somewhere between 18 and 23 characters here.

LOCALIZATION BEST PRACTICES

- 40% buffer
- Avg. Character counts
- Easy language preview
- Avoid string concatenation

Having debug commands that allow both writers, translators, testers, and UI artists to see how the interface looks in other languages has been extremely beneficial.

Earlier on, our toolset required us to set the new language and reboot. Now we can do it on the fly, which has made the process much more painless.

String concatenation isn't inherently wrong, but when done in the content setup by a UI artist, you can run into problems.

PARTY CRASHER +1
Legendary Shotgun

🎯 **297** ATTACK
I said INVITE ONLY!

PARTY CRASHER +1
{string 1} {string 2}

🎯 **297** ATTACK
I said INVITE ONLY!

ACOPLADO +1
Leyenda Escopta

🎯 **297** ATAQUE
He dicho ¡SOLO POR INVITACIÓN!



When you replace variables, you are having a fixed sentence structure.

These are tooltips for gun. Rarity and type: "legendary shotgun"
If you concatenate 2 separate strings together, they are permanently fixed in that order.

Are there any Spanish speakers in here? I'm guessing this probably sounds wrong to you - like a certain pointy-eared muppet did the translation for us.

LOCALIZATION BEST PRACTICES

- 40% buffer
- Avg. Character counts
- Easy language preview
- Avoid string concatenation
- Editors and testers on staff
- Dev language

When we worked on Halo, we'd ship off the text strings for translation, we'd get them back and plug them into the game. This greatly increased translation times and was more error prone since they didn't have a build to look at while they were doing the work – there was no context for them.

Dev language – next page.

*String Editor ✕
Designer View

Search Paths

Filters
 Any string contains Match case?

Namespace	Namespace File Path	ID	Release Id	String	Notes	Progress	Design Time S	Writer String
hud_strings.localized_string	content\ui\common\strings\hud_s	speed_dashes		--	Used in Sparrow Racing HUD speedometer	Final	9/24/2015 4:20:49 PM	--
hud_strings.localized_string	content\ui\common\strings\hud_s	stat_incomplete_string		---:---	Scoreboard DNF	Final	9/18/2015 2:40:45 PM	---:---
hud_strings.localized_string	content\ui\common\strings\hud_s	stat_percentage_string	r1_alpha	{0}%		Final	2/27/2015 3:10:14 PM	{0}%
hud_strings.localized_string	content\ui\common\strings\hud_s	stats_button_key	r1_alpha	&l_b_button; Stats		Final	2/27/2015 3:10:14 PM	&l_b_button; Stats
hud_strings.localized_string	content\ui\common\strings\hud_s	success		Success		Final	7/2/2015 1:33:59 PM	Success
hud_strings.localized_string	content\ui\common\strings\hud_s	suicide	r1_alpha	Misadventure		Final	2/27/2015 3:10:14 PM	Misadventure
hud_strings.localized_string	content\ui\common\strings\hud_s	summon_vehicle	r1_alpha	Summon Vehicle		Final	2/27/2015 3:10:14 PM	Summon Vehicle
hud_strings.localized_string	content\ui\common\strings\hud_s	summoning_vehicle	r1_alpha	Summoning Vehicle		Final	2/27/2015 3:10:14 PM	Summoning Vehicle
hud_strings.localized_string	content\ui\common\strings\hud_s	super_multiplier_string	r1_alpha	{0}x		Final	2/27/2015 3:10:14 PM	{0}x
hud_strings.localized_string	content\ui\common\strings\hud_s	super_ready	r1_alpha	Super Ready	Consider something else. Super ready sounds dorky.	Working	2/27/2015 3:10:14 PM	Super Charged
hud_strings.localized_string	content\ui\common\strings\hud_s	switch_off		Turn Off		Final	2/27/2015 3:10:14 PM	Turn Off
hud_strings.localized_string	content\ui\common\strings\hud_s	switch_on		Turn On		Final	2/27/2015 3:10:14 PM	Turn On

Dev language is a language sku just like Spanish, Portuguese, and Italian. This language, however, that does not ship.

Why do we have a Dev Language?

When UI artists are building pages, they need to plop in placeholder strings so the page structure is filled out. Sometimes these strings are so straightforward, that they don't change by the time we release the final game, and, as I explained earlier, some things we make up on the fly.

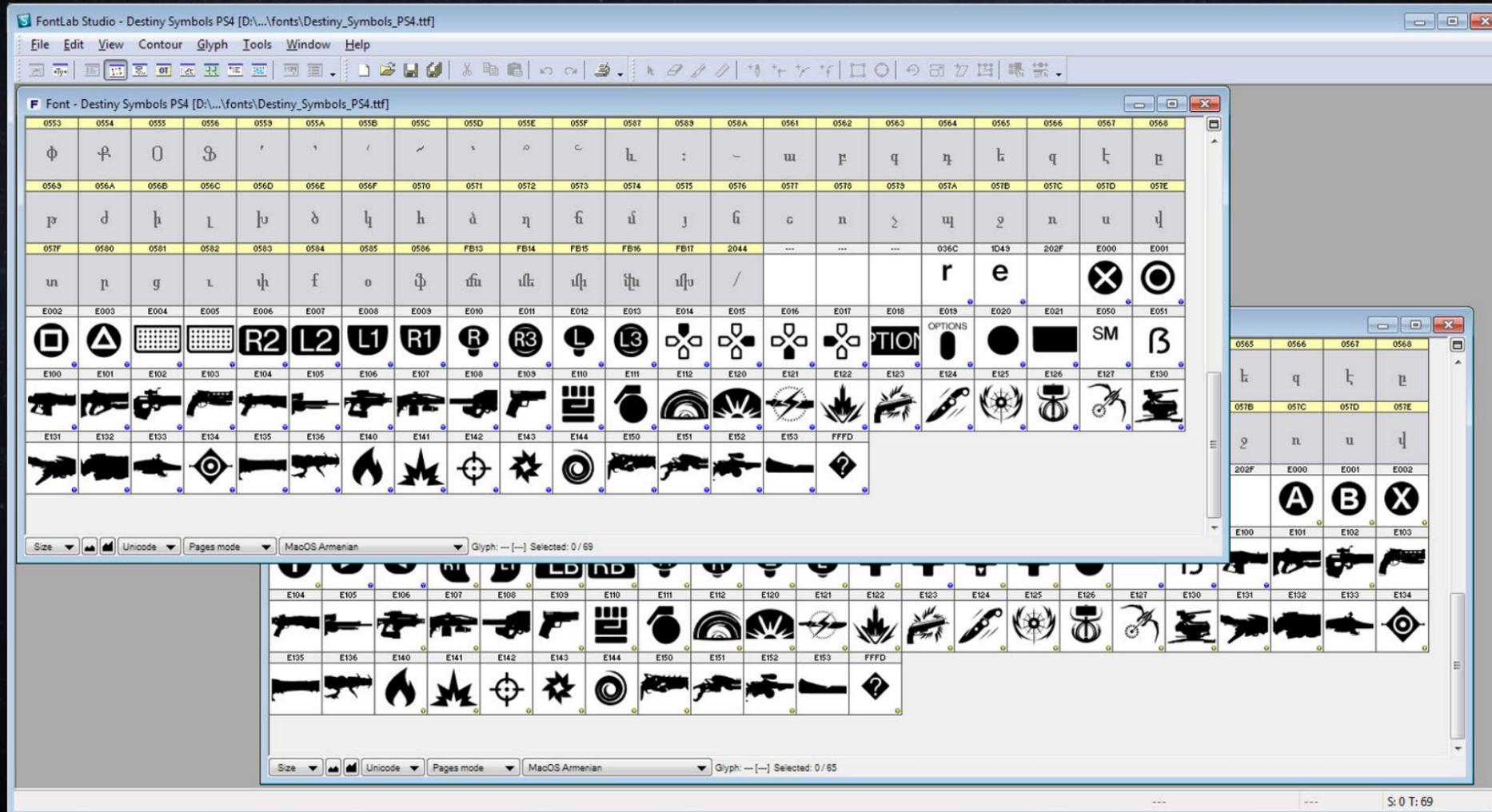
Sometimes they are an inside joke, like changing "Inverted" to "perverted." But even when the information is totally accurate but we want to have our writers vet them for grammar, terminology, and fictional consistency. So we avoid shipping placeholder strings.

Dev strings get the idea across, and if there is anything we want to add, we can add notes about our intent, so the English writers understand the purpose of that string, and if there is any formatting that needs preserved. Our email alias is also available in case they have questions.

LOCALIZATION BEST PRACTICES

- 40% buffer
- Avg. Character counts
- Easy language preview
- Avoid string concatenation
- Editors and testers on staff
- Dev language
- Choose the right font

Many fonts are missing international characters. This can make localization difficult without selecting a different font for other languages. This may be inevitable for Asian fonts.

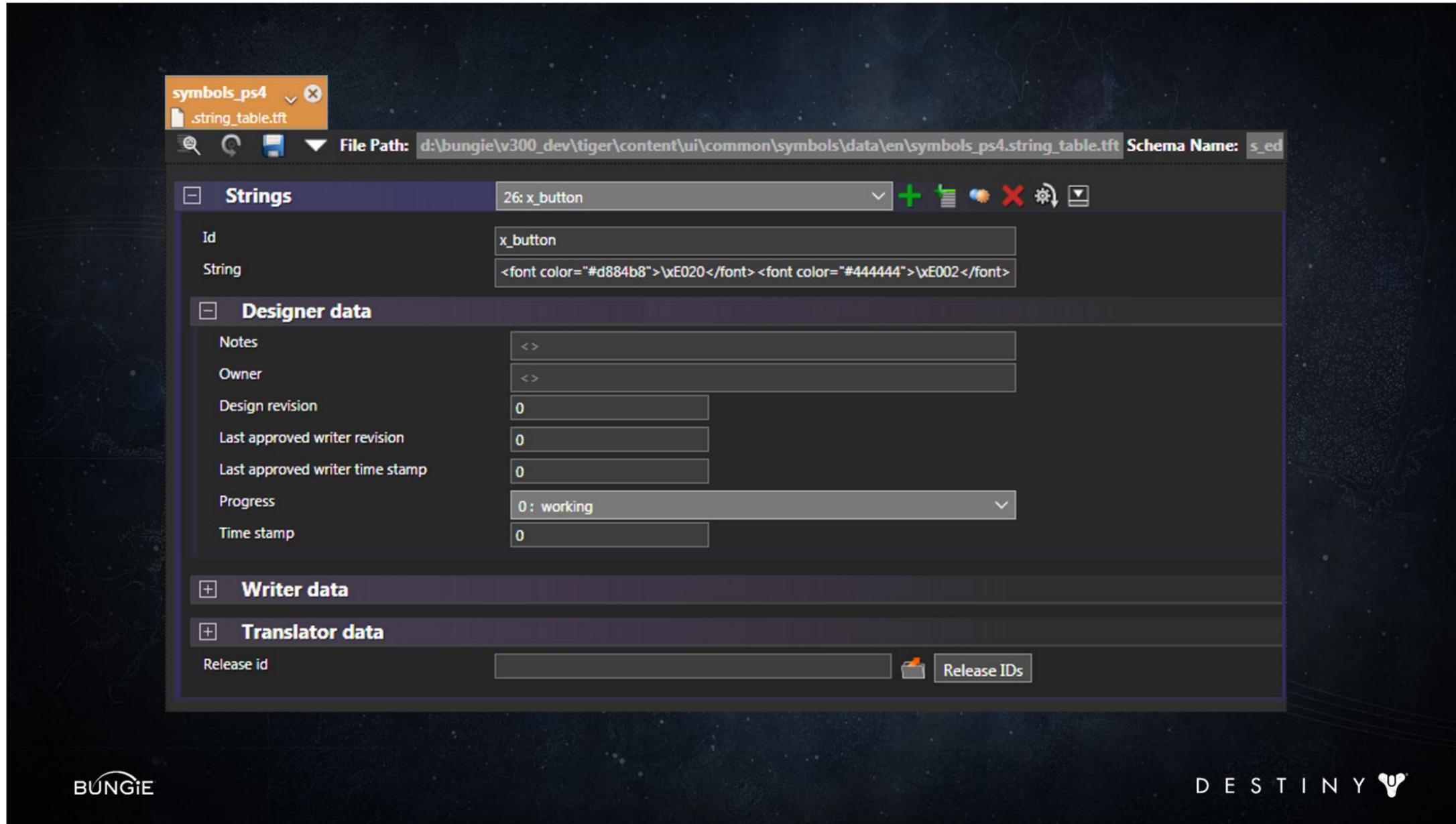


What we did was find the font we wanted, then at runtime the string parser checks font for characters. If it can't find it, it then searches our custom font for the missing characters.

Also notice the button glyphs. Our fonts are rendered on the fly as TrueType, and any glyphs in the font are vector based true type glyphs.

We used to render all fonts into bitmaps, so each character and button icon needed maintained any time we added a new font, added a new size, etc. Using TrueType gives us infinitely more flexibility since the font and glyphs are rendered at runtime.

Unfortunately, there's no way to specify more than one color in any single text character. To get around this, we came up with a clever solution...

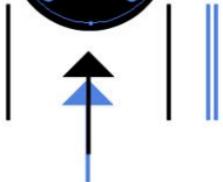


For our own unique dingbats and buttons, writers specify unique string hashes. Those refer to a tag where we use markup language and Unicode references to define the character and its coloring. For multicolored buttons, a single hash references a string that contains two characters and each one is assigned a color.

Hold   to interact



Hold  to interact



Of course, normally this would produce two characters side-by-side.

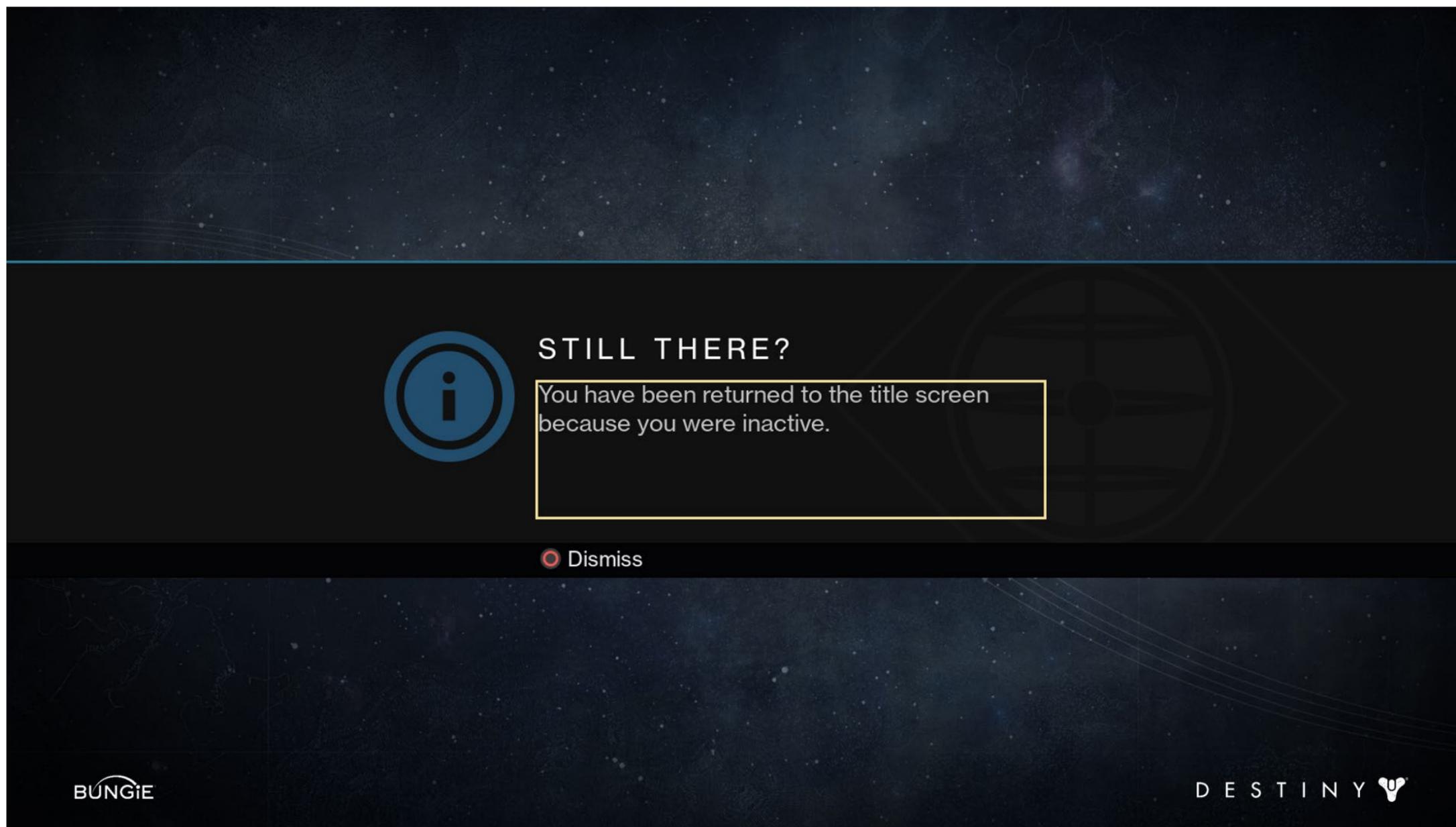
However, when we author the font, the character that provides the secondary color is intentionally misaligned and has a width of 0 so it lines up directly behind the character preceding it.

So – this tech allows us to specify font sizes, custom glyphs, and international characters at runtime without touching the original font.

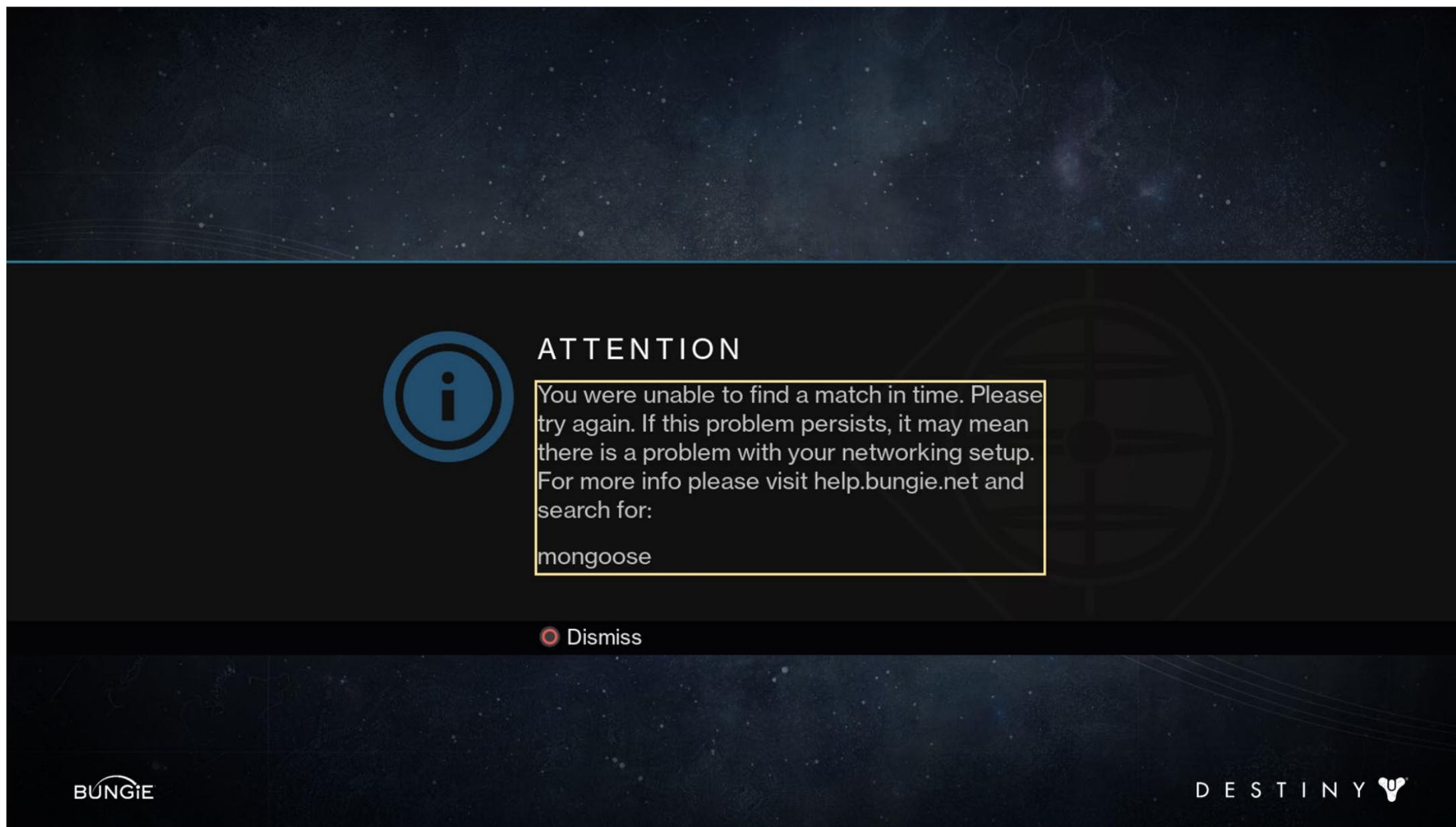
LOCALIZATION RELEASE VALVES

- 'Scale to fit' text fields

All things considered, no system is perfect. Sometimes writers go over the recommended space, sometimes the layout doesn't allow for much flexibility. These are some ways we add flexibility into the localization process.



We set up a text field and give it plenty of space, however, if more is required, we give it flexibility.

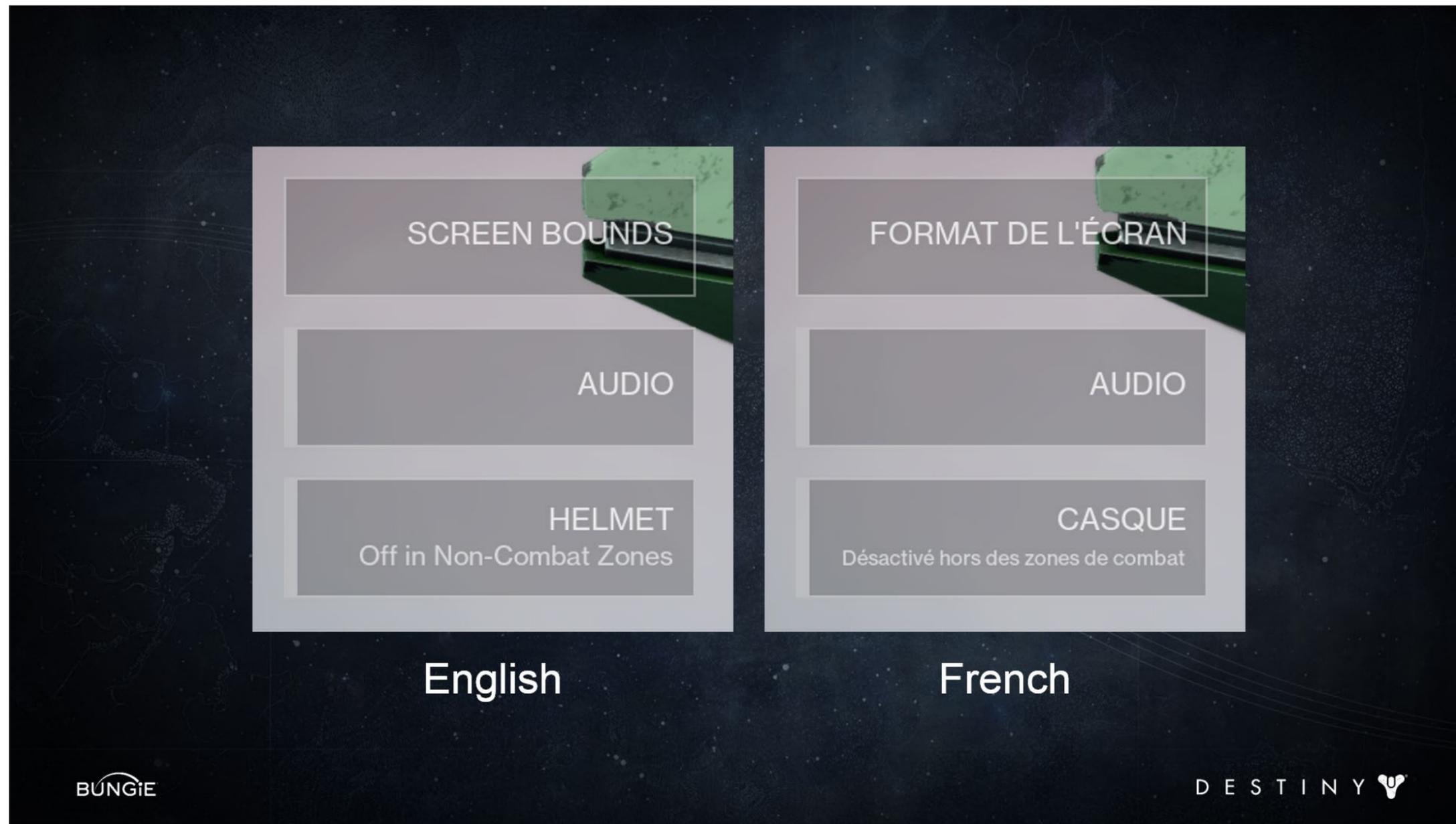


The vertical height is calculated on the fly, and if the content doesn't fit, we can push the field out taller. The height of that field is calculated, then that value is factored into the rest of the screen, so all the other elements can adjust accordingly.

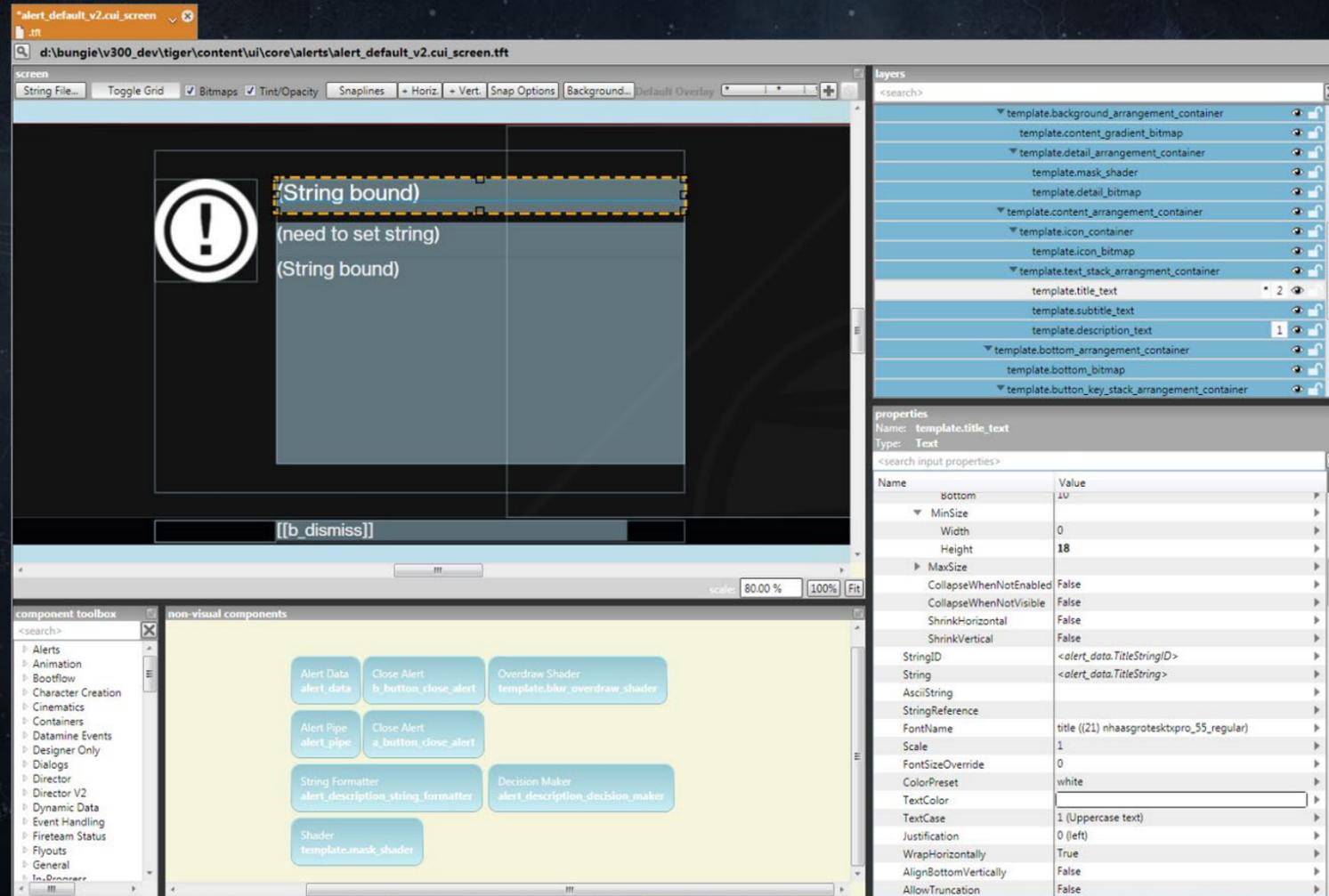
LOCALIZATION RELEASE VALVES

- 'Scale to fit' text fields
- Font scaling to fit single line text fields

Another way we use 'scale to fit' is the font itself. This is one of the great things about using trueType faces that render at runtime.



These are two shots of some buttons from our settings menu. As you can see on the bottom one, one phrase ended up being written longer than we anticipated. The translation into French didn't quite fit, so the text field is flagged to scale the font down until it does.



This is our internal tool, Bonobo. The middle panel is the preview area, and the upper right is the list of elements. A text field is selected here. Below that list in the bottom right is the attributes of the text field we have selected.

When a single line of text has 'scale to fit' specified, we are able to set a minimum text size here, to insure it never scales too far down. However, if that isn't going to work, we are able to take even more drastic measures.

LOCALIZATION RELEASE VALVES

- 'Scale to fit' text fields
- Font scaling to fit single line text fields
- Allow truncation

The truncation setting cuts strings off prematurely if they don't fit, and adds an ellipsis at end.

Portuguese



Batalha sem fim

A história da sua participação em Assaltos da Vanguarda contra a Treva.

1 Assaltos da Vanguarda

German



Nimmerendende Schlachten

Die Geschichte davon, wie der Hüter auf Vorhut-Strikes ging und so half, die...

1 Vorhut-Strikes

We are not currently using this setting anywhere, but it works like this: It fills up as much of the field as it can, leaving space for an ellipsis. This works well when the same information can be accessed elsewhere, rather than having the player lose that data.

LOCALIZATION RELEASE VALVES

- ‘Scale to fit’ text fields
- Font scaling to fit single line text fields
- Allow truncation
- Scrolling text

Lastly, there's Scrolling text – if all else fails, and the text simply won't fit and you absolutely must see it all.

There are a few cases of this in the game where we implemented a new feature late in the development process, and there was no time remaining to safely rewrite and translate the strings that went in these brand new locations.



(Video)

Look at this field, it's really tight. Now if I change language to Portuguese, the text increases in length and begins to scroll.

It's not the most elegant thing we've done, but sometimes, when all else fails, you need to use a little brute force.

DESIGNING THE DIRECTOR

BUNGIE

DESTINY 

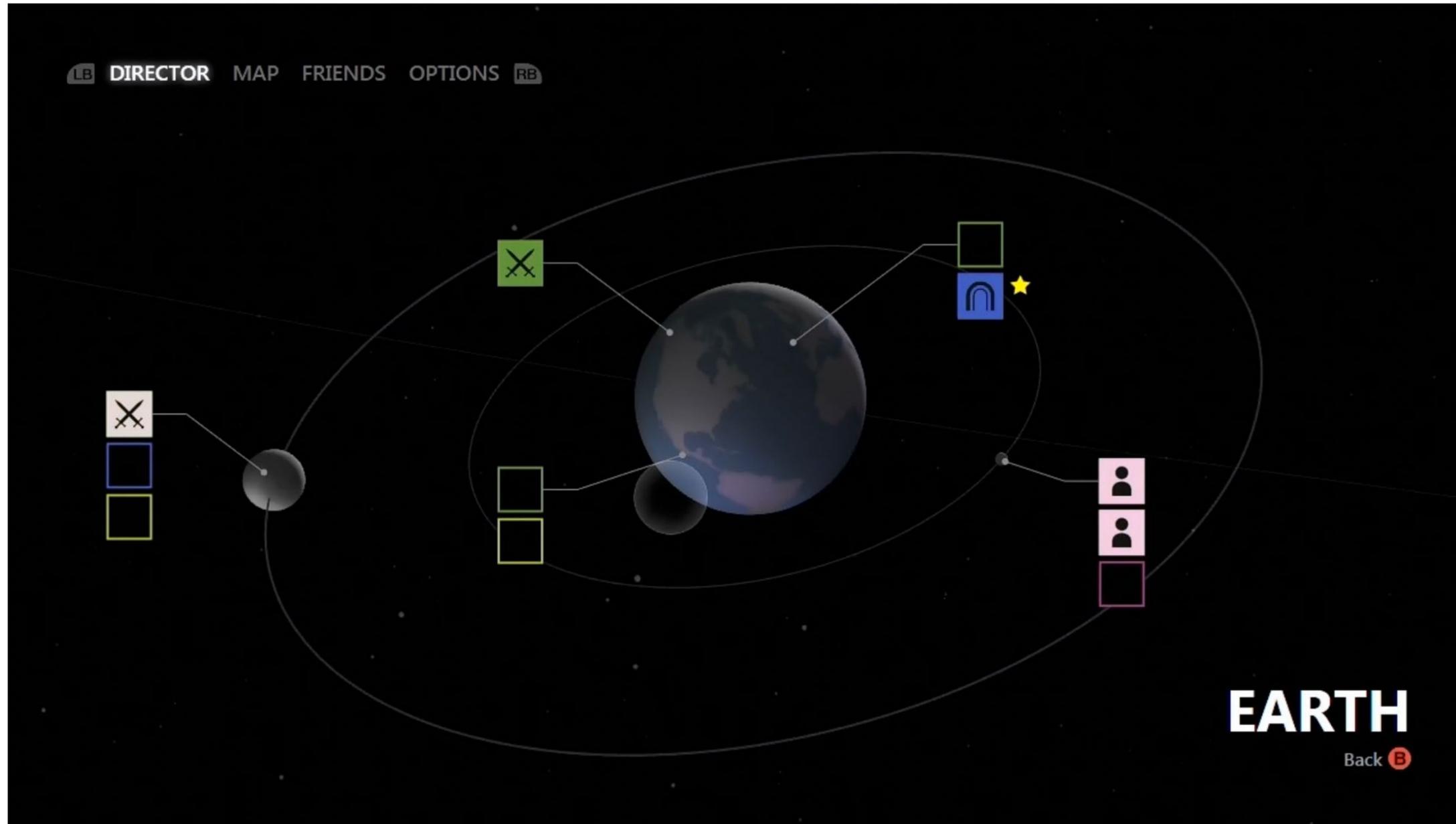
Our last topic of the day is the Director.

THE DIRECTOR

- Create an always available, always populated list of starting points for adventure.
- Incentivize players to be activity omnivores.
- Offer direction to inexperienced players.

The Director is the first piece of UI we prototyped for Destiny back in 2010.

Jason gave us the following goals to work from:



(Video)

Keep in mind this was a very early mockup, before we had any locations or story figured out. The design was based on the solar system and exploration.

However, this version had issues:

- There was too much digging in and out
- Everything equal, what should I do next?



We were expecting a lot of our Halo players to try playing our game, many of them just want to shoot other dudes, where is that?

Though we "make the games that we want to play" we had to be mindful of those of us in the studio that just like to play deathmatch. Eventually, we just started calling those types of players, "Todd, the drunken frat boy." How does he get to play what he wants?



We started iterating.
We tried lots of ideas.

Trying to design the UI while the rest of the game was being designed concurrently, plans would change daily. Anyone in game development knows that's just how it goes. Sometimes we'd create problems for ourselves by trying to solve for things like two destinations on one planet, or the gas giants, that didn't even make it into the sipping game.



So trying to not get hung up on the details was a challenge. Some designs didn't make it past b/w motion comps, some made it all the way into the engine.



We had about 8-9 major revisions, each one solving the issues the last one had.



We finally landed here. We had streamlined the UX, solved lots of issues, made it easy to do what you want and do so efficiently.

...BUT we had totally sucked the cool out of the most important piece of UI in the game. Looking back, the solar system was so neat, it provided a sense of exploration, wonder, and dozens of other intangibles.

This was an Important lesson we learned – when iterating and improving, you need to look at ALL past revisions and not the most recent one.

Jason was the first to recognize this and call it to our attention. And while we'd accomplished his first set of goals, he gave us a few more to really clarify the important things in this design.

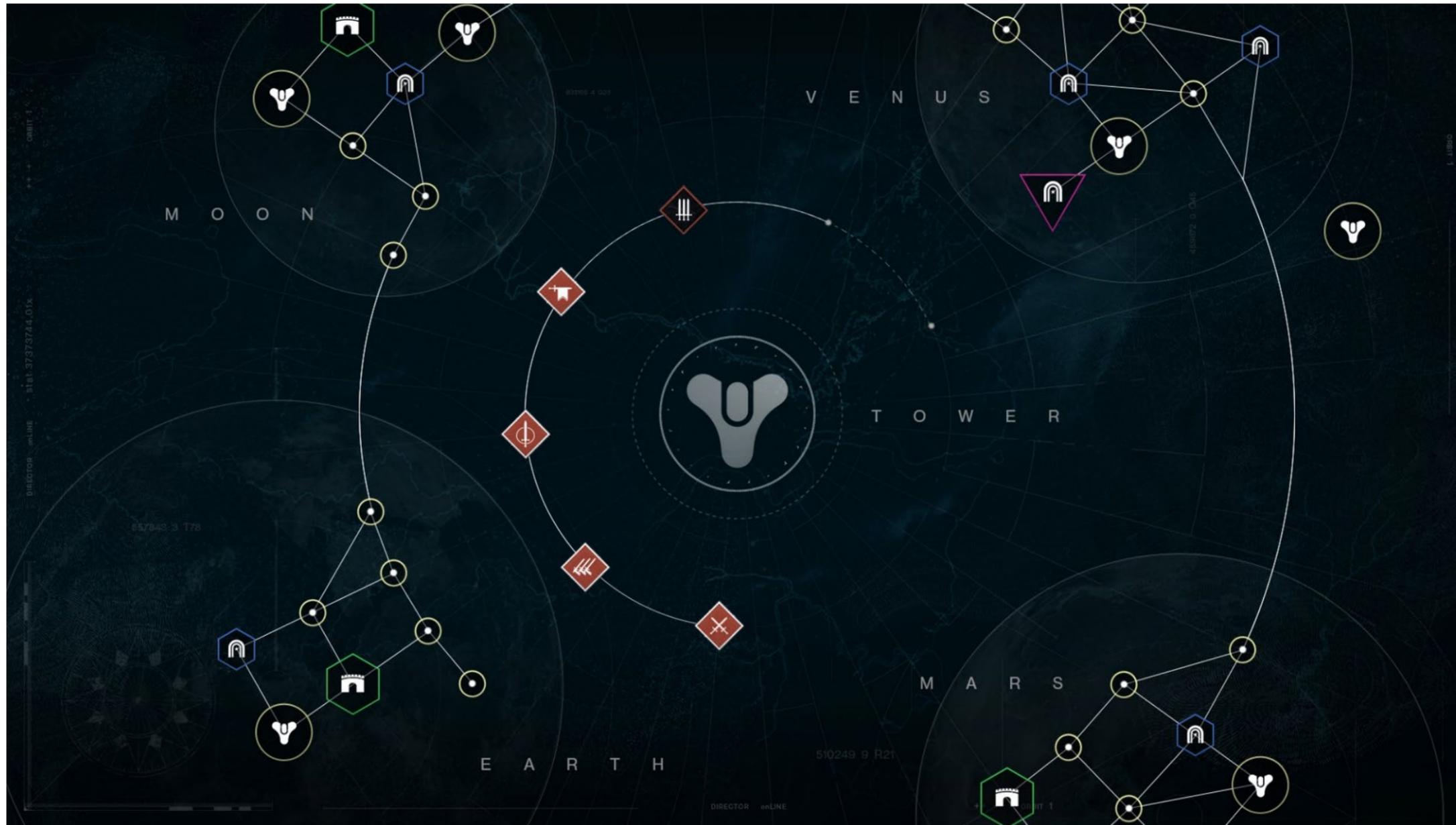
THE DIRECTOR

- Create an always available, always populated list of starting points for adventure.
- Incentivize players to be activity omnivores.
- Offer direction to inexperienced players.
- Offer a sense of place.
- Show you your past & future, provide a sense of accomplishment and aspiration.
- Display dependencies between activities.

It was the 11th hour, we had a few months to ship, so we went for it, and changed the director one more time.

We knew it was right, but we were very cautious. This was our last chance, and we had to do it right.

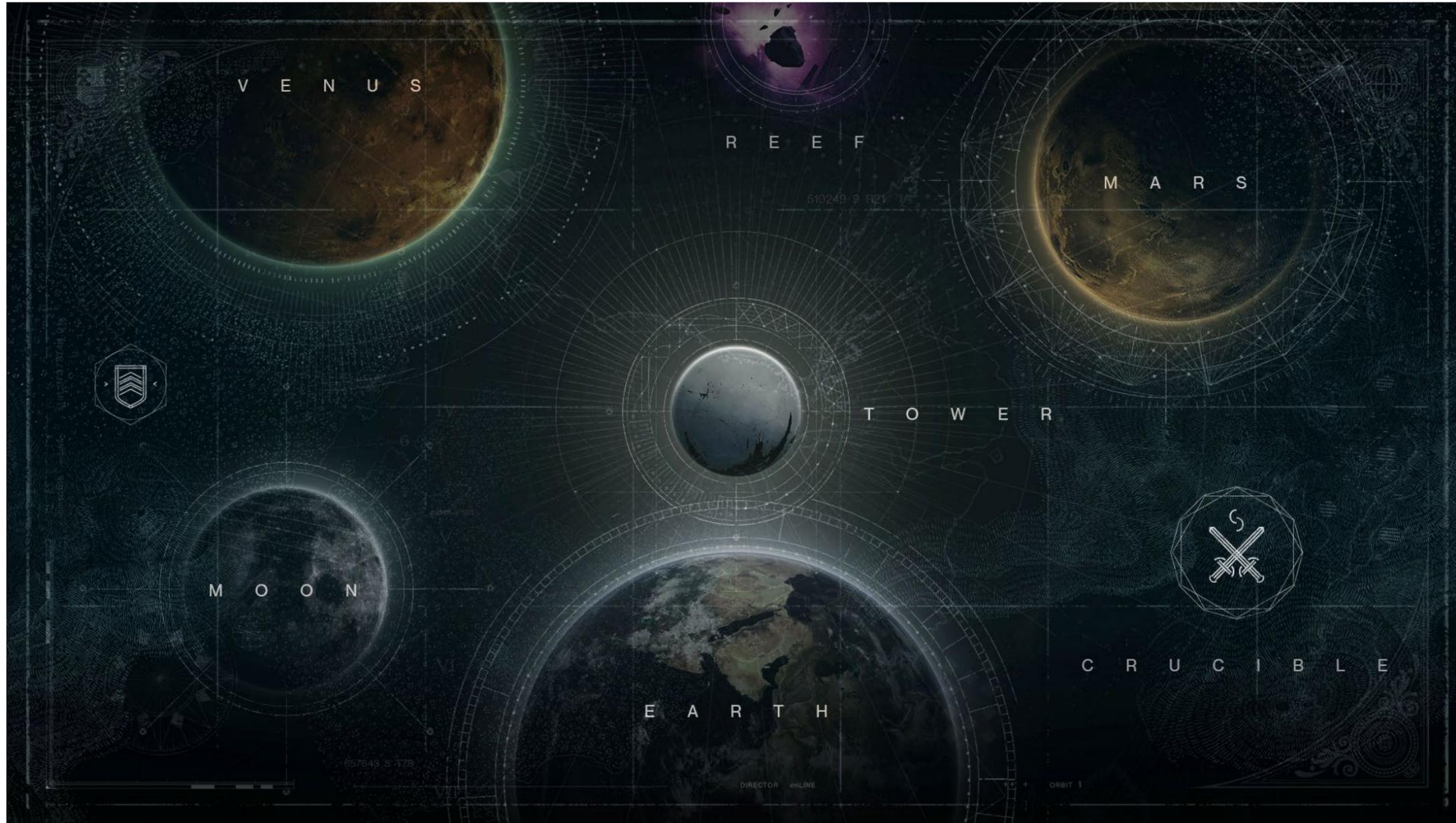
We went through several visual iterations...



...and built a working prototype as rapidly as possible.

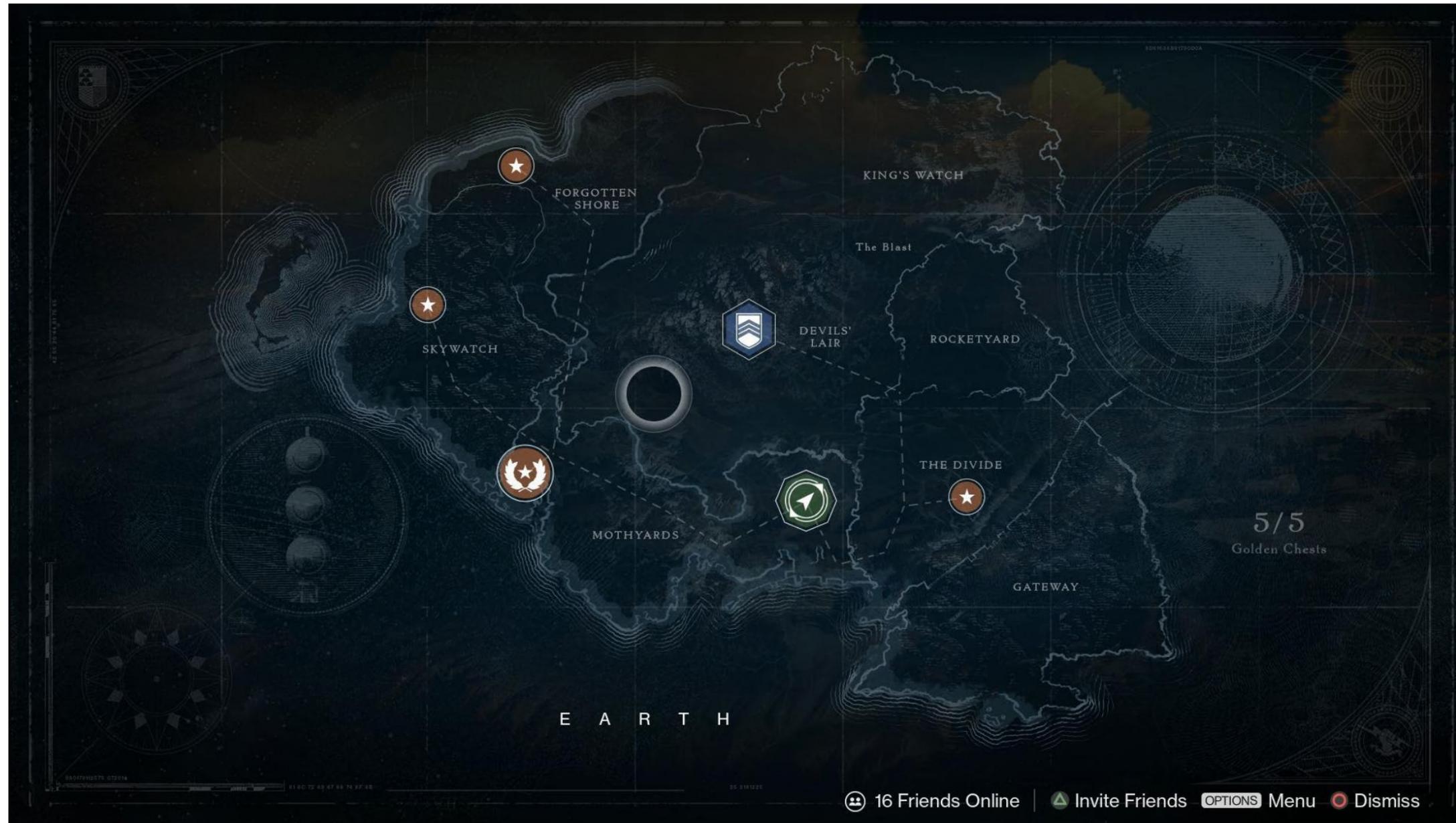
We pilfered a guy from the web team to build a rapid prototype using HTML, CSS & Java script.

We were completely out of time. The specs were mostly finished and most details figured out, so we started executing. It's not optimal, but we had to move.



In the end, I'm very happy with result, love the design, and the solutions we arrived at.

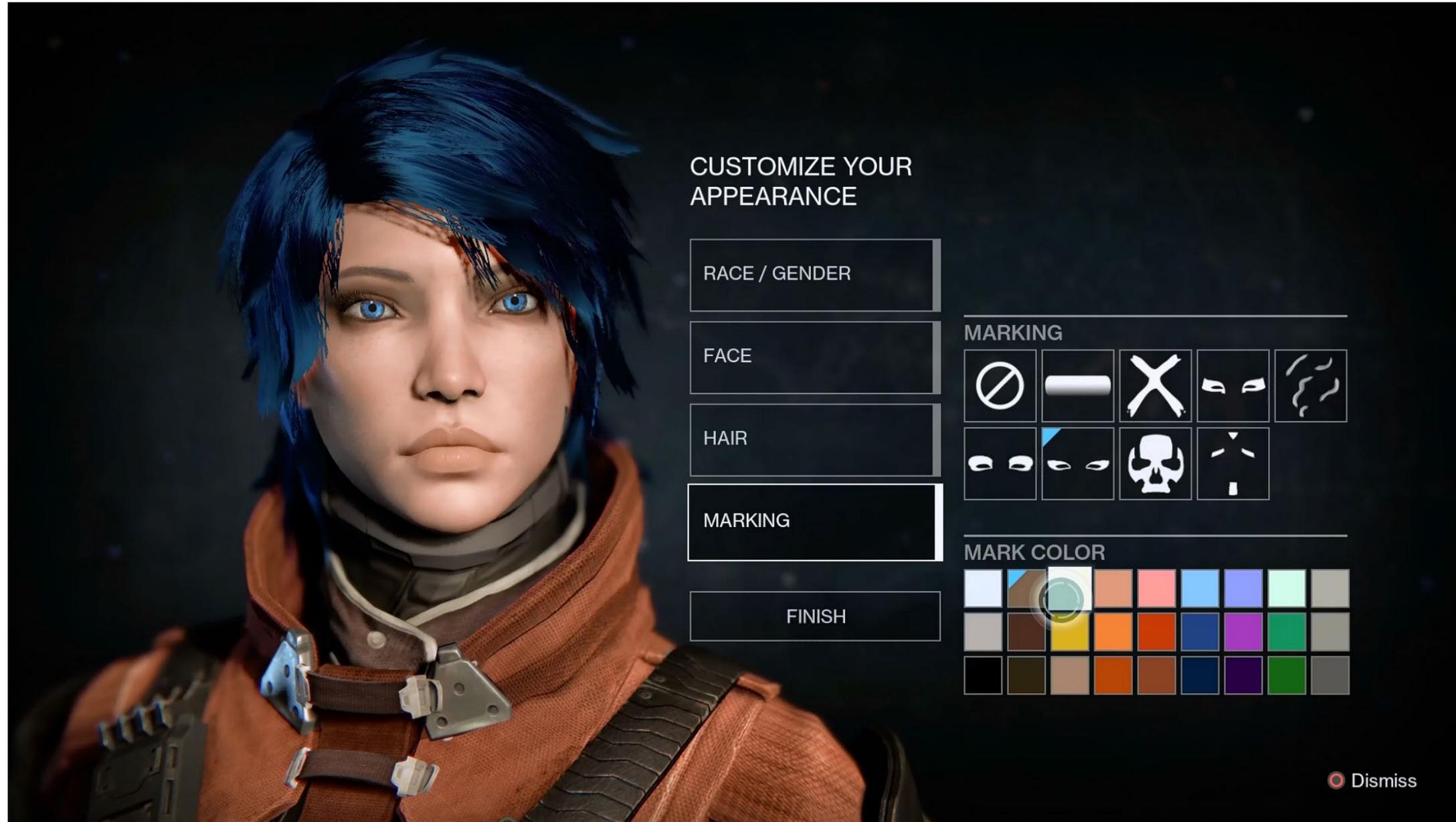
The visuals referenced ancient cartography in a sci-fi setting.





So I've talked about our UI and shown you stills and bits and pieces. I'd like to wrap things up by showing you a brief video tour that showcases the various elements of our UI in action so you can see how the things that I've talked about today all come together.

This is what ended up sticking to the wall.



(Video)

CONCLUSION

I hope this insight into our design process has helped. Maybe there's something specific you can take away in regards to the free cursor, Our Localization Practices, Creating Gear icons, and Designing The Director.

Our processes are constantly changing. We're always looking for ways to improve the UI in Destiny. To make it simpler and better for users to find the fun.

QUESTIONS?

David Candland

Twitter: @drcandland

XBL: Evil Otto

Feel free to follow me on twitter or Xbox Live

At this point I'll take questions.

W E ' R E H I R I N G



WWW.BUNGIE.NET/CAREERS
CAREERS@BUNGIE.COM

fin.