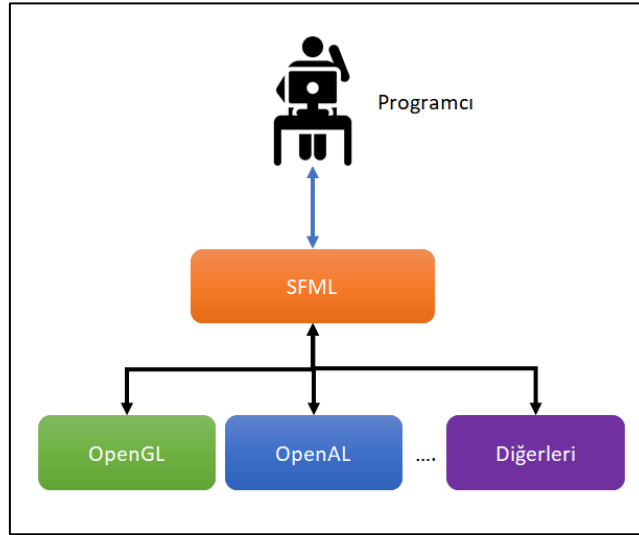


HAFTA 1: SFML Giriş

SFML Nedir?

Sfml (Simple and Fast Multimedia Library) video, ses veya grafik çizim ihtiyacı olan uygulamaları kolayca geliştirmek için tasarlanmış bir kütüphanedir. SFML yapacağı işlemler için başka kütüphaneleri kullanmaktadır. Örneğin 3D çizim için OpenGL kütüphanesi kullanılmaktadır. SFML kütüphanesinin amacı programcılara daha kolay kullanabilecekleri bir API(application programming interface) sunmaktır.



Şekil 1.1. SFML Kütüphanesinin Kullandığı Kütüphaneler.

SFML Windows, Linux ve Mac OS X gibi bir çok platformda çalışabilmektedir. Ayrıca C, .NET, C++, Java, Ruby, Python ve Go gibi dillerde kullanılabilir. SFML kütüphanesi temelde 5 modüle ayrılmıştır. Bunlar;

Ses Modülü: Müzik ve ses çalmak için veya kaydetmek ile görevlidir.

Window Modülü: OpenGL tabanlı bir pencere üretmekte ve giriş-çıkış olaylarını kontrol eder.

Grafik Modülü: Çizim için gerekli olan bütün araçları içerisinde barındırır.

Network Modülü: Başka bilgisayarlar ile uygulama arasında paket alışverişinden sorumludur.

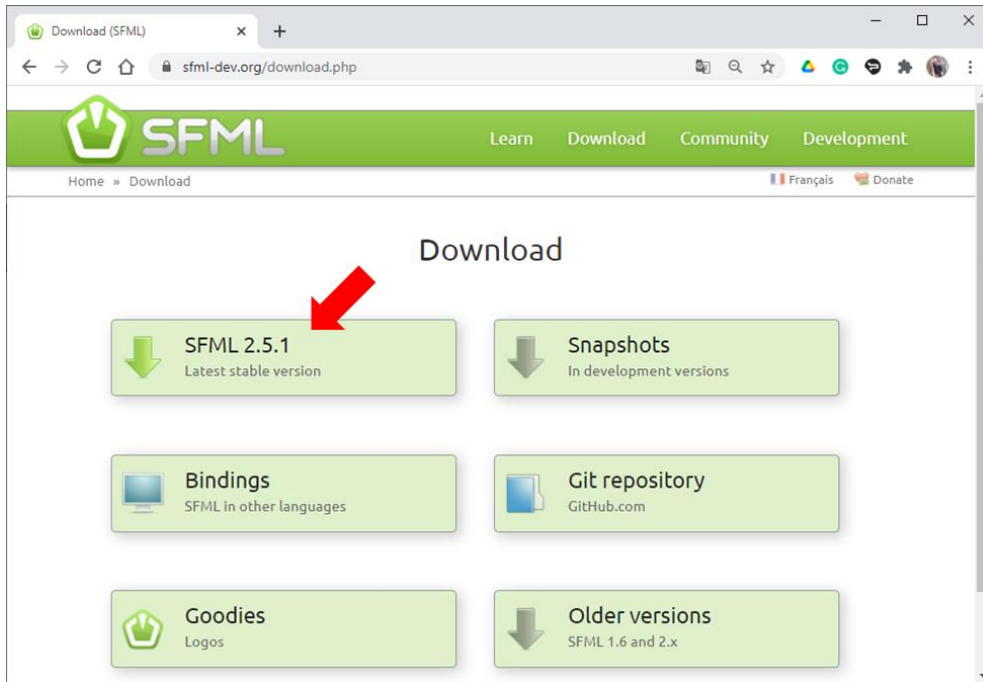
Sistem Modülü: Temel veri yapılarını, İplik, saat ve diğer önemli araçları barındırır.

SFML Lisansı

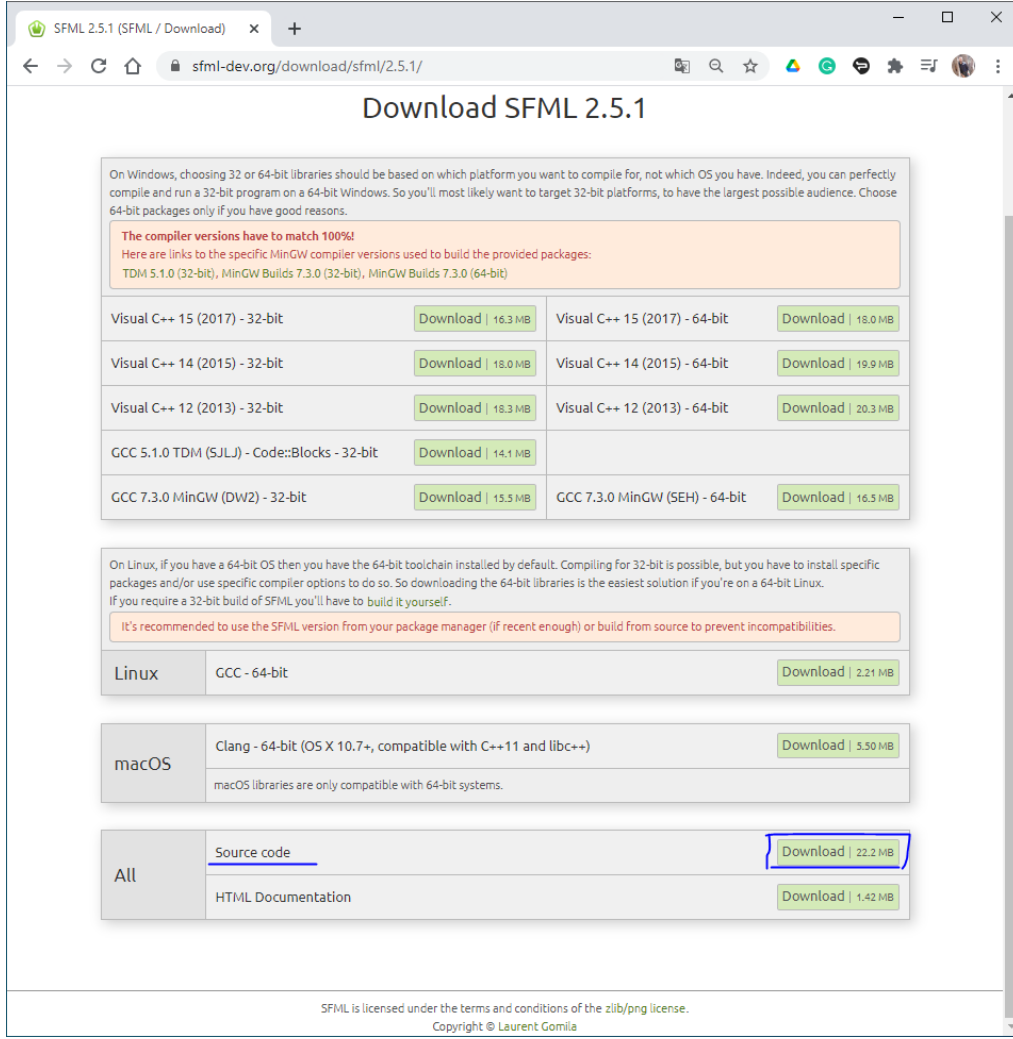
SFML kütüphanesi zlib/libpng lisansına sahiptir. SFML kütüphanesini kullanarak yaptığınız uygulamaları satabilirsiniz. Detaylı lisans bilgileri için <https://opensource.org/licenses/Zlib> adresini ziyaret edebilirsiniz.

SFML Kurulumu

SFML Kütüphanesi indirmek için <https://www.sfml-dev.org/download.php> sayfasına gidip en son sürümün bağlantısına tıklayabilirsiniz. Bu notları yazarken en son sürüm 2.5.1'dir. Kullanıcılar isterse SFML kütüphanesinin kaynak kodları kullanılarak istenilen platformlar için tekrardan derleme işlemi gerçekleştirilebilir. Şekil 1.2'de gösterilen bağlantıya tıkladığında karşımıza şekil 1.3'de gösterilen sayfa çıkacaktır. Buradan çeşitli geliştirme platformları için derlenmiş hazır kütüphaneleri indirebileceğimiz gibi en aşağıda bulunan kaynak kod dosyalarını da indirebiliriz. Notları yazarken SFML kütüphanesi ders için kullanacağımız Visual Studio 2019 için derlenmediği için kaynak dosyaları indireceğiz. Kaynak dosyaları ile CMake aracı kullanarak SFML kütüphanesini visual Studio 2019 için derlenebilecek hale getireceğiz.



Şekil 1.2. SFML Kütüphanesi İndirme Sayfası.

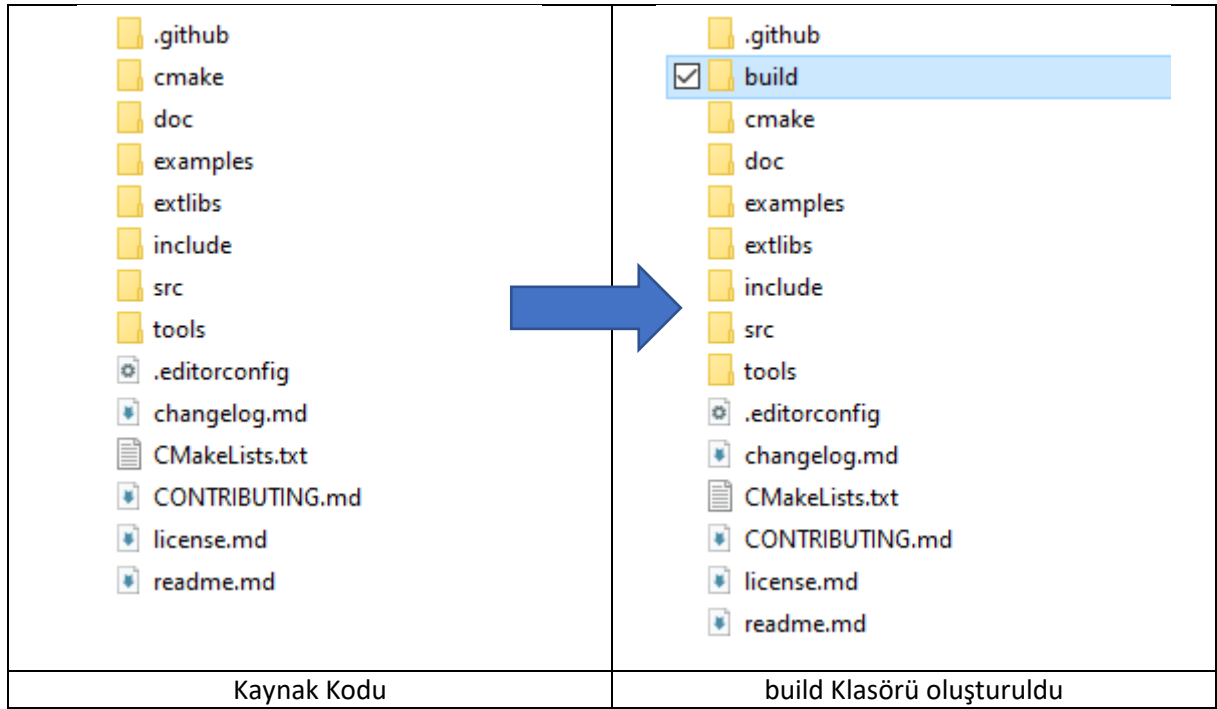


Şekil 1.3. SFML Kütüphanesinin Kaynak Kodunu İndirme Bağlantısı

CMake ile SFML'i Derlenmeye Hazırlama

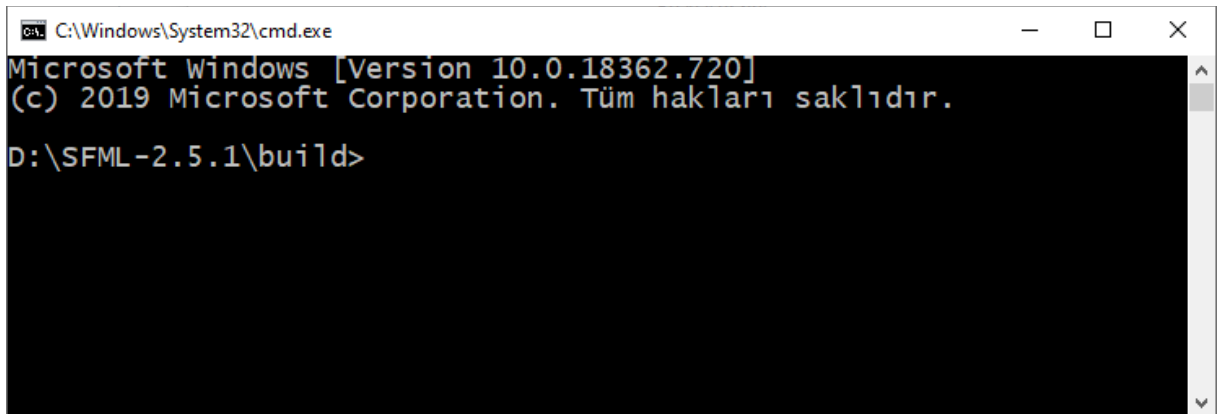
SFML kaynak kodunu Visual Studio 2019'da derlenebilecek hale getirmek için CMake uygulamasını kullanmamız gerekir. CMake aracı projeleri istenilen platformlarda derlenebilecek hale getirmektedir. SFML CMake uyumlu olarak tasarlanmıştır bu sayede SFML kütüphanesini CMake aracının desteklediği bütün platformlarda derlenebilecek hale getirebiliriz. CMake kütüphanesinin indirilmesi ve kurulması bu dersin içerikleri dışarındadır. İhtiyaç duyanlar için internet üzerinde Cmake kurulumunu anlatan bir çok kaynak mevcuttur.

SFML kaynak kodu indirildikten sonra sıkıştırılmış dosyalar açılmalıdır. Ardından kaynak kod klasörü içerisinde "build" adlı bir klasör oluşturulmalıdır. Klasörün ismi önemli olmadığı için farklı bir isim seçilebilir. CMake işlemi sonucunda elde edilecek dosyalar bu klasör içerisine yerleştirilecektir.



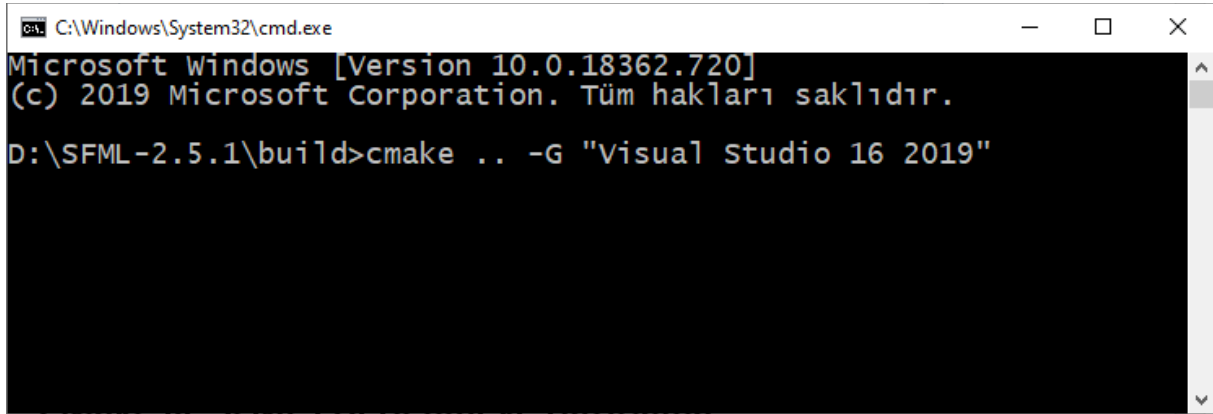
Şekil 1.4. SFML Kütüphanesinin Kaynak Kodu

CMake uygulamasının pencere uygulaması kullanılacağı gibi komut satırı uygulaması da kullanılabilir. Biz bu derste komut satırı uygulamasını kullanılacaktır. Komut satırına indikten sonra SFML içerisinde oluşturulan “build” klasörüne gireceğiz. Kolaylık olması için SFML klasörünü D sürücüsü içerisine yerleştirebilirsiniz. Komut satırı içerisindeyken CMAKE komutu kullanarak SFML kütüphanesini Visual Studio 2019 da derlenebilecek hale getireceğiz. Bunun için “Visual Studio 16 2019” parametresini kullanacağız. Şekil 1.6’da bu komut gösterilmiştir.



Şekil 1.5. Komut Satırında SFML Klasörüne Geçiş

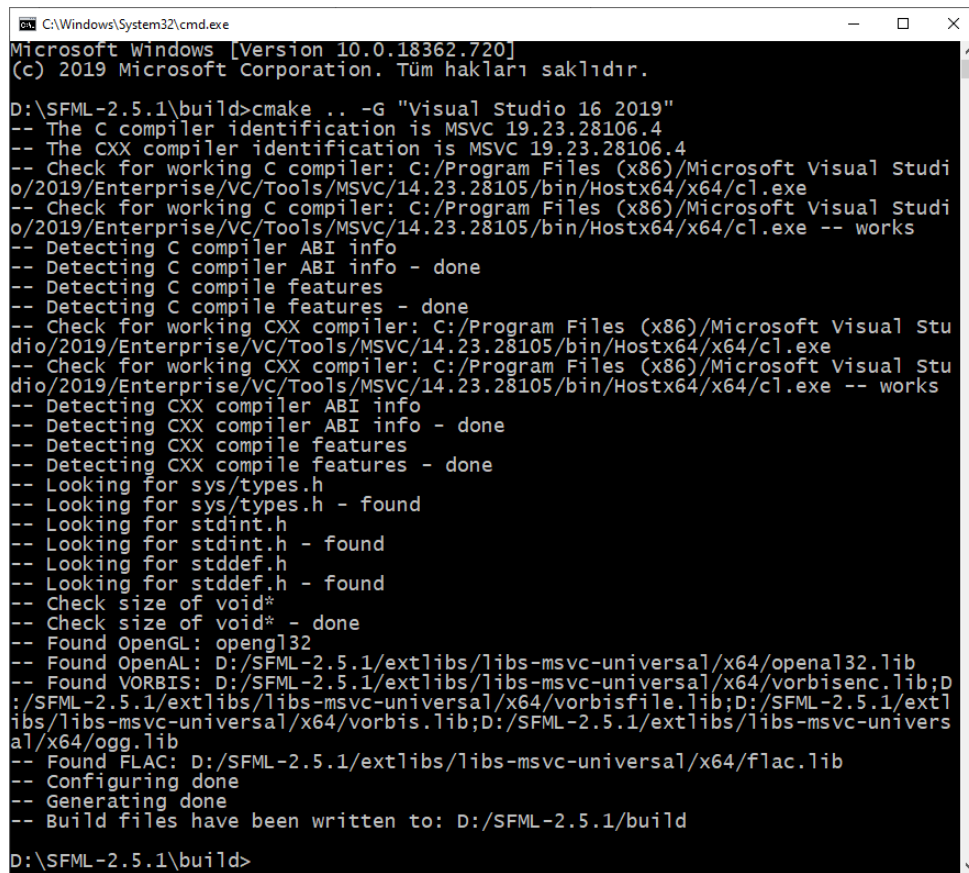
Cmake kullanılırken “..” parametresi kullanılmıştır. Bu parametre inşa edilecek projenin yolunu vermektedir. “..” kullanarak Cmake’in çağrıldığı klasörden bir önceki klasörün inşa edileceğini belirtmiş oluyoruz. İkinci parametre olarak -G kullanılmıştır. Bu parametre CMake programına çalışacağı platformu bizim vereceğimizi belirtmek için kullanılmaktadır. Bu parametre kullanılmadığında CMake varsayılan platformu kullanacaktır. Son parametre olarak da kullanacağımız platformu veriyoruz. Komut çalıştırıldığında şekilde 1.7’de görüldüğü gibi inşa işlemi çalıştırılacaktır. Bu işlem sonucunda Visual Studio 2019 için bir solution dosyası elde edeceğiz.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.

D:\SFML-2.5.1\build>cmake .. -G "Visual Studio 16 2019"
```

Şekil 1.6. Komut Satırında CMake kullanımı



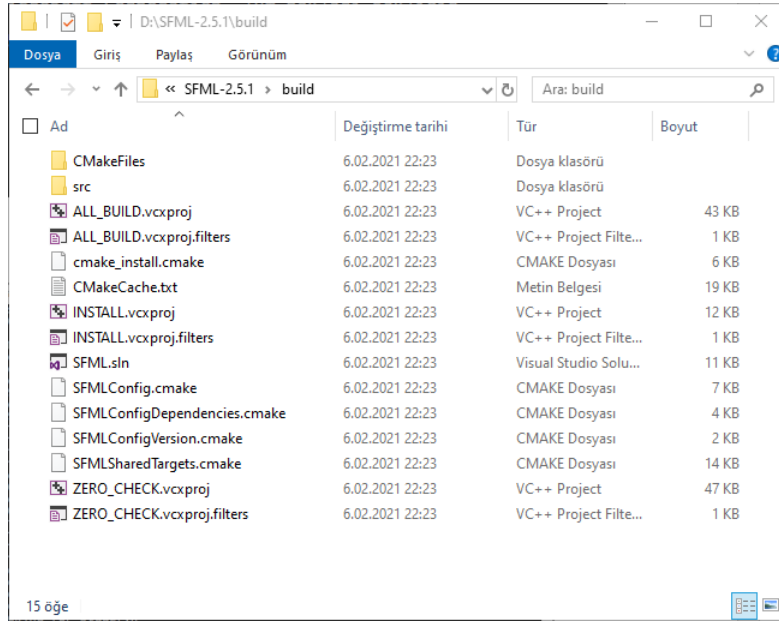
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.

D:\SFML-2.5.1\build>cmake .. -G "Visual Studio 16 2019"
-- The C compiler identification is MSVC 19.23.28106.4
-- The CXX compiler identification is MSVC 19.23.28106.4
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studi
o/2019/Enterprise/VC/Tools/MSVC/14.23.28105/bin/Hostx64/x64/cl.exe
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studi
o/2019/Enterprise/VC/Tools/MSVC/14.23.28105/bin/Hostx64/x64/cl.exe -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Stu
dio/2019/Enterprise/VC/Tools/MSVC/14.23.28105/bin/Hostx64/x64/cl.exe
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Stu
dio/2019/Enterprise/VC/Tools/MSVC/14.23.28105/bin/Hostx64/x64/cl.exe -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for sys/types.h
-- Looking for sys/types.h - found
-- Looking for stdint.h
-- Looking for stdint.h - found
-- Looking for stddef.h
-- Looking for stddef.h - found
-- Check size of void*
-- Check size of void* - done
-- Found OpenGL: opengl32
-- Found OpenAL: D:/SFML-2.5.1/extlibs/libs-msvc-universal/x64/openal32.lib
-- Found VORBIS: D:/SFML-2.5.1/extlibs/libs-msvc-universal/x64/vorbisenc.lib;D
:/SFML-2.5.1/extlibs/libs-msvc-universal/x64/vorbisfile.lib;D:/SFML-2.5.1/extl
ibs/libs-msvc-universal/x64/vorbis.lib;D:/SFML-2.5.1/extlibs/libs-msvc-univers
al/x64/ogg.lib
-- Found FLAC: D:/SFML-2.5.1/extlibs/libs-msvc-universal/x64/flac.lib
-- Configuring done
-- Generating done
-- Build files have been written to: D:/SFML-2.5.1/build

D:\SFML-2.5.1\build>
```

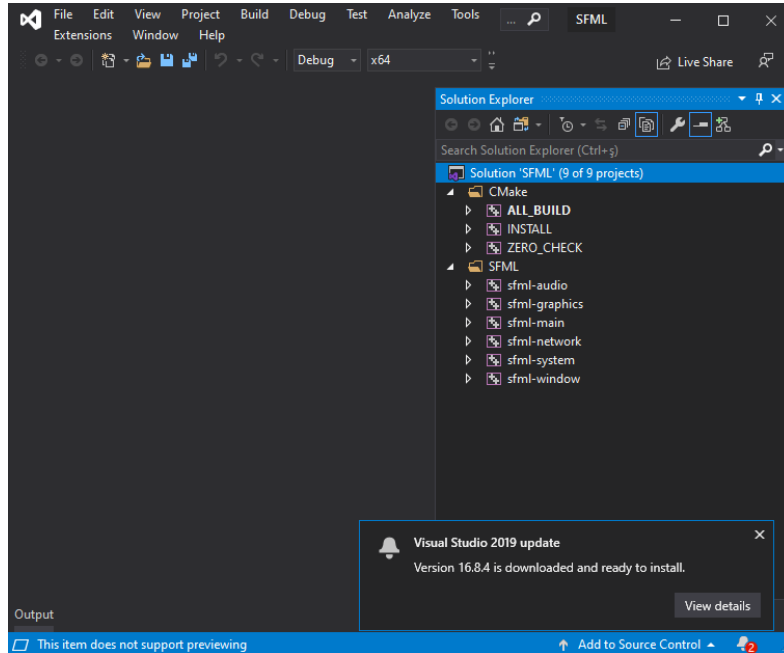
Şekil 1.7. Komut Satırında CMake işleminin sonucu

İşlem sonucunda klasör içerisinde SFML solution dosyası oluşacaktır. Şekil 1.8’de bu dosya gösterilmektedir. Bu dosyayı çift tıklayarak çalıştıracğız.



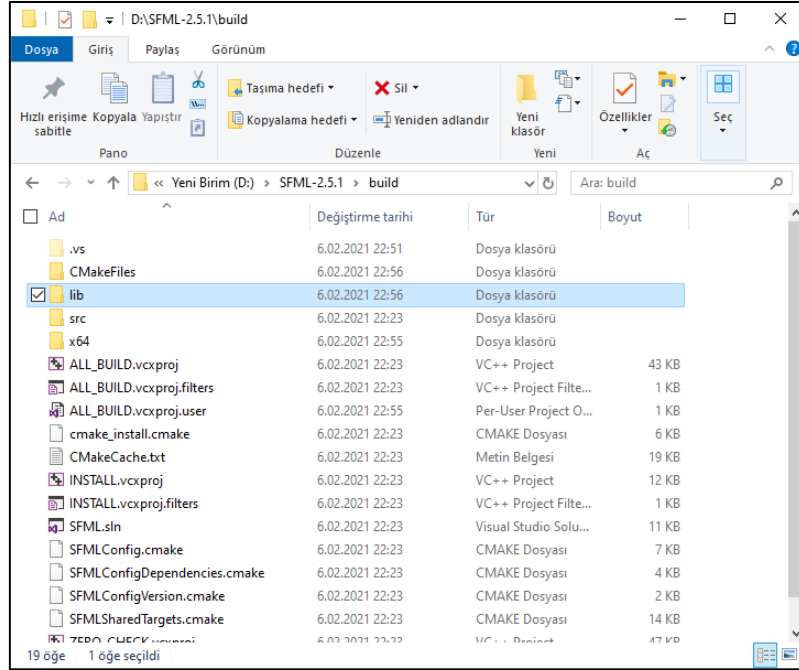
Şekil 1.8. SFML Solution Dosyası

Solution dosyası açıldığında tek yapılması gereken “Build” menüsünden “Build Solution” seçeneğini tıklamaktır. İnşa işlemi bittiğinde projelerimizde kullanılacak olan kütüphane dosyaları oluşturulmuş olacaktır.

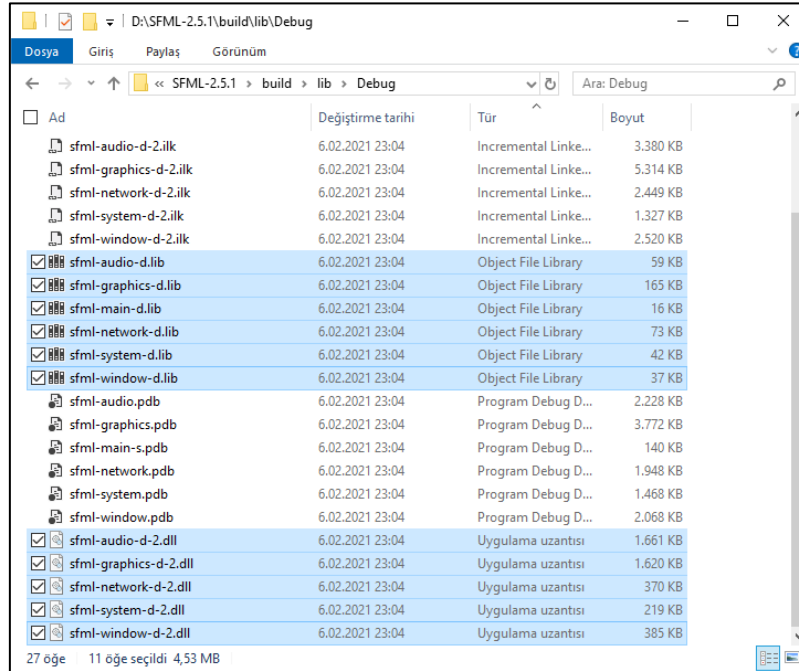


Şekil 1.9. SFML Solution Dosyası

Derleme işlemi bittiğinde “build” klasörü içerisinde “lib” adlı bir klasör oluşturulmuştur. Derleme sonucunda oluşturulan kütüphaneler bu klasör içerisindedir. İnşa işlemi “Debug” modunda yapıldığında kütüphanelerde “-d” eklentisi görülmektedir. Bu sayede projelerimizde hata bulma işlemleri daha kolay olacaktır.

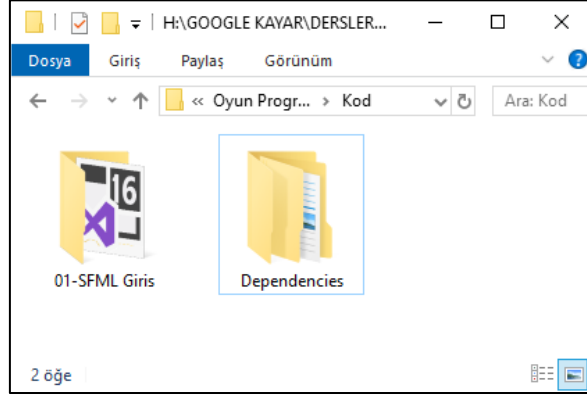


Şekil 1.10. Kütüphane Klasörü



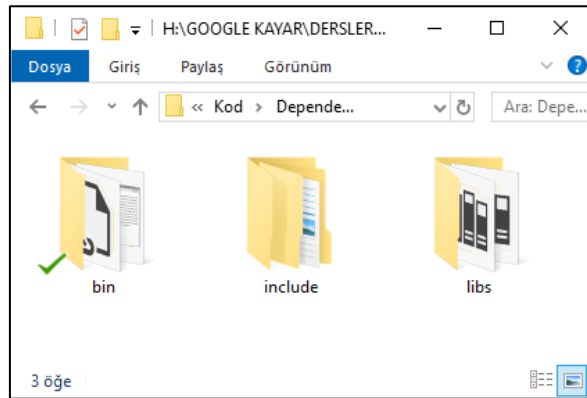
Şekil 1.11. SFML Kütüphane dosyaları

Sırada kütüphane dosyalarımızı projelerimizi saklayacağımız klasöre taşımakta. Bu notları hazırlarken projeler için “**Kod**” isimli bir klasör oluşturuldu. Ardından bu klasör içerisine “**Dependencies**” isimli bir klasör oluşturuldu. İsterseniz farklı bir isimde kullanabilirsiniz.

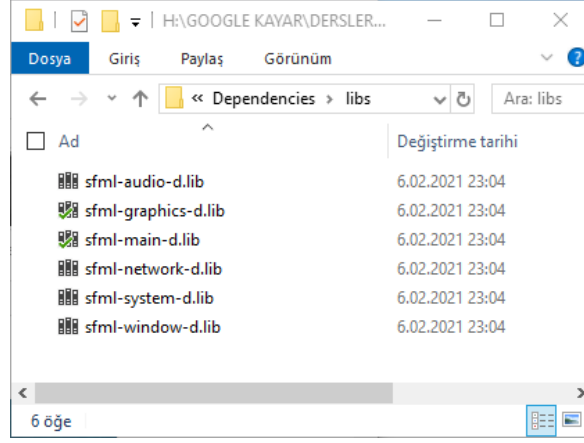


Şekil 1.12. Kod klasörünün içeriği

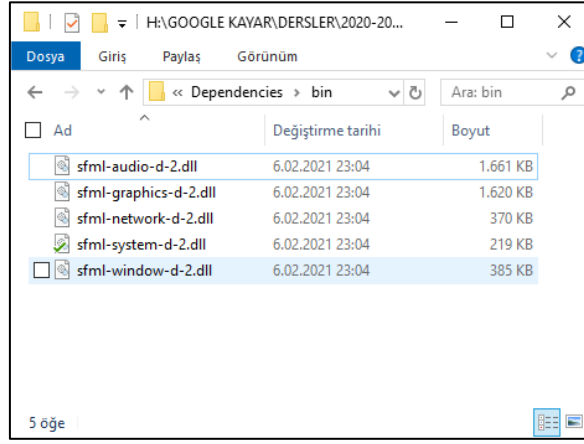
“**Dependencies**” klasörü içerisine aşağıdaki gibi üç klasör oluşturacağız. SFML kütüphanesinin derlenmesi sonucunda elde ettiğimiz **.lib** uzantılı dosyaları “**libs**” klasörüne kopyalayacağız. Aynı şekilde **.dll** uzantılı dosyaları da “**bin**” klasörüne kopyalayacağız. SFML kaynak kodu içerisindeki **include** klasörünün içeriğini de “**include**” klasörü içerisine kopyalayacağız. Bu klasörlerin içeriği sırasıyla Şekil 1.14, 1.15 ve 1.16’da gösterilmiştir.



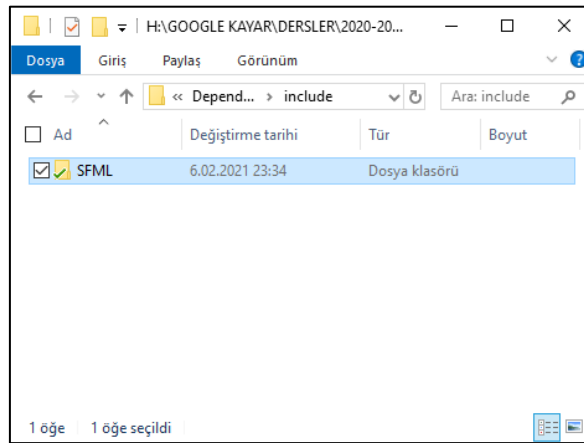
Şekil 1.13. Kod klasörünün içeriği



Şekil 1.14. “libs” klasörünün içeriği



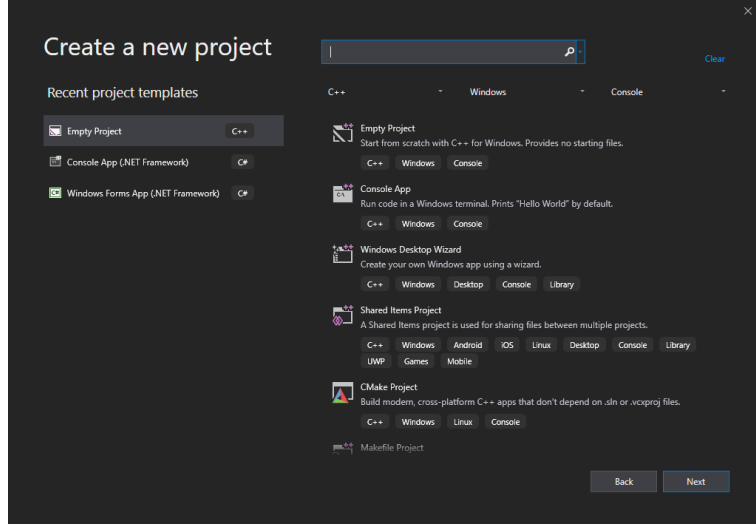
Şekil 1.15. “bin” klasörünün içeriği



Şekil 1.15. “include” klasörünün içeriği

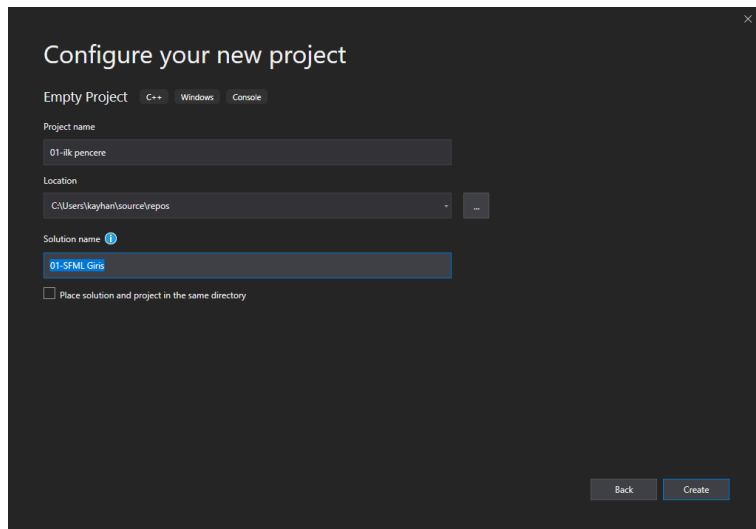
Visual Studio 2019 ile İlk Projeyi Oluřturma

Öncelikle Visual Studio 2019 programı çalıştırılır ve yeni bir proje oluřturma seçeneęi tıklanır. Karřımıza ařaęıdaki panel çıkacaktır. Buradan **“Empty Project”** seçeneęini tıklayacaęız.

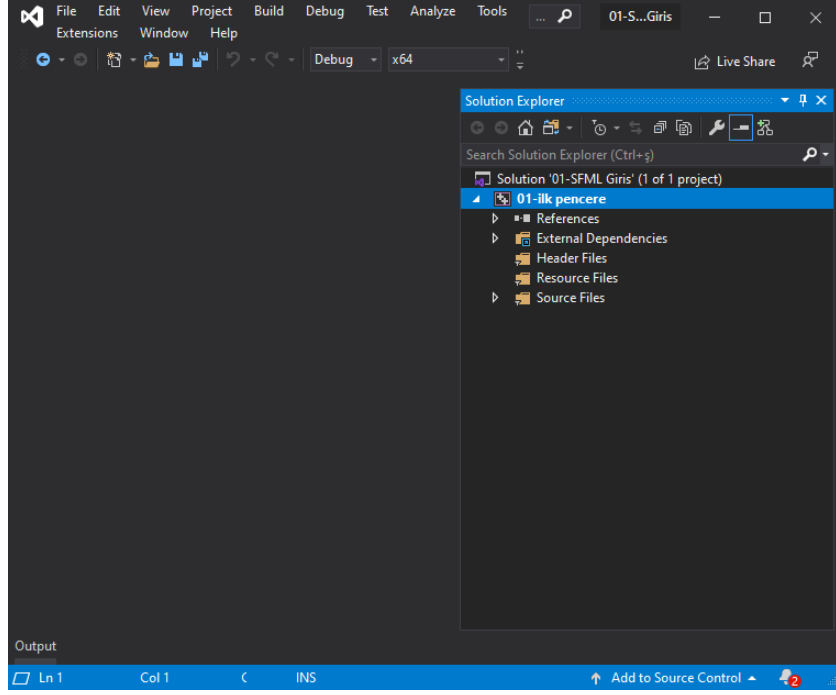


řekil 1.16. Boř Proje Oluřturma

Daha sonra oluřturacaęımız projenin ismini belirteceęiz. Visual Studio projeleri **“Solution”** isimli bir blok ierisinde tutmaktadır. Bir biri ile alakalı olan projeler bir **“solution”** ierisinde bulunur. **“Solution”** ile projelerin aynı klasörde olmasını istemedięimizden en alttaki seçeneęi kaldırıyoruz. Burada seçim size kalmıř. Solution ismi ile ilk projemizin ismini farklı seçiyoruz. **“solution”** 1. Haftanın konusunu temsil ederken projeler daha özel isimler alacaktır. Seçimler tamamlandıktan sonra **“Create”** butonuna tıklanır ve proje oluřturulur.



řekil 1.17. Proje ve solution özellikleri



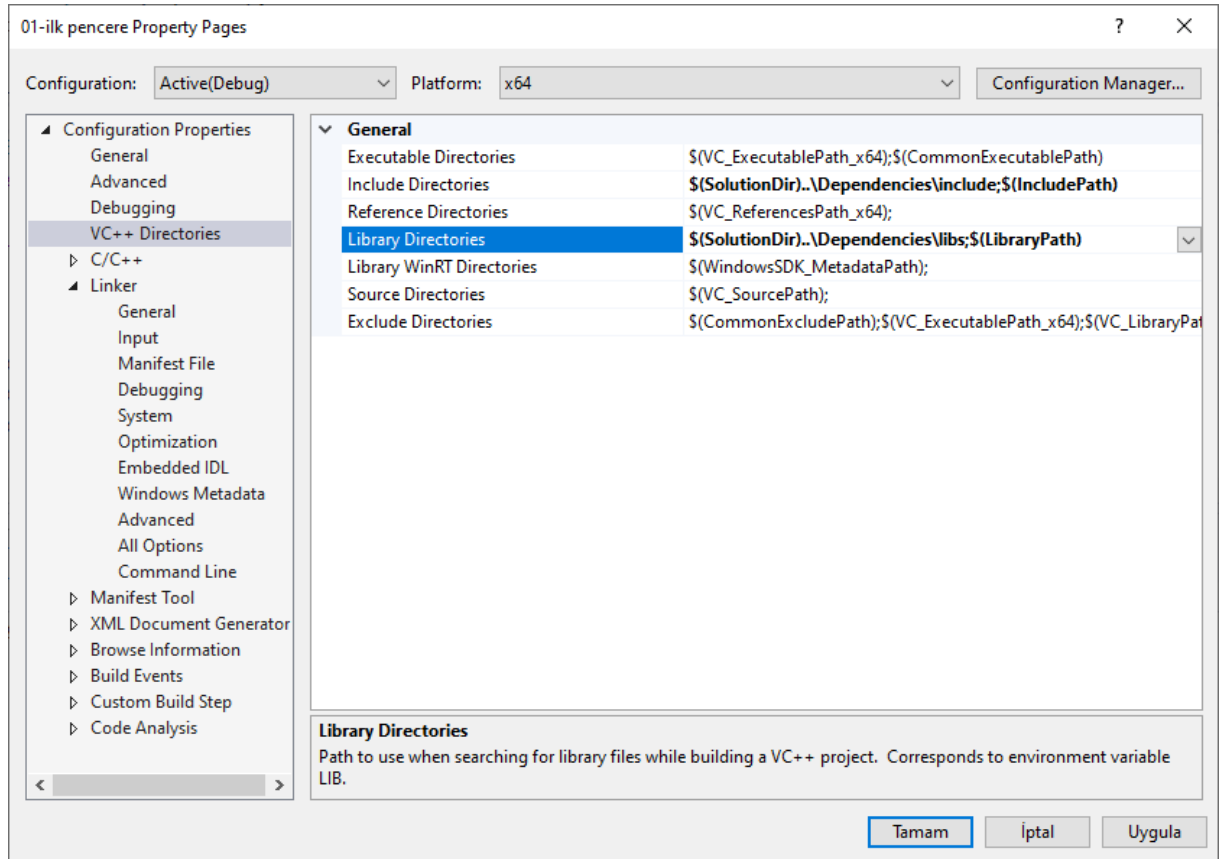
Şekil 1.18. Oluşturulan Projenin Görüntüsü

Projeyi SFML kütüphanesine bağlama

Oluşturacağımız projelerin SFML kütüphanesine bağlanması işleminde pratik bir teknik kullanacağız. Bu teknik sayesinde projelerimizi istediğimiz bilgisayara veya başka bir klasöre taşıdığımızda sorunsuz bir şekilde çalışmaya devam edecekler. Bunun için projelerimizin ait oldukları “Solution” klasörlerini projelerimizin bağımlı olduğu kütüphanelerin bulunduğu “Dependencies” klasörünü Şekil 1.12 de görüldüğü gibi aynı klasör içerisine yerleştirdik. Bu klasör hiyerarşisini koruduğumuz sürece projelerimiz doğru şekilde derlenip çalışacaktır.

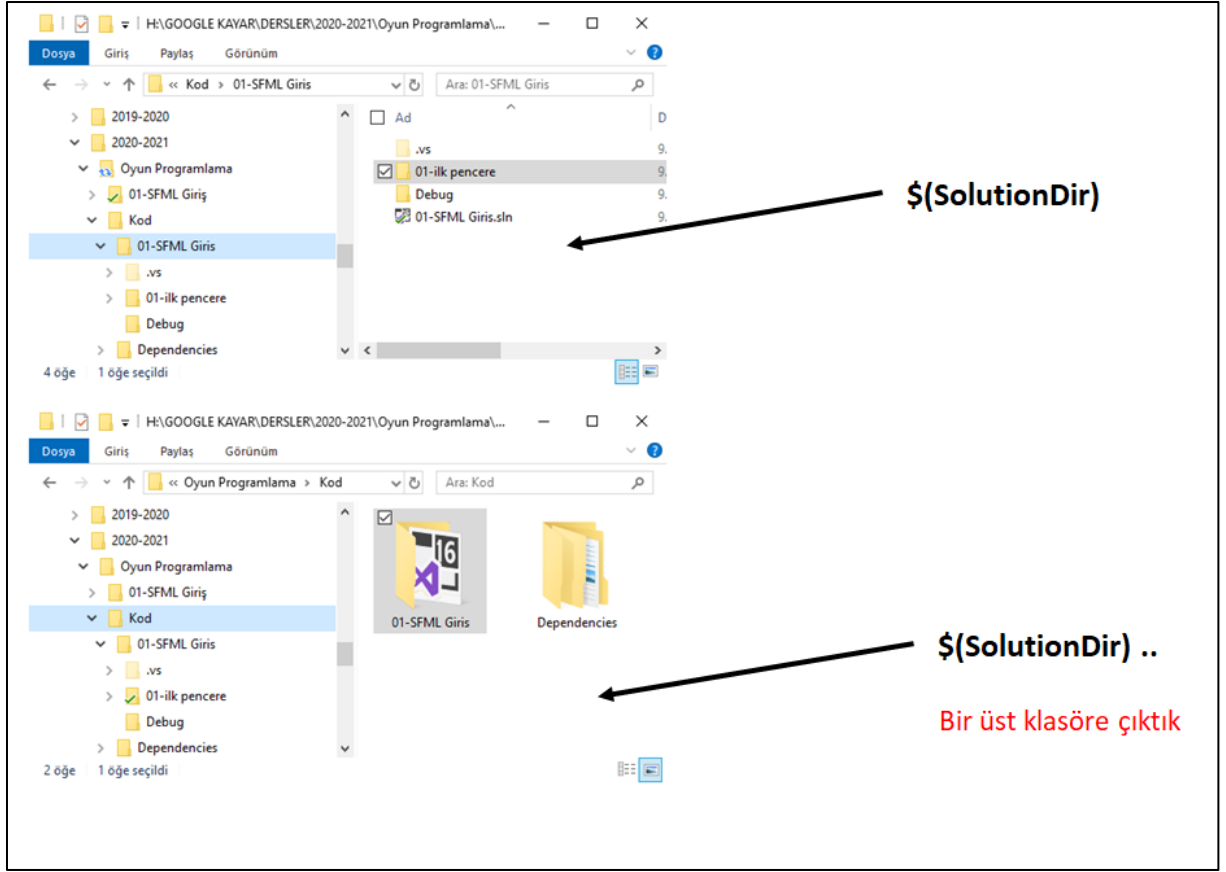
Bu işlemi gerçekleştirmek için göreceli adres gösterme tekniğini kullanacağız. Projemizin bağlı olduğu kütüphanelerin bulunduğu klasörlerin tam yolunu vermektense projemize göre konumlarını kullanacağız. Kütüphanelerin konumu projelerimize göre aynı konumda kaldığı sürece sorun yaşanmayacaktır. Örneğin bu derste “Dependencies” klasörünün bulunduğu alana projelerimizin klasörlerinden ulaşmak istersek iki defa üst klasöre çıkmamız gerekecektir. Bu ilişki korunduğu sürece projelerimiz her konumda çalışacaktır.

Peki projelerimizle SFML kütüphanelerini nasıl bağlayacağız. Bunun için öncelikle proje seçili iken “view” menüsünden “Property Pages” seçeneğini tıklamamız gerekir. Karşımıza Şekil 1.19’daki panel çıkacaktır. Bu panelde “VC++ Directories” seçeneğini tıklayacağız. Buradan projemize bağlı olacak ekstra klasörlerin yollarını göstereceğiz. İlk olarak “Include Directories” seçeneğine aşağıda görüldüğü gibi SFML kütüphanesinin “include” klasörünün yolunu gösteriyoruz.



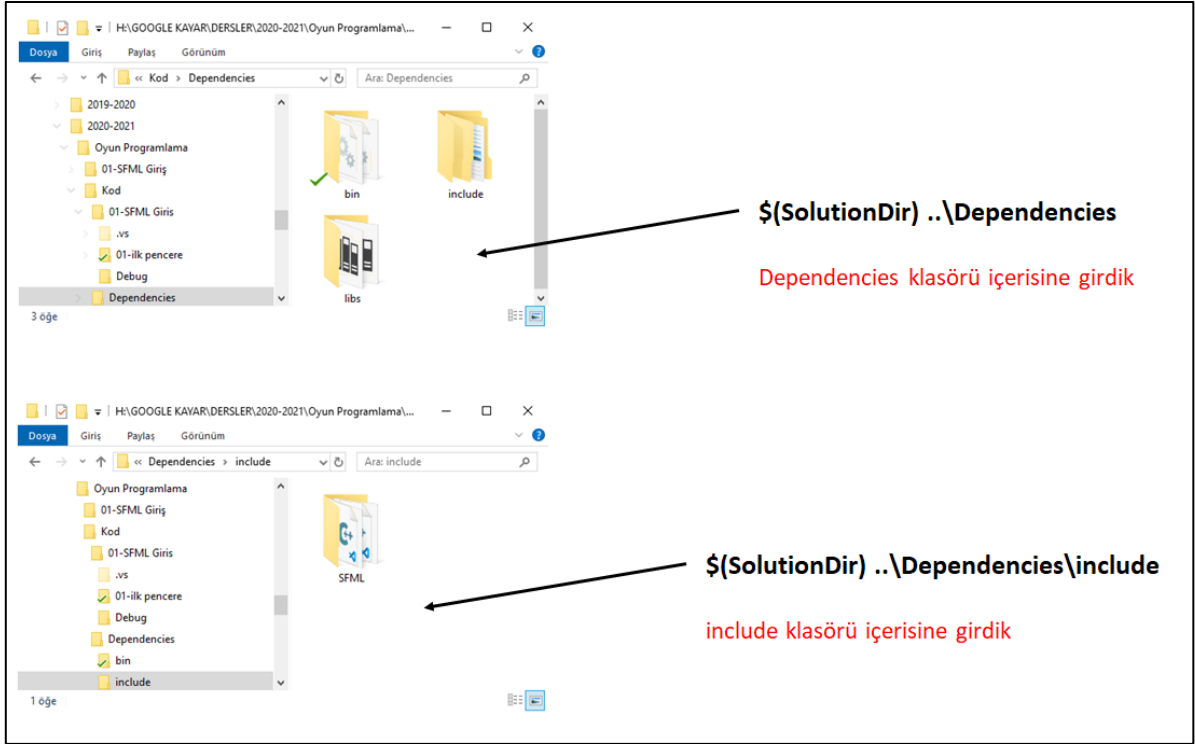
Şekil 1.19. “Property Pages” Paneli

Dikkat ediyorsanız yolu gösterirken “\$(SolutionDir)” değişkenini kullanıyoruz. Bu değişkenin değeri Visual Studio tarafından verilmektedir. Değişken projenizin bağlı olduğu “Solution” dosyasının bulunduğu klasörün tam yolunu tutmaktadır. Böylece projemizi bilgisayarın hangi klasörüne taşırsak taşıyalım \$(SolutionDir) değişkeni bize projemizin ait olduğu “Solution” dosyasının bulunduğu klasörün yolunu verecektir. “..” nokta kullanarak bu klasörden bir alt klasöre iniyoruz. Bu işlemin sonucu aşağıdaki şekilde gösterilmektedir.



Şekil 1.20. \$(SolutionDir) değişkeni

Şekil 1.20. 'e dikkatle bakarsanız “\$(SolutionDir)..” ifadesini kullandığımızda “Solution” klasörümüzü barındıran klasörün içerisine girdiğimizi göreceksiniz. “Dependencies” klasörünü de tam bu noktaya yerleştirmiştik. İlerde projemizi taşımak istersek tek yapmamız gereken “Kod” klasörünü taşımak olacaktır. Projelerimizin ihtiyaç duyduğu bütün kütüphanelerin bulunduğu “Dependencies” klasörü de projelerimize ile beraber taşınacaktır. Şekil 1.21’ de include klasörüne erişimin nasıl yapıldığı gösterilmektedir.

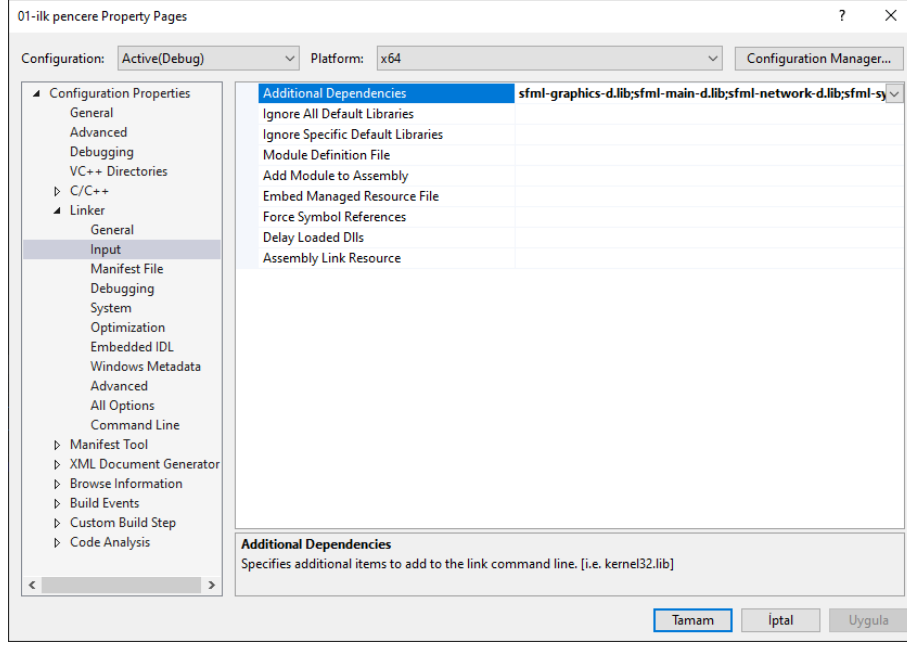


Şekil 1.20. \$(SolutionDir) değişkeni

Aynı işlemleri “.lib” dosyalarını barındıran klasör içinde yapmamız gerekiyor. İşlem “include” klasörü ile aynı adımları barındırmaktadır. Tek fark tam yol “\$(SolutionDir)..\Dependencies\lib” olacaktır.

Kütüphane yolları gösterildikten sonra projelerimizin bağlama aşamasında kullanılacak olan .lib dosyalarının belirtilmesi gerekmektedir. Bunun için yine “Property pages” panelinden “Linker” seçeneğini tıklıyoruz. Buradan da “input” seçeneğine tıklıyoruz. Karşımıza Şekil 1.21.’deki panel çıkacaktır. Bu paneldeki ilk seçenek olan “Additional Dependencies” aşağıdaki .lib dosyalarının isimlerini yerleştiriyoruz.

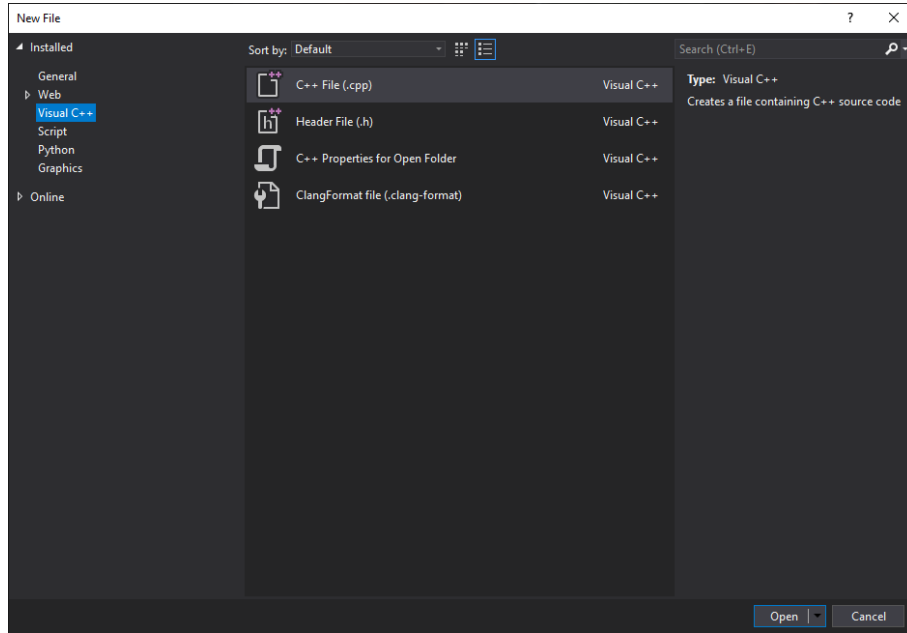
```
sfml-graphics-d.lib  
sfml-main-d.lib  
sfml-network-d.lib  
sfml-system-d.lib  
sfml-audio-d.lib  
sfml-window-d.lib
```



Şekil 1.21. .lib dosyalarının isimleri değişkeni

İlk Uygulama

İlk uygulamamız için bir kaynak dosyasına ihtiyacımız olacak . Bu işlem için File menüsünden “New” seçeneği tıklanır . ardından karşımıza çıkan panelden “file” seçeneğini seçeceğiz. Karşımıza aşağıdaki panel çıkacaktır. Bu panelin solundan “Visual C++” seçeneğini tıklayacağız ve sağda karşımıza çıkanlardan “C++ File” seçeneğini tıklayacağız. Dosyasınıza istediğiniz ismi verebilirsiniz (Türkçe karakter kullanmayın)

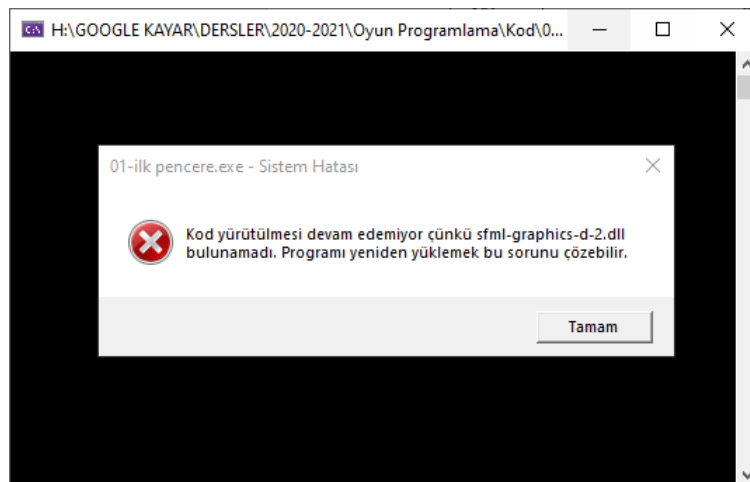


Şekil 1.22. Yeni Dosya Ekleme

Kod dosyası içerisine aşağıdaki kodu yapıştırın. Programı direk olarak çalıştırırsanız Şekil 1.23.'deki hatayı alacaksınız. Derleme sonucu oluşan çalıştırılabilir dosyamız sfml kütüphanesine ait olan .dll dosyalarına ihtiyaç duymakta ve onları bulamadı içinde hata vermektedir.

```
01 #include <SFML/Graphics.hpp>
02
03 int main()
04 {
05     sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
06     sf::CircleShape shape(100.f);
07     shape.setFillColor(sf::Color::Green);
08
09     while (window.isOpen())
10     {
11         sf::Event event;
12         while (window.pollEvent(event))
13         {
14             if (event.type == sf::Event::Closed)
15                 window.close();
16         }
17
18         window.clear();
19         window.draw(shape);
20         window.display();
21     }
22
23     return 0;
24 }
```

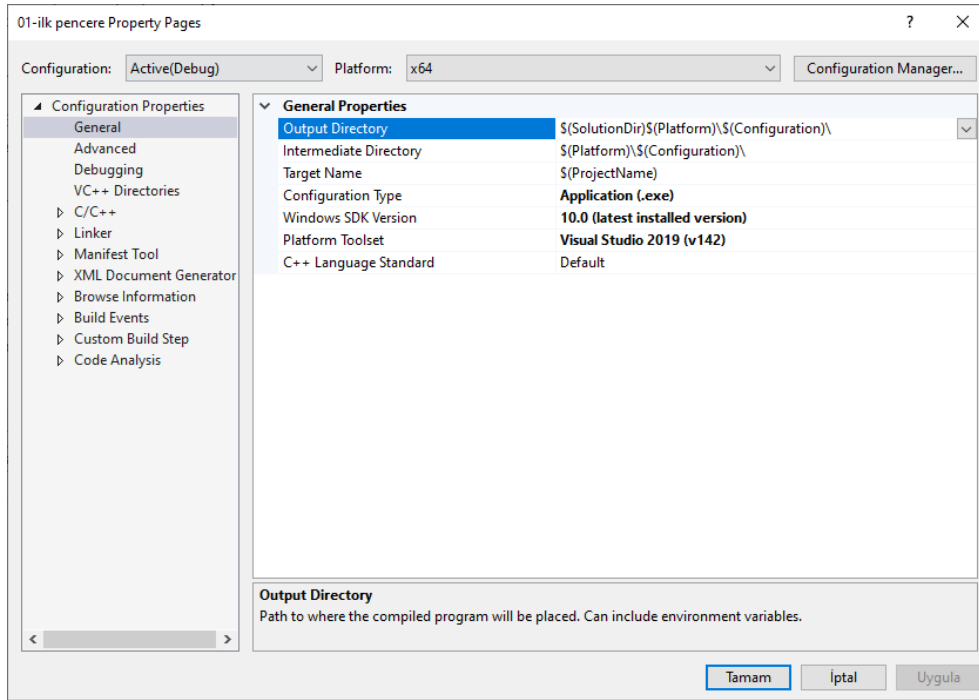
Kod 1.1



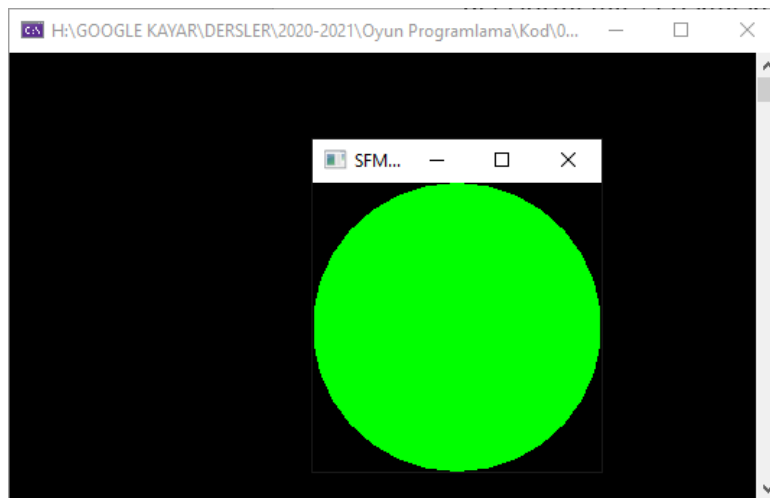
Şekil 1.23. DLL hatası

SFML kütüphanesinin .dll dosyaları Dependencies klasörü içerisindeki "bin" klasörü içerisinde bulunmaktadır. İşletim sisteminde herhangi bir değişiklik yapmadan bu problemin üstesinden

geleceğiz. Amacımız derleme sonucunda oluşan çalıştırılabilir dosyayı .dll dosyalarının bulunduğu klasör içerisinde oluşturmak ve burada çalıştırmak olacak. Bunun için yine “property pages” panelini açmamız gerekmektedir. Panel içerisinde General bölümünden Output Directory seçeneğini değiştirmemiz gerekmekte. Bu seçenek çalıştırılabilir dosyanın oluşturulacağı klasörü belirtmektedir. Buraya “\$(SolutionDir)..\Dependencies\bin\” değerini gireceğiz. Bu sayede çalıştırılabilir dosyamız .dll dosyaları ile aynı klasörde bulunacaktır. Değişiklik yapıldıktan sonra programı derleyip çalıştırdığımızda Şekil 1.25.’deki görüntüyü elde edeceğiz.



Şekil 1.24. Çalıştırılabilir dosyanın klasörü



Şekil 1.24. ilk Programın sonucu