GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

Mission Kontrol: Revolutionizing Combat Testing for Mortal Kombat 1

Etienne Palmer-Campbell
Gameplay & Tools Engineer
NetherRealm Studios

NETHERREALM STUDIOS

#GDC2025

Thank you everybody for coming today.

In a challenging fighting game such as Mortal Kombat with thousands of unique inputs, how can you ensure that all your developers are capable of independent in-game testing?

Today I want to demonstrate why you don’t always need to be good at video games to work on them. I’ll do that by telling the story of Mission Kontrol: a tool that revolutionized combat testing for Mortal Kombat 1.

My name is Etienne Palmer-Campbell. I’m a Gameplay and Tools Engineer working at NetherRealm studios.

Content Warning

MATURE 17+
M
ESRB

Blood and Gore,
Intense Violence

Right up front I’d like to give a content warning because this presentation includes some gameplay footage from Mortal Kombat 1 which is rated mature for Blood, Gore, and Intense Violence.

MORTAL 11 KOMBAT
UNREAL ENGINE
MORTAL 1 KOMBAT

Background

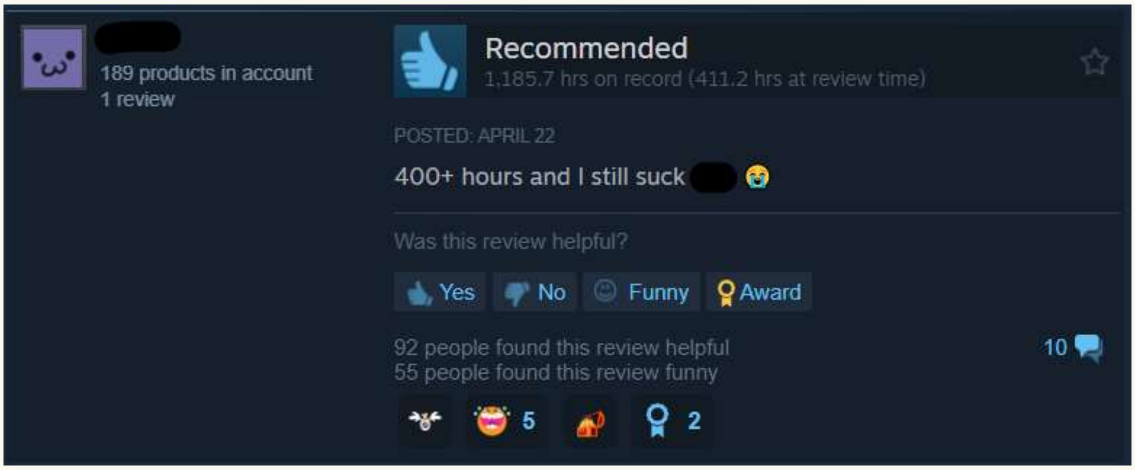
I'll start with a little background on myself and the games we make. I've been with NetherRealm since 2018 at the start of my career, which was midway through the development of Mortal Kombat 11 *[click]*. My time on that project was mostly spent providing support to other gameplay engineers while I learned the ropes and handled minor gameplay features.

After finishing work on MK11 and it's DLC, I moved on to some prototype work in Unreal Engine 4. *[click]* This was a precursor for the engine migration we did when transitioning from Mortal Kombat 11 to our latest project: Mortal Kombat 1.

[click]
During the Mortal Kombat 1 project, some of my primary engineering responsibilities included: work

for the Combat AI, Tutorial features, and creating debug combat testing tools, which is what I'll be talking about today.

So, the context for this presentation is going to be about fighting games made in Unreal Engine, but the engine components I reference do carry forwards into the latest version of UE5 and I believe the takeaways can be applied more broadly than fighting games.


[redacted]
189 products in account
1 review

Recommended
1,185.7 hrs on record (411.2 hrs at review time)

POSTED: APRIL 22

400+ hours and I still suck [redacted] 😭

Was this review helpful?

Yes No Funny Award

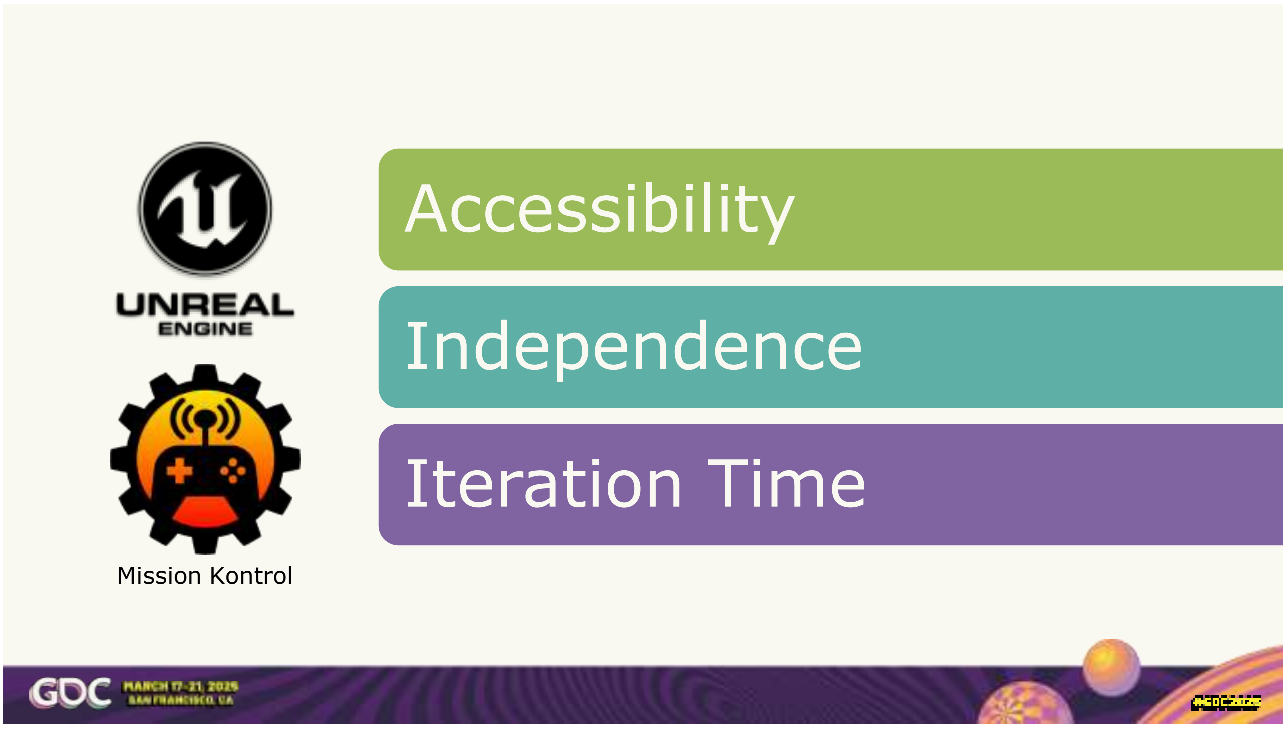
92 people found this review helpful
55 people found this review funny

10

5 2

The fighting games we make at NetherRealm are known for how challenging they can be. Players often need to invest a lot of time practicing their skills before feeling comfortable with the controls. I’ll hold off on the details of our game for now, because what I’m getting at is that developers are not an exception to this!

Anything that makes your game harder to play also makes it harder to work on. This difficulty is often increased further by the game always changing during development. We use tools to mitigate that problem.



Accessibility

Independence

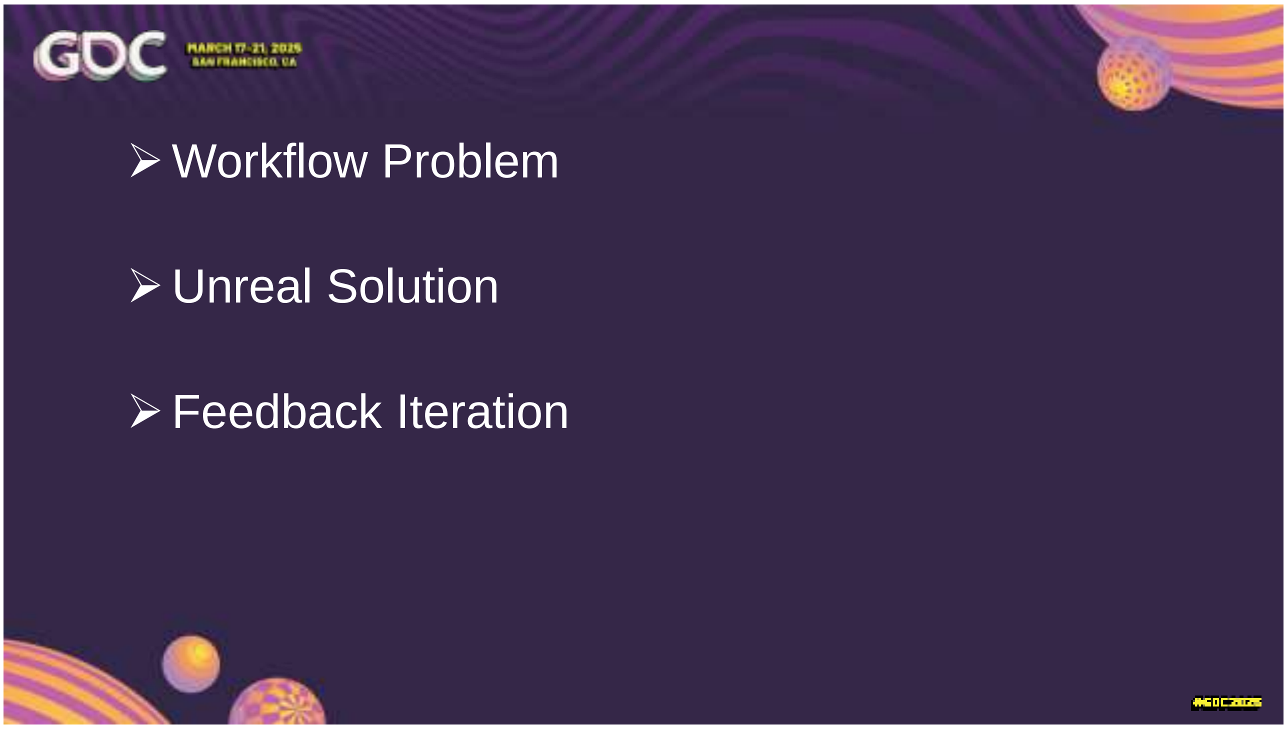
Iteration Time

Mission Kontrol

GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

What I’m going to show you today is how to create specialized gameplay testing tools within the Unreal Engine – as well as what this can do to promote [click] accessibility, independence, and reduced iteration times for your whole team when performing gameplay feature testing.

Maintaining the stability of a shared game project is a shared responsibility, so it’s important for every developer to have what they need to test their work efficiently.


GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

- Workflow Problem
- Unreal Solution
- Feedback Iteration

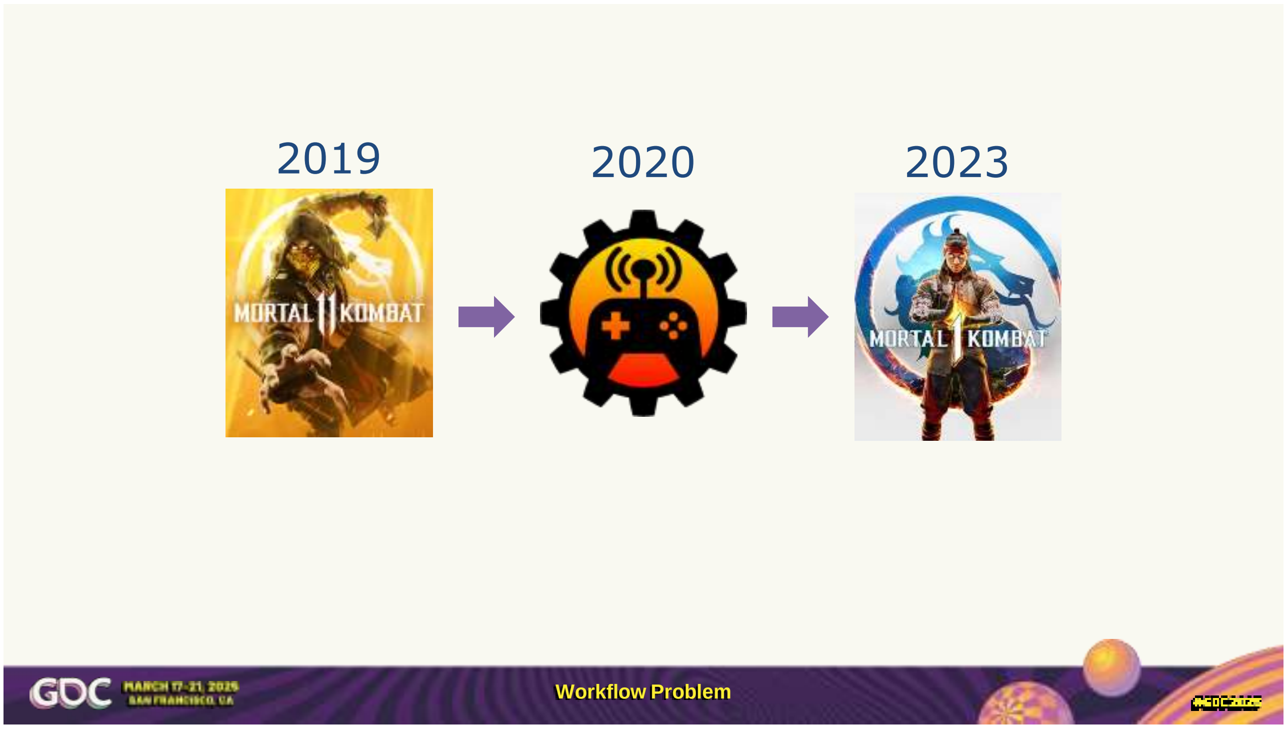
#GDC2025

I'll start by describing the problem we identified and why it was important to address.

Then I'll talk about how and why we decided to solve that problem using only the tools available within the game engine.

Finally, I will show you the iteration and feedback process we went through to evolve the tool into what it is today.

[click]
Let's begin with some more context about that original workflow problem.

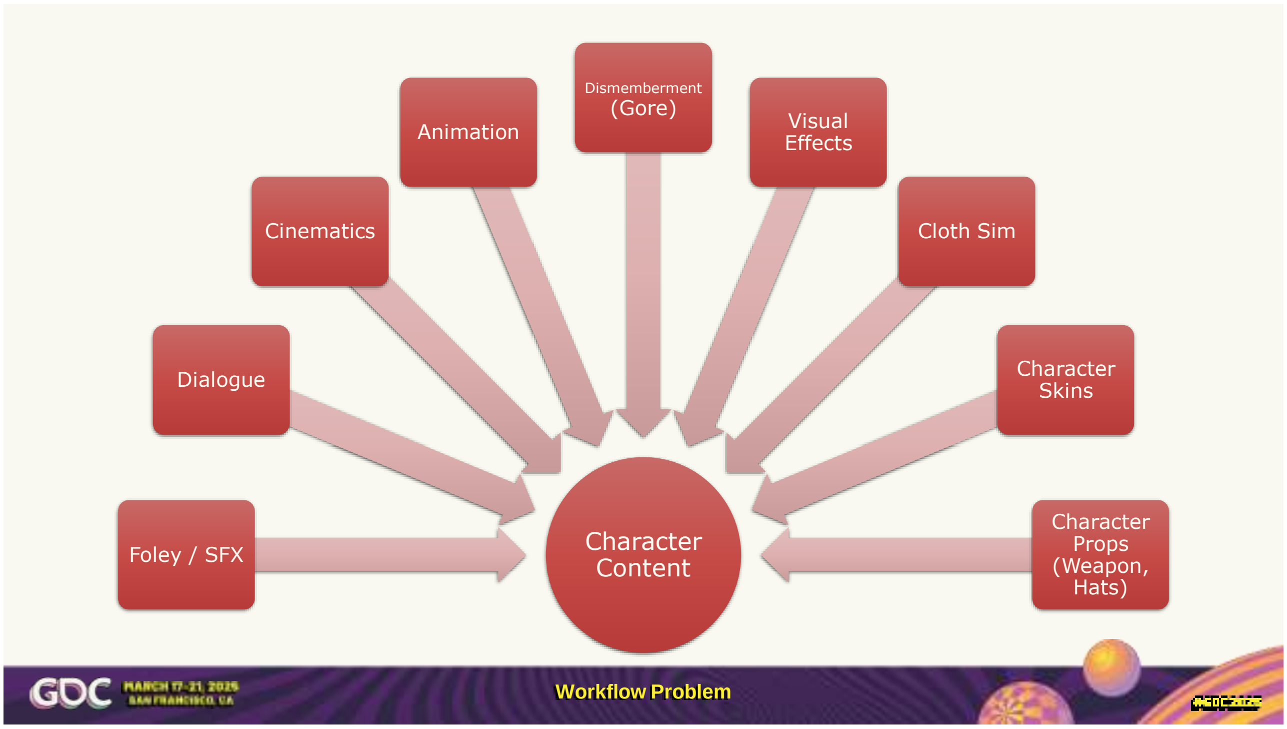

2019
2020
2023
MORTAL 11 KOMBAT
MORTAL 1 KOMBAT
GDC MARCH 17-21, 2025 SAN FRANCISCO, CA
Workflow Problem
#GDC2025

The Mission Kontrol project started in 2020, during the transition from Mortal Kombat 11 to Mortal Kombat 1.

For those of you who are unfamiliar with the gameplay of modern Mortal Kombat, let me introduce you with a trailer for one of our recently released DLC characters. This will help set the stage for understanding our problem space.

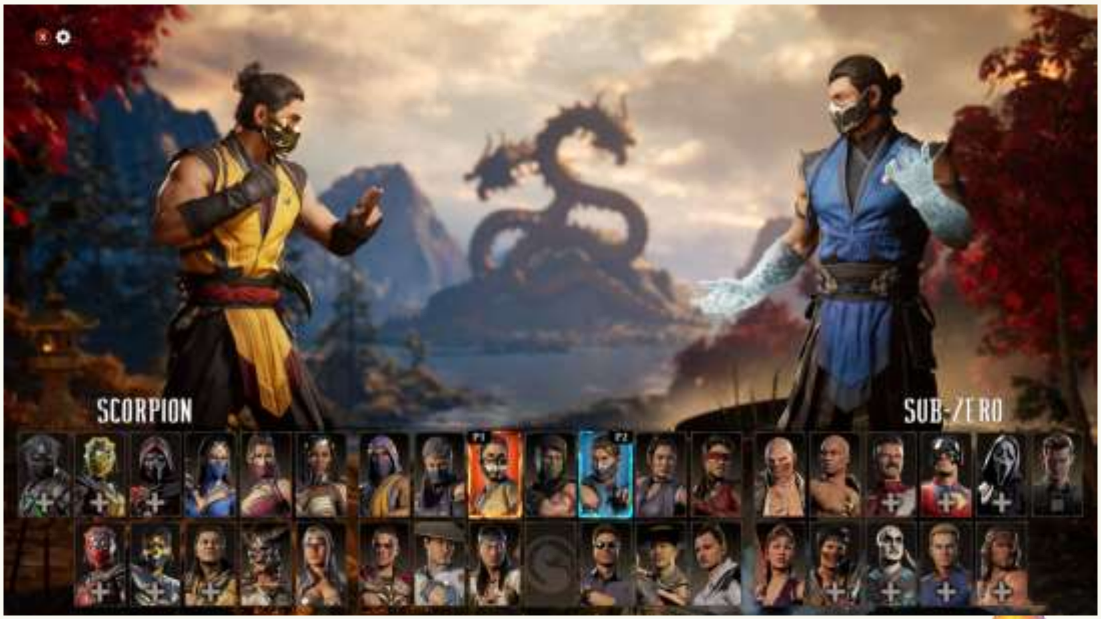
I want to demonstrate how much effort goes into a character. While watching, keep track of the different types of content the characters demonstrate. Particularly during the cinematic sequences towards the end.

[Wait for trailer to finish]



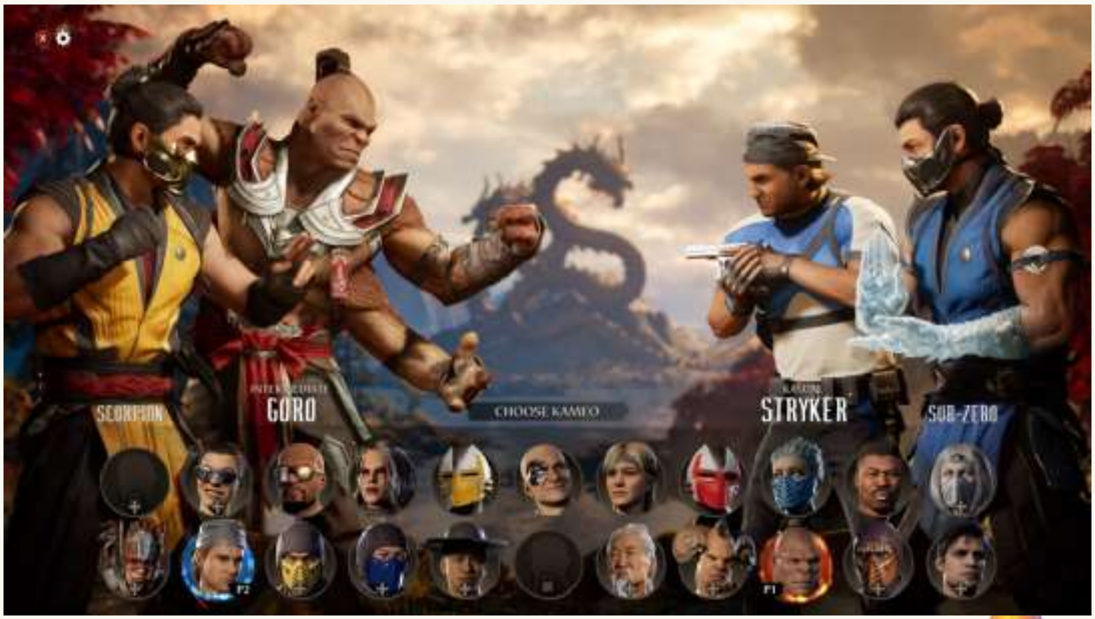
As you can see, a single character in our game requires a lot of polished content from a wide variety of disciplines. Like any fighting game, our core gameplay revolves around these characters, which means everybody in the studio needs to be able to rapidly iterate on it.

One of the most expensive features shown at the end of the trailer is the Fatalities. These in-engine cinematic sequences live at the core of the Mortal Kombat identity, each character has several of them along with many other similar finishing moves like Brutalities or Fatal Blows. This type of content exercises every one of those systems.



Workflow Problem

When you multiply that by our full roster of characters, you end up with a mountain of content. MK1 launched with 23 playable main characters, with 12 additional characters released as DLC to date. The latest DLC is being released this month, T1000 from Terminator 2.

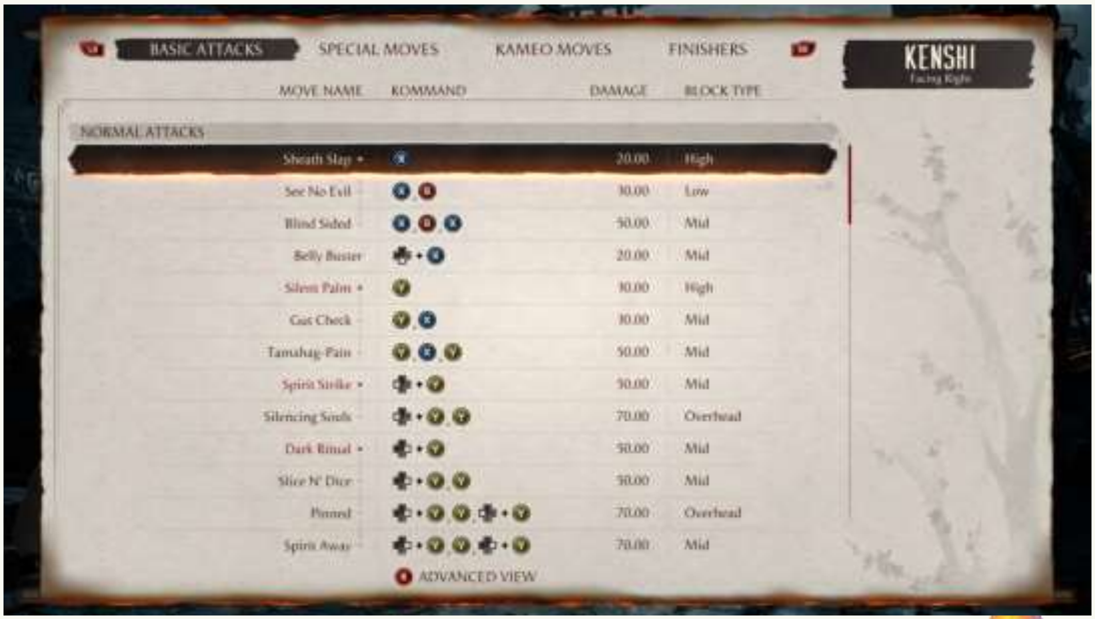


Workflow Problem

This game also introduces Kameo fighters. These are essentially sidekick characters whose abilities get added on to your main character and are featured heavily during the cinematic moments.

These don't add quite as much content as a full character, but it's still comparable. It also introduces a lot more permutations to worry about when characters interact with each other.

So the problem is all the content we've discussed so far is hidden within gameplay, and as I mentioned earlier, our games are designed to be challenging.



MOVE NAME	KOMMAND	DAMAGE	BLOCK TYPE
NORMAL ATTACKS			
Sheath Slap •	X	20.00	High
See No Evil	X, B	30.00	Low
Blind Sided	X, B, X	50.00	Mid
Belly Buster	✚ + X	20.00	Mid
Silent Palm •	Y	30.00	High
Gut Check	Y, X	30.00	Mid
Tamahag-Pain	Y, X, Y	50.00	Mid
Spirit Strike •	✚ + Y	50.00	Mid
Silencing Souls	✚ + Y, Y	70.00	Overhead
Dark Ritual •	✚ + Y	50.00	Mid
Slice N' Dice	✚ + Y, Y	50.00	Mid
Pinned	✚ + Y, Y, ✚ + Y	70.00	Overhead
Spirit Away	✚ + Y, Y, ✚ + Y	70.00	Mid

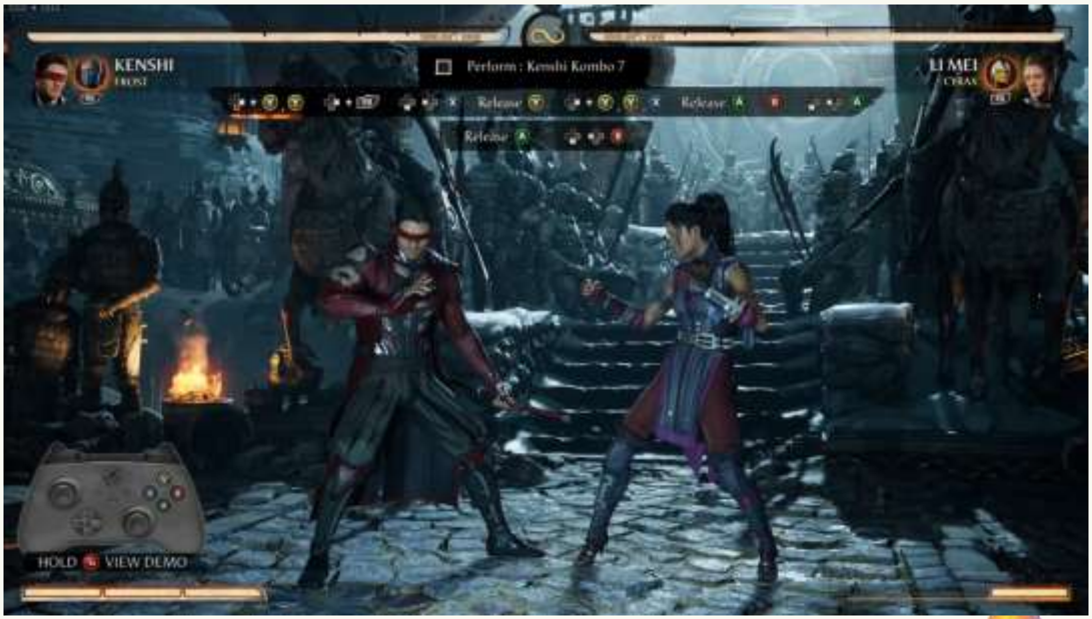
B ADVANCED VIEW

Workflow Problem

That core gameplay loop always involves a lot of complex button combos which vary for every character across increasingly large rosters. Each of those characters has their own unique set of attacks which we call a move list. Each of those attacks, including the Fatalities and other cinematic sequences, is mapped onto a unique button combination. Some characters have upwards of 100 different moves.

Players need to memorize and be able to execute at least a few of these to play the game effectively. Many players will only pick a single character to focus on growing their skills with, but as a developer I may end up working on all of them.

While this move list screen is helpful, during development it always may not be available or accurate until the later stages of the project, which leaves us to track down this information on the fly.

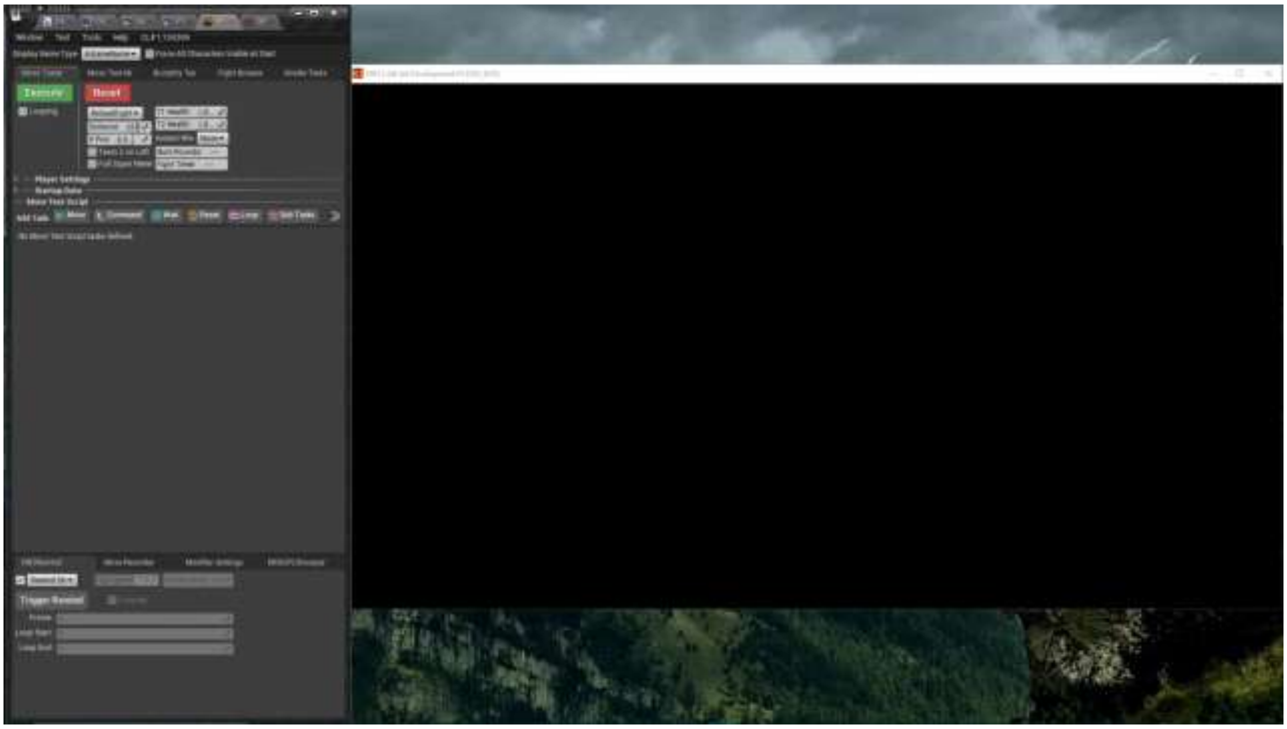


Workflow Problem

The quantity of moves isn't the only issue here. One of the pillars of fighting games is that these many attacks can be strung together to create combo sequences, which often requires careful timing and challenges your dexterity. This becomes an accessibility problem.

For example, take a look at this combo challenge. This is single player content designed to practice and test advanced skills. The controller in the lower corner is a live input display.
[play clip]

As one of the developers responsible for this feature, I need a way to consistently reproduce complex interactions to validate features like this. While I do enjoy fighting games, let's just say my engineering skills are more relevant than my fighting game skills, so I’m not well practiced enough to get this right on my first try. But when I’m fixing bugs, I want to get it right on my first try, every time, and I want it to be easy.

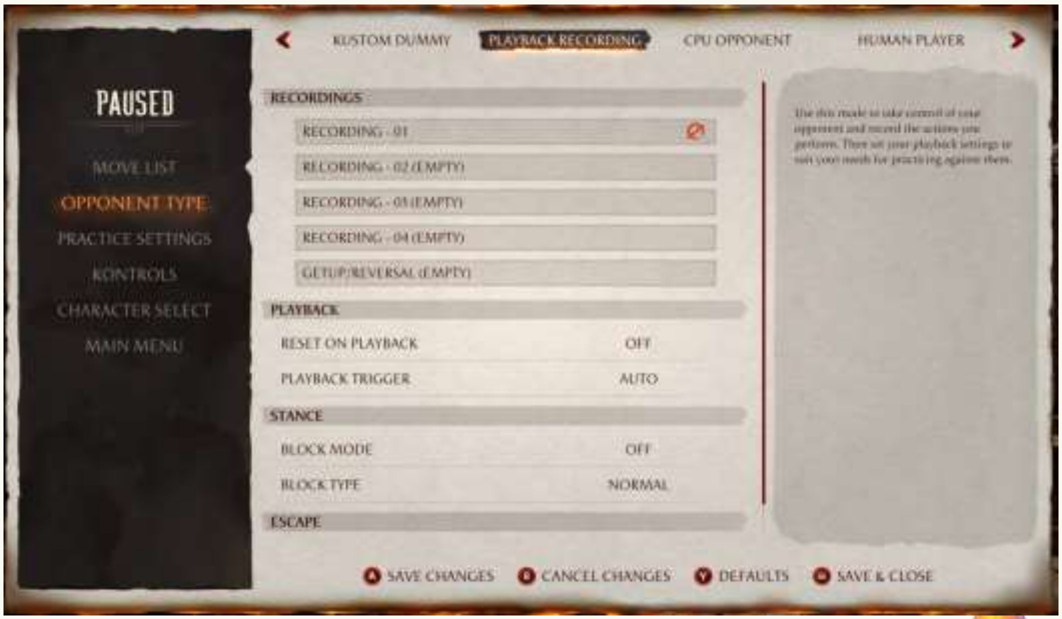


Introduce Mission Kontrol. This is a testing tool made for automating combat gameplay, intended to be accessible for any developer. It provides a remote interface for users to view character move lists and request the automated execution of those moves. This clip is a preview of the latest version of the tool. I’m using it to replicate a resurrection scenario that you may recognize from the gameplay trailer we watched.

[play video]

One of Conan’s moves allows him to come back from the dead, and it requires a little bit of setup to make it happen. Mission Kontrol makes it simple to reproduce that scenario repeatedly by automating the inputs, as well as putting the game into the correct state beforehand.

If I didn’t have this tool and I didn’t know how to perform any of those moves myself, I might have to ask somebody from the design or QA team to come to my desk and help me reproduce.



PAUSED

MOVE LIST
OPPONENT TYPE
PRACTICE SETTINGS
KONTROLS
CHARACTER SELECT
MAIN MENU

KUSTOM DUMMY | PLAYBACK RECORDING | CPU OPPONENT | HUMAN PLAYER

RECORDINGS
- RECORDING - 01
- RECORDING - 02 (EMPTY)
- RECORDING - 03 (EMPTY)
- RECORDING - 04 (EMPTY)
- GETUP/REVERSAL (EMPTY)

PLAYBACK

RESET ON PLAYBACK	OFF
PLAYBACK TRIGGER	AUTO

STANCE

BLOCK MODE	OFF
BLOCK TYPE	NORMAL

ESCAPE

Use this mode to take control of your opponent and record the actions you perform. Then set your playback settings to suit your needs for practicing against them.

SAVE CHANGES · CANCEL CHANGES · DEFAULTS · SAVE & CLOSE

GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

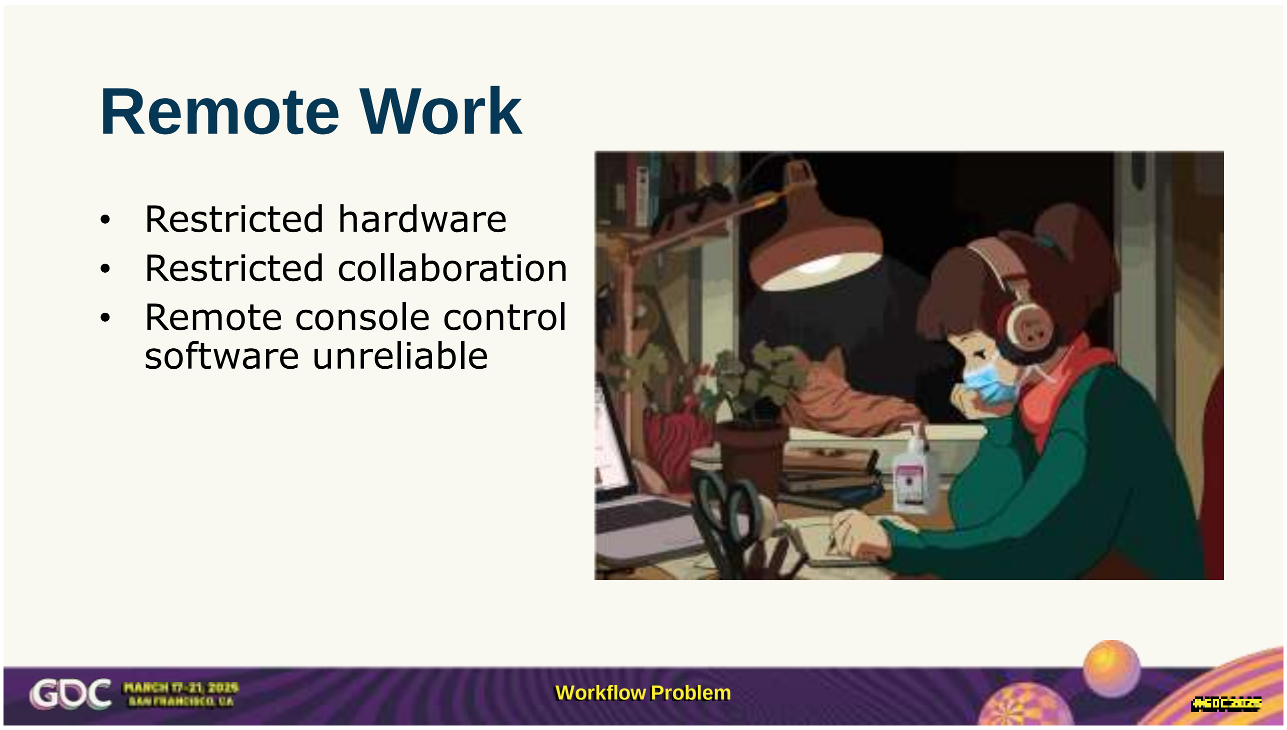
Workflow Problem

#GDC2025

Before Mission Kontrol, we did have other tools that helped with this type of problem. One of those tools is input recordings. Our games typically feature a training mode where you can capture and play back your own inputs. This is invaluable for training, and for local testing. There are also first party console developer tools that can capture and playback raw inputs before they get to the game.

The downside here is that it’s not independent if you don’t know how to perform the move. It requires cooperating with somebody who knows how to do it, making it a two-person job.

So, going into this project we already had enough reason to investigate more efficient workflows.



Remote Work

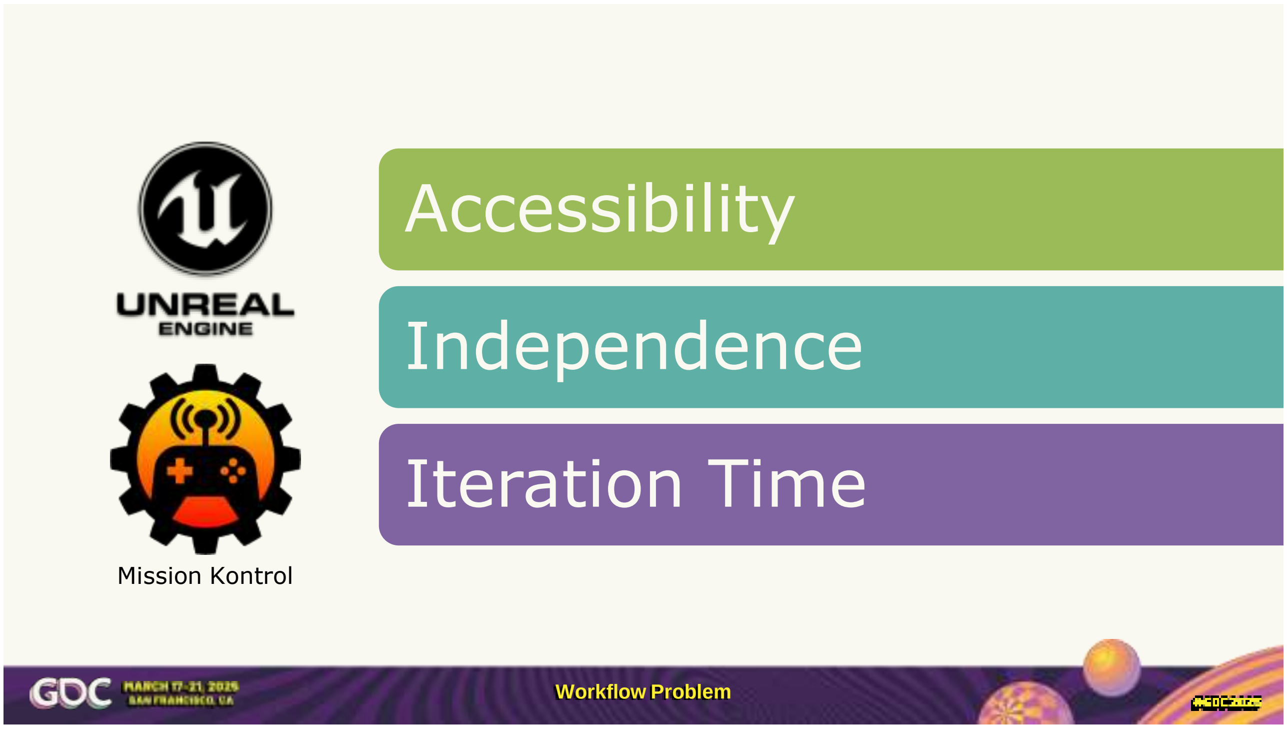
- Restricted hardware
- Restricted collaboration
- Remote console control software unreliable

However, just as we began researching this tool at the start of 2020, the United States entered lockdown for the COVID-19 pandemic, and the workflow problem grew larger.

Everybody started working remotely, making it difficult to collaborate on reproducing issues. Depending on the hardware they were able to take home, many devs suddenly did not have physical access to their devkits (which are at the studio) or the ability to use a physical controller to remotely play the game on consoles.

The problem grew from “What buttons do I press?” into “How do I press buttons?”.

The existing solution for this includes first party remote control software for the playstation and xbox platforms, which allow you to interact with the console remotely. However, these tools are not always as reliable or easy to use for testing as a local connection. It gets the job done most of the time, but it doesn’t make it any easier. This has continued to be relevant as working from home has become more commonplace.



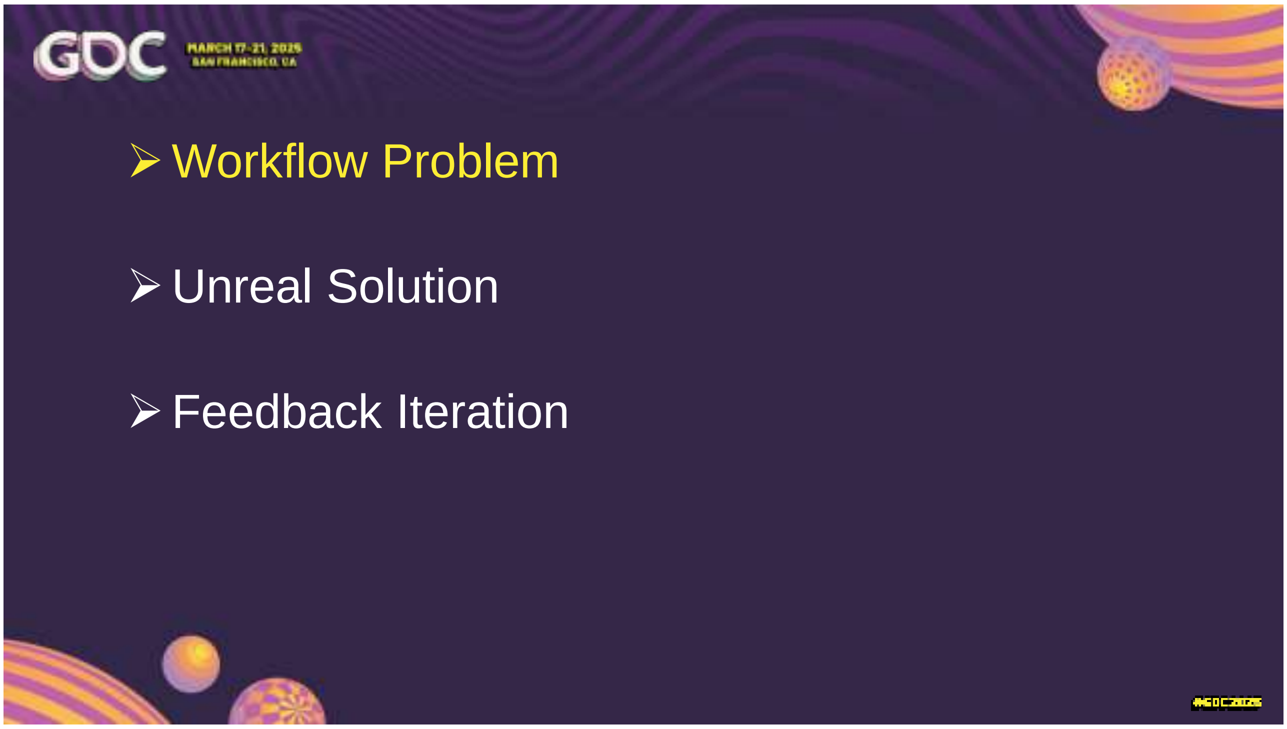
Accessibility

Independence

Iteration Time

Mission Kontrol

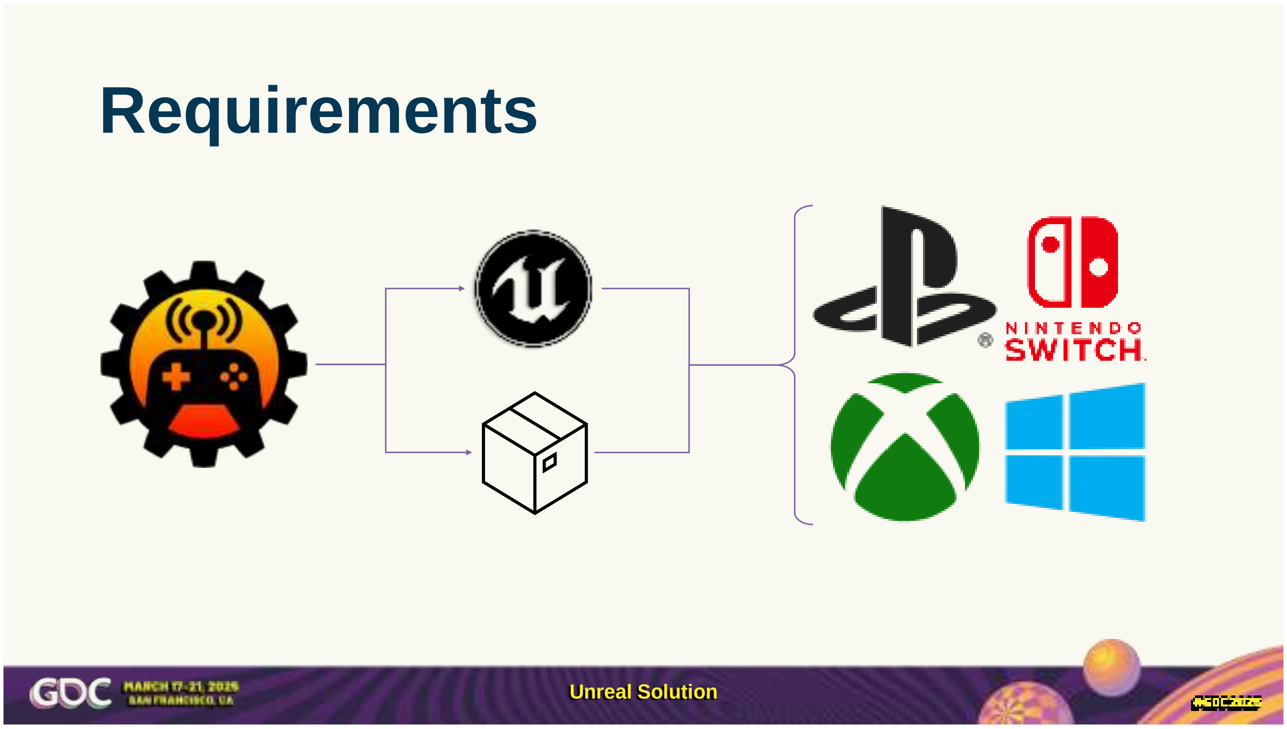
So, taking a step back, you can see how each of those aspects of our workflow are impeded by the nature of our game and work environment. The problem of input complexity may not be relatable for developers in every genre. However, the message here is that if there’s anything interfering your ability to efficiently test core features, then it’s a problem worth investigating.



- Workflow Problem
- Unreal Solution
- Feedback Iteration

So, we know what the problem is. [click]

How did we leverage the Unreal Engine to solve it? The first step was plotting out the requirements and putting together a high-level view of the components. I’ll go over our overall plan, as well as run through some technical details specific to Unreal Engine.

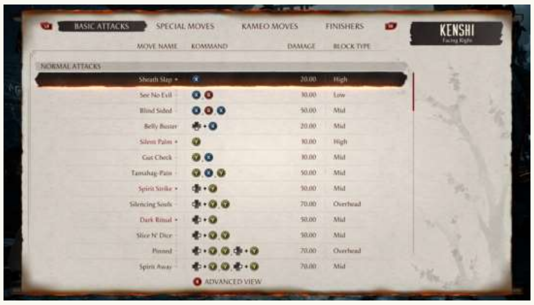


Requirements

At the application level, we knew that we needed:

- [click] The option to have a portable application, not dependent on Editor. This makes the tool easier to use for certain workflows that don't always need the full editor, such as engineers, QA testers, or external teams who don't have access to the Editor.
- [click] We also needed to connect to game instances on any of our release platforms, including those which are managed by external teams such as Nintendo Switch and PC ports.

Requirements

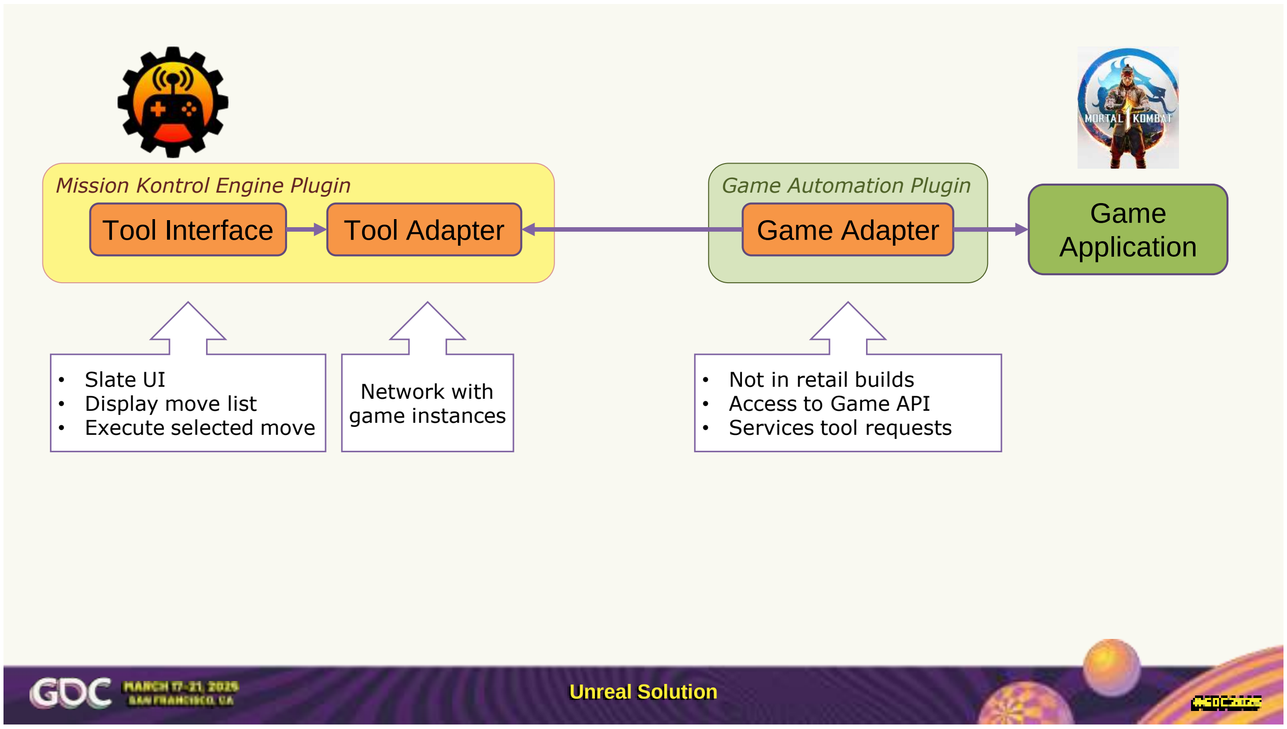


NORMAL ATTACKS

MOVE NAME	KOMMAND	DAMAGE	BLOCK TYPE
Sheath Slap •	X	20.00	High
See No Evil	X, B	30.00	Low
Blind Sided	X, B, X	50.00	Mid
Belly Buster	✚ + X	20.00	Mid
Silent Palm •	Y	30.00	High
Gut Check	Y, X	30.00	Mid
Tamahag-Pain	Y, X, Y	50.00	Mid
Spirit Strike •	✚ + Y	30.00	Mid
Silencing Souls	✚ + Y, Y	70.00	Overhead
Dark Ritual •	✚ + Y	50.00	Mid
Slice N' Dice	✚ + Y, Y	50.00	Mid
Pinned	✚ + Y, Y, ✚ + Y	70.00	Overhead
Spirit Away	✚ + Y, Y, ✚ + Y	70.00	Mid

B ADVANCED VIEW

For functionality, our starting point goal was to request character move list data from the game application, present it to the remote user, and then use that data to request automated move execution.



With those requirements in mind, we plotted out a model of the tool following this general structure:

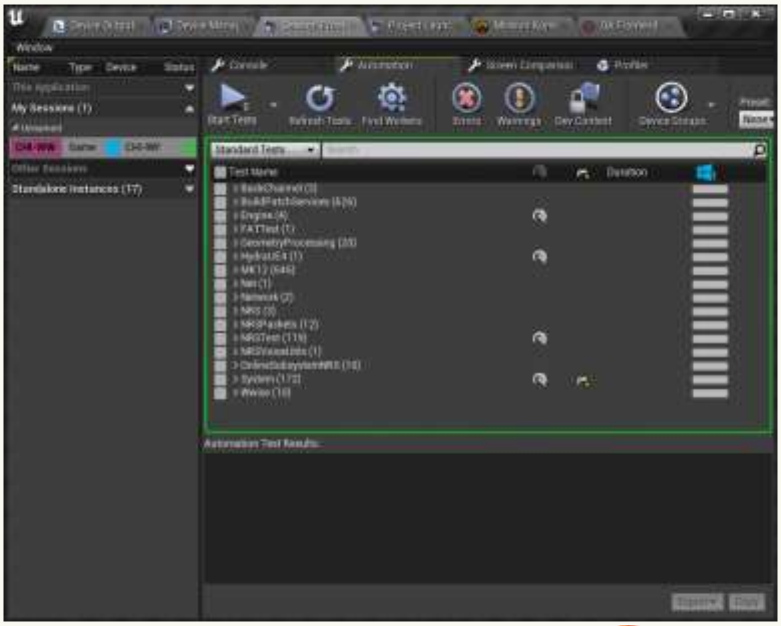
- Mission Kontrol Tool Plugin – Unreal engine plugin using Slate UI to build the tool interface. Slate is the UI framework used by the unreal editor application, which I'll talk more about later.
- Tool Adapter – module on the tool plugin which manages networking with eligible game instances. This includes establishing connections and sending or receiving data.
- Game Adapter – debug only plugin on the game application which networks with the tool application and services automation or data requests.

So how does the Unreal Engine fit into all of this?

Slate UI

"Messaging" Runtime Module

Unreal Frontend



Session Frontend inside of Unreal Frontend program (UE 4.27)

There are 3 unreal components that make up the core of what we needed.

[click]

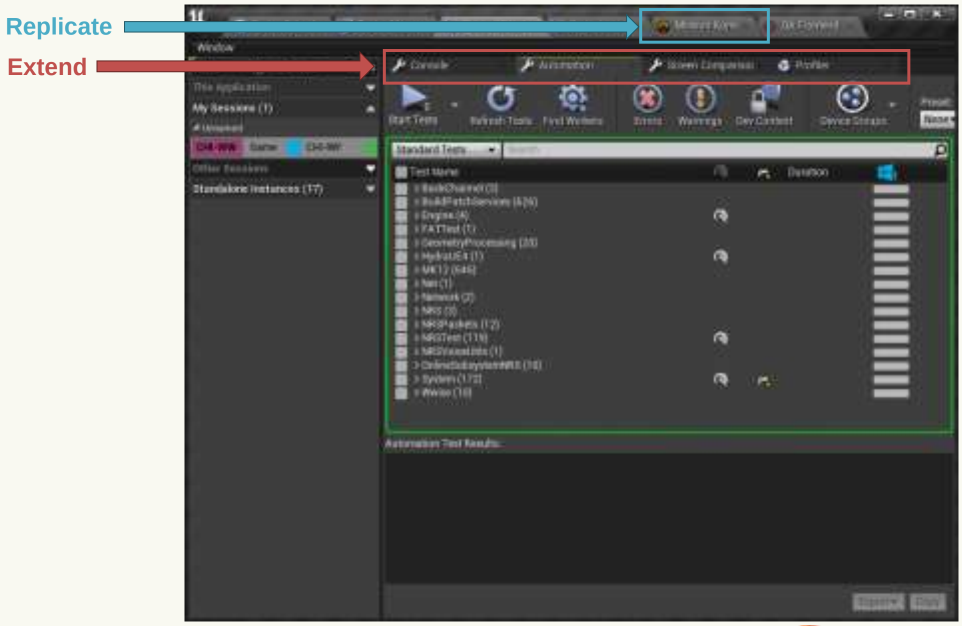
The Slate UI framework,

[click]

The engine runtime module called "Messaging" which handles the networking between Unreal applications on the same network or same process, which is the case when you are running the game from inside the editor application.

[click]

Finally, Unreal Frontend. This is a lightweight standalone program built separately from the editor which hosts several debugging tools. One of those tools is called “Session Frontend”. Reverse engineering this tool was my starting point for Mission Kontrol. Session Frontend provides a model for how to use all three of these components to create a remote debugging tool. As a first step, I created a shell for the tool interface using Slate.



Session Frontend inside of Unreal Frontend program (UE 4.27)

Unreal Solution

There are a couple different paths we considered taking from this point that you should also consider.

[click]

One option was to directly extend Session Frontend by adding additional panels inside of it. This would then borrow the same remote connection service.

[click]

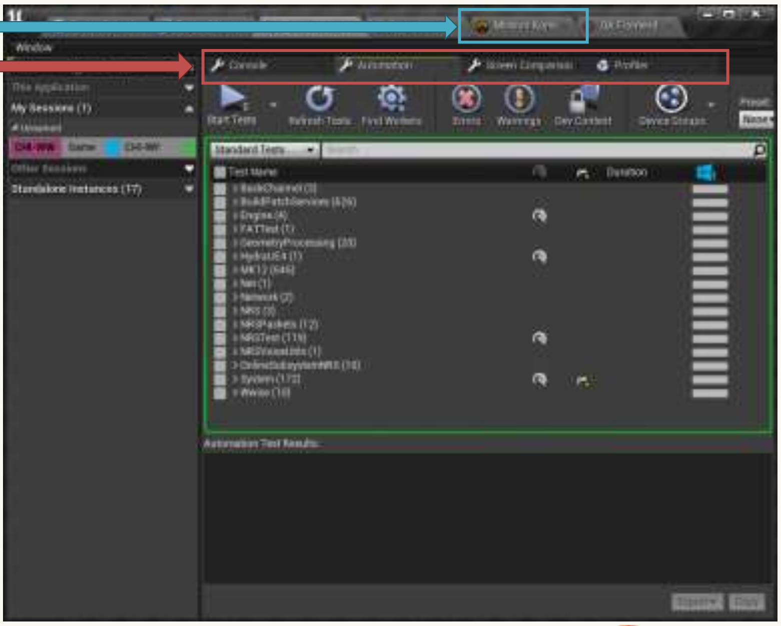
The second was to create an independent tool modeled after Session Frontend.

Replicate

Extend

Why Replicate?

- More Flexible UI
- Independent Networking
- Avoid Engine Modification



Session Frontend inside of Unreal Frontend program (UE 4.27)

The second option does involve more work, but there are a few reasons we chose this path for Mission Kontrol. Mostly it has to do with our intended scope:

We knew that we wanted Mission Kontrol to have it’s own internal tab well, similar to Session Frontend. This allows it to serve as a toolbox with room to grow – by adding additional panels.

We wanted greater control over the networking component and the way that game instance connections are managed for the user. Making changes to the Session Browser, which is the component on the left that manages connections for Session Frontend, would have involved engine modifications.

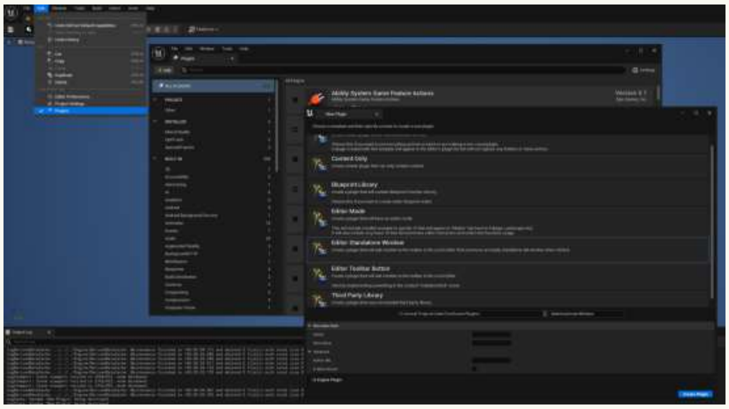
Generally, we try to avoid making engine modifications because it increases the cost of integrating new engine updates and poses a risk of breaking other engine features.

If this doesn’t apply to you then you might consider extending instead, which could be cheaper for a simpler tool.

So here are the steps we took to create our own version of a frontend tool.

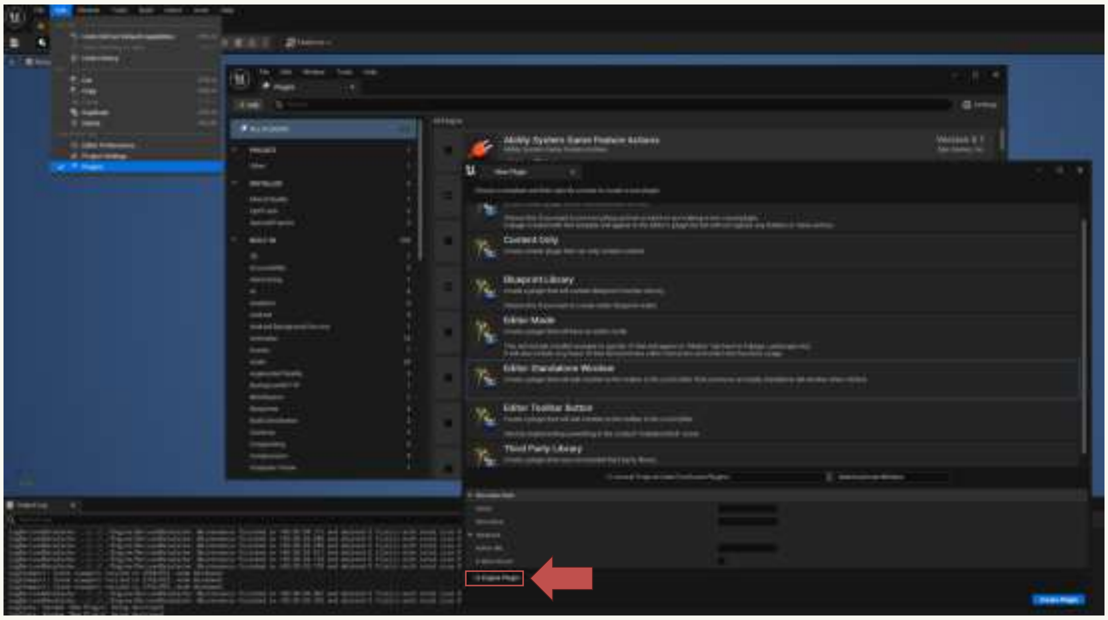
Create the Plugin

- Editor Standalone Window
- Engine Plugin



https://dev.epicgames.com/documentation/en-us/unreal-engine/slate-editor-window-quickstart-guide-for-unreal-engine

The first is creating the engine plugin for the tool interface. You can do this from the Editor by opening Plugins window, then adding a new plugin of type "Editor Standalone Window". You'll need to specify Engine plugin here so it can be added to the Unreal Frontend later.

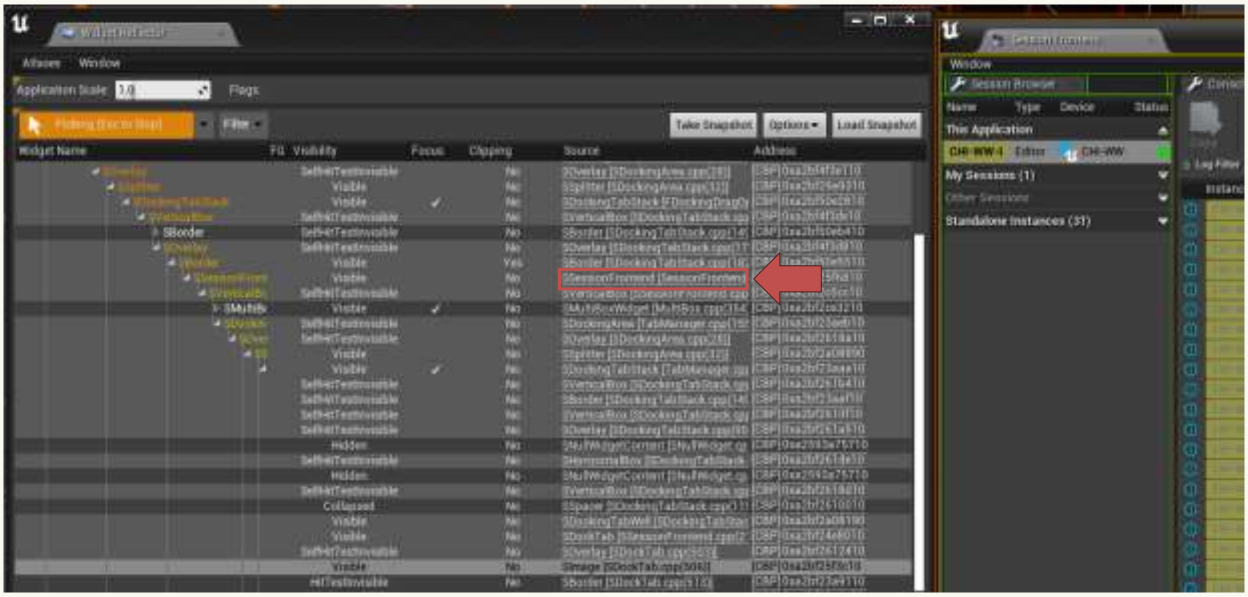


https://dev.epicgames.com/documentation/en-us/unreal-engine/slate-editor-window-quickstart-guide-for-unreal-engine

Unreal Solution

This image is of UE5, but the steps are the same as in UE4. At the time I originally did this the Slate documentation was relatively sparse, but UE5 has better documentation walking you through this part of the process so I recommend checking that out.

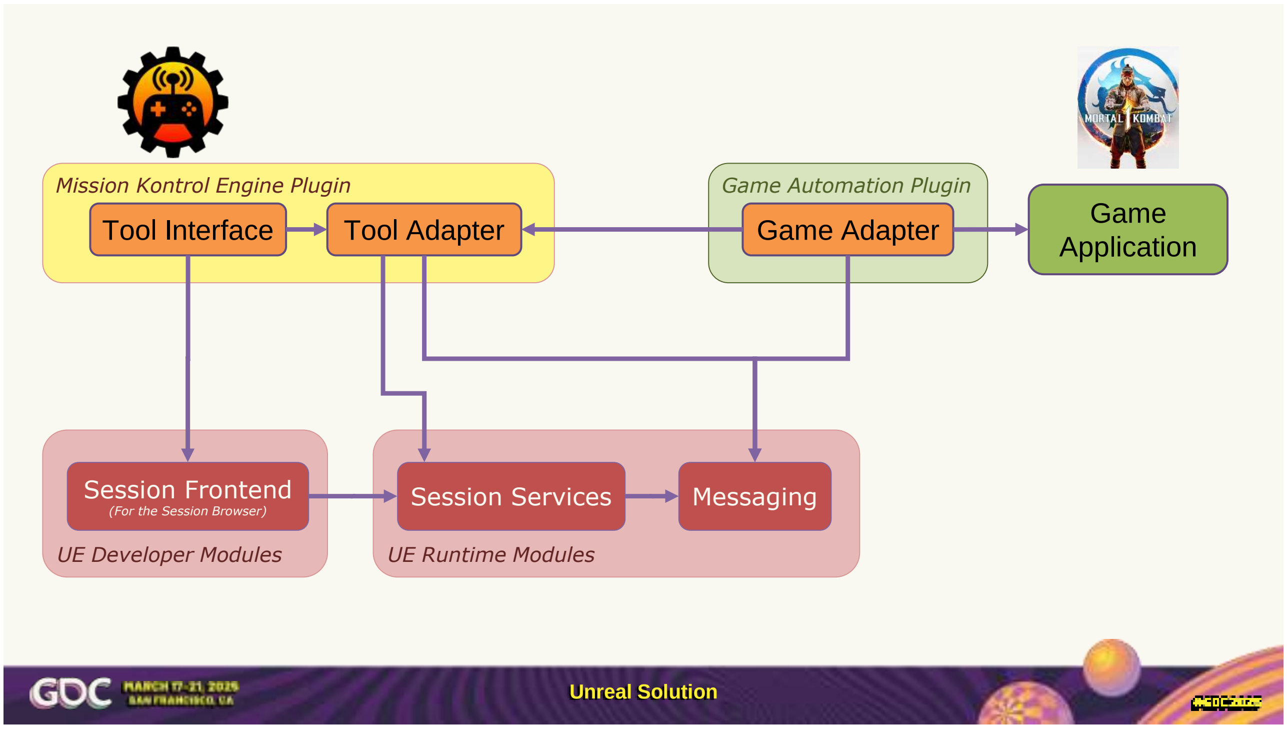
Creating the plugin will generate some boilerplate code files, which is where you'll begin writing Slate code. Despite my lack of UI experience at the time I was still able to use Slate to implement just about everything people could ask for by referencing the large library of Slate widgets that are used across the Editor. I found the easiest way to learn was the Widget Reflector tool.



Widget Reflector // Session Frontend (UE 4.27)

The Widget reflector displays the implementation details for any of the UI elements you can see in the editor, allowing you to browse examples of Slate code by using the editor as a gallery. This is how I modeled after existing Editor UI patterns such as the Session Frontend. You would also find this useful if you decided to extend instead of replicate. Since I wanted my own version of the Session Browser panel, I used the widget reflector to track down the implementation. [click]

Clicking that link takes me directly to the source code for the Session Frontend widget, and from there I found the Session Browser code. This not only gave me access to all the Slate widgets I would need to display a list of connections, but it also directed me towards the underlying networking modules, which is the next major step.

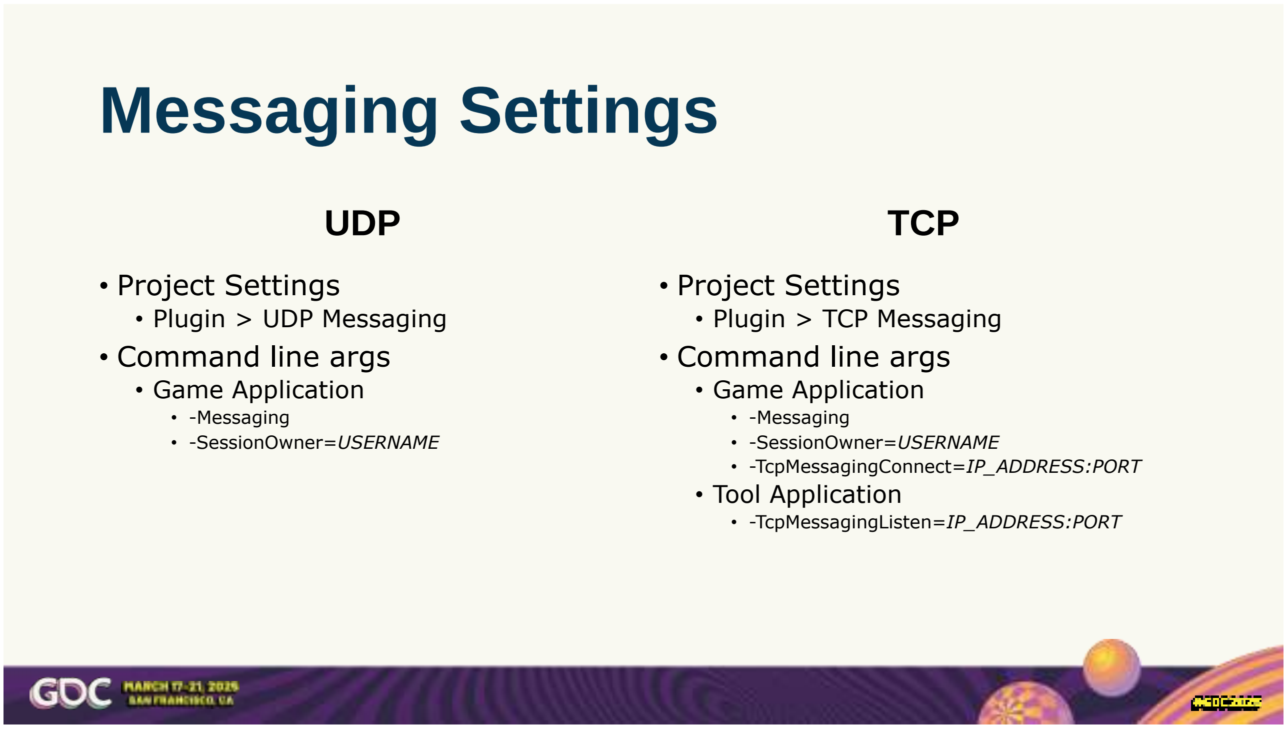


The engine runtime modules that facilitate networking and communication for Session Frontend are called [click] Messaging and Session Services.

The Messaging module is an API for setting up what are called message endpoints and sending data between them. This is needed to communicate between the Game Adapter and Tool Adapter.

The Session Services module is an API for managing messaging connections. It maintains a list of game instances that are discoverable on your network and manages which of them is considered active in the Session Browser used by Session Frontend. The active session can be accessed via the API, which allows your tool to borrow the same connection as Session Frontend, as well as the same Session

Browser UI.



Messaging Settings

UDP

- Project Settings
  - Plugin > UDP Messaging
- Command line args
  - Game Application
    - -Messaging
    - -SessionOwner=*USERNAME*

TCP

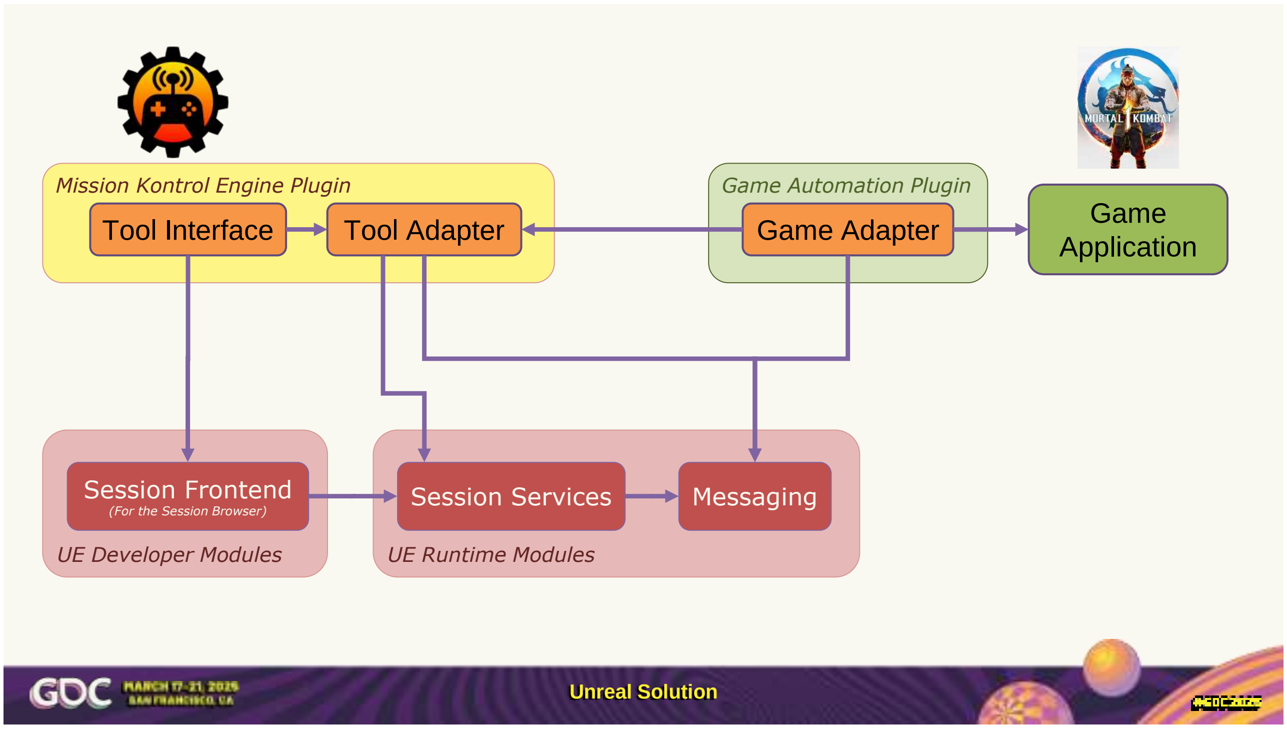
- Project Settings
  - Plugin > TCP Messaging
- Command line args
  - Game Application
    - -Messaging
    - -SessionOwner=*USERNAME*
    - -TcpMessagingConnect=*IP_ADDRESS:PORT*
  - Tool Application
    - -TcpMessagingListen=*IP_ADDRESS:PORT*

GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

Two important notes about the Messaging system is that it will be disabled by default in any non-editor or shipping build, and the default protocol is UDP.

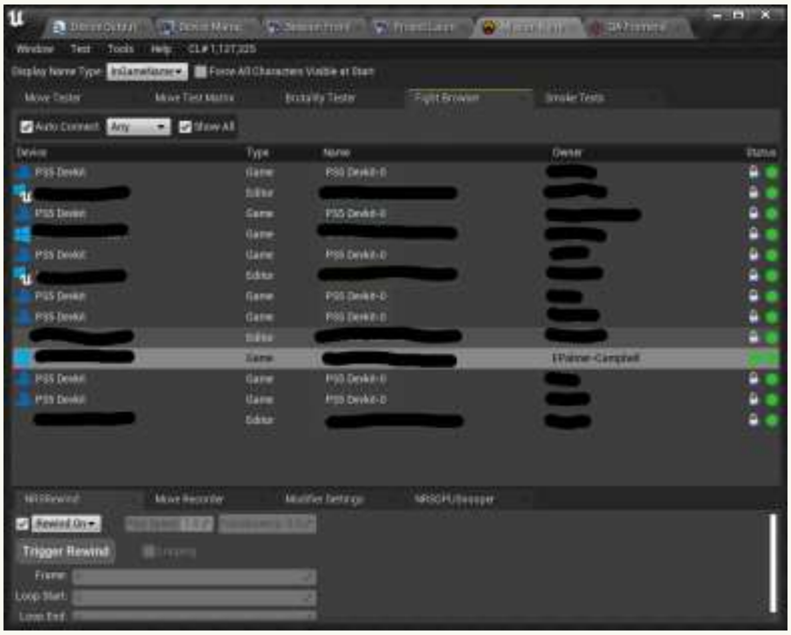
These settings can be changed by editing the default project configuration, or on a case-by-case basis with command line arguments when launching the application.

We used TCP because it prevented bugs with messages being received in the wrong order during high traffic. We also used command line instead of keeping it enabled by default, because the IP address needs to be specified for TCP and that information is dependent on the context you're launching from. Your mileage may vary on this, since it does complicate things for the end user.



Going back to the modules diagram, I should mention that we did not directly use the Session Browser or the Session Services API in Mission Kontrol. [click]

We created our own version of Session Browser called the Fight Browser, and managed game instance connections independently of Session Services. My recommendation would be not to do that, and to begin by leveraging the Session Browser as this diagram suggests.



Display Name Type: InGameName ▾ ☐ Force All Characters Visible at Start

☑ Auto Connect: Any ▾ ☑ Show All

Device	Type	Name	Owner	Status
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Game	[redacted]	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●
[redacted]	Game	[redacted]	EPalmer-Campbell	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●

☑ Rewind On ▾

Frame:

Loop Start:

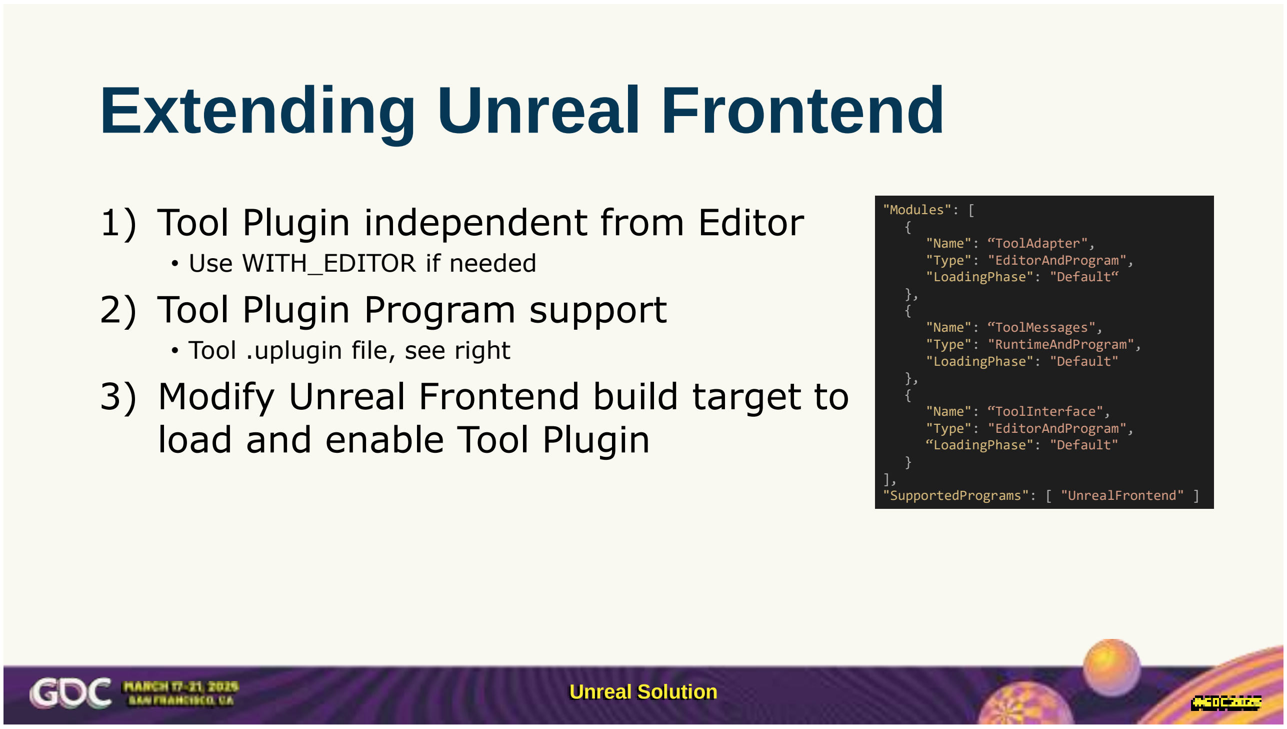
Loop End:

Mission Kontrol Fight Browser tab (UE 4.27)

Unreal Solution

This is the Fight Browser. This was more work but allowed for more control over how connections are managed.

This is where I could have prepared better for the long-term. It was understood that we would want to use this framework for other tools going forwards, but the immediate goal of Mission Kontrol was very specific and held my focus. While implementing the adapter modules responsible for networking, I didn't spend enough time ensuring that they would be generic enough to allow independent tools borrowing the same connection, like Session Services. The consequences came later when we needed to add unrelated tools either as extensions of the Mission Kontrol plugin, or with a dependency on it. This goes against the intended scope of Mission Kontrol, but it was a sacrifice we made to avoid the cost of a refactor. It’s important to keep encapsulation in mind when implementing core components like that to avoid accumulating tech debt. Alternatively, don’t re-invent the wheel.



Extending Unreal Frontend

1) Tool Plugin independent from Editor
   - Use WITH_EDITOR if needed
2) Tool Plugin Program support
   - Tool .uplugin file, see right
3) Modify Unreal Frontend build target to load and enable Tool Plugin

```
"Modules": [
   {
      "Name": “ToolAdapter",
      "Type": "EditorAndProgram",
      "LoadingPhase": "Default“
   },
   {
      "Name": “ToolMessages",
      "Type": "RuntimeAndProgram",
      "LoadingPhase": "Default"
   },
   {
      "Name": “ToolInterface",
      "Type": "EditorAndProgram",
      “LoadingPhase": "Default"
   }
],
"SupportedPrograms": [ "UnrealFrontend" ]
```

So, we’ve gone over everything needed to create a remote debugging tool, but right now it’s still only available in the editor. The missing piece is Unreal Frontend.

For the tool to be available to a standalone program, the tool plugin must be independent from any Editor modules. It’s okay if that’s conditional based on the build target.

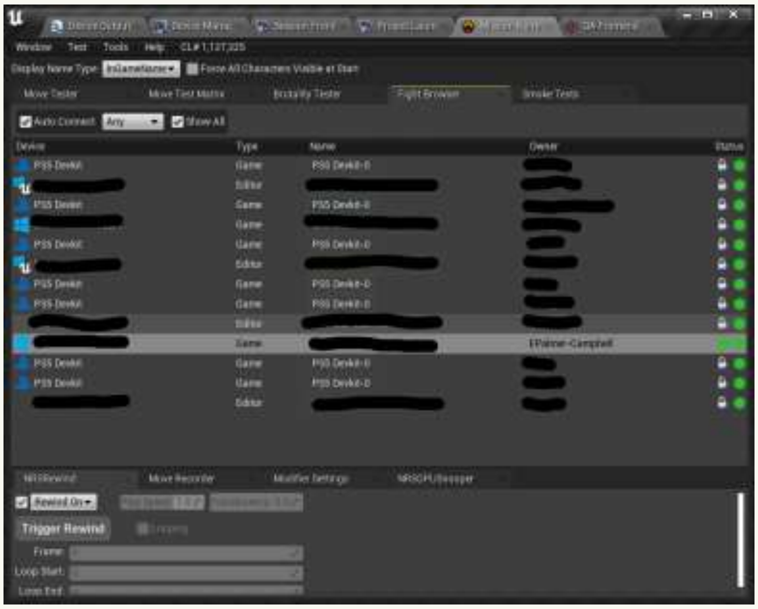
Then, all the modules in the tool plugin must be allowed to load for Programs, and the plugin itself must explicitly support Unreal Frontend.

From here we decided to directly modify and extend Unreal Frontend. Alternatively, you could avoid the engine modification by creating your own standalone Unreal program, but that is considerably more work.

Slate UI

“Messaging” Runtime Module

Unreal Frontend



Window Test Tools Help CL# 1,127,225

Display Name Type: InGameName ▾ Force All Characters Visible at Start

Move Tester | Move Test Matrix | Brutality Tester | Fight Browser | Smoke Tests

✓ Auto Connect Any ▾ ✓ Show All

Device	Type	Name	Owner	Status
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Game	[redacted]	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●
[redacted]	Game	[redacted]	EPalmer-Campbell	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
PS5 Devkit	Game	PS5 Devkit-0	[redacted]	●
[redacted]	Editor	[redacted]	[redacted]	●

NRSRewind | Move Recorder | Modifier Settings | NRSCPUStepper

✓ Rewind On ▾

Trigger Rewind

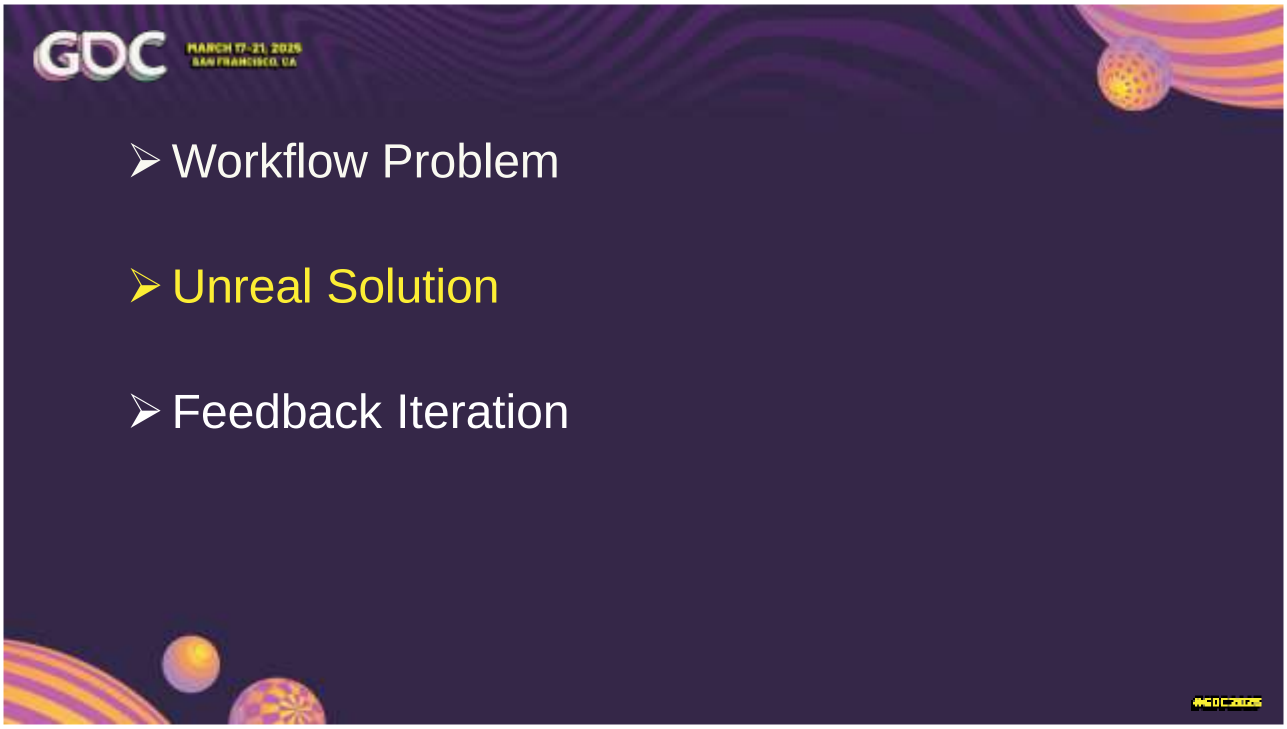
Frame:

Loop Start:

Loop End:

Mission Kontrol inside of Unreal Frontend program (UE 4.27)

All together this enabled me to prove out the initial framework for Mission Kontrol. I now had a basic tool interface which could be hosted from the editor or a standalone program and connect to a networked game instance on any of our console platforms.

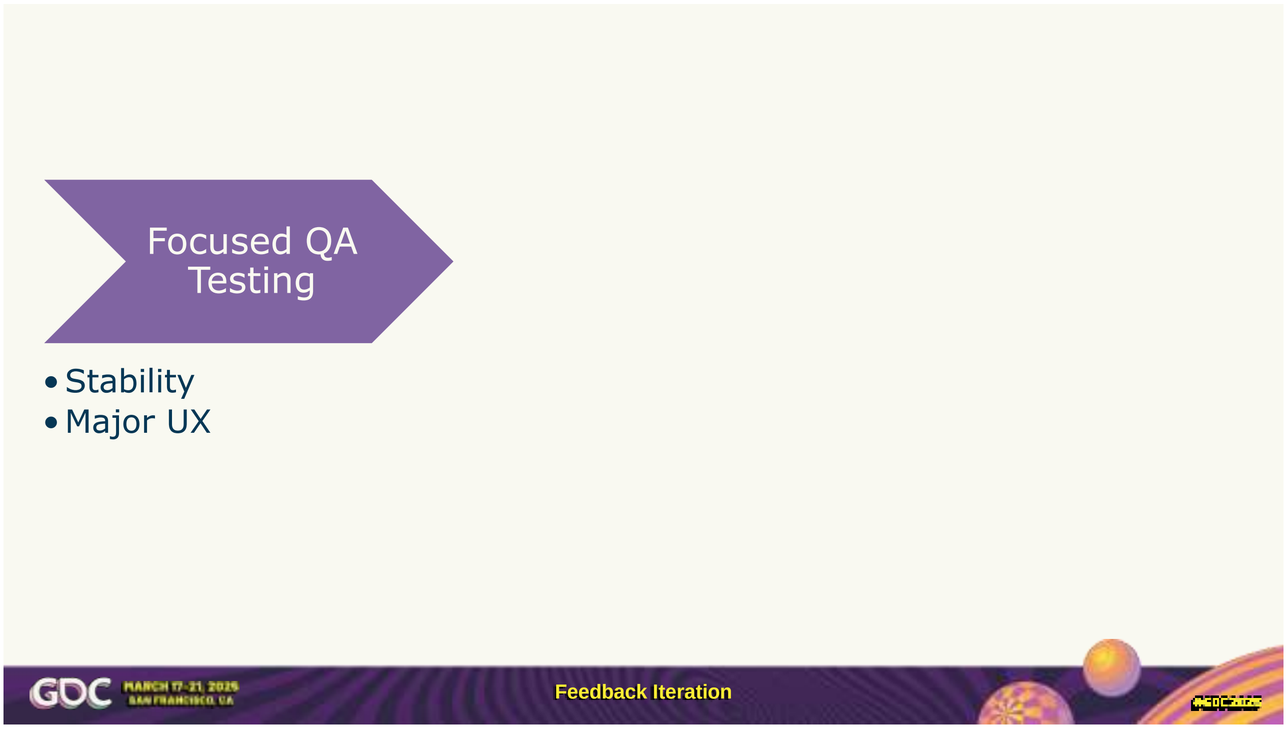

MARCH 17-21, 2025
SAN FRANCISCO, CA

- Workflow Problem
- Unreal Solution
- Feedback Iteration

#GDC2025

Relying on those existing Unreal tools as a model for this foundation was critical for me to produce something useful quickly enough to keep pace with our rapid development cycle. We wanted people to learn how to use the tool while setting up their initial workflows for the project and then provide feedback to help it evolve alongside the game. This process was important to start early in development to leave ample time for iteration.

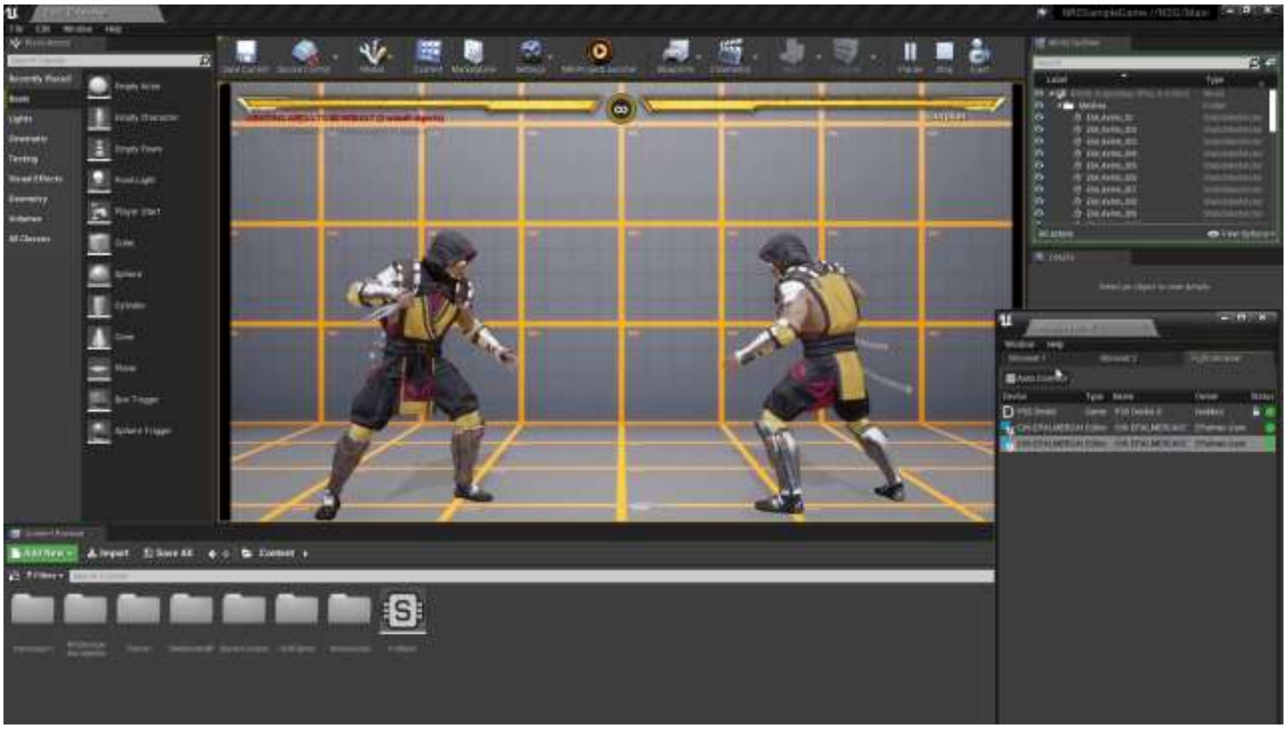
So now comes the follow through. [click] Several months later, development on MK1 is accelerating and the game is starting to take shape.



Focused QA Testing

- Stability
- Major UX

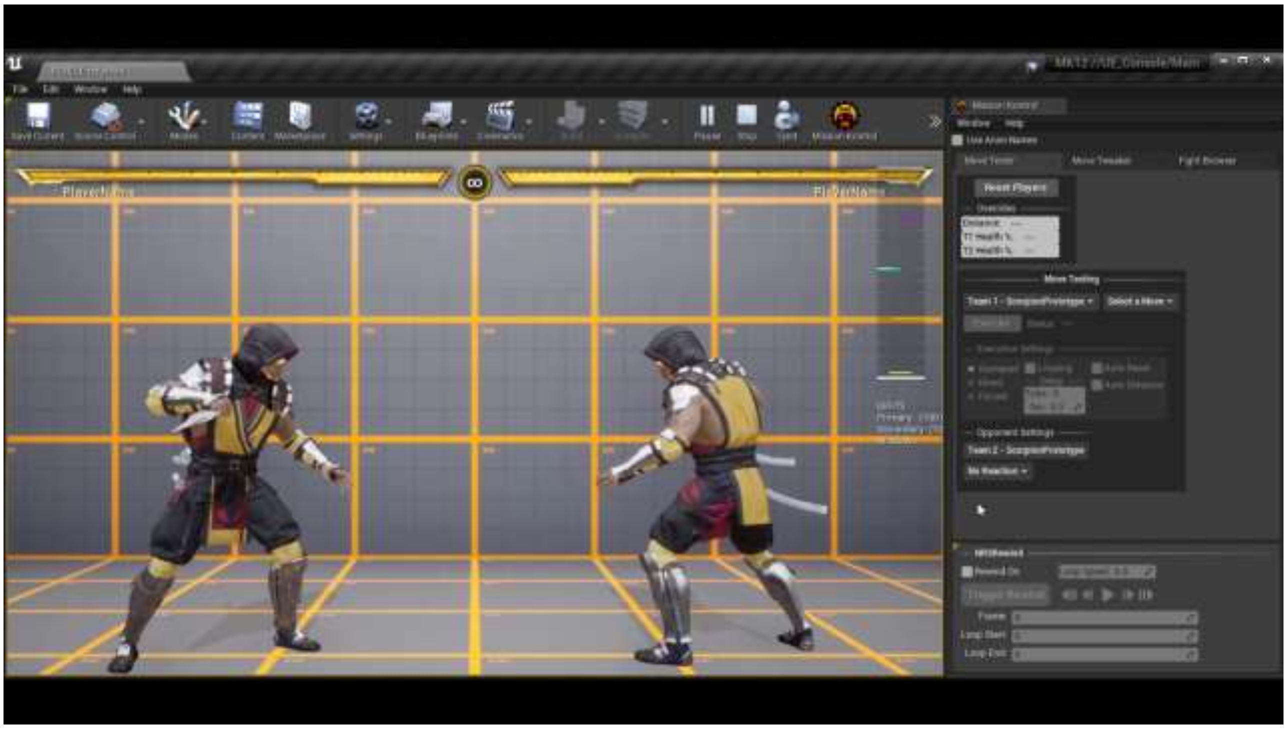
For the first stage of iteration, we chose our QA team as the user group. This is not only because software testing is their specialty, but also because they are going to be one of the biggest clients for this tool, so their feedback is valuable. Our goal here is to focus on stability and major UX improvements based on that feedback.



This clip demonstrates one of the first versions of the tool. The window in the bottom right corner of the screen is the tool interface. Right now, it's showing the available game instance connections.

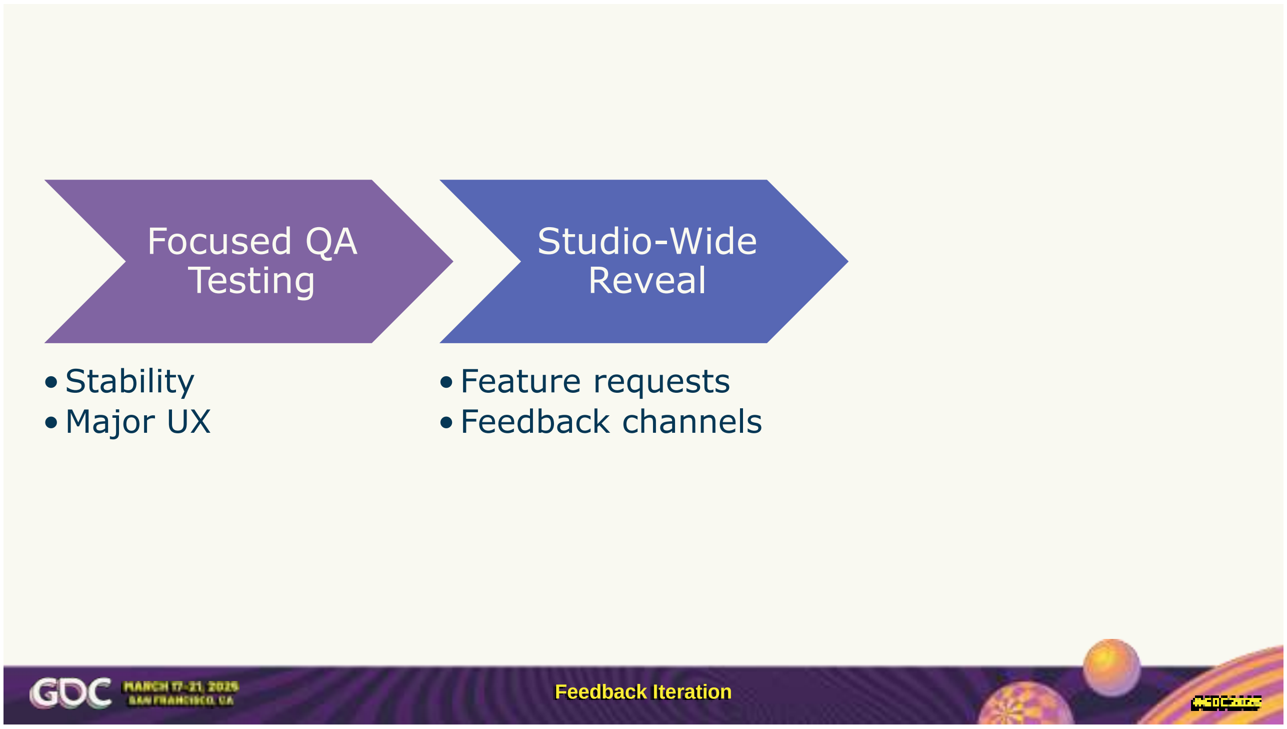
[click]
And now it is displaying a list of all the available attack commands for player 1 on the left. One of those commands is then selected and executed in game. That list of moves is game data that was requested by the tool upon connecting. It's important to note here that this data is not something which was scraped from content by the tool but rather communicated by the running game instance.

This satisfies all the requirements from the initial planning. Pretty simple to begin with but it's already more useful than what we had before.



Based on their feedback we made a couple big improvements. Most notably the ability to request actions on both players at the same time and setup a multiplayer interaction, and a panel for controlling our frame stepping tool, which is demonstrated here.

[Play video]
This frame stepping tool has been with us for a while now, but the typical way of controlling it is via the gamepad. There are debug only button chords for activating it and interacting with it. Since the user is specifically not using a gamepad in this context, this provides an alternative interface because the two tools are frequently used at the same time.


Focused QA Testing
Studio-Wide Reveal

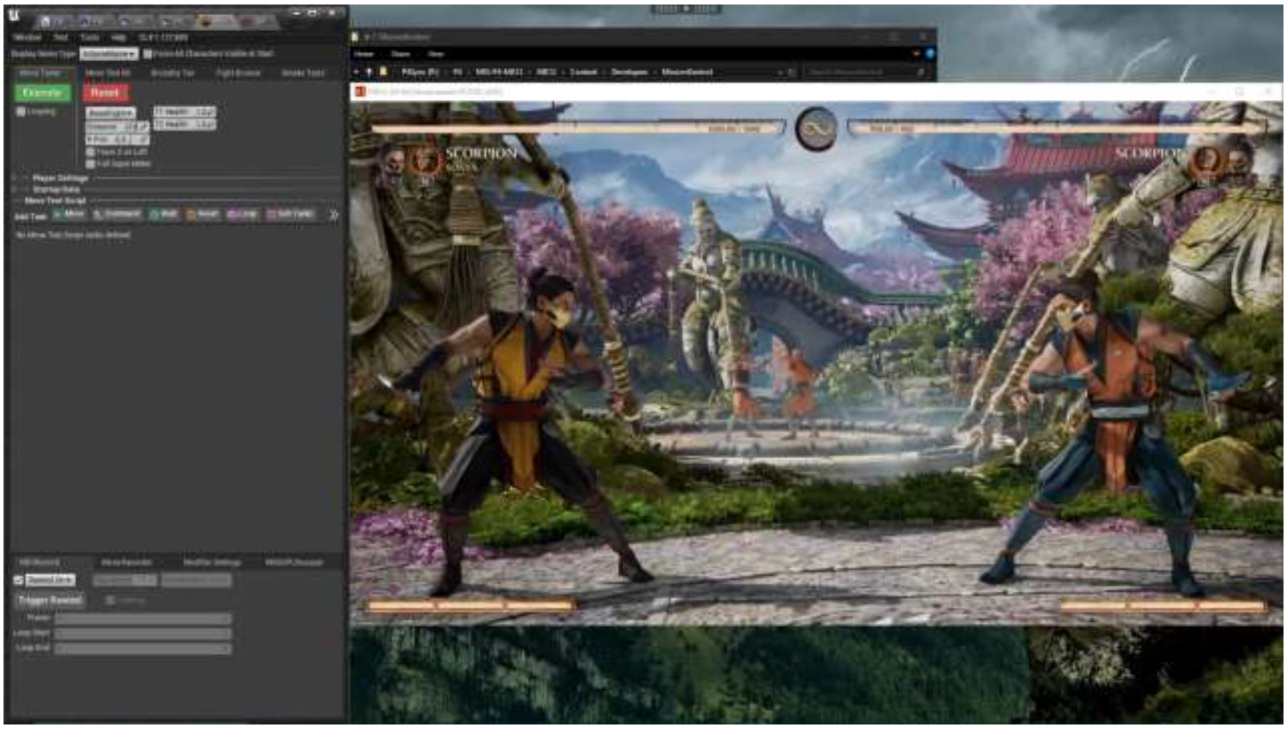
- Stability
- Major UX

- Feature requests
- Feedback channels

After this stage, we're ready to go wide.

[click]
We introduced the tool to the rest of the studio in a live demonstration and training presentation. At the end of this presentation, we did some Q&A, including an open request for any feature ideas. Some of the most useful features came from this initial wave of feedback.

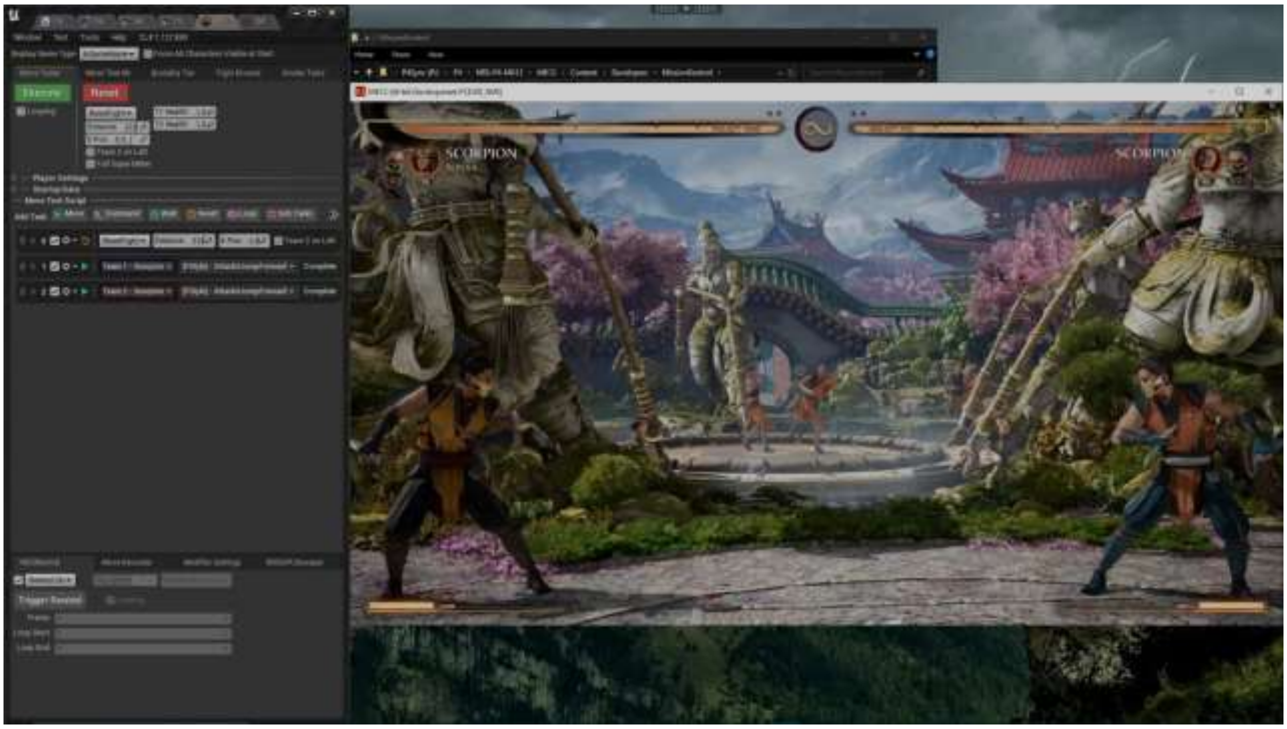
Afterwards, we directly asked disciplinary leads to incorporate the tool into their team workflows and created a public communication channel on Slack to use as a feedback forum. Once people had some time to play with it and saw the benefits, the ideas kept coming. This is where the tool transformed into something we now view as essential to our workflow.



This is the latest version of Mission Kontrol which I previewed earlier. I won’t go over everything that’s changed, but I’ll talk about a couple of the most useful features that came out of this stage of iteration.

[play video]
The first is the concept of test script. Rather than executing moves one at a time, now you can create a sequence of multiple tasks, and there’s a wider range of actions that a task can be, such as executing a move, executing a console command, resetting game state, or waiting between actions so that you can piece together a combo sequence or complex multiplayer interaction.

What I’m doing here is creating a script that makes both players jump and hit each other at the same time, which is an important test case to cover that requires careful timing.

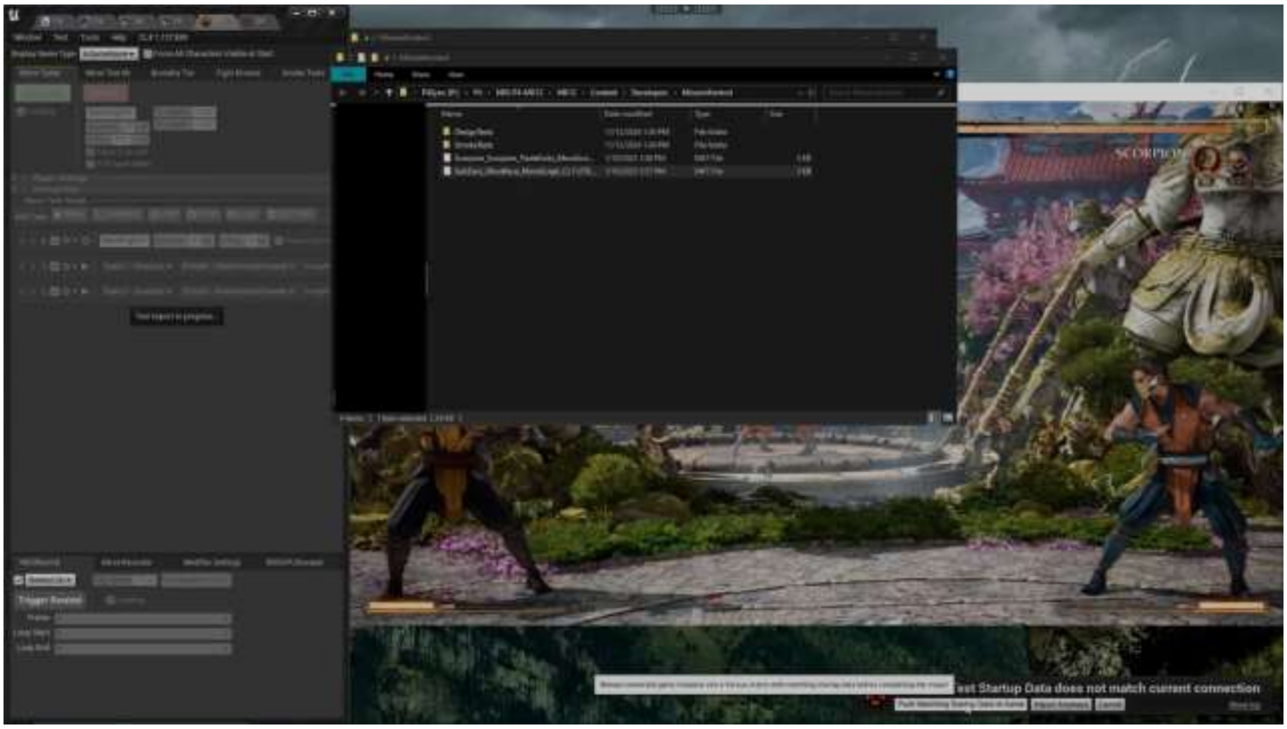


Possibly the most useful thing to come from this was the ability to export these test scripts in JSON format, which can then be shared or attached to tracking tickets to assist with bug reproductions. Functionally, this is similar to an input recording but with some extra capabilities.

What I’m doing here is exporting the test script for the trading kicks and then importing a different test script which was created for a different set of characters.

The test scripts contain some meta data to help make sure that the correct characters and game settings are loaded before attempting to import. Here I’m being prompted to reload the game with the correct characters during the import process.

This is important so that the game is prepared to communicate the expected character data upon importing the script, because the characters need to be loaded for the data to be accessible.



est Startup Data does not match current connection

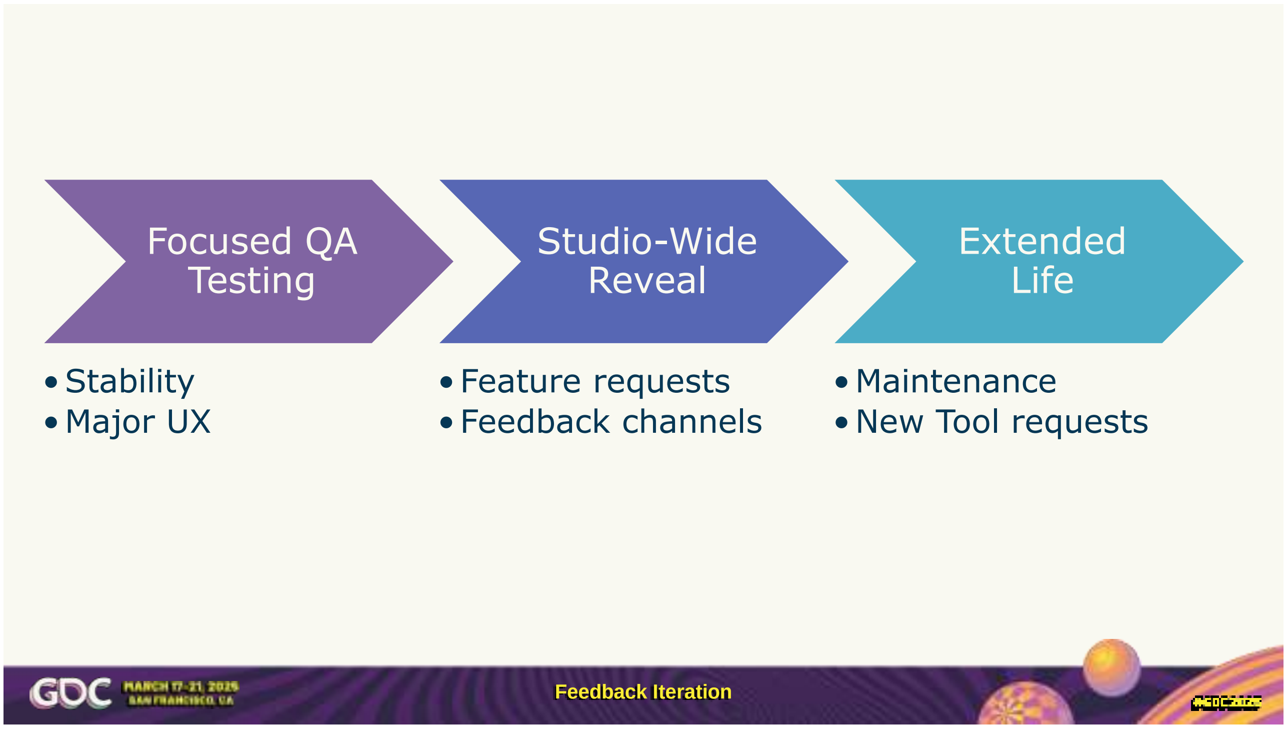
This particular script demonstrates another scenario that is difficult to reproduce by hand because of the spacing between characters and timing of the attacks.

If I make small adjustments to the initial distance between players, something a little different happens each time the reaction that Ghostface has when he hits the ice clones. This is exactly the type of scenario that might result in a weird cloth or visual fx bug that only occurs under specific circumstances.

Having this degree of precise control over the game state - using a tool that's accessible to the whole studio - has made testing and reproduction an entirely different process compared to previous projects. It really can't be understated just how valuable it is to create a portable bug reproduction in a complex action game.

[click]
The final stage covers the rest of the project. Gameplay tools are working with a moving target, and regular maintenance will be needed to stay functional while the game evolves. This is another reason why the feedback channel is so useful, since it gives people a direct line to notify the tool engineers when anything gets broken.

After the dust settles from that initial wave of new features, we continue to get requests for new tools as our priorities change. As I mentioned I won’t cover everything but here’s a handful of the most elaborate tools that were created later using the same foundation as Mission Kontrol.


Focused QA Testing
Studio-Wide Reveal
Extended Life

- Stability
- Major UX

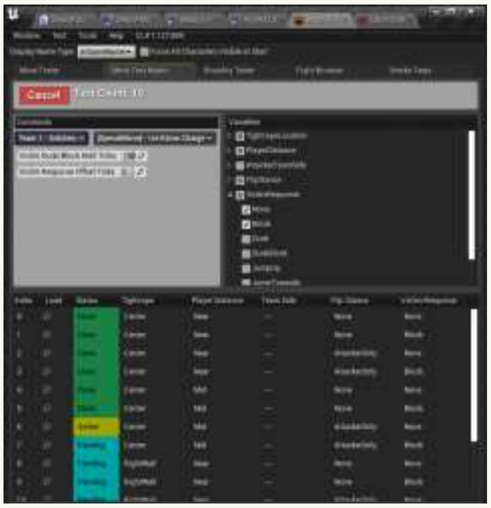
- Feature requests
- Feedback channels

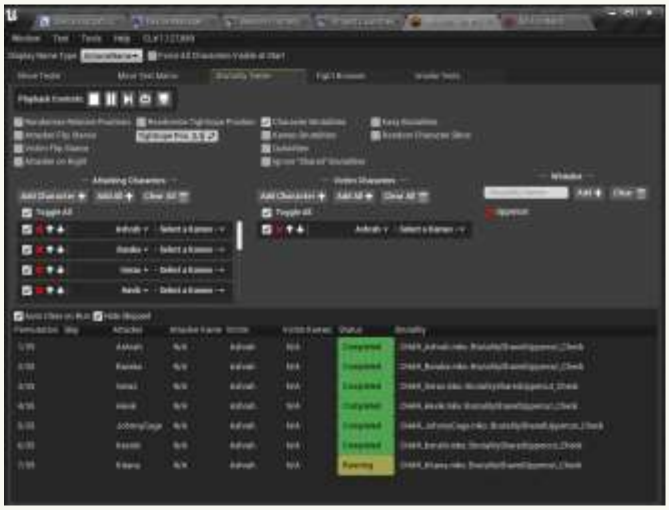
- Maintenance
- New Tool requests

The final stage covers the rest of the project.

[click]
Gameplay tools are working with a moving target, and regular maintenance will be needed to stay functional while the game evolves. This is another reason why the feedback channel is so useful, since it gives people a direct line to notify the tool engineers when anything gets broken.

After the dust settles from that initial wave of new features after the reveal, we continue to get requests for new tools as our priorities change. As I mentioned I won’t cover everything but here’s a handful of the most elaborate tools that were created later using the same foundation as Mission Kontrol.


Move Test Matrix


Brutality Tester

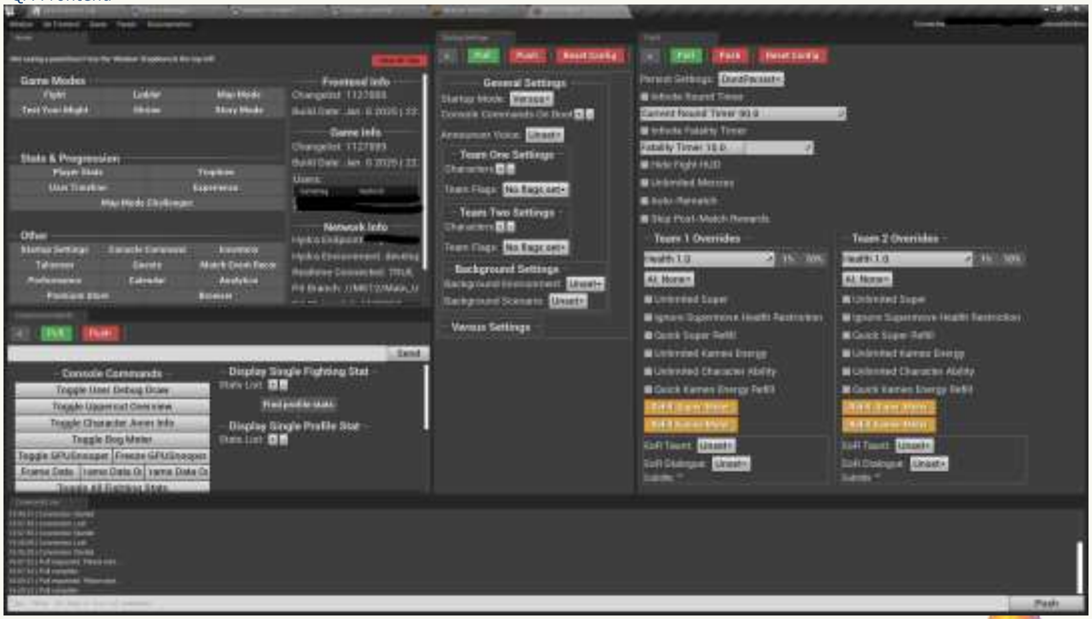
GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

Feedback Iteration

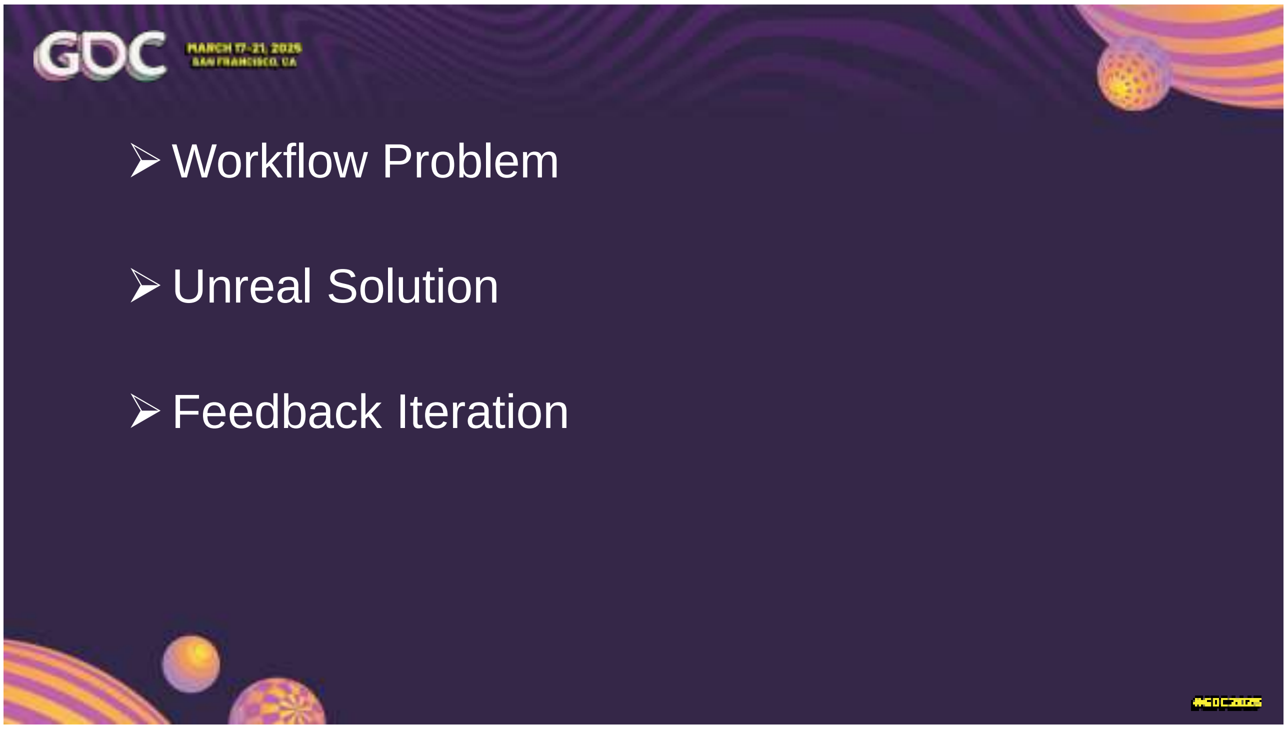
These two, the Move Test Matrix on the left and the Brutality Tester on the right, are elaborations of the move tester that were created to assist with smoke testing. They function similarly, the shared premise being that they generate and execute large batches of test scripts based on user parameters, for easy hands-free smoke testing. Once you hit execute, you just sit back and watch as a series of test scripts are executed for a comprehensive variety of scenarios.

It's important for us to test brutalities like this because there's a lot of them, they are intentionally difficult to execute, and they need to look good for each permutation of characters because the victim needs to look good while they're being torn apart, and the attacker needs to look good while they're doing it. A lot of polish goes into this feature.

QA Frontend



This last one is called QA Frontend, which is totally packed full of useful debugging tools that our QA team relies on to quickly iterate over the massive amount of content in our game. Mission Kontrol has an intentionally limited scope so that it can have a focused interface dedicated to the core combat gameplay loop, but QA Frontend is a collection of smaller tools that cover the entire game.


GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

- Workflow Problem
- Unreal Solution
- Feedback Iteration

#GDC2025

With that, we’ve seen all the key components involved in creating a full gameplay debugging tool suite. Problem identification, in-engine custom tooling support, and being available and responsive to feedback during the earliest stages of a project.

Thank you everybody for attending my talk today. I hope that I could inspire you to investigate ways that your game could benefit from something like Mission Kontrol. With the proper engine support, you can find opportunities to dramatically improve iteration times for your whole team.

Thank you!

GDC MARCH 17-21, 2025 SAN FRANCISCO, CA

Q & A

Etienne Palmer-Campbell

Gameplay & Tools Engineer

NetherRealm Studios

#GDC2025

Now we can move on to questions.