T.C.
MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
ELEKTRİK ELEKTRONİK MÜH.

DÖNEM PROJESİ: 2 OYUNCULU SATRANÇ

Hazırlayan:
Osman Aslancan
170519024
1.Sınıf

Ders Adı:
Bilgisayar Programlama

Ders Öğretmeni:
Anıl Baş

2 OYUNCULU SATRANÇ

OSMAN ASLANCAN

T.C. MARMARA ÜNİVERSİTESİ, TEKNOLOJİ FAKÜLTESİ, ELEKTRİK ELEKTRONİK MÜH.

Bilgisayarda bir satranç oyunu yapabilmek için ilk olarak satrancın kurallarını bilmemiz gerekiyor. Bu kuralları öğrenmeye başladıkça oyunumuzun ne kadar karmaşık olacağını anlayabiliyoruz. Zaten benim bu oyunu yapmak istememdeki amaçta bu karmaşıklıklara güzel çözümler getirerek kendimi geliştirmek. Oyunumu dizaynlarken herhangi bir yerden nasıl yaparım diye bakmadım hepsi yavaş yavaş oyunu yaparak gelişti. Satrancı 4'üncu yapışım. Her yapışımda oyunumu daha da geliştiriyorum.

Amaç ve Hedefler

- Oyunumun ismi 2 oyunculu satranç fakat ben ileride oyuna yapay zekâ yapılmak istenirse buna destek vermek istiyorum. Bunun için oyunumda çeşitli haritalar yaptım bu haritalar sayesinde nerde taş olduğunu, kimin nereye hareket edebildiğini ve nereyi tehdit ettiğini görebiliyoruz.
- Oyunumu OOP (Object Orianted Programming) kullanarak yapmak istiyorum. Daha önceki oyunumu da böyle yapmaya çalışmıştım fakat tam başarılı olamamıştım.
- Oyunumun değişken isimlerinin İngilizce olmasını istiyorum böylelikle daha global bir koda sahip olacağım.
- Oyunumu yaparken Dev C++ yerine Visual Studio kullanacağım. Çünkü Visual Studio'nun sağladı çok daha fazla araç var. Yapmaya çalıştığım oyun çok karmaşık olduğundan bu araçlara ihtiyaç duyuyorum. Fakat ödev Dev C++'da yapılması istendiği için daha sonra oyunumu Dev C++'da derlenebilir hale getireceğim.
- Oyunumun platform layeri için yani çalıştığı işletim sistemiyle konuşmasını sağlamak için SDL (Simple Directmedia Player) kullanacağım. SDL kullanılarak birçok profesyonel oyun yapıldı. Örnek olarak Portal'ı verebilirim. Seçmemin başka bir nedeni ise multiplatforma destek vermesi.

Adım Adım Nasıl Çalışır?

Uygulama ilk olarak SDL kullandığım için SDL_main fonksiyonundan başlar. Daha sonra bir oyun değişkeni oluşturur. Oluşturduğu oyun objesinin İnit fonksiyonunu çağırır ve bu fonksiyonda SDL kütüphanesini başlatır, satranç tahtasını ve taşları oluşturur ve oluşturduğu taşları tahtaya dizer.

İnit fonksiyonu bittikten sonra oyun objesinin Handle_Events fonskiyonunu çağırır. Bu fonksiyon ilk olarak kullanıcıdan herhangi bir input var mı diye kontrol eder. İnput var ise inputa göre tepki verir. Daha sonra tahta ve taşları render eder. Uygulama oyun bitene kadar bu handle_Events fonksiyonunda döngü içindedir, input alır ve aldığı inputa cevap verir.

Programın nasıl çalıştığını basit bir şekilde anlattığımıza göre şimdi biraz daha derine inelim ve objeleri tek tek inceleyelim. Her şey oyun objesiyle başladığı için ilk olarak oyunla başlayalım.

Game (Oyun)

Game oyundaki tüm objeleri barındıran objedir. Tüm objelerin bir obje içinde olması fonksiyonlar oluştururken işimizi çok kolaylaştırır. Çünkü fonksiyonlara parametre olarak tek eklememiz gereken şey oyun objesi olur. Fakat game objesi çok bir büyük obje olduğundan her fonksiyona girdiğimizde kopyalamak programımızı çok yavaşlatır. Bu yüzden her zaman fonksiyonlara game objesinin pointerini parametre olarak ekliyoruz. Neden her şey oyun objesinde olduğunu anlattığımıza göre oyun nasıl başlar ona bakalım.

Game::Init() (Oyun Nasıl Başlar ?)

Oyun ilk olarak SDL ve SDL_image kütüphanelerini başlatarak başlar. Daha sonra bulunduğu işletim sisteminden window isteyerek bir window oluşturur ve bu windowda bir renderer oluşturur.

Window ve rendererimiz hazır olduğuna göre satranç tahtamızı ve taşlarımızı oluşturabiliriz.

Satranç Tahtası (struct Board)

Bu basit bir structır. Texture ve çizdirme fonksiyonundan oluşur. İnit edildiğinde textureleri diskten belleğe yükler. Satranç tahtasını çizerken de bulunduğu x ve y eksenlerinin toplamlarının tek veya çift olmasına göre farklı texturelar koyarak yapar.

Tools::AlignPieces (Taşları dizmek)

Taşları dizerken hangi rengin aşağıda olacağını anlamak için Game.h dosyasındaki ColorAtBottom makrosunu kullanırız. Bunu anladıktan sonra taşları olması gereken yerlere dizeriz ve Piece Controllerdeki dizimize ekleriz.

PieceController

Piece controller oyundaki taşları kontrol etmek istediğimizde kullandığımız objedir. Bu objede eğer taşlara tek tek ulaşmak istersek her taşın pointeri bulunan bir dizi bulunmaktadır ve eğer taşlara toplu bir işlem yapmak istersek bunun için fonksiyonlar bulunmaktadır.

Peki birbirinden farklı objeler nasıl tek bir dizi içinde tutulabilir? Bu sorunun cevabı için piece objesini incelemeliyiz.

Piece (Satranç Taşı)

Piece objesi oyundaki tüm taşlar için bir kalıp(template) görevi görür. Satrançtaki tüm taşlar birbirine benzerdir. Bu onları bir kalıp içine almamızı kolaylaştırır. Piece objesini kalıp olarak kullanmak için oluşturduğumuz taşları piece objesinden inherit[1] etmemiz gerekiyor. Taşların farklı olan fonksiyonları için ise virtual[2] fonksiyonlar kullanmamız gerekiyor.

Bu virtual fonksiyona örnek vermek için piyon taşına bakalım. Örneğin bir piyon taşı oluşturdum daha sonra piyon taşı piece objesinden inherit edildiği için ona bir piece gibi davrandım yani pieceye cast ettim. Eğer fonksiyonum virtual olmasaydı ben piyon taşının Calcmove fonksiyonunu çağırmak

[1] https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm
[2] https://www.geeksforgeeks.org/virtual-function-cpp/

istediğim zaman çalışan fonksiyon piece objesinde olan move fonksiyonu olacaktı. Fakat bu bir virtual fonksiyon olduğu için çalışan fonksiyon piyon taşının CalcMove fonksiyonu olacak.

Piece objesini incelediğimize göre şimdi haritalar nedir ne işe yarar ona bakalım.

PlaceMap, MoveMap, DangerMap (Konum, hareket, tehdit haritaları)

Bu haritalar oyunu takip etmemizi sağlar. Zaten adlarında da geçtiği gibi PlaceMap taşların konumlarını, MoveMap taşların nereye hareket edebildiğini ve DangerMap ise taşların nereyi tehdit ettiğini bize gösterir. MoveMap ve DangerMap 2 türdür; biri takım için diğeri ise tek bir taş için.

Eğer ileride bir yapay zekâ yapmak istersem bu haritalar sayesinde yapay zekâ oyunda ne olup bitiyor görebilecek. Peki bu haritalar nasıl hesaplanıyor onu inceleyelim.

PlaceMap, PieceController::CalcPiecePositions

CalcPiecePositions fonksiyonu oyundaki tüm taşların x ve y koordinatlarına bakıyor daha sonra Placemapta onların bulunduğu yerleri işaretliyor. Böylelikle nerede taş olduğunu görebiliyoruz.

MoveMap, CalcMove,Piece::Move, PieceController::CalcMoveMap

MoveMap

Her taşın nereye hareket edebileceğini hesaplamamız ileride yapay zekâ için şimdi ise şah matı hesaplamak ve kullanıcıya taşı nereye koyabileceği hakkında ipucu vermek için kullanılacak.İlk olarak bir taşa ait olan movemap nasıl hesaplanıyor ona bakalım.

CalcMove

CalcMove fonksiyonu bir virtual fonksiyondur. Çünkü her taşın hareketi türüne göre değişir. Bu fonksiyonlarda çok fazla sorgu olduğu için kod tekrarına çok giriliyor. Bu eski oyunumda büyük sıkıntı olmuştu çünkü küçük bir şeyi unuttuğunda bile yüzlerce satır kod değiştirmeniz gerekiyor. Ben bu sorundan kurtulmak için makrolar kullandım. Böylelikle eğer bir şey değiştirmek istersem tek bir makro değiştirerek bundan kurtuluyorum.

Makrolar kullanarak her taşın türüne göre sorgumuzu yapıyoruz eğer hareket geçerli ise movemapda orayı true yapıyoruz.

Sorguya örnek olarak kaleyi verelim. Kalenin üst tarafını incelerken bir while loop ile sürekli kalenin üst tarafına bakıyoruz. Eğer baktığımız yerde herhangi bir taş yoksa bir üst kareye bakarak devam ediyoruz. Eğer baktığımız yerde bir taş var ise ve bizim rengimizde değilse o taşın bulunduğu yeri true yapıyoruz ve loopdan çıkıyoruz. Çünkü karşı takımın taşının olduğu yere hareket edebilirsiniz. Eğer bizim rengimizde ise orayı true yapmadan çıkıyoruz. Çünkü kendi takımınızın taşının olduğu kareye hareket edemezsiniz. Ve son olarak eğer tahtanın sonuna gelirsek direk döngüden çıkıyoruz.

Piece::Move

Bu fonskiyon bir virtual fonksiyon değil çünkü her taş için aynı çalışıyor eğer gitmek istenilen yer Movemapta true ise gidebilir değil ise gidemez. Eğer taş gitmek istediği yere giderse oyun tüm haritaları yeniden hesaplıyor.

PieceController::CalcMoveMap

Bu fonksiyon oyundaki tüm taşların hareket haritasını hesaplıyor ve takımına göre 2 tane haritaya koyuyor. Bunu yaparken tek tek taşların CalcMove fonksiyonunu çağırıyor daha sonra takımının haritası ile fonksiyonu çağırdığı taşın haritasını OR ile birleştiriyor. Eşitleme yerine OR kullandım çünkü eğer eşitlersem fonksiyon bittiğinde elimde olan harita hesaplanan son taşın haritası olur.

DangerMap, Piece::CalcDanger,PieceController::CalcDangerMap

DangerMap

DangerMap her taşın nereyi tehdit ettiğini bize söyler. Oyunda Şah ve Şah mat hesaplamada kullanılıyor. Eğer oyunda şahın bulunduğu yer karşı takım tarafından tehdit ediliyorsa oyunda şah vardır anlamına geliyor.Şimdi taşlarda nasıl hesaplanıyor ona bakalım.

Piece::CalcDanger

Piece::CalcMove fonksiyonu bir virtual fonksiyondur. Gene burada da movemapda kullandığım gibi makrolar kullanıyorum. MoveMap hesaplama ile danger map hesaplama çok benzer. Örneğin kaledeki movemap ile dangermap hesaplamadaki tek fark movemap hesaplarken eğer önümüze çıkan taş bizim rengimizde ise orayı true yapmıyorduk bunda true yapıyoruz. Çünkü bizim rengimizde olan taşların bulunduğu yerleri de tehdit ederiz.

PieceController::CalcDangerMap

Bu fonksiyon ile CalcMoveMap fonksiyonu haritayı hesaplama konusunda aynı işi yapıyor. Fakat bu fonksiyonda harita hesaplandıktan sonra şahların bulundukları yerler karşı takım tarafından tehdit ediliyor mu diye kontrol ediliyor. Eğer tehdit ediliyorsa oyun **Şah mat** durumunu hesaplıyor. Bu biraz karışık olduğu için biraz daha ileride anlatacağım.

Özel Durumlar

Şah, Şah mat

Program oyunda bir şah olduğunu daha öncede dediğim gibi CalcDangerMap fonksiyonunda taşların tehdit ettiği yerlere bakarken anlıyor. Şah olduğunu anlayınca bunun bir şah mat olup olmadığını hesaplıyor. Bunun için MoveMap ve DangerMapı kullanıyor. İlk olarak şah hareket edebiliyor mu diye kontrol ediyor. Şahın hareketi hesaplanırken de DangerMap kullanılıyor. Eğer şahın gidebiliceği yer dangermapda true ise gidebileceği yer false yapılıyor.

Eğer şahın gidebileceği hiçbir yer yok ise Oyun kuklayı (Dummy) aktive ediyor. Şah yiyen takımın gidebileceği tüm yerleri bildiğimiz için gidebileceği tüm yerlere tek tek Dummy koyuyor ve CalcDangerMap fonksiyonunu çağırıyor. Eğer CalcDangerMap fonksiyonundan gene şah gelirse devam ediyor. Eğer kukla şah yiyen takımın gidebileceği tüm yerlere konduğu halde şah olmayan bir durum bulunamazsa bu oyun şah mattır deyip oyunu bitiriyor. Eğer denemelerden birinde şah olmayan bir durum bulursa oyun mat değildir deyip devam ettiriyor.

Yani oyun şah matı nasıl hesapladığı özetleyecek olursak ilk olarak şah hareket edebilir mi kontrol ediyor. Daha sonra şah yiyen takımın yapabileceği tüm hamleleri tek tek yapıyor ve şah olmayan bir durum bulmaya çalışıyor. Eğer bulamazsa oyun şah mattır. Bulursa oyun devam ediyordur.

Rok (Castling)

Rok satrançta özel bir hamledir. Özel tek bir durum olduğu için makro yapmak istemedim. Sorgusunu şaha koydum. Rok için ilk olarak şahın bonusu var mı kontrol ediyoruz yani eğer hareket ettiyse rok yapamaz. PlaceMap ve DangerMap ile örneğin kısa rokta şahın sağını kontrol ediyoruz. Daha sonra eğer sağdaki kalenin de bonusu var ise movemapta şahın 2 sağını işaretliyoruz ve özel olarak yaptığım castling objesini dolduruyoruz. Daha sonra özel olduğu için bu durum tek bir taş için virtual function yapmak istemedim Piece::move fonksiyonunda eğer hareket eden taşın tipi şah ise hareketin Castling objesindeki Pos değişkeniyle aynı olup olmadığına bakıyoruz. Eğer aynı ise roku gerçekleştiriyoruz.

Geçerken Alma (Capture While Passing (CWP))

Geçerken alma[3] piyonlara özel bir hareket. Daha fazla detay için linke bakabilirsiniz. Bunu anlamak için eğer bir önceki turda bir piyon 2 ilerlemişse ona geçerken alma yapabilirsiniz diyoruz ve o taşı işaretliyoruz. Piyon hareketini hesaplarken de piyonun sağına ve soluna bakıyoruz eğer sağında ve solunda herhangi bir taş var ve bu taş karşı takımın piyonu ise işareti var mı diye bakıyoruz. Eğer var ise özel structı dolduruyoruz. Piece::move fonksiyonunda ise hareket eden piyon ve özel structın pozisyonuna hareket etti ise arkasına hareket ettiği piyonu yok ediyoruz.

Piyon Dönüşümü (Promotion)

Piyonların MoveMapını hesaplarken eğer bir piyon rengine göre sınıra geldi ise piyonun Promotion fonksiyonunu çağırırız. Bu fonksiyon bize bir menü çizer ve bizden cevap bekler. Daha sonra piyonu bizim seçtiğimiz taşa çevirir.

Oyunu yapmamda yardımcı olan kanal ve siteler.

https://handmadehero.org/watch (Handmade_hero) Casey Muratori'den oyun yapma hakkında çok şey öğrendim.

https://www.youtube.com/user/TheChernoProject OOP'yi öğrenmemde yardımcı oldu.

https://wiki.libsdl.org/FrontPage

[3] https://en.wikipedia.org/wiki/En_passant