XDK없이 XBOX게임 개발하기(UWP on XBOX)

유영천

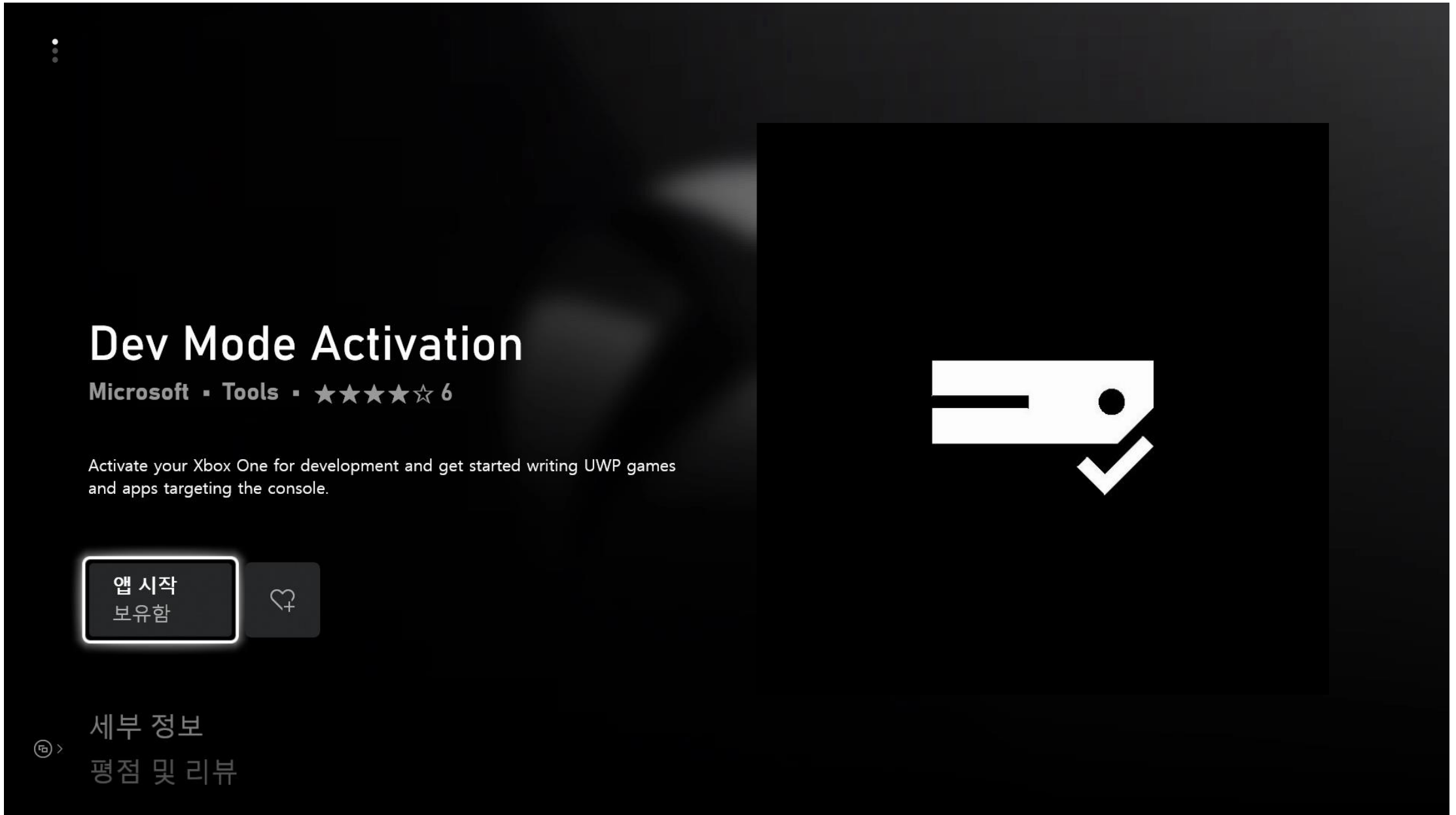
https://megayuchi.com

Tw: @dgtman

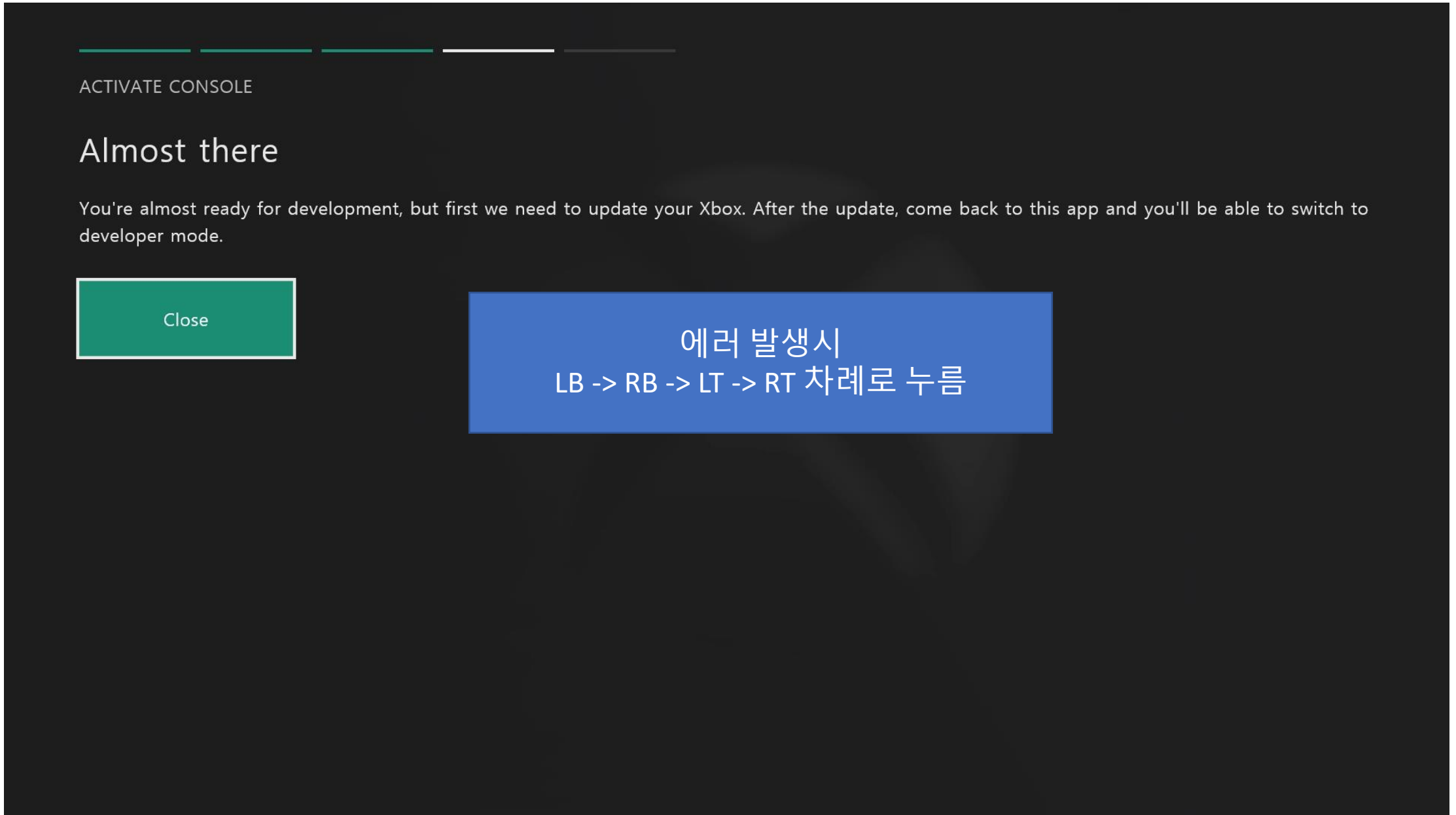
XBOX 세팅.

https://docs.microsoft.com/ko-kr/windows/uwp/xbox-apps/devkit-activation

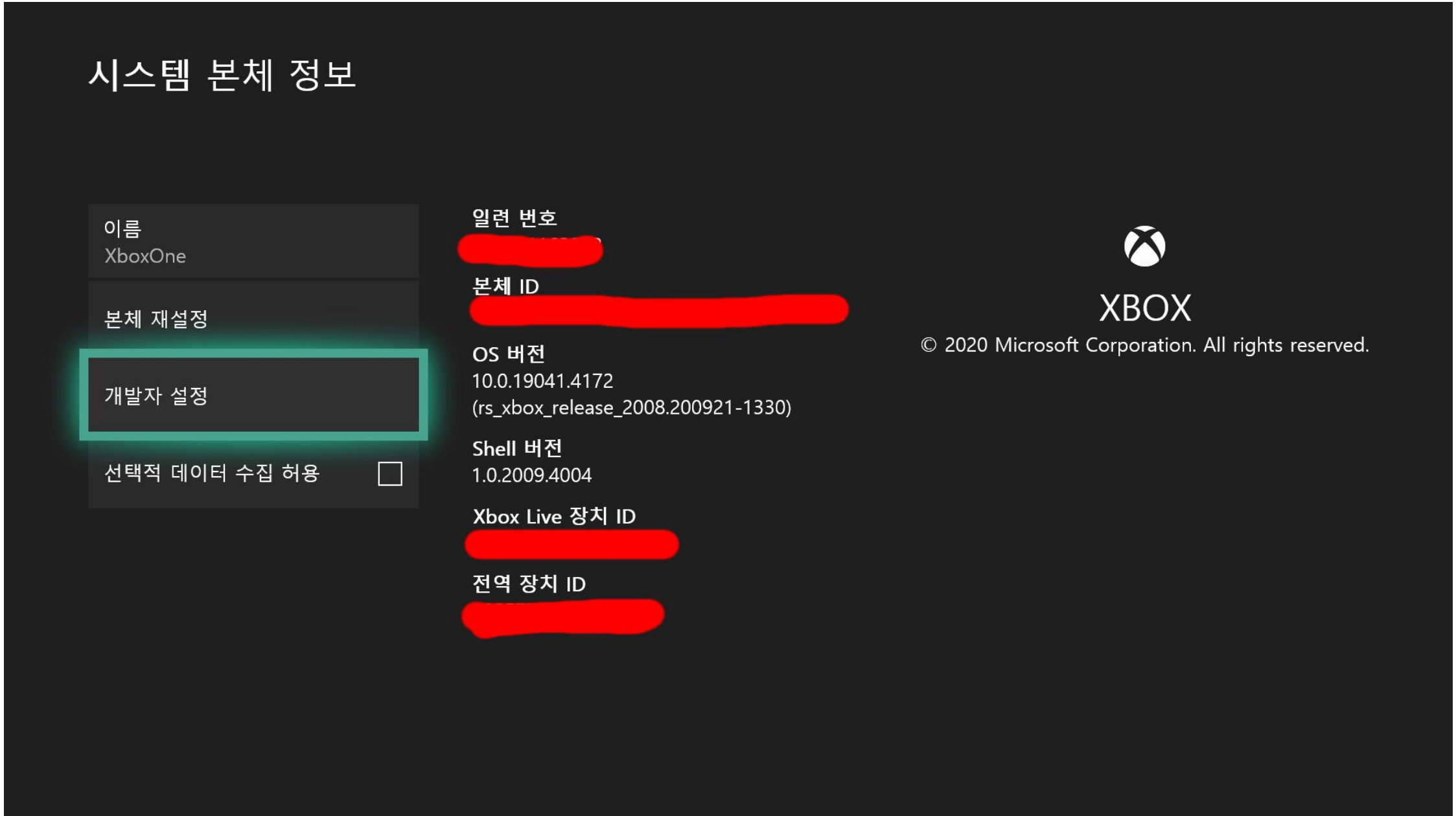
XBOX Dev Mode Activation



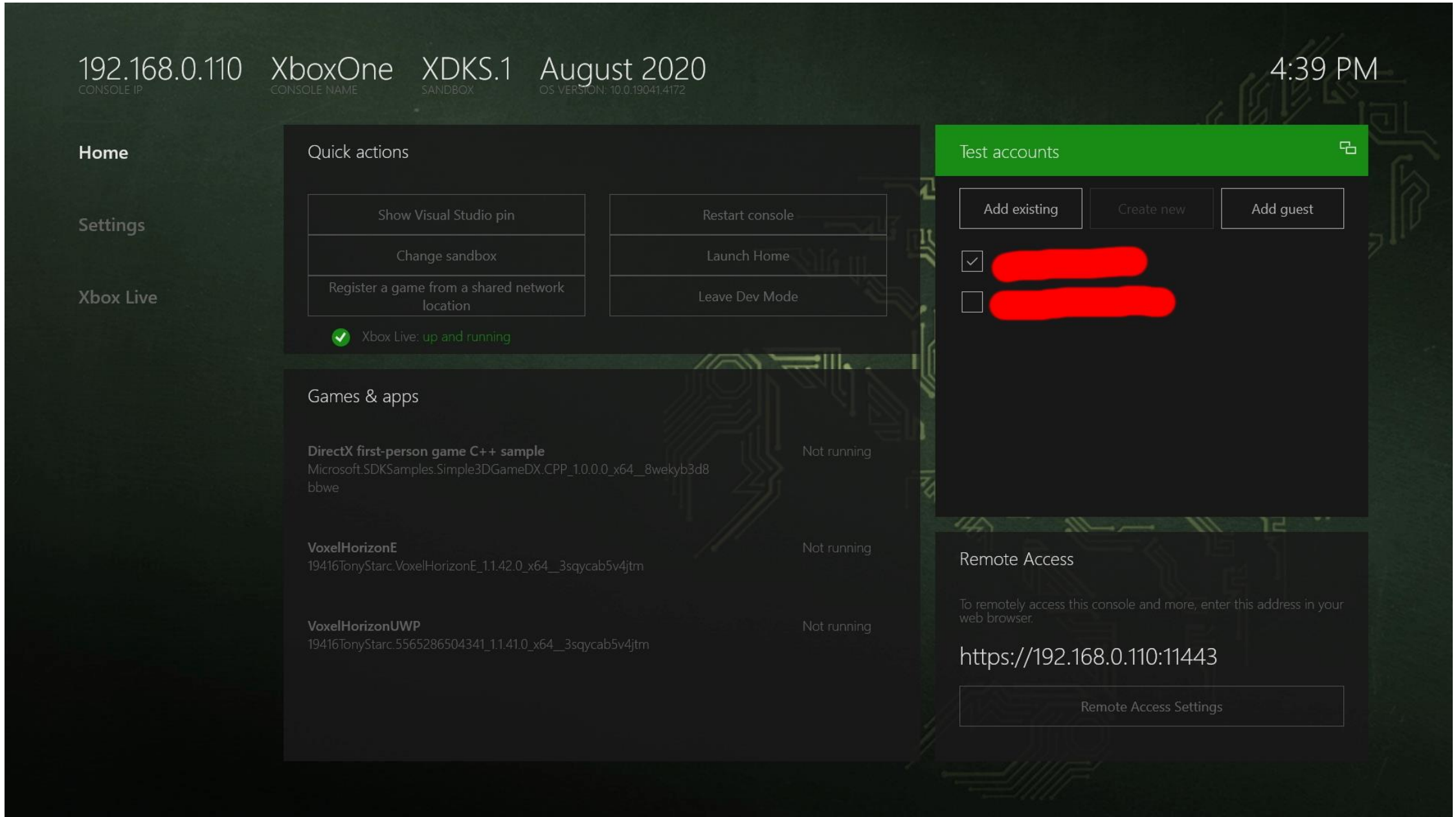
XBOX Dev Mode Activation



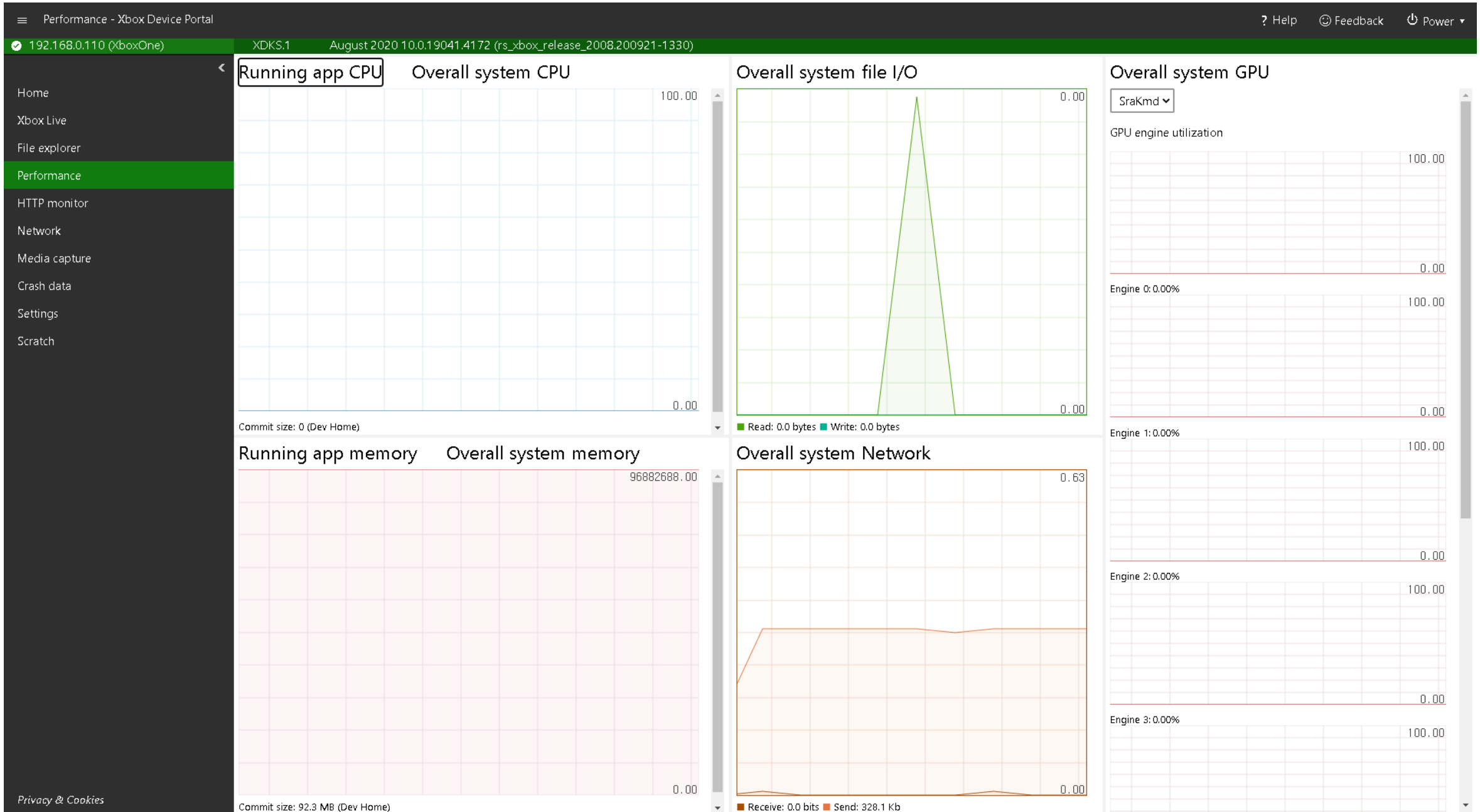
XBOX Dev Mode Activation



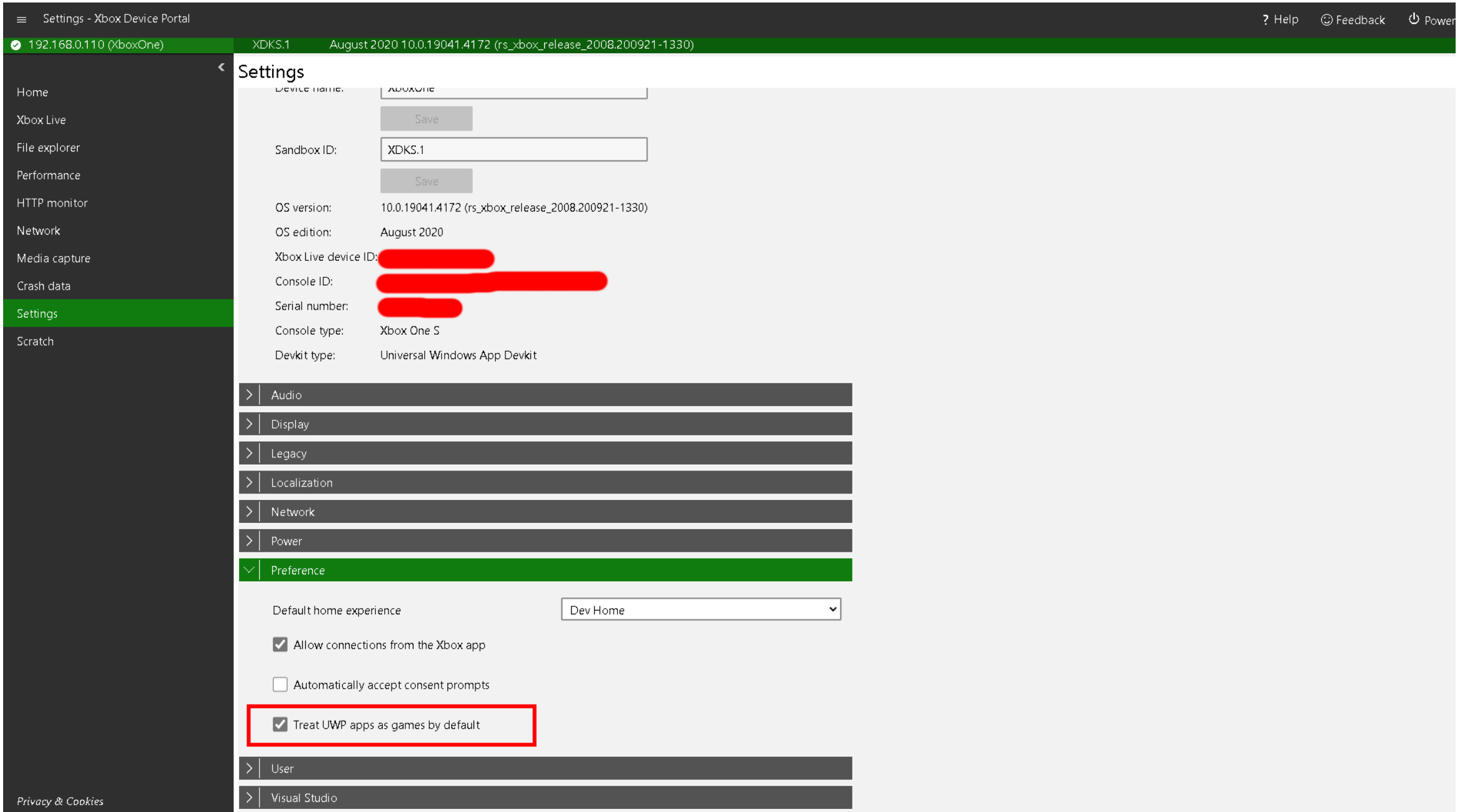
XBOX Dev home



XBOX portal – https://xboxone:11443/



XBOX portal – game mode로 세팅



UWP API소개

UWP(Universal Windows Platform)

- Windows Platform의 새로운 API
- 모바일과 터치,웹 억세스에 특화
- 모든 Windows Device API의 대통합
- X86/x64/ARM/ARM64 지원
- 초기엔 Windows Store에 올리기 위해서는 반드시 UWP API로 앱을 개발해야 했다.

UWP(Universal Windows Platform) - 2015년

One Universal Windows Platform

아름다운 UWP 생태계

UWP(Universal Windows Platform) – 2020년

One Universal Windows Platform

어디서 약을 팔아...

UWP App(Store App)의 최근 추세

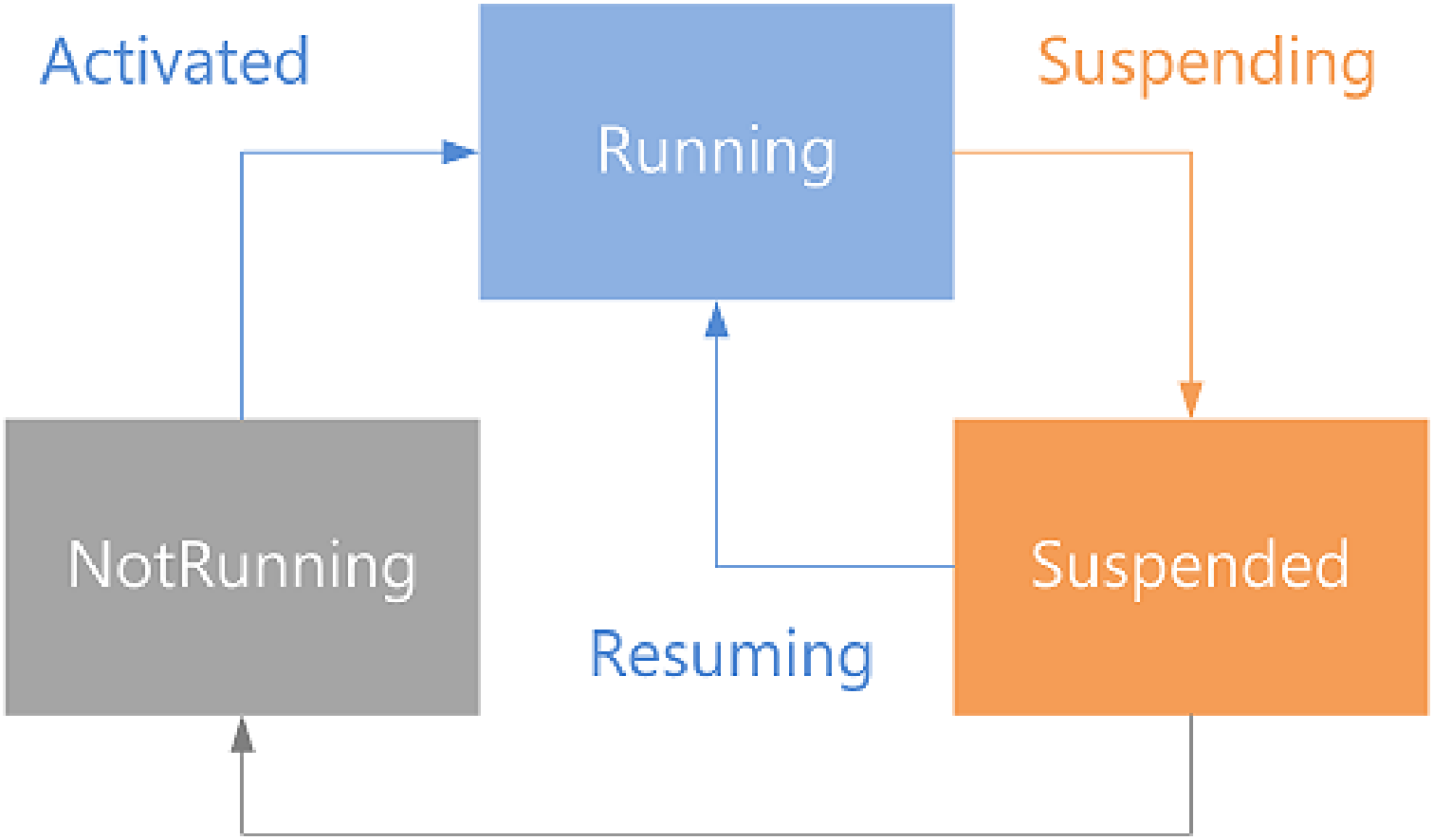
- 현재는 Windows Store에 올릴 수 있는 형태로 패키징된 앱을 모두 UWP App으로 지칭.
- 5년 사이에 Desktop Bridge기술이 정착됨.
- 고전적인 win32 Desktop App도 UWP App으로 패키징 가능.
  - Sandbox로 작동.
  - 일부 UWP API 사용 가능.
  - 스토어에도 업로드 가능.
  - 유명한 win32 App들 상당수가 스토어에 Desktop Bridge로 패키징되어 올라와있다.
- Desktop API로 UWP API가 흡수되는 모양새.
- UWP의 실패를 인정하고 싶지 않은 MS는 양쪽 모두 살려서 통합한다고 주장
  - WinUI 3
- 뒷통수 잘 치는 MS가 언제 WinUI를 땅에다 파묻을지 모르지만 일단 계획대로라면 Dekstop App에서 현대적인 XAML UI를 사용할 수 있게 됨.
- 즉 Windows Store에 앱을 올리기 위해서라면 UWP API를 학습할 이유가 없다.

아직 UWP API를 사용해야만 하는 영역

- XBOX Game/App 개발
  - XDK없이 개발 - UWP on XBOX
  - XDK도 UWP API사용
  - GDI를 전혀 사용하지 않으므로 계속 이대로 갈걸로 추측함.
- Hololens
  - 하드웨어가 조약하고 GDI는 아예 쓸 일이 없는지라 결코 완전한 형태의 win32 API가 포팅될 일은 없을것임.

UWP API로 UWP App개발

App Lifecycle



App Lifecycle – 이것만 알아둡시다.

- void App::OnLaunched()
  - App이 실행됨. 최초 실행되었거나, 죽었다가 실행되었거나…
    - Suspend-> Killed -> Resume -> OnLanunched
    - 최초 설치됨 -> OnLaunched
- Void App::OnSuspending()
  - 더 이상 스케쥴링되지 않음.
    - Phone이나 태블릿모드에서 back버튼, Windows버튼 눌렀을때.
    - 전화왔을때.
    - Desktop에서 창을 최소화시켰을때.
- void App::OnResuming()
  - Susened상태에서 다행히(?) 죽지 않고 App으로 복귀한 경우.
    - OS는 메모리가 부족할 때 suspend 상태의 앱부터 죽인다.

C++/CX

- UWP API(Windows Runtime API)를 호출하기 위한 MS의 C++확장
- UWP의 모든 API는 객체지향. C++/CX의 ref class로 구현되어있다.
- 레퍼런스 카운팅 기반 C++
- 컴파일러가 분석해서 알아서 AddRef()와 Release()를 호출하는게 아니고…ref class가 스마트 포인터를 내장하고 있다.

C++/CX

- C++에 C++/CLI 문법을 차용했다. -> C# 비슷한 느낌이 있다.
- ref class는 내부적으로 IInspectable COM 객체
- ref class의 핸들로 포인터 연산자 *대신 ^을 사용한다.
- Lambda 표현을 많이 사용함.
- ppl task를 많이 사용한다.

winrt/cpp

- 변덕쟁이(아유카와 ~~마도카냐?~~) MS가 C++/CX도 걷어찼다.
- 컴파일러 확장 기능 없이 표준 Modern C++(C++ 14이상)로 WinRT API의 소비계층을 구현.
- 현재는 C++/CX쓰지 말고 빨리 winrt/cpp로 갈아타라고 협박중.
- 어차피 게임에선 winrt기능 많이 안쓰므로 그냥 C++/CX로 해도 된다. 나중에 포팅하든가.
  - https://docs.microsoft.com/ko-kr/windows/uwp/cpp-and-winrt-apis/move-to-winrt-from-cx
  - https://docs.microsoft.com/ko-kr/windows/uwp/cpp-and-winrt-apis/interop-winrt-cx

제약사항

- https://docs.microsoft.com/ko-kr/windows/uwp/xbox-apps/system-resource-allocation
- CPU
  - 앱: 2-4개의 공유 코어.
  - 게임: 4개의 전용 코어 + 2개의 공유 CPU 코어
- GPU
  - 앱: GPU자원의 최대 45%
  - 게임: GPU자원의 최대 100%.
- DirectX 지원
  - 앱: DirectX 11 Feature Level 10.
  - 게임: DirectX 12 or DirectX 11 Feature Level 10
- x64만 지원

엄청 발목 잡는다!
결국 DX12렌더러를
만들어야 한다!

개발 수순

1. Win32 Desktop App으로 먼저 개발
2. DX11로 먼저 개발.
3. UWP on XBOX제약 사항에 따라 필요하면 DX12 지원.
    1. Compute Shader를 사용하는가?
4. UWP API로 포팅
5. XBOX에 배포 및 테스트

Desktop Application -> Windows 10 UWP App

- Desktop Application(이하 win32 App)과 Windows Store App(이하 UWP App)은 상당히 많이 다름.

- UWP앱에서 일부 win32 API를 사용할 수 있지만 UI 프로그래밍에서의 공통점은 아예 없다고 봐야…

- win32 app의 UI 코드 비중이 높다면 UWP app으로의 포팅이 어려움.

win32 Game -> UWP Game

- 게임의 코드는 대부분 C/C++.
- 대부분의 게임이 Graphics API로 DirectX 를 사용.
- DirectX API는 99%이상 동일.
- 게임에선 win32 API를 많이 사용하지만 GDI는 거의 사용하지 않음.
- Windows 8/8.1에 비해 Windows 10은 훨씬 많은 win32 API를 사용 가능.게임에서 필요한 win32 API는 대부분 사용 가능.

win32 Game -> UWP Game

- 게임의 코드는 대부분 C/C++.
- 대부분의 게임이 Graphics API로 DirectX 를 사용.
- DirectX API는 99%이상 동일.
- 게임에선 win32 API를 많이 사용하지만 GDI는 거의 사용하지 않음.
- Windows 8/8.1에 비해 Windows 10은 훨씬 많은 win32 API를 사용 가능.게임에서 필요한 win32 API는 대부분 사용 가능.

Win32 Game -> UWP Game

MMORPG Client를 만든다고 하면…

Win32

- Language : C/C++
- Network : winsock
- Graphics API : Direct X 11/12
- Multi Threading :
  - _beginthread(),_endthread()
- OS UI Framework : GDI

UWP

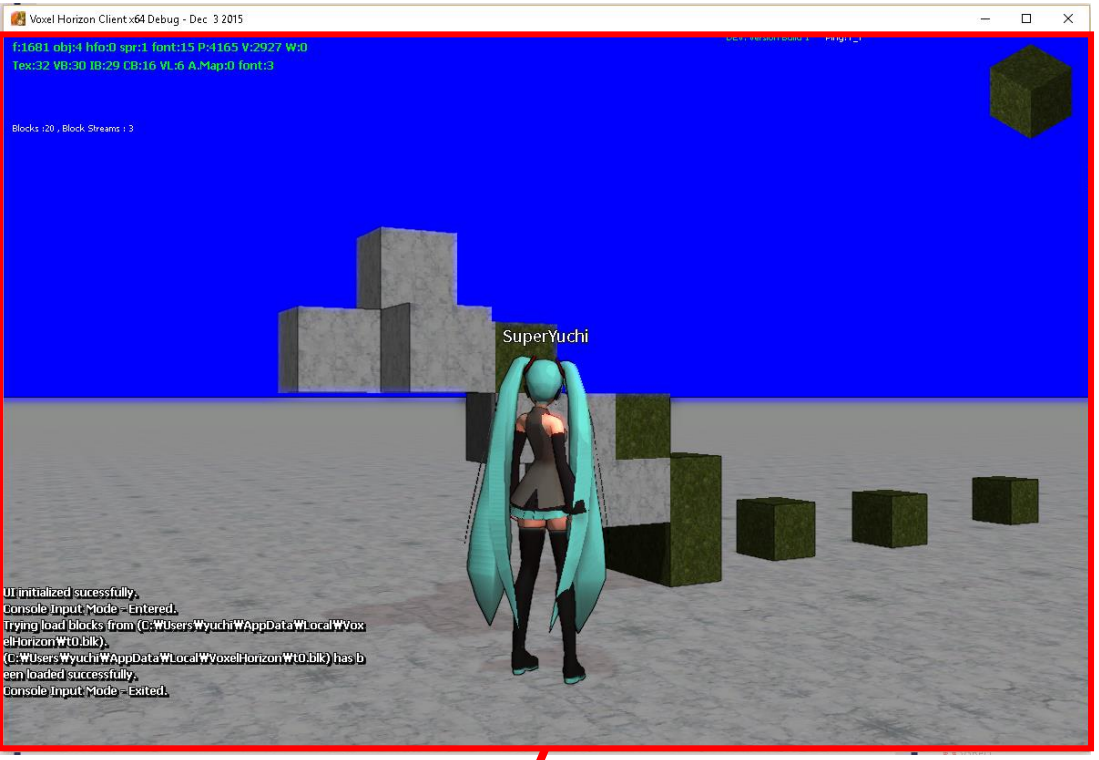
- Language : C/C++ , C++/CX or winrt/cpp
- Network : winsock
- Graphics API : DirectX 11/12
- Multi Threading :
  - _beginthread(),_endthread()
- OS UI Framework : Direct2D/Direct Write

포팅시 대응관계

- 렌더링할 윈도우 :
  - HWND -> `Windows::UI::Core::CoreWindow^ window`
- 입력 :
  - Keyboard/Mouse -> Keyboard/Mouse
  - Xinput -> Xinput , Windows::Gaming::Input

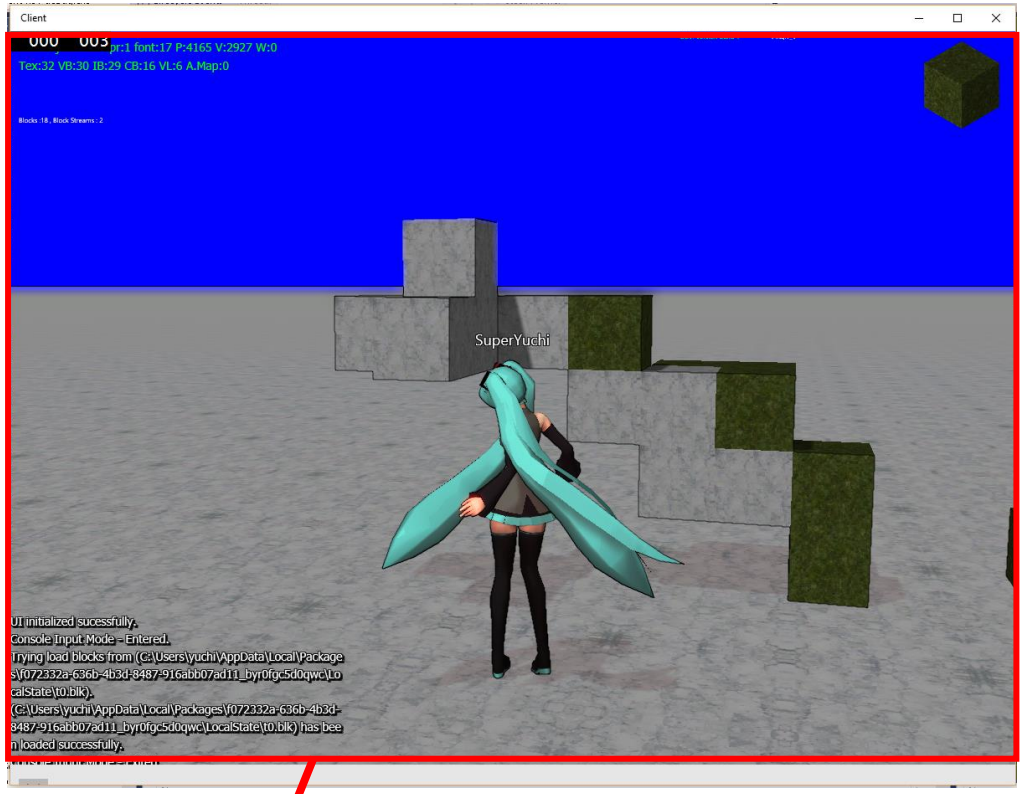
- 네트워크
  - winsock -> winsock , StreamSocket

Desktop Application – win32



GDI Windows Handle - HWND

UWP App



`Windows::UI::Core::CoreWindow`

```
void App::Initialize(CoreApplicationView^ applicationView)
{
    applicationView->Activated +=
        ref new TypedEventHandler<CoreApplicationView^, IActivatedEventArgs^>(this, &App::OnActivated);

    CoreApplication::Suspending +=
        ref new EventHandler<SuspendingEventArgs^>(this, &App::OnSuspending);

    CoreApplication::Resuming +=
        ref new EventHandler<Platform::Object^>(this, &App::OnResuming);

    CoreApplication::Exiting += ref new EventHandler<Platform::Object^>(this, &App::OnExiting);
}
```

앱 수명관리 핸들러 설정

```
void App::SetWindow(CoreWindow^ window)
{
    m_CoreWindow = window;

    DisplayInformation^ currentDisplayInformation = DisplayInformation::GetForCurrentView();

    window->SizeChanged += ref new TypedEventHandler<CoreWindow^, WindowSizeChangedEventArgs^>(this, &App::OnWindowSizeChanged);
    window->VisibilityChanged += ref new TypedEventHandler<CoreWindow^, VisibilityChangedEventArgs^>(this, &App::OnVisibilityChanged);
    window->Closed += ref new TypedEventHandler<CoreWindow^, CoreWindowEventArgs^>(this, &App::OnWindowClosed);

    currentDisplayInformation->DpiChanged += ref new TypedEventHandler<DisplayInformation^, Object^>(this, &App::OnDpiChanged);
    currentDisplayInformation->OrientationChanged += ref new TypedEventHandler<DisplayInformation^, Object^>(this, &App::OnOrientationChanged);

    DisplayInformation::DisplayContentsInvalidated += ref new TypedEventHandler<DisplayInformation^, Object^>(this, &App::OnDisplayContentsInvalidated);

    window->IsInputEnabled = true;
}
```

윈도우 관련 핸들러 설정

```
void App::OnInitComplete()
{
    m_CoreWindow->PointerPressed +=
        ref new TypedEventHandler<CoreWindow^, PointerEventArgs^>(this, &App::OnPointerPressed);

    m_CoreWindow->PointerMoved +=
        ref new TypedEventHandler<CoreWindow^, PointerEventArgs^>(this, &App::OnPointerMoved);

    m_CoreWindow->PointerReleased +=
        ref new TypedEventHandler<CoreWindow^, PointerEventArgs^>(this, &App::OnPointerReleased);

    m_CoreWindow->PointerWheelChanged += ref new Windows::Foundation::TypedEventHandler<Windows::UI::Core::CoreWindow ^, Windows::UI::Core::PointerEventArgs ^>(this, &ClientUWP::App
    m_CoreWindow->KeyDown += ref new TypedEventHandler<CoreWindow^, KeyEventArgs^>(this, &App::OnKeyDown);
    m_CoreWindow->KeyUp += ref new TypedEventHandler<CoreWindow^, KeyEventArgs^>(this, &App::OnKeyUp);
    m_CoreWindow->CharacterReceived += ref new TypedEventHandler<CoreWindow^, CharacterReceivedEventArgs^>(this, &App::OnChar);

    Windows::Devices::Input::MouseDevice::GetForCurrentView()->MouseMoved += ref new TypedEventHandler<MouseDevice^, MouseEventArgs^>(this, &App::OnMouseMoved);

    Gamepad::GamepadAdded += ref new EventHandler<Gamepad^>(&OnGamepadAdded);
    Gamepad::GamepadRemoved += ref new EventHandler<Gamepad^>(&OnGamepadRemoved);

    gestureRecognizer = ref new Windows::UI::Input::GestureRecognizer();
    gestureRecognizer->GestureSettings =
        Windows::UI::Input::GestureSettings::Hold |
        Windows::UI::Input::GestureSettings::HoldWithMouse |
        Windows::UI::Input::GestureSettings::RightTap |
        Windows::UI::Input::GestureSettings::ManipulationTranslateX |
        Windows::UI::Input::GestureSettings::ManipulationTranslateY |
        Windows::UI::Input::GestureSettings::ManipulationScale |
        Windows::UI::Input::GestureSettings::ManipulationRotate |
        Windows::UI::Input::GestureSettings::ManipulationTranslateInertia |
        Windows::UI::Input::GestureSettings::ManipulationScaleInertia |
        Windows::UI::Input::GestureSettings::ManipulationRotateInertia;

    // Register all the delegates
    _tokenRightTapped = gestureRecognizer->RightTapped::add(
        ref new Windows::Foundation::TypedEventHandler<
        Windows::UI::Input::GestureRecognizer^, Windows::UI::Input::RightTappedEventArgs^>(
            this, &App::OnRightTapped));
```

입력 핸들러 설정

게임루프

UWP

```
// This method is called after the window becomes active.
void App::Run()
{
    while (!m_windowClosed)
    {
        if (m_windowVisible)
        {
            m_CoreDispatcher->ProcessEvents(CoreProcessEventsOption::ProcessAllIfPresent);

            if (m_pGame)
            {
                m_pGame->Process();
            }
        }
        else
        {
            m_CoreDispatcher->ProcessEvents(CoreProcessEventsOption::ProcessOneAndAllPending);
        }
    }
}
```

win32

```
// Main message loop:
while (1)
{
    BOOL    bHasMsg = PeekMessage(&msg, nullptr, 0, 0, PM_REMOVE);

    if (bHasMsg)
    {
        if (msg.message == WM_QUIT)
        {
            break;
        }
        TranslateMessage(&msg);
        DispatchMessage(&msg);


    }
    else
    {
        g_pGame->Run();

    }
    g_bCanUseWndProc = FALSE;
}
```

샘플코드 분석

https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/Simple3DGameDX

종료 처리

- 원래 UWP App(UWP API로 작성된)은 명시적 종료 처리가 없다.
- 미친 MS가 애플 잘되는거 보고 못된것만 배워가지고…
- 종료 시점에서 체크해야할 것들을 몽땅 놓치게 된다.
- 명시적 종료를 만들것.
  - Heap corruption 체크
  - Resource leak 체크
  - 기타 버그체크등..

Tip

- Win32 + DirectX 11로 먼저 개발. 막바로 UWP로 개발하는거 자살행위.
- 종료처리 반드시 할 것. 종료처리시 heap corruptio이나 resource leak을 확인한다.
- Win32빌드와 최~~~~대한 코드를 공유할것.
- UWP 환경에서 지원되지 않는 함수나 자료구조는 모양이라도 똑같이 맞출것.
  - MessageBox(), HWND등..
- Win32 빌드와 UWP빌드의 코드를 한방에 비교할 수 있는 diff 스크립트를 짜둘것.

문제점

- Feature Level 10 제한 때문에 결국 DirectX 12를 사용해야한다. XDK는 이 제약이 없다고 알고 있다.
- DX12코드 짜는게 엄청 빡시다.
- 정작 DX12로 짜도 구형 XBOX ONE/XBOX ONE S에선 DX12가 정상작동하지 않는다. 성능이 크게 떨어지고 화면이 깨지거나 반점이 생기는등 문제가 발생한다. XBOX ONE X, XBOX ONE Series X/S에선 테스트해보지 않았음.
- 죽을 고생을 해서 개발해도, ID@XBOX를 통해서 출시하지 않는한 온라인(MO,MMO)게임으로는 출시할 수 없다. (Live Creator프로그램 규정)

참고자료

- https://docs.microsoft.com/ko-kr/windows/uwp/xbox-apps/
- https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/Simple3DGameDX
- https://walbourn.github.io/directx-and-uwp-on-xbox-one/
- https://www.slideshare.net/dgtman/f1-c-windows-10-uwp