

**T.C**

**DOKUZ EYLÜL ÜNİVERSİTESİ**

**FEN FAKÜLTESİ**

**ENGİNAR OYUN MOTORU**

**Ahmet TÜMİŞ**

**Erhan GÜVEN**

**Nur ÇINAR**

**Denizhan KİLERCİOĞLU**

**Mayıs, 2022**

**İZMİR**

# ÖZET

Bu çalışmada genel olarak oyun geliştiricileri ve yazılımcılar tarafından oyun motoru geliştirmesine karşı oluşan önyargının kırılabilmesi hedeflenmiştir. Bu önyargılar arasında sistemlerin karmaşıklığının, altından kalkılamayacak bir süreç olmadığını göstermek adına sıfırdan oyun motoru yazılmıştır. Oyun motoru geliştirme sürecinin tecrübe edilmesi ve oyun geliştiricilerinin günlük olarak kullandıkları fonksiyonların çalışmasındaki incelikleri ve bu fonksiyonların yazımının ne kadar önemli olduğunun farkına varılması amaçlanmıştır.

**Anahtar Kelimeler:** Oyun Motoru, Enginar, Fizik Motoru, Rendering, Texture, Game Object, Game Management, OpenGL.

# ABSTRACT

In this study, it is aimed to break the prejudice against game engine development by game developers and software developers in general. Among these prejudices, a game engine was written from scratch to show that the complexity of the systems is not an insurmountable process. This project is aimed to experience the game engine development process and to realize the subtleties of the functions that game developers use daily and how important it is to write these functions.

**Keywords:** Game Engine, Enginar, Physics Engine, Rendering, Texture, Game Object, Game Management, OpenGL.

# **İÇİNDEKİLER**

[ÖZET 2](#_Toc103364583)

[ABSTRACT 3](#_Toc103364584)

[İÇİNDEKİLER 4](#_Toc103364585)

[ŞEKİLLER 5](#_Toc103364586)

[1 PROJE KONUSU 6](#_Toc103364587)

[2 GİRİŞ 6](#_Toc103364588)

[3 YÖNTEM VE ALGORİTMALAR 7](#_Toc103364589)

[3.1 Oyun Motorları 7](#_Toc103364590)

[3.2 Oyun Motorlarının Yapıtaşları 9](#_Toc103364591)

[*3.2.1* *Fizik Motoru* 9](#_Toc103364592)

[*3.2.2* *Rendering* 9](#_Toc103364593)

[*3.2.3* *Game Manager* 10](#_Toc103364594)

[*3.2.4* *Game Object* 10](#_Toc103364595)

[*3.2.5* *Data Manager* 11](#_Toc103364596)

[*3.2.6* *Memory Allocation* 11](#_Toc103364597)

[*3.2.7* *Sound Managment and Rendering* 11](#_Toc103364598)

[4 UYGULAMA 12](#_Toc103364599)

[4.1 Game Managment 13](#_Toc103364600)

[*4.1.1* *Transform* 13](#_Toc103364601)

[*4.1.2* *Game Object* 15](#_Toc103364602)

[*4.1.3* *Game* 15](#_Toc103364603)

[4.2 Rendering 17](#_Toc103364604)

[*4.2.1* *OpenGL* 17](#_Toc103364605)

[*4.2.2* *Texture* 17](#_Toc103364606)

[*4.2.3* *Window* 18](#_Toc103364607)

[*4.2.4* *Sprite* 19](#_Toc103364608)

[*4.2.5* *Texture Manager* 20](#_Toc103364609)

[4.3 I/O (Input / Output) 20](#_Toc103364610)

[*4.3.1* *Input Manager* 20](#_Toc103364611)

[*4.3.2* *Data Manager* 20](#_Toc103364612)

[4.4 Sound System 20](#_Toc103364613)

[4.5 Physics 20](#_Toc103364614)

[*4.5.1* *Collider* 20](#_Toc103364615)

[*4.5.2* *Collision* 21](#_Toc103364616)

[*4.5.3* *Rigid Body* 22](#_Toc103364617)

[4.6 Helpers 22](#_Toc103364618)

[*4.6.1* *EnginarMath* 22](#_Toc103364619)

[*4.6.2* *Matrix* 22](#_Toc103364620)

[*4.6.3* *Vector* 23](#_Toc103364621)

[*4.6.4* *Constants* 23](#_Toc103364622)

[5 SONUÇ ve DEĞERLENDİRME 24](#_Toc103364623)

[KAYNAKÇA 25](#_Toc103364624)

#

# ŞEKİLLER

[Şekil 1 Oyun Motorunun Yapısı 8](#_Toc103357547)

[Şekil 2 Motor Proje Yapısı 12](#_Toc103357548)

[Şekil 3 Transform işlemlerinin yapıldığı dosyadan bir ekran görüntüsü 14](#_Toc103357549)

[Şekil 4 Oyun loop’unun çalışma şekli 16](#_Toc103357550)

[Şekil 5 Örgü geometrisi üzerine atanmış bir texture 17](file:////Users/denizhankilercioglu/Downloads/abo.docx#_Toc103357551)

[Şekil 6 Texture yaratma diagramı 18](#_Toc103357552)

[Şekil 7 4:3 Çözünürlük oranına sahip bir Window 19](#_Toc103357553)

[Şekil 8 Örnek bir sprite animasyonu 19](file:////Users/denizhankilercioglu/Downloads/abo.docx#_Toc103357554)

[Şekil 9 AABB (Axis-Aligned Bounding Box) çarpışma 21](#_Toc103357555)

[Şekil 10 Circle-circle Çarpışma 22](#_Toc103357556)

# PROJE KONUSU

Bu çalışmada genel olarak oyun geliştiricileri ve yazılımcılar tarafından oyun motoru geliştirmesine karşı oluşan önyargının kırılabilmesi hedeflenmiştir. Bu önyargılar arasında sistemlerin karmaşıklığının, altından kalkılamayacak bir süreç olmadığını göstermek adına sıfırdan oyun motoru yazılmıştır.

# GİRİŞ

Oyunlar, canlıların hayatta kalabilme mücadelesi sürecinde karşılaşacakları unsurlar için tatbiki bir yeterlilik sınamasıdır. Antik çağlardaki hayvanlardan, günümüz insan toplumuna kadar oyun bütün canlıların hayatlarında yer almıştır.

Oyunların içgüdüsel olarak tetiklenmesi günümüzde oyunların eğlence aracı olarak kullanılmasının yanı sıra eğitim aracı olarak kullanılmasını da sağlamaktadır. Bu açıdan, ücret karşılığı yapılan bir işten veya çoğunlukla estetik veya ideolojik unsurların bir ifadesi olan sanattan farklıdır. Bu farklılıklara rağmen oyunlar bir iş veya sanatsal ifadeler olarak kullanılabilir.

Oyunların ortak noktaları, oynayan tarafların yeterliliklerini sergilemeleri sonucunda ulaştıkları tatmin hissidir, bu yeterlilik sonucunda zevk duyulur, saygınlık edinilir veya ödüllere sahip olunabilir.

Oyunların temel bileşenleri hedefler, kurallar, meydan okumacılık, etkileşim ve takım halinde oynanan bir oyun ise takım çalışmasıdır. Kaynağı ilkel olduğu için fiziksel ve zihinsel uyarım oyunlarda ön plandadır. Bu yüzden oyunlar günümüzde pratik becerilerin geliştirilmesinde, fiziksel ve zihinsel egzersiz yapılmasında, eğiticiliğin kullanılmasında, stimülatif veya psikolojik olarak önemli bir rol oynar.

# YÖNTEM VE ALGORİTMALAR

## Oyun Motorları

Bilgisayarların yaygınlaşması ve sadece bilimsel araçlar olarak kullanılmaktan normal kullanıcıların evlerine de girecek hale gelmesi, oyunların bilgisayarlar için yapılmasını hızlandırmıştır. Geçmişte bilgisayar teknolojilerinin gelişimine askeri araştırmalar büyük oranda katkıda bulunurken, ucuzlayan ve erişimi kolaylaşan bilgisayarların evlere girmesi oyunun sektörleşmesine ve hızla artan ilgi sektörde bir patlamaya neden olmuştur.

Bilgisayarların icadından itibaren işlemci komutları ile yazılan oyunlar, programlama dillerinin de gelişmesi ile programlama dilleri ile yazılmaya başlamıştır. Programlama dilleri ile kümülatif bilginin komponentleştirilebilmesi, oyun motorlarının doğumunu sağlamıştır.

Oyun motorlarının özünde bir bilgisayarın ana kartından farkı yoktur. Ana kartın bütün donanımları birleştiren, girdi alarak çıktılar üreten devre elemanları olduklarını düşünürsek, oyun motorları da bir oyunun bütün yapıtaşlarını içeren ve kullanıcıdan aldığı girdileri oyun aracılığı ile kullanıcıya çıktı olarak döken kodlar bütünüdür.

Tipik olarak bir oyun motorunun temel işlevleri;

* 2D veya 3D grafikler için bir işleme motoru,
* Fizik işlemleri için fizik motoru veya çarpışma algılama,
* Ses çıkışı için ses sistemi,
* Komutların girilebilmesi için script yönetimi,
* Objelerin hareketlendirilebilmesi animasyon desteği,
* Kullanıcı harici oyuncular için yapay zeka,
* Başka kullanıcılar ile oynamak için ağ oluşturma,
* Oyunun senaryosunu takip edebilmek için akış yönetimi,
* Sinematikler için bellek yönetimi,
* Performans için iş parçacığı oluşturma,
* Farklı diller ve bölgeler için yerelleştirme desteği,
* Sahne grafiği ve video desteği sunmaktır.

Oyun motoru geliştiricileri, genellikle aynı oyun motorunu farklı oyunlar üretmek veya oyunları birden fazla platforma taşımak için yeniden kullanarak oyun geliştirme sürecinde zamandan ve maliyetten tasarruf sağlar.

 Oyun motorları, yazılımcıların ihtiyaçları doğrultusunda değişkenlik göstermektedir. Buna örnek olarak çoklu oyuncuların olacağı bir oyunda ağ yapısının kullanılması, hikaye tabanlı bir oyunda sinematikleri kullanılması verilebilir. Ancak oyun motorları ihtiyaçlara yönelik olarak çeşitlense de hepsinin içerdiği ortak yapılar vardır. Bunlar fizik motoru, görüntü ve ses renderingi, oyun, girdi ve veri yöneticileri, oyun objeleri ve hafıza yönetimidir. Bu yapılar en temel işlev olan kullanıcı etkileşimini sağlamak için gereklidir. Görsel, işitsel ve dokunsal etkileşimin sağlıklı bir şekilde sağlanabilmesi için bu yapıların kusursuz bir uyum ile çalışması çok önemlidir. Bu proje kapsamında da bu yapıların oluşturulması ve ortaya oynanabilir bir oyunun çıkması hedeflenmiştir.



Şekil 1 Oyun Motorunun Yapısı

## Oyun Motorlarının Yapıtaşları

Enginar oyun altı adet yedi sistemin birleşiminden oluşmuştur. Bu alt sistemlerin uyum ile çalışması motorun göstereceği performansta önemli bir rol oynamaktadır.

### Fizik Motoru

Fizik motoru 3 yapıdan oluşur; pozisyon hesaplanma, çarpışma tespiti, çarpışma çözümleme.

 Pozisyon hesaplama, fiziğe ve girdiye bağlı olarak objenin pozisyonunun fiziksel Dünya’daki koordinatlarının güncellenmesiyle ilgilenir. Hız, açısal hız, konum, dönüş değeri gibi veriler burada hesaplanmaktadır, pozisyon hesaplama klasik mekanik dalındaki fonksiyonları kullanır.

 Çarpışma tespiti, objelerin birbirleriyle olan etkileşimlerini objelerin ve ortamın parametlerini kullnarak hesaplar. Bu hesaplamalarda objelerin şekli, ortamın yerçekimi gibi değişkenler göz önünde bulundurulur. Netwon fiziğine dayalı olarak yapılan işlemler çarpışma çözümlemeye yollanır.

 Çarpışma çözümleme, gerçekleşen çarpışmadan sonra yapılacak eylemi ifade eder. Momentum değerlerini, esnek olan ya da olmayan çarpışmaları, çarpışma sonrasında durma, sekme veya oluşabilecek farklı tepkilerin hesaplanmasında kullanılır.

### Rendering

Rendering, kullanıcının motor ile arasındaki etkileşimin en büyük parçalarından biridir. Bu etkileşim dahilinde, ekrana objeleri çizdirmek, iskelet yapılarını hareketlendirmek, modelleri içeri aktarmak, materyalleri ve yüzeyleri manipüle etmek ve render updatelerini çağırmak vardır.

Ekrana obje çizdirme, ekran kartı cihazına verilen talimatların ekrana piksel piksel çizdirilmesi ile oluşur. Objenin çizilebilecek hale gelmesi için her güncelleme çağrısından önce objenin konumunun, fiziklerinin, açılarının hesaplanması ve obje ilk çağırıldığında bellekten yüzey texturelerinin yüklenmesi gerekmektedir. Motorda en büyük kaynak (resource) yüküne neden olur.

İskelet yapılarını hareketlendirme, objelerin eklemlerinin birer fizik objesi olarak görülüp motor tarafından gerekli hesaplamalar ile simüle edilmesi ile oluşur. Bir insan figürünün hareket ettirilebilmesi için optimum düzeyde eklemin simüle edilip güncelleme çağrılarında ekran kartı adaptör cihazına yollanması gerekir.

Modelleri içeri aktarmak, hem dizayn sırasında hem de çalışma sürecinde, objelerin belleğe alınması ile yapılır. Bellek erişim hızı yüksek olduğu ancak maliyetli ve sınırlı bir alan olduğu için memory allocation yapılırken olabilecek en optimize şekilde objeler çağırılır.

Materyalleri ve yüzeyleri manipüle etmek, high level ve low level sistemlerin senkronize olarak çalışması ile gerçekleşir. Bellekten alınan texture ve materyal verilerinin 2 boyutlu uzaya yerleştirilerek vertexlerinin (köşelerinin) şekillerini değiştirmek, efektler uygulamak, ve rendering güncellemesinden önce çizilecek şeklin genel hatlarını hazır hale getirmek için kullanılır.

Render updateleri (güncellemeleri), renderingin en önemli ve final aşamasıdır, ekrana çizdirilmesi istenen final görüntü (kare) ekran kartına bu aşamada yollanır. Ekran kartı kullanılan ekranın yenileme hızına bağlı olarak güncellemeyi stabil tutmalıdır. Kare zamanlamasının stabil olmaması durumunda ekranda takılmalar ve yırtılmalar gibi istenmeyen durumlar oluşur. Bu yüzden renderlanması için yollanan her karenin tam zamanında yollanması ve diğer karelerin hesaplanıp yollanması ile arasındaki sürenin de düzenli olması gerekir. Performans için just in time rendering veya görüntü kalitesi icin frame queueing yapılabilir.

### Game Manager

Bu bölge objelerin engine loop ile arasındaki bağlantıyı kurarak iletişimini sağlar.

Objelerin her anlamdaki değişimi o frame içerisinde buffer’a eklenerek loop’a gönderilir ve orada hesaplamalar yapıldıktan sonra game manager’a geri gönderilir. Bu sayede değişim gerçekleşir. Kısaca oyun motorunun traceleridir.

### Game Object

Sahneye koyulabilecek her obje bir game object’tir. Component içerir ve bu component’lere göre değişir.

### Data Manager

Veriyi kaydetmek ve okumak için kullanılabilecek bir yardımcı kütüphanedir. Verilerin binary veya text olarak kaydedilmesini ve okunmasını içerecek fonksiyonlar bulunur ve motor ile yapılan oyunlar bu kütüphaneyi kullanarak save/load sistemi kurabilir.

### Memory Allocation

Bellek yönetimi, motorun yürüyeceği değişkenlik gösteren sistemlerde motorun performansını en çok etkileyecek kısım olduğu için kullanılan fonksiyonların bellek yönetimini dikkate alınarak kodlanması gerekmektedir.

C++’taki standart “new” ve “delete” işlemleri bir oyun motoru için yavaş kalacağından belleğe ekleme ve silme işlemlerinin çok daha hızlı olması için kendi bellek yönetimi sistemi kurulması planlanmaktadır. Stack allocator, pool allocator gibi memory allocating yöntemlerini daha doğru yerlerde kullanabilmek için genel bir sistem tasarlanacaktır.

### Sound Managment and Rendering

Bir oyun motorunun başarısı derinden etkileyen ancak çok fazla gündeme gelmeyen bir sistem de ses yönetimi ve renderingidir. Özünde ses dosyasının zamanı geldiğinde çalmaktan başka bir işlemi olmasa da, cihazların ses adaptörlerinin standartı, ekran adaptörlerinin standartları kadar oturmuş olmadığından yüksek çözünürlüklü ses dosyalarının çalınması için ciddi dosya ve kaynak işlemesi yapılması gerekmektedir.

Belleğe alınan ve gerektiğinde bellekten çıkarılan ses dosyaları fonksiyon çağrıları yapıldığı zaman çalınır. Ses dosyaları binary formattadır ve içindeki dalga sayısı ile uzunluğu, ne kadar süre çalınacağı, ne zaman biteceği sisteme söylenir. Sistem de işletim sistemi tarafından sunulan ilk ses adaptörüne çağrı yaparak sesin çalınmasını sağlar.

# UYGULAMA

Motorun yazımı için dil olarak C++ seçilmiş, API olarak ise SDL2 kullanılmıştır. SDL2 kullanılmasının ana nedeni OpenGL’in eksiklerini tamamlamak ve low level sistem initialization’larından kurtulmaktır. Bu sayede platform spesifik API’lerin kullanılması önlenmiş olup, daha hazır bir pakette sunulmuştur.

Motorun proje yapısı ise (şekil 2); assets, common, graphics, io, physics, soundü, x64, ext dependencies şeklindedir. Ayrıca ana dizinde SDL için gereken DLL dosyaları ve main.cpp dosyası bulunmaktadır.



Şekil 2 Motor Proje Yapısı

## Game Managment

### Transform

Bir koordinatın herhangi bir anlamı olması için, o koordinattan yola çıkarak bir referans noktası olması gerekir. Ekrana çizdirilen bir noktanın konumunu ölçmek için bir yöntem olmadığı sürece noktanın bir koordinatı yoktur.

Noktanın pencerenin sol tarafından ve üstten olan uzaklığı belirlendiğinde, bu iki ölçüm, sayfanın sol üst köşesinde bir referans noktası olduğu için ekrana relatif olarak noktanın koordinatını verir.

Dönüşümler (Transform), bu koordinat sistemleri arasında, matrisleri kullanarak kolay bir şekilde dönüştürmek için bir yol sağlar.

Bir oyuncu karakterinin köşelerini temsil eden dört nokta tanımlandığında bu noktalar, karaktere göre karakter anahatlarının köşelerini temsil eder. Bu köşelerin ekrana çizilirken çeşitli dönüşümler ile manipüle edilmesi karakterin pencerede hareket etmesini sağlar.

Enginar’ın kullandığı üç temel dönüşüm fonksiyonu vardır, bunlar; setPosition, setScale, setRotation’dır. Görevleri ise;

* setPosition, ekrana çizdirelecek objenin pozisyonunu belirlemek için kullanılır.
* setScale, ekrana çizdirelecek objenin boyutunun ayarlanmasında, aynı zamanda belirli bir eksende yönünün değiştirilmesini sağlar
* setRotation, ekrana çizilecek objenin döndürülmesini sağlar.



Şekil 3 Transform işlemlerinin yapıldığı dosyadan bir ekran görüntüsü

### Game Object

Oyun objesi bir grup bileşenin bir araya gelmesini sağlar. Bu bileşenler, objenin dönüşümü, oluşturucusu (constructor), fizik ve oyun mantığıdır.

Oyun objesinin mantığı gerçek Dünya’daki obje kavramına dayanmaktadır. Her objeninin fiziksel ve soyutsal olarak özellikleri bulunur.

Nesne tabanlı programlama yapısını kaynak alarak oluşturulan sınıflardan türeyen objeler kompleks değişkenlere benzer ve işaretçi görevi görür. İşaretçi görevi görmeleri oyun objesinin hafızada bir bütün olarak durmasını ve yazılan koddan ayrı bir abstraction’ı olmasını sağlar.

Bu bakımdan objelerin constructor’ları, fizik ve oyun mantığı gibi diğer oyun objelerine, dönüşümde depolanan değerlere erişim sağlayabilir.

Motorda obje içerisinde üç ana bileşen bulunur. Bunlar;

* addComponent, objenin türüne göre türe bağlı componentin oluşmasını sağlar,
* getComponent, belirtilen componentlerin id’lerini döndürür,
* update, sprite veya rigidbody componentlerinin güncellemesini döndürür.

### Game

Game, aygıtların aktifleştirildiği, objelerin oluşturulduğu ve en önemlisi oyun loop’unun bulunduğu yerdir.

Her oyun yürütülmeden önce OpenGL ve OpenAL’e bağlı aygıların aktifleştirilmesi gerekir. Bu aygıtların aktifleştirilmesi, ekran kartını kullanacak bir pencerenin oluşturulmasını, ses dosyalarının yürütülmesini ve gerekli bütün veriler için hafızada yer açılmasını sağlamaktır.

Aygıtlar aktifleştirildikten sonra oyun objeleri oluşturulur ve objelerin parametleri belirlenir, bu objeler spritelar, rigid body fizikleri, collider’lar olabilir.

Bütün cihazların aktifleştirilmesi ve ardından da objelerin oluşturulması ile game loop’a giriş yapılır. Game loop kullanıcı tarafından çıkış talimatı verilene kadar çalışan bir sonsuz döngüdür, oyuna yapılan girdiler, oyunun logic’i, zamanlamalar, güncellemeler, hafızada yapılacak işlemler ve en önemlisi rendering bu kısımda yapılır. Oyun “yürütülmüş” olur.



 Şekil 4 Oyun loop’unun çalışma şekli

## Rendering

### OpenGL

OpenGL, vektör grafiklerinin hesaplanması, manipüle edilmesi ve yaratılması işlemlerinin hızlı bir şekilde gerçekleşmesi için grafik işlem biriminin (GPU) kullanılmasını kolaylaştıran bir uygulama programlama arayüzüdür (API).

 Projede kullanılan SDL2, bazı platformlarda donanım render işlemleri için OpenGL kullanır.

### Texture

Texture, örgü geometrisi yüzeyine atanmış bir bitmap görselidir. Son kullanıcının herhangi bir objeyi algılaması ve tanımlandırabilmesi için kullanılır.

Şekil 5 Örgü geometrisi üzerine atanmış bir texture

Oyun motorunda kullanılan texture oluşturma yöntemi, yaratılacak örgü geometrisine texture olması beklenen görselin dizininin sağlanması ile başlar. Render işlemlerinin gerçekleştiği obje yardımıyla texture oluşturulur. Oluşan texture Window’a atanarak son kullanıcının algılayabildiği bir texture oluşur.



Şekil 6 Texture yaratma diagramı

### Window

Window, oyun motoru tarafından derlenmiş bir oyunu son kullanıcıya eriştiren bir arayüzdür. Window’un oluşturulması, işletim sistemine ait pencerenin yaratılması ve kullanılması ile gerçekleşir. Oluşturulan window ile oyun motoru arasındaki bağlantının kurulabilmesi için SDL2 ile Renderer oluşturulur. Bu bağlantının sağlanmasıyla birlikte oyun motorundan derlenen oyun, son kullanıcı tarafına eriştirilmiş olur.

 Window’un kapatılabilmesi, tam ekran yapılabilmesi, saklanabilmesi ve boyutunun değiştirebilmesi gibi yöntemlere sahiptir.

 Oyun motorunu kullanılması öngörülen oyun geliştiricileri, window’a ait çözünürlük ve oyunun başlığı gibi elementleri window objesini yaratırken belirleyebilir.



Şekil 7 4:3 Çözünürlük oranına sahip bir Window

### Sprite

Sprite, oyun objelerinde var olan texture dizesinin ve animasyon için gerekli önceden tanımlanmış değişkenlerin mevcut olduğu bir objedir.

 Bir oyun objesinin animasyonunun gerçekleşebilmesi için, sprite içerisinde var olan dizedeki texture’ları ardı ardına, sıralı bir şekilde ve belirli periyotlarda geçiş yapılması gereklidir. Animasyon süresince texture’lar arasındaki geçiş aralıkları ve texture’ların sıralandırılması gibi işlemler sprite objesinde belirlenmektedir.



Şekil 8 Örnek bir sprite animasyonu

### Texture Manager

Texture Manager, texture’ların belirli bir standartta entegre edilebilmesi için yaratılmış kurallar ve yöntemler bütünüdür. İçerisinde, yaratılması öngörülen bir texture için gerekli olan Window objesinin referansı mevcuttur.

## I/O (Input / Output)

### Input Manager

Input manager, xinput destekli SDL keyboard eventlerini üstlenen bir class’tır. Klavyede eventleri sürekli dinlenir, girdinin hangi scancode’a ait olduğu belirlenir ve oyun kısmında o tuşun karşılığında yapılacak güncellemeyi yapar. xinput’un sahip olduğu yaklaşık 250 tane girdi, karşılığı olan SDL\_SCANCODE formatına dönüştürülür.

### Data Manager

Oyun assetlerinin kullanımı için yazılmış bir class’tır. Assetleri ihtiyaca yönelik olarak hafızaya alır veya çıkarır. Yapı olarak ise /assets/ dizini altında /graphics/ ve /sounds/ klasörlerindeki dosyalara ulaşır.

## Sound System

Motorda ses işlemek için SDL API’nin sunduğu fonksiyonlardan yararlanılmıştır. Ses verilerini miksleyen ve ses akışına yerleştiren bir callback fonksiyonu kullanılır. Bundan sonra, istenen format ve çözünürlükteki ses dosyası SDL\_LoadWAV() fonksiyonu kullanılarak hafızaya alınır ve sesin oynatılacağı ses cihazı başlatılır.

İşlemler bitince SDL\_PauseAudio(0)'u çağırılır ve bu andan itibaren callback fonskiyonlar ile ses dosyaları yürütülebilir. Ses çıkışı kullanıldıktan sonra SDL\_CloseAudio() fonksiyonu ile çıkış kapatılır.

## Physics

### Collider

2D Oyunlarda genel olarak 3 adet collider türü kullanılmaktadır; Circle, Box ve Edge. Oyun motorumuzda Circle ve Box collider’ları bulunmaktadır. Collider’lar fizik hesaplama sırasında bir objenin fiziksel alanını belirleyen geometrik alanları formülize eder. Collider Component’ı base class’tır. BoxCollider ve CircleCollider bu class’tan inherit edilir. Collider bir component olduğundan bir GameObject’e eklenip çıkarılabilir.

Collider (Base Class): Her collider türünün ortak olarak tag’i, tipi ve geometrik şeklini güncellediği bir fonksiyonu vardır. Dolayısıyla bu özellikler ve fonksiyon Collider base class’ının içerisinde yer almaktadır.

updateShape(Texture\* texture): Bu metod collider’ın konum ve şeklinin transformation’a göre güncellenmesini sağlar.

BoxCollider: 2 boyutlu bir dörtgen collider’dır. Oluştururken orijin noktası (x,y), genişliği, yüksekliği ve base class’ın ihtiyaç duyduğu tag’i belirtmek gerekir.

CircleCollider: 2 boyutlu bir çember collider’dır. Oluştururken orijin noktası(x,y) ve yarıçapının verilmesi yeterlidir.

### Collision

Collision oyun motorlarında objelerin çarpışmasının tespiti ve bunun üzerine tepki vermesidir. Normalde oyun motorları yazılırken çarpışma için kütüphane kullanılır fakat biz basit ve optimize bir çarpışma mekanizması istediğimizden herhangi bir kütüphane kullanmaksızın sıfırdan yazdık. Farklı Collider’ların çarpışma tespiti için farklı yöntemler gerektiğinden, her Collider kombinasyonu için farklı kontrol metodu kullanıldı. AABB çarpışması, iki box collider’ın çarpışma işlemini kontrol eder.



Şekil 9 AABB (Axis-Aligned Bounding Box) çarpışma

İki circle collider’ın çarpışma işlemini kontrol eder. A ve B iki circle collider’a sahip obje olsun. A ve B arasındaki mesafe yarıçapları toplamından küçük ise iki obje çarpışıyor anlamına gelir.



Şekil 10 Circle-circle Çarpışma

### Rigid Body

Oyun motorlarında rigid body bir objenin fiziksel dünyayla etkileşime girebilecek biçimde özelliklere sahip olmasını ve fiziksel özelliklere (hız, ivme, yerçekiminden etkilenme, kuvvet vb) sahip olmasını sağlar. Rigidbody de Collider gibi bir component’tır ve bir gameobject’e eklenip çıkarılabilir. Rigidbody oluşturulurken bir kütleye ihtiyaç duyar. Yerçekimi ivmesi default olarak 9.81 m/s2’dir fakat isteğe göre oluştururken bu da ayarlanabilir. Gameobject güncellenirken her seferinde Rigidbody component’ını da (varsa) günceller. Rigidbody güncellendiği sırasında ivmesini ve hızını değiştirir. Bunları yaparken sürtünme, yerçekimi ve o anki kuvveti dikkate alarak günceller.

## Helpers

Oyun motorlarında temel işlevlerin yerine getirilmesi amacıyla helper script'ler yazılmaktadır. Bu motorda belirli matematiksel işlerin kolaylaştırılması ve daha optimize olması adına matematik kütüphanesi, matrix ve vector class'ları ve definitionlar bulunmaktadır.

### EnginarMath

Bu script matematik kütüphanesidir. İçerisinde belirli sayıların definitionları (örnek olarak e sayısı, pi sayısı, kök2) ve fastInverseSqrt gibi hızlı karekök alma metodları vardır.

Bu sayıların definitionlar’ının bulunma nedeni floating point işlemleri her çağırılmada hesaplanmasını önleyip hız artışı sağlamaktır.

### Matrix

Matrix kütüphanesi genel olarak iki boyutlu dizilerin matris şeklinde gösterilmesini sağlar. İçerisindeki Matrix operatörleri sayesinde matrix işlemleri yapılır. Dönüşüm (transform) için kullanılacak cebirsel işlemler burada yapılır.

### Vector

Vector, Matrix'ten inherit edilen 2'ye 1 veya 1'e 2'lik bir matristir. İçerisinde matris'e ek olarak kendi operatörleri vardır ve bunlar sayesinde oyun motorundaki çoğu işlem gerçekleştirilir. Örneğin Position, scale gibi 2 değerli verilen vector olarak saklanabilir ve bunların üzerinde gerkeçleştirilebilecek dot product, cross product gibi eylemler vector içerisinde saklanabilir.

### Constants

Bu scriptte motor genelinde kullanılacak sabitler yer alır. Bu sabitler arasında yerçekimi sabiti veya açılacak pencerenin boyutu yer alır.

# SONUÇ VE DEĞERLENDİRME

Oyun motoru üzerinde 2D ve fizik temalı birçok oyun yapılabilir. Aynı zamanda seviye içerikli oyunları da destekleyerek sürekli ve rekabetçi bir ortam sunar.

Bizim seçtiğimiz oyun ise Tetris’tir. Bu oyun için kontrollü serbest düşme, obje şekil değişikliği gibi fiziksel olayları gerekmektedir. Enginar bu tüm özellikleri sağlar. Kontrollü serbest düşme için yer çekimi ivmesinin değişikliği dahil diğer çekim yasaları formülleri çok kolay farklılaştırabilir.

Sonuç olarak yapılan oyun motoru diğer oyun motorlarına göre daha basit olmuştur. Oyun dünyasına yeni katılan veya çok kompleks olmayan oyun yapmak isteyen kullanıcılar için Enginar sade ve yeterli bir tercih olabilir. Çok fazla özellik olmaması sayesinde zaman kaybı en aza indirgenmiş olmuştur. Bunun yanında gerekli özelliklerin de eklenmesiyle istenen esneklik de sağlanmıştır.

Kısaca bir oyun motorunun çalışması için gerekli olan bütün işlevler yazılmış, ihtiyaca yönelik olarak optimize edilmiştir.

# KAYNAKÇA

WEB\_1. (2022). General Idea of Game Engine Structures, <https://github.com/love2d/love>

WEB\_2. (2018) General Idea of How OpenGL Works, <https://www.khronos.org/opengl/wiki/>

WEB\_3. (2022). Offical Wiki of SDL2. <https://wiki.libsdl.org/>