SVG KAT PLANLARINDAN OYUN ORTAMI OLUŞTURULMASI

CREATING GAME ENVIRONMENT FROM SVG FLOOR PLANS

MUSTAFA AKAY

DOÇ. DR. BURKAY GENÇ

Danışman

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin

Bilgisayar Grafiği Anabilim Dalı için Öngördüğü

YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

Ağustos 2022

MUSTAFA AKAY tarafından yazılan **"Svg Kat Planlarından Oyun Ortamı Oluşturulması "** adlı bu çalışma aşağıdaki jüri tarafından BİLGİSAYAR GRAFİĞİ ANABİLİM DALI' nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Dr. Fatih Sağlam
Başkan . . . . . .

Doç. Dr. Burkay Genç
Danışman . . . . . .

Dr. Ufuk Çelikcan
Jüri . . . . . .

Bu tez Hacettepe Üniversitesi Bilişim Enstitüsü tarafından YÜKSEK LİSANS TEZİ olarak onaylanmıştır.

Prof. Dr. Arif Altun
Bilişim Enstitüsü Müdürü

ETİK

Hacettepe Üniversitesi Bilişim Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tüm bilgi ve belgeler akademik kurallara uygun şekilde elde edildiğini,
- tüm görsel-işitsel, yazılı bilgi ve sonuçlar bilimsel etik kurallarına uygun olarak sunulduğunu,
- başka eserlerin kullanılması durumunda bilimsel standartlara uygun olarak ilgili çalışmalara atıf yapıldığını,
- alıntılanan tüm çalışmalara tam olarak atıfta bulunulduğunu,
- Tezle ilgili verilerde oynama yapılmadığını,
- ve bu tezin herhangi bir bölümü, herhangi bir üniversitede başka bir tez çalışması olarak sunulmadığını

beyan ederim.

... / ... / .....

MUSTAFA AKAY

ETHICS

In this thesis study, prepared in accordance with the spelling rules of Institute of Graduate Studies in Science of Hacettepe University,

I declare that

- all the information and documents have been obtained in the base of the academic rules.
- all audio-visual and written information and results have been presented according to the rules of scientific ethics
- in case of using others works, related studies have been cited in accordance with the scientific standards
- all cited studies have been fully referenced
- I did not do any distortion in the data set
- and any part of this thesis has not been presented as another thesis study at this or any other university.

... / ... / .....

MUSTAFA AKAY

YAYINLAMA FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan **"Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge"** kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H. Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

- ☐ Enstitü yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.
- ☐ Enstitü yönetim kurulu gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren .... ay ertelenmiştir.
- ☐ Tezim ile ilgili gizlilik kararı verilmiştir.

... / ... / .....

MUSTAFA AKAY

ABSTRACT

CREATING GAME ENVIRONMENT FROM SVG PLANS

MUSTAFA AKAY
Master Of Science, Computer Animation and Game Technologies
Supervisor: Assoc. Prof. Dr. BURKAY GENÇ
August 2022, 81 pages

Advances in computer technologies have paved the way for developments in games. Graphics and game environment are important game components. There are various studies to create a game environment. In this thesis, a method that aims to create a three-dimensional environment by taking two-dimensional drawings as input is proposed to create a game environment.

SVG is a suitable format for drawing a floor plan. In this thesis, SVG format floor plans data are formatted. Polygons and line segments are obtained from the formatted data. These geometric shapes will be classified as interior wall, corridor wall and exterior wall and transferred to the three-dimensional environment and different textures, items and models can be added to them. As a result, an environment that can act as a first person is obtained in the enriched three-dimensional game environment created in Unity. This three-dimensional environment can be used for games and simulations.

A rich three-dimensional game environment has been achieved with the item placement algorithm and textures. The methods proposed in this thesis can be a great resource for game developers and those who want to simulate build plans.

Keywords: Svg File Analysis, Raycast Analysis, Unity 3D, Game Development

ÖZET

SVG KAT PLANLARINDAN OYUN ORTAMI OLUŞTURULMASI

MUSTAFA AKAY
Yüksek Lisans, Bilgisayar Animasyonu ve Oyun Teknolojileri Bölümü
Danışman: Doç. Dr. BURKAY GENÇ
Ağustos 2022, 81 sayfa

Bilgisayar teknolojilerindeki gelişmeler oyunlardaki gelişmelerin önünü açmıştır. Grafikler ve oyun ortamı, önemli oyun bileşenlerindendir. Oyun ortamı oluşturmak için çeşitli çalışmalar vardır. Bu tezde oyun ortamı oluşturmak için iki boyutlu çizimleri girdi olarak alarak üç boyutlu ortam oluşturmayı amaçlayan bir metot önerilmiştir.

SVG kat planı çizimi için uygun bir formattır. Bu tezde, SVG formatındaki yapı planları verileri biçimlendirilmektedir. Biçimlendirilen verilerden poligonlar ve doğru parçaları elde edilmektedir. Bu geometrik şekiller iç duvar, koridor duvarı ve dış duvar olarak sınıflandırılarak üç boyutlu ortama aktarılarak bunlara farklı dokular, eşyalar ve modeller eklenebilmektedir. Sonuçta Unity'de oluşturulan zenginleştirilmiş üç boyutlu oyun ortamında birinci kişi(first person) olarak hareket edilebilen bir ortam elde edilmektedir. Bu üç boyutlu ortam, oyunlar ve simülasyonlar için kullanılabilecektir.

Eşya yerleştirme algoritması ve dokular ile üç boyutlu olarak zengin bir oyun ortamı elde edilmiştir. Bu tezde önerilen metotlar oyun geliştiricileri için ve yapı planlarını simüle etmek isteyenler için harika bir kaynak olabilir.

Keywords: Svg Dosya Analizi, Raycast Ortam Analizi, Unity 3D, Oyun Geliştirme

TEŞEKKÜR

Tez konusunun belirlenmesinden sonuçlanmasına kadar tüm çalışmalarımda değerli yardımlarıyla tez çalışmasında yol gösteren Sayın Doç. Dr. Burkay Genç' e ve bana her yönden destek olan sevgili annem Selma Akay' a sonsuz teşekkür ederim.

İçindekiler

Sayfa

TABLOLAR

Sayfa

ŞEKİLLER

KISALTMALAR

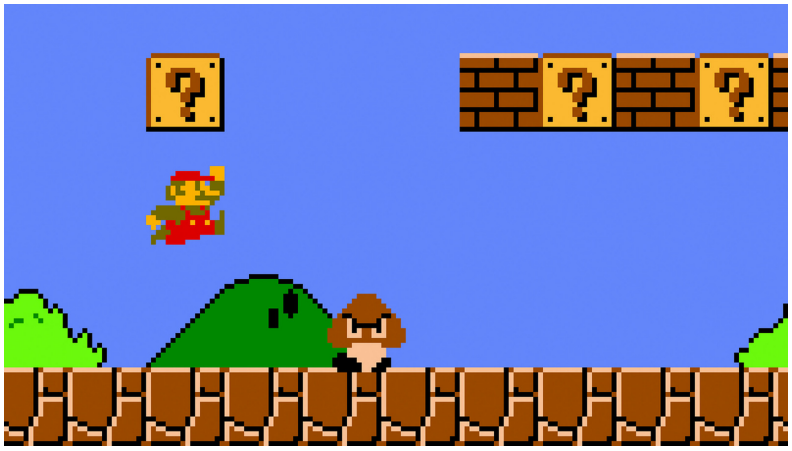
SMIL	:	**S**ynchronized **M**ultimedia **I**ntegration **L**anguage
RDF	:	**R**esource **D**escription **F**ormat
X3D	:	**E**xtensible **3 D**imension
BIM	:	**B**uilding **I**nformation **S**ystem
SVG	:	**S**calable **V**ector **G**raphics
XML	:	**E**xtensible **M**arkup **L**anguage
HTML	:	**H**yper **T**ext **M**arkup **L**anguage

1. Giriş

1.1. Problemin Tanımı

Bilgisayar grafiği, grafik yazılımları ve donanımlarıyla oluşturulabilen tüm görselleri ifade eden bir terimdir. Bilgisayar grafiği temelde iki kategoriden oluşmaktadır. Bunlar iki boyutlu ve üç boyutlu grafiklerdir[5]. Bilgisayar oyunları bilgisayar grafiğinin en önemli kollarından biridir. İnsan sosyal hayatında önemli bir yer tutmuş oyunların bilgisayar ortamına girmesi de fazla zaman almamıştır. Kullanıcı arayüzü yazı olan oyunlardan gerçekçi görüntülerin olduğu oyunlara bilgisayar donanım teknolojilerindeki ilerlemelerle geçilmiştir[6]. Bilgisayar grafiklerindeki gelişmeler ile oyunlarda zenginleştirilimiş grafikler kullanılmaya başlanmış ve bu bilgisayar oyun grafiklerinin doğmasına neden olmuştur.

İki boyutlu olarak çıkan bilgisayar oyunları zamanla üç boyutlu olarak geliştirilmeye başlanmıştır. İki boyutlu oyunlar ile üç boyutlu oyunlar arasında temel farklar vardır. Bunlar birkaç cümlede özetlenebilir. İki boyutlu oyunların yapımı kolaydır, basit kurallar ve hedefler vardır. İki boyutlu oyunlarda mekanikler basittir ve hareket bir doğrultuda meydana gelir. İki boyutlu bir oyun bireysel olarak tasarlanabilirken üç boyutlu oyunlar genelde ekipler tarafından oluşturulur. Günümüzde, bu kategoriye ilgi az olmasına rağmen halen iki boyutlu grafiklerde de oyunlar geliştirilmektedir.[7] Üç boyutlu oyunlar daha karmaşık yapısı, görevleri ve grafikleriyle oyunculara saatler harcatabilmektedir. Üç boyutlu oyunlar oynanış ve grafiklerle ilgili birçok seçeneğe sahiptir. Üç boyutlu oyunlar, karmaşıklık ve derinlikten dolayı iki boyutlu oyunlardan daha fazla tercih edilse de iki boyutlu oyunlar hala birçok insan tarafından oynanmaktadır[8]. İlk olarak iki boyutlu olarak piyasaya sürülen klasik Mario oyunu ve üç boyutlu olarak yapılan Mario oyunu Şekil 1.1 ve Şekil 1.2'de gösterilmiştir.



Şekil 1.1 Klasik iki boyutlu Mario oyunu[1]

Şekil 1.2 Üç boyutlu Mario oyunu[2]

Üç boyutlu oyunlar birbirlerini tamamlayan birçok parçadan oluşmaktadır. Bunlar; oynanış tasarımı, modelleme, animasyon, kodlama ve test aşaması olarak özetlenebilir. Doksanlı yıllardan sonra gerek grafiklerin karmaşıklığının artması gerek kullanıcı-oyun etkileşimindeki gelişmeler, oynanıştaki ayrıntıların hızlı bir şekilde gelişmesine neden olmuştur. Bunun yarattığı etki, gerçekçi tecrübe ve sürükleyicilik üç boyutlu oyunların özellikle çocuklarda ve gençlerde popüler olmasına neden olmuştur.

Son yıllarda bilgisayar ve mobil teknolojilerinin gelişmesi ile oyunlara ulaşım kolaylaşmıştır. Böylece gerek oyun arzında gerek oyun oynama sürelerinde artış gerçekleşmiştir. Oyuna ulaşımdaki kolaylık rekabetin artmasına yol açmıştır. Oyunlardaki talep artışı oyun stüdyolarını daha hızlı ve etkili grafikler çıkarmaya yöneltmiştir. Bu, daha nitelikli oyun geliştirme yazılımları ve araçlarının geliştirilmesine önayak olmuştur. Yeni geliştirilen bu yazılımlar stüdyolar ve bağımsız oyun geliştiricilere daha hızlı ve kolay oyun geliştirmeyi amaçlamaktadır.

Her ne kadar popüler oyun geliştirme araçları oyun geliştiricilere birçok imkan ve kolaylık sunsa da oyun ortamı geliştirme halen geliştiriciler için en çok uğraş verilen bölümlerden biridir. Oyun ortamı geliştirme içerisinde eşyalar, yapılar, ışık, karakterler ve bunların dokularını tasarlama gerektiren çok karmaşık ve zaman alan bir süreçtir

Üç boyutlu oyun ortamı oluşturmayı kolaylaştırmak için çeşitli güncel fikirler vardır. Yapay zeka ile oyun ortamı oluşturmak bunlardan en popüler olanıdır. Burada daha önce oluşturulmuş oyun ortamlarının makine öğrenmesi ile eğitilmesi ile model oluşturulur[9]. Bu model bir oyun ortamı önerecektir. Bir diğer metot ise iki boyutlu çizimler sonucu oluşturulan planı üç boyutlu oyun ortamına dönüştürmektir.

İki boyutlu çizilebilir, düzenlenebilir bir ortam ile oyun ortamının üç boyutlu şekilde otomatik oluşturulması bu süreci kısaltmaktadır. Bundan dolayı oyun geliştiricisinin geliştirmek istediği yapıyı iki boyutlu çizerek oluşturması fikri ortaya çıkmıştır. Bu tezde iki boyutlu planların üç boyutlu oyun ortamına çevrilmesi problemine çözüm bulmak amaçlanmıştır.

1.2. Tezin Hedefi

Oyun geliştirmede oyun ortamı oluşturma temel aşamalardan biridir. Oyun geliştirme safhalarından olan üç boyutlu mekan tasarımları fazlasıyla zaman alır. Bireysel oyun geliştiricilerinde sadece oyun kurgusu yapmayı isteyenler için mekan tasarımı bir problemdir. Oyun geliştirme bir ekip işi olması, ortam tasarımcısı rolünün görev ve sorumluluğun fazla olması pratik oyun ortamı gelişimi arayışlarında sorun teşkil etmektedir. Bu tezde oyun geliştiricilerinin daha az emek harcayarak belirli konseptlerde oyun ortamı oluşturmaları için bir metot önerilmiştir. Oyun geliştiricisi oyun ortamını çizimlerle yapacağı için oyun geliştirme suresi uzun olmayacaktır. İki boyutlu çizilebilir ve kod olarak düzenlenebilir formatların başında SVG formatı gelmektedir. Bu tezin geliştirme aşamalarından ilki SVG formatında çizilen veya kod ile oluşturulan doğrusal planlarını Unity tarafından okunabilecek şekilde doğru parçası ve poligon noktalarına dönüştürülmesi için metot geliştirilmesidir.

Bu tezin geliştirme aşamalarından ikincisi ise algoritma ile ayrıştırma sonucu doğru parçası noktalarına ayrılmış SVG planına ortam analizi uygulamaktır. Doğru parçası noktaları olarak gelen veriler sadece iki boyutlu çizgiler ifade etmektedir. Bu bilgilerin zengin bir oyun ortamı oluşturabilmesi için analiz edilerek yorumlanmasına ihtiyaç vardır. Bu tezde iki boyutlu planlardaki poligon ve çizgileri; koridor duvarı, iç duvar ve dış duvar olarak üç farklı kategoride sınıflandıran bir algoritma geliştirilmesi amaçlanmıştır. Sınıflandırma sayesinde farklı kategorideki duvarlar yapısına uygun şekilde doku giydirilecek ve her birinin üç boyutlu şekli farklı olacaktır.

Bu tezin geliştirme aşamalarından üçüncüsü; elde edilen veriler kullanılarak üç boyutlu zenginleştirilmiş bir oyun ortamı oluşturmaktır. Burada alışveriş merkezi, ev ve şato konseptleri ile üç boyutlu oyun ortam oluşturulabilmektedir. Daha önce elde edilen ortak duvarlar, dış duvar ve koridor duvarı verilerine göre her konseptteki duvarların dokularının, yüksekliklerinin ve renklerinin farklı olması amaçlanmıştır. Konseptlerde oluşturulan duvar dokuları kullanıcının isteğine göre değişikliğe açıktır.

1.3. Çalışma Planı

ARAŞTIRMA:

- Çizilebilir kod formatlarının incelenmesi
- SVG'nin iki boyutlu çizim araçları ve kod formatlarının araştırılması
- SVG'nin hangi yazılımlar kullanarak ayrıştırılabileceğinin araştırılması
- Üç boyutlu canlandırma için yazılım araştırması
- Önceki çalışmalarda kullanılan yöntemler incelenerek en iyi yöntemin bulunması
- Daha önce yapılmış oyun ortamlarının incelenmesi
- Yöntemlerin performans analizi ve belirlenen yazılımlarla uyumluluğu
- Raycast ile ilgili yazılım araştırmaları

GELİŞTİRME:

- SVG'nin ayrıştırlacağı Python kütüphanelerinin indirilip çalışma ortamı oluşturulması
- Ortam analiz, duvar sınıflandırma algoritmalarının oluşturulması ve kodlarının yazımı
- SVG'den ortam analizine, ortam analizinden üç boyutlu oyun canlandırmaya veri aktarımının oluşturulması
- Belirlenen konseptlerin tasarımlarının yapılması
- Unity'de konseptlerde görsellerin kodlarla oluşturulması
- Konseptler arası geçişlerin ayarlanması
- Oluşturulan farklı planların Unity'de üç boyutlu ortamda testi

YORUMLAMA:

- Yazılımların planlamalara göre çalışıp çalışmadığının kontrolü
- Elde edilen sonuçların yorumlanması

1.4. Tez Çalışmasının Uygulama Alanı

Yapılarda kullanılan iki boyutlu planlar üç boyutlu ortama aktarılması, gerçek ortamın sanallaştırılması ve sanallaştırılan ortamların kullanıcıların deneyimine sunulması sağlanacaktır. SVG formatından yeniden elde edilmiş veriler Unity'de üç boyutlu olarak gösterilecektir. SVG planları; ev, alışveriş merkezi, şato gibi konseptler olarak üç boyutlu ortama aktarılabilmektedir. Aktarılan yapının duvarları yapının özelliklerine göre dokulardan oluşturulmaktadır. Duvarlarının renkleri kullanıcı tarafından ayarlanabilir olacaktır. Yapının cinsine göre zemin için de doku oluşturulmaktadır. Bu uygulamada kullanıcı birinci kişi kamerasıyla üç boyutlu ortamda gezebilmektedir. Kullanıcı üç boyutlu platform üzerinde kamera ile hareket edebilecektir. Kullanıcı platform üzerinde Mouse ile istediği yere bakarak ortamı test edebilmektedir. Bu arayüz Unity programında C# ile yazılmıştır. SVG analiz edilip elde edilen veriler Unity'ye gönderilmeden önce kullanıcının isteğine göre şekillendirilmektedir. Oyun tasarımcısı istediği bir planı iki boyutlu çizerek üç boyutlu ortamını tecrübe edebilmektedir. Oyunlarda bölüm tasarımı, emek ve zaman isteyen ciddi bir iştir. Duvarlar, zemin, eşyalar, çevredeki nesneler vb. birçok konuda zaman harcanır. İki boyutlu planları üç boyutlu oyun ortamına aktaran bu tez sayesinde bölüm tasarımı ve oyun ortamı oluşturma işi çok daha az zaman alır hale gelmektedir. Oyun oluşturmak isteyenler için iki boyutlu plan üzerinden üç boyutlu bölümler oluşturmaları sağlanmaktadır. Oyunun üç boyutlu yapısı ile uğraşmayıp sadece kodlama ve tasarım ile uğraşmak isteyenler özgün oyun ortamları yapabileceklerdir. Değişik yapıları Unity ortamında tasarlamak yerine çizerek veya etiketlere veri yerleştirip zamandan kazanç sağlamaları amaçlanmıştır.

1.5. Tez Çalışmasının Kısıtları

Girdi olarak dörtgensel ve doğrusal şekiller içeren kat planları kabul edilmektedir. Raycast'in doğru şekilde sınıflandırma yapabilmesi için beş metre ve altındaki aralıklar kapı olarak değerlendirilip birleştirmektedir. Bundan dolayı girilen planlarda koridor genişliğinin beş metreden fazla olması beklenmektedir. Zenginleştirilmiş oyun ortamları alışveriş merkezi, ev ve şato konseptlerinde oluşturulmaktadır.

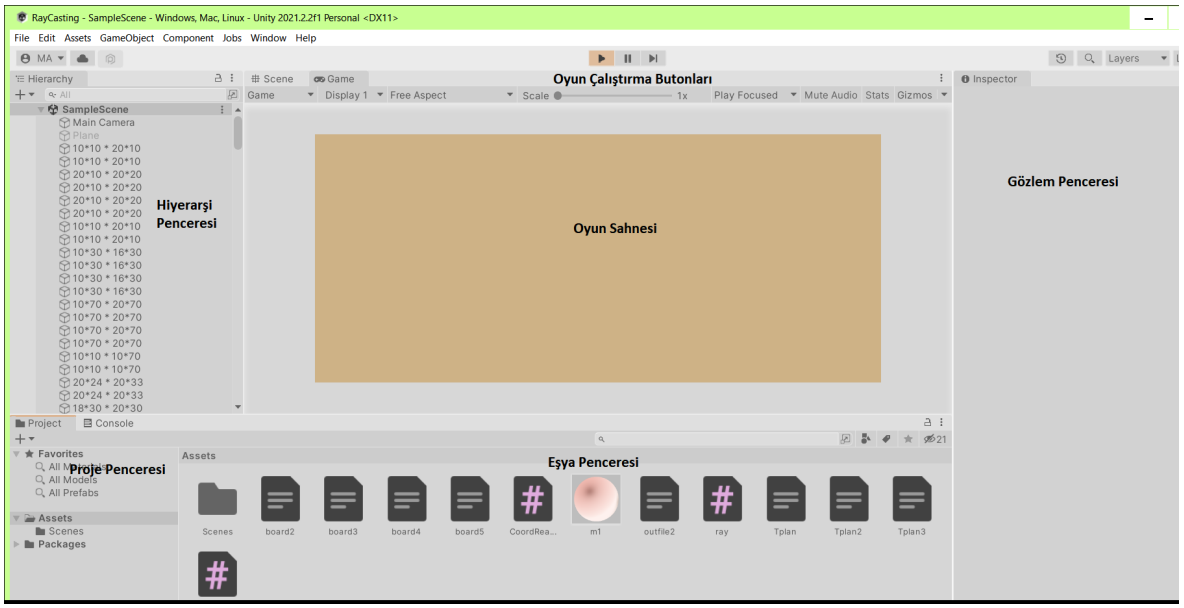
2. Genel Bilgiler

2.1. Raycast Yöntemi

Modern Bilgisayar oyunlarında çevredeki nesneleri belirlemek için birçok metot kullanılır. Raycast yöntemi bunlardan biridir. Raycast'in türkçe karşılığı ışın yayılımı olarak çevrilmektedir. Bu metodun uygulanmasında öncelikle bir merkez noktası belirlenir. Bu noktadan belli bir doğrultuda bir Ray(ışın) gönderilir. Bir engele çarpıp çarpmadığı kontrol edilir. Raycast ışını bir engele çarptığında koşul sağlanmış olur. Raycast tek bir doğrultuda uygulanacağı gibi her yönde yani 360° uygulanabilir. Bu tezde her yönde uygulanmıştır. Daha çok birinci şahıs nişancı oyunlarda hedef belirlemek ve hedefe hasar vermek kullanılır[10].

2.2. Unity ve Bileşenleri

Unity oyun geliştirme ve üç boyutlu oyun modellemede etkili ve yaygın kullanılan bir uygulamadır. Unity'de kullanıcı girişine göre üç boyutlu tasarım yapılabilir. Tasarlanan üç boyutlu ortama çeşitli modeller ve eşyalar eklenebilir, oluşturulan ortamın rengi, dokusu ve yapısı değiştirilebilir. Kullanıcının ortamı görmesi kamera özelliği ile ayarlanır. Kullanıcının ortamı görmesini sağlayan kamera özelliği verilen komutlara göre değiştirilebilir[11].



Şekil 2.1 Unity ana penceresi bileşenleri

Unity genelde bilgisayar ve mobil ortamlara oyun simülasyon ve kullanıcı etkileşimli uygulamalar geliştirmek için kullanılan oyun motoru ve yardımcı araçlar içeren uygulamadır. Unity'de C# ve Javascipt dilleri kullanılarak oyunlar ve simülasyonlar oluşturulmaktadır. Unity'de yapılan uygulamalar Android, IOS ve Windows gibi platformlarda yayınlanabilmektedir. Bu tezde yol ve yön bulma çözümlerine geliştirilen algoritmalarla ulaşılmaya çalışılmıştır. Unity'de oyun grafiği araçları ve kodlar birlikte kullanılmaktadır. Oyun grafik nesneleri yaratılıp üzerine kod, oyun motoru nesneleri ve oyun motorunun sağladığı fiziksel özellikler eklenebilmektedir. Bu özelliği ile Unity kullanıcı için pratik oyun oluşturma ortamlarından biridir.

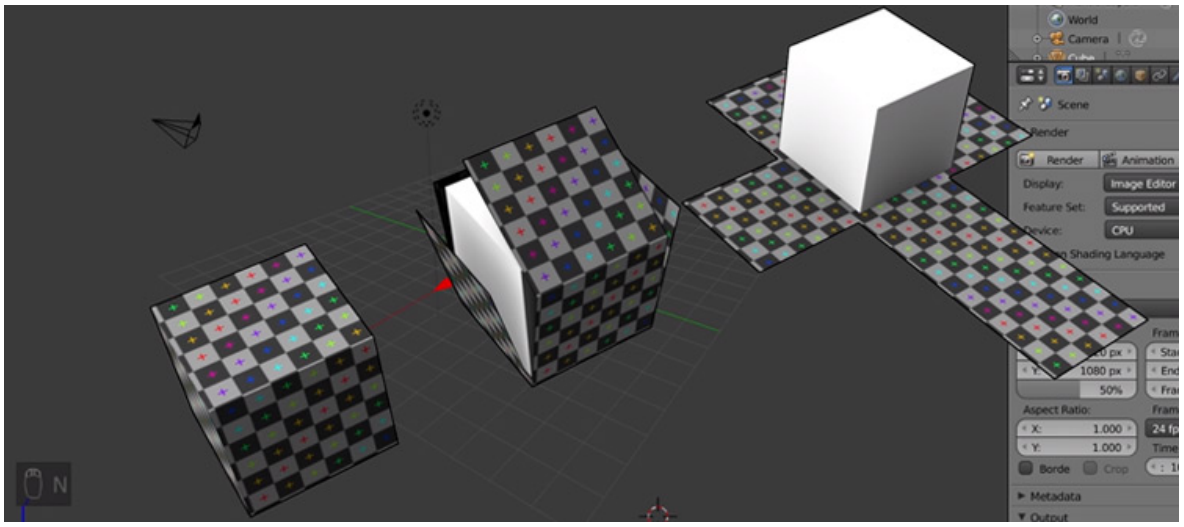
C# nesne tabanlı bir dildir. Java'ya benzer bulunan C# Microsoft tarafından ortaya çıkarılmıştır. Çoklu platformlar tarafından desteklenmektedir. Birçok güncel uygulamada tercih edilmektedir. Unity de bunlardan biridir[12].

Bu tezde ışın yayılımını sağlamak için Unity'nin fonksiyonu olan *Physics.Raycast* kullanılmıştır[11]. Bu fonksiyonla belirli açılarda Raycast ışınları gönderilebilmektedir. Unity'de *Physics.Raycast* ve *Physics.Spherecast* Raycast fonksiyonları olarak kullanılmaktadır. *Physics.Raycast* fonksiyonu içerisinde; *origin*, *direction*, *maxDistance*,

layerMask, *queryTriggerInteraction* parametreleri bulunmaktadir. *Origin* Raycast'in başlangıç noktasını, *direction* Ray' in hangi yönde gönderileceğini, *maxDistance* ışının uzunluğunu, *layerMask* ışın çarptığında *Collider*'ın dikkate alınıp alınmayacağını, *queryTriggerInteraction* ise çarpışmanın dikkate alınıp alınmayacağını belirtir. *Physics.Spherecast* fonksiyonu içerisinde; *origin*,*radius*, *hitinfo*, *maxDistance*, *layerMask*, *queryTriggerInteraction* parametreleri bulunmaktadır [13]. *Origin* Raycast'in başlangıç noktasını, *radius* Ray'in hangi yarıçapta gönderileceğini, *direction* ışınların hangi yönde yayılacağın, *hitinfo* çarpılan nesne hakkında daha fazla bilgi içeriğini, *maxDistance* ışının uzunluğunu, *layerMask* ışın çarptığında *Collider*'ın dikkate alınıp alınmayacağını, *queryTriggerInteraction* ise çarpışmanın dikkate alınıp alınmayacağını belirtir. Bu tezde Raycast, duvarların ayrıştırılmasında kullanılmıştır.

Unity'de sahnelerde kullanabileceğiniz, çeşitli özellikler eklenebilen oyunu oluşturan temel nesneler Gameobject olarak adlandırılır. Bu nesnelere Collider, Rigidbody, doku(texture) vs. gibi özellikler eklenebilmektedir. Bu tezde duvarlar Gameobject olarak oluşturulmuştur. Bu Gameobject'lere RigidBody ve Collider eklenmiştir. *Instantiate()* fonksiyonu daha önce tanımlanan nesnelerin tekrar oluşturulmasında ve çoğaltılmasında kullanılmaktadır. Linerenderer, Unity'de çizgi(line) oluşumunu sağlamaktadır. Çizgi'nin materyal, kalınlık, renk, pozisyon gibi özellikleri belirlenmektedir [14]. Collider Unity'de bir nesnenin oyun motoru tarafından görünür olup olmadığını ifade eder. Collider nesnelerin çarpıştırıcı özelliğidir. Ayrıca nesnelerinin birbirne girmesine engel olmak için fiziki özellik kazanmalarını sağlar. Nesneye collider eklendiğinde artık çarpışmalar vs. için o nesne oyun motoru tarafından değerlendirilmeye alınacaktır.

Prefab tekrar kullanılacak bir nesnenin kopyalanmasını, kaydedilmesi ve değiştirilmesinde kullanılır. Bir Gameobject'in kopyalanması yerine Prefab olarak kullanılması hafıza açısından daha verimlidir. Prefab kullanımı geniş kapsamlı projelerde karmaşıklığı azaltır. Unity'de Uv, iki boyutlu koordinatları birleştirerek üç boyutlu nesne oluşturmak için kullanılır. Uv de u yatay, v ise dikey koordinatları temsil eder. Uv iki boyutlu istenen bir alanı örerek doku oluşturmaya yarar. Köşeler, kenarlar ve çokgenler oluşturarak yüzeyler elde edilir[15].

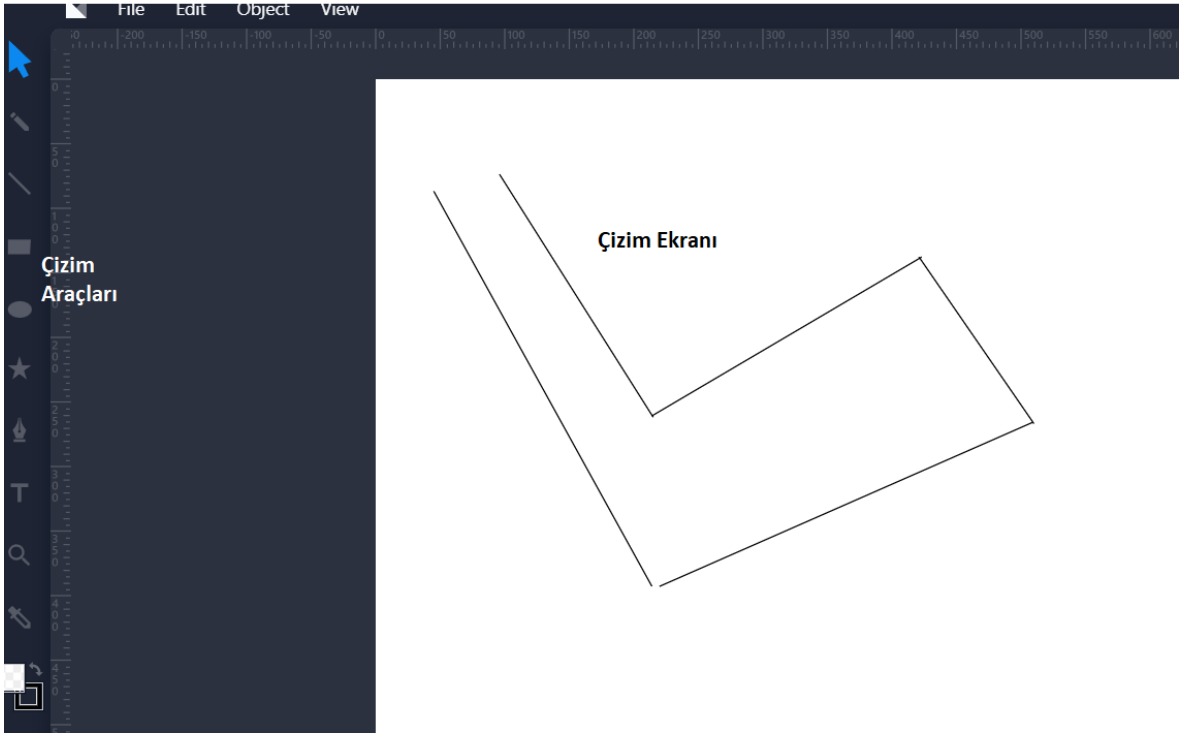


Şekil 2.2 Uv ile üç boyutlu haritalandırma[3]

2.3. SVG Formatı

XML dosyaların daha kolay anlamlandırılması için tasarlanmış pratik şekilde dosya oluşturmaya yarayan işaretleme dilidir. Genelde web ile alakalı dosyalarda kullanılır. Etiketler işaret tabanlıdır. XML yeni etiketler oluşturulmaya müsade eder. Bundan dolayı XML tabanlı birçok yeni dosya biçimi oluşturulmuştur. SVG de XML yapısından yeni etiketler kullanılarak türetilmiştir. XML tabanlı olması kolay üretilebilir ve düzenlenebilir yapmaktadır[16]. Genellikle web sitelerinde ve web uygulamalarında kullanılmaktadır. Bu tezde çizim formatı olarak SVG format kullanılmaktadır. Format iki boyutlu yapı planları için uygun olsa da üç boyutlu oyun geliştirme araçları için uygun bir format değildir. Bu tezde SVG formatı çeşitli yöntemler kullanılarak üç boyutlu oyun ortamı oluşturma konusunda uygun hale getirilmiştir. SVG formatı diğer görsel formatlara göre daha düşük boyutlardadır. SVG vektörel bir format olması nedeniyle mekan tasarımına daha uygundur. SVG, XML tabanlı olduğundan tasarım sürecinde kod ile düzenlemeye uygundur. SVG formatında daha karışık şekiller için ”Path” kullanılır. Path’i oluşturan ”M,m,L,l,,V,v,H,h,C,c,z” gibi harf kodları, geometrik şeklin nasıl olacağını belirler. Path’in başlangıç noktasını M harfi belirlemektedir. M den sonra gelen iki sayı Pathin kesin başlangıç noktasıdır. Başlangıç noktası m harfi olarak ifade edildiğinde ise yolun o an bulunduğu noktaya verilen sayılar kadar hareket edilmesi istenir. L; SVG de line yani yolu temsil eder. L harfi kendinden

sonra girilen x ve y koordinatlarına bir çizgi çeker. Bir diğer sembol olan l harfi ise mevcut noktadan girilen x ve y koordinatı miktarınca hareket edilmesini sağlar. V harfi kendinden sonra girilen y koordinatına bir çizgi çeker. Bir diğer sembol olan v harfi ise kendinden sonra girilen y koordinatı miktarınca hareket edilmesini sağlar. H harfi kendinden sonra girilen x koordinatına bir çizgi çeker. Bir diğer sembol olan h harfi ise kendinden sonra girilen x koordinatı miktarınca hareket edilmesini sağlar. C harfi kendisinden sonra girilen üç nokta koordinatına göre eğri çizecektir. SVG dosyaları içerisindeki şekillerde her harf vektörün sonraki konumu hakkında bilgi verir. Polyline ve Line çizimlerinde etiket içinde doğru parçalarının koordinatları doğrudan verilmektedir[17]. Bu tezde geliştirilen metotlar kullanılarak SVG Path'lerini, her harf koduna bakılarak sonraki noktanın konumu hesaplanmaktadır. Hesaplanan noktalardan doğru parçaları oluşturulmaktadır. Elde edilen bu çizgi bilgileri Unity'ye aktarılmaktadır. Aktarılan bu veriler Unity'de düzenlenip gelen verilere göre üç boyutlu odalar oluşturulmaktadır. Tüm plandan bu şekilde üç boyutlu bir yapı simülasyonu oluşturulmaktadır.

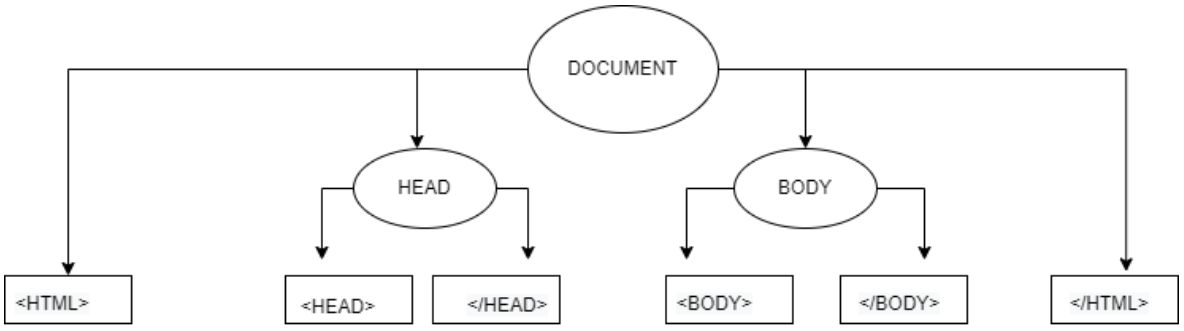


Şekil 2.3 SVG planı oluşturmak için çizim aracı[4]

2.4. Python Veri Analizi ve Veri Aktarımı

Veri iletişimi bilgisayarla yapılan uygulamalarda temel taşlarından biridir. Bu iletişimde veriler başka bir ortama aktarılırken ortam tarafından tanınmaya uygun yapıda olmayabilir. Veri iletişiminde alıcının sorun yaşamadan bilgileri temin edebilmesi çeşitli protokoller ile sağlanır. Protokoller uygun formatlarla çalışır. Eğer veriler alıcının tanımlayabildiği bir formatta ise bu bilgiler sorunsuz şekilde değerlendirmeye alınabilecektir. Bilinmeyen bir yapıdaysa, veri alınsa da format uygun olmadığı için verilerin ayrıştırılması ya da uygun formata çekilmesi gerekmektedir.

Özellikle veri bilimi ve makine öğrenme uygulamalarında veri ayrıştırma çok geniş şekilde kullanılmaktadır. Ayrıştırma işlemi veri elde etmede en önemli işlemlerden biridir. Bir formattan kullanımı daha kolay bir formata veri çevrimini ifade eder. Lexical analiz kod metnini mevcut anahtar kelimelere göre jetonlara ayırma işlemine denir[18]. Ayrıştırılmış verilerin jetonlarla daha rahat şekilde kullanılmasını amaçlar. Etiketli veri kullanma ve bunları ayrıştırma web sayfalarında XML, HTML ve farklı formatlarda kodların yorumlanmasında yaygınca kullanılmaktadır[19]. SVG kod yapısı etiketlerden oluştuğu için ayrıştırmaya uygun bir formattır.



Şekil 2.4 Etiketlerine ayrılmış HTML formatı

Python, 1991 yılında ortaya çıkan makine öğrenmesi, veri işleme dahil birçok alanda kullanılan betik tabanlı yüksek seviye bir dildir. Python betik tabanlı olduğundan öğrenmesi kolay bir dildir. Python'ın farklı alanlarda birçok kütüphanesi olmasından dolayı birçok alandan projelerde kullanılmıştır. PySVG; Python ile SVG dosyası oluşturmayı sağlar.

Lxml kütüphanesi Python'da ayrıştırma işlemi için yapılmış kütüphanelerden biridir. XML; SVG kodlarını girdi olarak alınmasını sağlar. Bu kütüphanenin Etree modülü SVG dosyasını ayrıştırmak için kullanılır. Bu fonksiyonun kullanımı ile SVG, etiketlerine göre ayrılmaktadır. Bir diğer kullanılan kütüphane ise Python Shapely'dir. Python shapely deterministik mekansal analiz ve geometrik analiz konusunda pratik bir kütüphanedir. Coğrafi bilgi sistemlerinin birçok alanında kullanım kolaylığı sunar. Birçok mekansal veri paketi vardır. Doğru ve poligon ile ilgili birçok kullanışlı fonksiyona sahiptir. Shapely'nin poligonları kapsayıcı tek bir poligona çevirmek için kullanışlı fonksiyonları vardır. Shapely koordinatlardan doğru parçaları oluşturmada ve bunları kapsayıcı bir poligona dönüştürmede kullanılabilmektedir[20].

3. Literatür Özeti

Yapı platformlarında hangi formatların kullandığı ile ilgili araştırma yapılmıştır. Ortam analizi ve sınıflandırma için metotlar araştırılmıştır. Kat planlarında iki boyutlu SVG formatı çok fazla karşılaşılan bir modeldir. SVG formatındaki yapı planları iki boyutlu olarak gösterilmektedir. SVG formatı, vektörel olduğundan küçültülüp büyütüldüğünde kalitesini kaybetmemesi sonucu tercih sebebi olduğu görülmüştür.

Walsh ve diğerleri[21] Raycast yöntemi kullanan robotla yapılan haritalandırmada verimliliği arttırmayı amaçlamışlardır. Bunun sonucunda yüksek performans ile etraftaki engel ve duvarların yerlerinin belirlenmesi ve ortam haritalandırılması sağlanmıştır. Raycast ile haritalandırmanın donanımı yoğun kullandığı bilindiğinden, geliştirdikleri verimi artırmayı amaçlayan *Compressed Directional Distance Transform for Accelerating* metodu kullanmışlardır. Daha etkin bir bellek ve cpu kullanımı sağlayarak robotlarla Raycast haritalandırma yapmışlardır.

Çangır[22] üç boyutlu ortam sensörü olan Kinect kamera kullanarak çevredeki cisimleri belirlemiştir. Üç boyutlu ortamda nesne seçimi için metot geliştirmiştir. Kinect medya kamerayla optimal nesne seçimi için Unity de oyun ortamı oluşturmuştur. Nesne seçimi

sırasında Unity oyun motorunun Raycast metodunu kullanmıştır. Bu tezde Kinect aletinin üç boyutlu Raycast özelliğini kullanarak nesne seçiminde bulunmuştur. Raycast ışınını seçim için kullanarak farklı seçim aralıklı ortamlarda çalışma yapmıştır.

Carlier ve diğerleri[23] karmaşık SVG ikonları oluşturmak için üretici sinir ağları tasarlamıştır. Konvolüsyonel sinir ağları kullanılarak oluşturulmuş olan DeepSVG baştan grafik üretme yeteneğine sahiptir. Konvolüsyonel sinir ağları, örneklemeler arasında güçlü bağlantı kurma yeteneğinden dolayı tercih edilmiştir. Burada kompleks resimleri SVG formatında ifade etmeye çalışmışlardır. Farklı kategorilerden 100.000 tane yüksek kalite örnek SVG kullanmışlardır. Raster grafiklerinin karesel gridlerle ifade edilmesinden farklı olarak vektör grafikleri matematiksel formüller ve kodlarla ifade edilmiştir.

Salameh ve diğerleri[24] SVG resim demetini RDF grafiklerine dönüştürür. SVG formatı çözünürlükten bağımsız ve son derece küçük boyutlu görüntü kodlamasına sahip olduğu için işlenmemiş girdi formatı olarak tercih edilmiştir. Ayrıca SVG'ler XML tabanlı olduğu için yazı(text) bilgileri de taşıyabilmektedir. Geliştirdikleri sistem otomatik olarak SVG'den RDF format grafiklerine dönüşüm yapar. Ham SVG resimleri anlamlı ve etiketlendirilmişRDF tabanlı resimlere çevirir. Kullanıcıya daha önce kaydedilmiş renk, şekil gibi kriterlere göre dönüştürülmüş RDF resimlerinde arama şansı sunar.

Othman ve diğerleri[25] nesne belirlemede Unity kullanmışlardır. Burada Unity uygulaması gerçek hayata yakın bir simülasyon ortamı sağladığı için geliştirdikleri algoritmayı Unity'de Raycast kullanarak canlandırmışlardır. Unity'nin üç boyutlu görüş açısı verme özelliğini Raycast ile kullanmışlardır.

Kapetanakis ve diğerleri[26] SVG dosyalarının X3D dosya formatına çevrimi ile ilgili çalışma yapmışlardır. Bu çalışmada iki boyutlu'den üç boyutlu'ye web tabanlı da yeni bir çevirme mekanizması kullanılmıştır. SVG formatı XML hem iki boyutlu, XML türevi olan X3D üç boyutlu olarak gösterilebilimektedir. Bu çalışma XML tabanlı olduğu için webde kolayca kullanılabilmektedir. X3D formatında gerçek zamanlı iletişim web servis tarafından sağlanan bir alt yapıya sahiptir. X3D internet tarayıcısına eklenen bir eklenti ile webde kullanılabilmektedir. Bu projede SVG formatında iki boyutlu koordinatlarda olan veriler

X3D'de üç boyutlu koordinatlara çevrilmektedir. SVG'de açı olarak derece kullanılır, bu çevrim sırasında X3D radyan kullandığı için dereceden radyana çevirme yapılmaktadır. SVG formatında döndürme x ekseniyle yapılan açı olarak ifade edilmektedir X3D formatında ise döndürme *x, y, z ve theta* değerleri kullanılarak yapılmaktadır. X3D'de yükseklik pozisyon renk gibi değerler metadata olarak alınmıştır.

Eidenberger[27] SVG destekli SMIL'in webde öğrenme aracı olarak kullanıldığından bahsetmektedir. SVG'lerin öğretici eğitim materyali olarak kullanılması amaçlanmıştır. SMIL XML tabanlıdır ve interaktif multimedya içerik oluşturulmasını sağlar. SVG XML tabanlı olduğu için ve vektörel grafikleri kullandığı için multimedya uygulamalarında kullanılmaya uygundur. Burada kullanılan SVG JavaScript kodları ile birlikte yazılmıştır. Eğitim materyali oluşturmada SMIL ve SVG formatlar müzik ve videoların yanında görsel olarak kullanılmıştır.

Battiato ve diğerleri[28] tarafından geliştirilen *DDT(Domain Delaunay Triangulation)* yönteminde resimlerin Raster'den SVG formatına çevrilmesi sağlanır. Delaunay üçgenleme yöntemi kullanılmıştır. Raster yönteminde resim büyütüldüğünde görüntü kötü gelmektedir. SVG formatına çevrildiğinde büyütme işleminde görüntülerde kalite kaybı olmamaktadır. Gelen raster veri büyüklüğüne göre üçgen oluşturulacak alan da değişecektir. SVG formatı ile boyut azalacaktır ve bu raster verilerine göre avantaj sağlayacaktır.

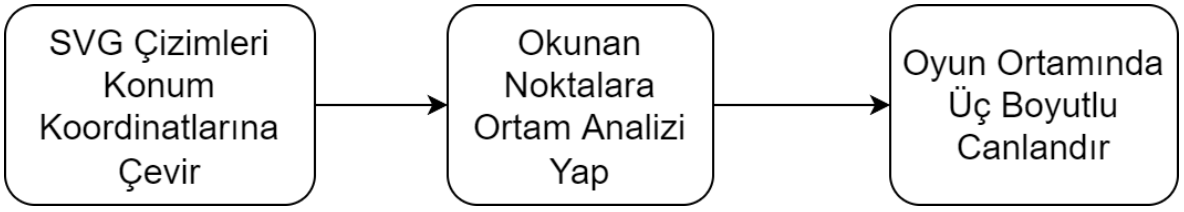
Byun ve diğerleri[29] iki boyutlu tasarımları, BIM (Building information system) kullanarak üç boyutlu ortama çevirmişlerdir. Bilgisayar Yardımlı Tasarım(CAD) kullanılmıştır. Dijital ortamda inşaat bilgilerinin modellenmesi amaçlanmıştır.

Gimenez ve diğerleri[30] mevcut binalardan, üç boyutlu modellemeye yardımcı olmak için ve uygun maliyetli yaklaşımlara duyulan ihtiyaçtan dolayı çözüm önermişlerdir. İki boyutlu taranmış planlardan üç boyutlu modeli otomatik olarak yeniden oluşturmak için bir çözüm önerilmiştir. Yeniden yapılandırma; geometri, topoloji ve anlam bilimi ile çözüm üretilmesine dayanır. Duvarlar, açıklıklar ve boşluklar gibi yapı elemanları tanınır. Oluşturulan üç boyutlu modelleri zenginleştirmek için çözümler önerilmektedir.

Yin ve diğerleri[31] iki boyutlu planları üç boyutlu BIM (Building information system)'e aktarılmasını incelemişlerdir. Bu işlem sırasında farklı algoritmaları karşılaştırmışlardır.

4. Önerilen Metot

Bu tezde girdi olarak alınan SVG dosyası çeşitli ayrıştırma metotları kullanarak Unity'de iki boyutlu ortama, sonrasında buradan alınan veriler Unity'de üç boyutlu ortama aktarılmıştır.



Şekil 4.1 Akış diyagramı.

4.1. SVG analizi

SVG içinde doğrusal planlar oluşturmak için genelde Line, Polyline ve Path kullanılmaktadır. SVG formatında yapı planları genelde Path ile oluşturulmaktadır. Path vektörel yapıda olduğundan, ayrıştırarak poligonlar ya da doğru parçaları elde etmek için algoritma geliştirilmiştir. Ayrıştırılması için algoritma Tablo 4.1'de gösterilmiştir. Line, Polyline etiketlerden gelen veriler nokta bilgilerini vermektedir. Bunlardaki bilgiler doğrudan alınmaktadır. Path ise vektörel veriler içermektedir. Python Lxml kütüphanesi kullanılarak SVG okunmaktadır. Okunan SVG, çağrılan metotlarla etiketlerine ayrılmaktadır. Path, polyline, Line şeklinde etiketler ile gelen veriler geliştirilen algoritmalar ile doğru parçaları veya poligon noktalarına ayrıştırılarak dosya olarak kaydedilmektedir.

Şekil 4.2 de SVG ayrıştırılması gösterilmiştir. Önce SVG dosyasından Path etiketli bölüm alınmaktadır. Sonrasında içeriği harflere göre ayrılır. Son olarak harf kodlarına göre vektörel bilgiler koordinat verilerine çevrilerek iki boyutlu plan oluşturulmaktadır.

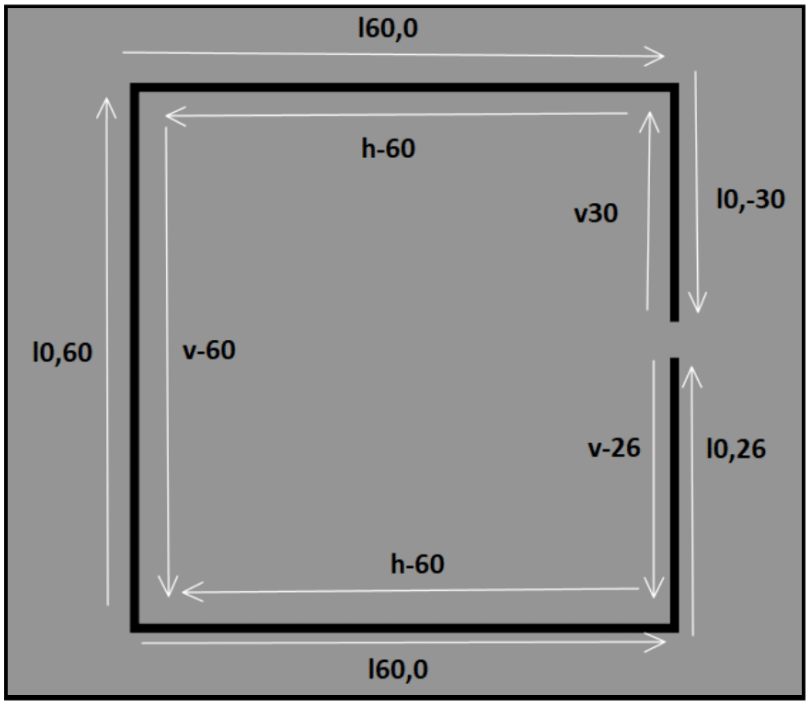
```
<svg width="640" height="290" xmlns="http://www.w3.org/2000/svg">
 <g>
  <title>Layer 1</title>
<path d="m0,0l60,0,l0,26v-26h-60l0,60,l60,0l0,-30,v30,h-60,v-60" id="svg_5"
stroke="#000" fill="none"/>
 </g>
</svg>
```

1)SVG dosyasında path etiketi ile başlayan metin ayrılır.

```
<path d="m0,0l60,0,l0,26v-26h-60l0,60,l60,0l0,-30,v30,h-60,v-60" id="svg_5" stroke="#000"
fill="none"/>
```

m0,0
l60,0
l0,26
v-26
h-60
l0,60
l60,0
l0,-30
v30
h-60
v-60

2)Path içerisindeki metin harf kodlarına ayrılır.



3)Ayrılan harf kodlarına göre iki boyutlu çizim yapılır.

Şekil 4.2 SVG dosyasının ayrıştırılıp plan elde edilmesi.

Algorithm 1 SVG ayrıştırma algoritması.

```
python XML tree ile SVG'yi oku
if etiket line ise then
    line noktalarını al
end if
if etiket polyline ise then
    polyline noktalarını al
    belirlenen noktaları listeye ekle
end if
if etiket path ise then
    if etiket d ile başlıyorsa then
        m noktasının önündeki koordinatları listeye ekle
        if sonraki harf küçük l ise then
            okunan x ve y değeri kadar hareket et
            belirlenen noktaları listeye ekle
        end if
        if sonraki harf L ise then
            okunan koordinatlara git
            belirlenen noktaları listeye ekle
        end if
        if sonraki harf h ise then
            değer kadar x koordinatlarında ilerle
            belirlenen noktaları listeye ekle
        end if
        if sonraki harf H ise then
            x koordinatına okunan değeri ata
            belirlenen noktaları listeye ekle
        end if
        if sonraki harf v ise then
            değer kadar y koordinatlarında ilerle
            belirlenen noktaları listeye ekle
        end if
        if sonraki harf V ise then
            y koordinatına okunan değeri ata
            belirlenen noktaları listeye ekle
        end if
        if sonraki harf c ise then
            ilk koordinatlardan son koordinata git
            belirlenen noktaları listeye ekle
        end if
    end if
end if
```

Tablo 4.1 SVG ayrıştırma algoritması.

4.2. Unity Raycast Ortam Analizi

Klasörde kaydedilen noktaların bulunduğu dosya, Unity'de iki boyutlu projede dizin adresi girilerek okunmaktadır. Dosyadan okunan metne belli karakterlere göre ayırma işlemi uygulanmaktadır.

Algorithm 2 Unity iki boyutlu ortam analizi için nokta okuma

```
1: dosya yolunu belirle
2: tüm metni oku
3: if metinde ']' yada '[' varsa then
4:     bu değerlere göre ayır
5:     ayrılan değerleri listede sakla
6: end if
7: for ayrılan değerler do
8:     her noktayı ikili şekilde ayır
9:     her noktayı nokta listesine ekle
10: end for
```

Tablo 4.2 Çizgi ekleme algoritması.

Bu işlem sonucunda önce ön etikete göre listeye alınmaktadır. Sonrasında bu liste içerisinde x ve y koordinatları olacak şekilde noktaların saklanacağı listeye eklenmektedir. Listeye alınan noktalar koordinatlarına göre sırasıyla eklenmektedir. Bu noktalar kullanılarak ardışık iki nokta arasında olacak şekilde çizgi Gameobject'leri oluşturulmaktadır. Oluşturulan çizgi Gameobject'leri teker teker çizgi listesine eklenmektedir. Bu şekilde Unity'de SVG dosyası iki boyutlu olarak tekrar çizilmektedir. Oluşturulan Line'ların ışınların çarptığında çarpışma olması için Gameobject'lere Rigidbody ya da Collider eklemek gerekmektedir. Her Line uzunluğuna ve merkez noktasına göre her çizgiye Collider özelliği eklenmektedir. Bunun sonucu ışın çarptığında Raycast ışının nereden ve hangi açı ile geldiğine dair çarpışma bilgileri elde edebilmektedir. Collider uzunluğunun ve merkez noktasının nasıl eklendiğine dair bilgiler Tablo 4.3'te gösterilmektedir.

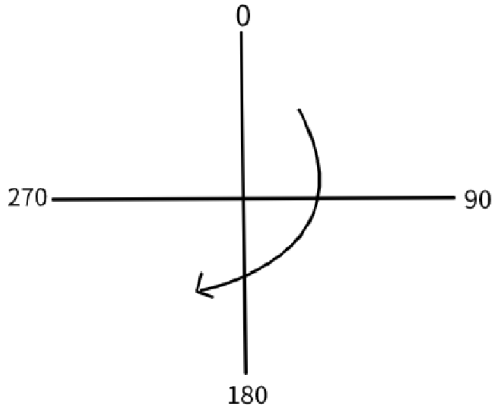
Algorithm 3 Çizgi ekleme algoritması

```
1:  tüm noktaları yükle
2:  for all nokta : tüm noktalar do
3:      çizgi nesnesi oluştur
4:      çizgi nesnesinin collider'ını başlangıç ve bitiş noktalarının ortasında oluştur
5:      çizgi nesnesi uzunluğunu başlangıç ve bitiş noktalarının arasındaki uzunluğu yap
6:      çizgi nesnesine trigger ekle
7:      çizgi nesnesine kinematic ekle
8:      çizgi nesnesinin başlangıç pozisyonunu başlangıç noktası pozisyonu yap
9:      çizgi nesnesine linerenderer ekle
10:     çizgi nesnesinin scale'ini (1,1,0) ayarla
11: end for
```

Tablo 4.3 Çizgi ekleme algoritması.

Bu tezin hedeflerinden biri SVG planı şeklinde verilen karmaşık şekilleri anlamlı oda ve koridor gibi bölgelere ayırmak için metot geliştirmektir. Bunu sağlamak için başlangıçta iç ve dış alanı ayırmak için bir algoritma geliştirilmiştir. Sonrasında iç alan; odalar ve koridor şeklinde bölümlere ayrılmaktadır. Bölümlere ayırma işlemi Raycast'de elde edilen bilgilerle dış duvar, ortak duvar ve koridor duvarı şeklinde ayırma ile olmaktadır. Burada en kritik işlem dış duvar, ortak duvar ve koridor duvarlarını etiketleme işlemidir. İlk Raycast merkez noktasından başladığımızda çevresindeki duvarlar başlangıç ve bitiş

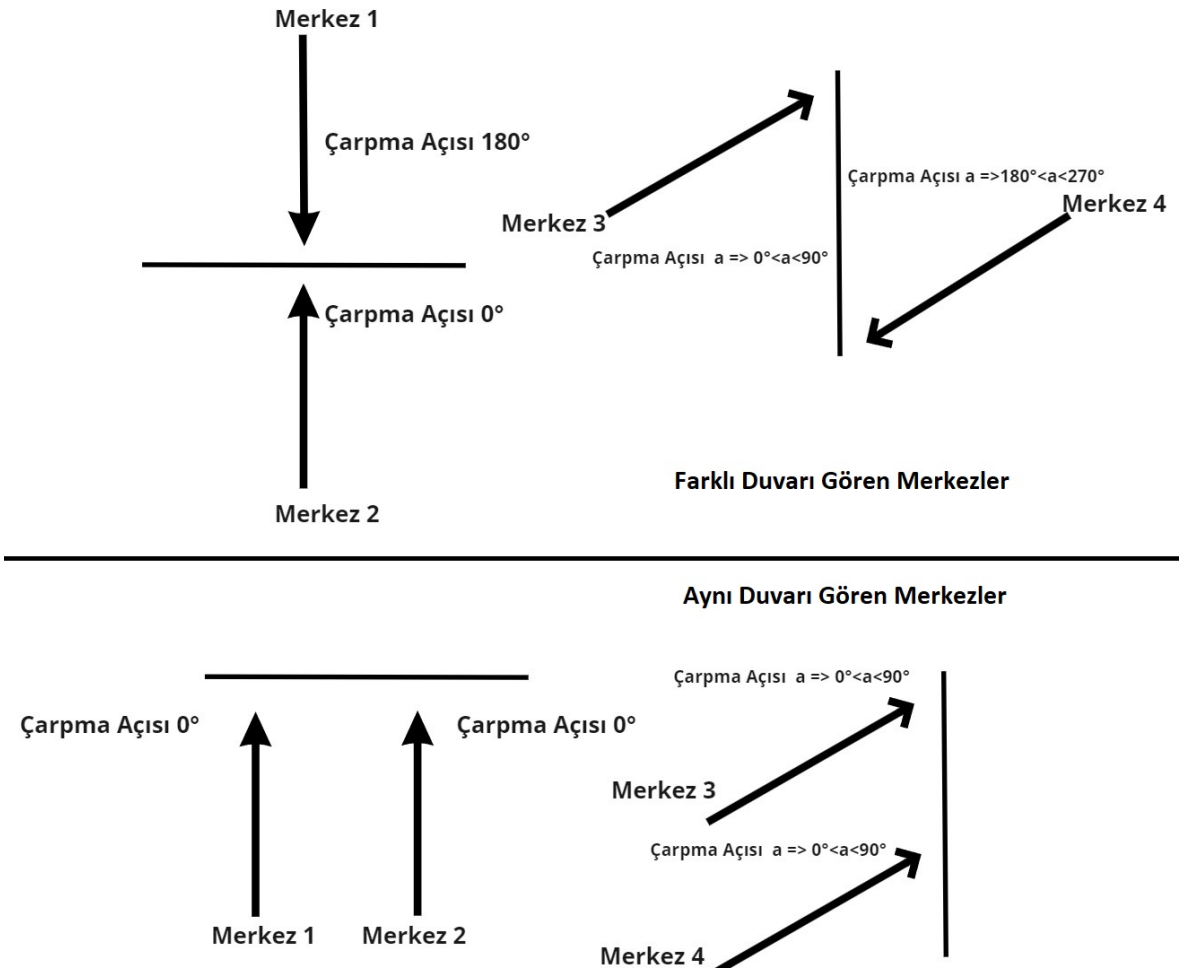
koordinatları ile dış duvar noktaları olarak listeye alınacaktır. Sonrasında diğer bir noktadan Raycast işlemi yaptığımızda eğer bu duvara tekrar bir ışın çarpıyorsa ve ışınların arasındaki fark 90 dereceden fazlaysa bu duvar ortak duvar noktaları listesine alınacaktır ve dış duvar listesinden çıkarılacaktır. Duvarlara çarpan ışınlar arası açı farkının 90 dereceden fazla olması duvarın iki Raycast merkezi arasında olduğunu ve ortak duvar olduğunu belirtmektedir. Şekil boyunca tüm Raycast noktalarında 360° boyunca ışınlar gönderilerek ışın çarpan duvarların bilgileri alınmaktadır.



Şekil 4.3 Unity'de Raycast ışınlarının derece cinsinden gönderim yönü.

Bu tezde Raycast metodu haritalandırma ve ortam bilgisi elde etmede kullanmıştır. Nokta listesinden okunan her noktanın x ve y koordinatlarının üç metre altında ve üstünde Raycast merkezleri belirlenmektedir. Raycast uygulanacak noktaların konum ve sayısı, haritalandırma ve yapının bölümlere ayrılması için gereklidir. Eğer Raycast merkez noktaları duvarlar tarafından engellenen yerlerde olursa yapı eksik şekilde haritalandırılacak dolayısıyla doğru şekilde duvarlara ayrılmayacaktır. Bu nedenle yeterli sayıda ve doğru konumlandırılmış Raycast merkez noktası önemlidir.

Bu merkez noktalarından 360° boyunca boyunca her 10°'de bir ışın gönderilmektedir. Merkez noktasından her yöne aralıklı olarak toplamda 36 Raycast ışını gönderilmektedir. Gönderilen ışınlar daha önce oluşturulan, duvarları temsil eden iki boyutlu çizgilere çarptıklarında, çarpışma bilgileri alınmaktadır. Bu bilgiler, ışının hangi açı ile gönderildiği ve hangi çizgiye çarptığını içeren bir listeye eklenmektedir. Bununla ilgili algoritma Tablo



Şekil 4.4 Raycast ışınlarının farklı ve aynı odalardan gönderimi sonucu çarpma açılarının gösterimi.

4.4'de incelenmektedir. Açı bilgilerin alımı projenin amaçlarından biri olan dış duvar, ortak duvar ve koridor duvarı arasındaki farkın belirlenmesinde bir ayraç olarak kullanılmaktadır.

Ortak duvar, dış duvar, koridor duvarı seçimi tamamen ışınlardan gelen duvar ve açı bilgileri ile belirlenmektedir. Raycast'in doğru şekilde sonuç vermesi için ışınların etrafındaki boşlukların kapanması gerekmektedir. Her Raycast ışını eğer dört yön boyunca farklı duvarlara çarpıyorsa bu Raycast merkezi oda içerisindedir olarak değerlendirilmektedir. Bu noktanın raycast ışınlarının çarptığı bütün odalar bir liste içerisine eklenmektedir. Bu liste eğer daha önce eklenmediyse ya da kendisini kapsayacak başka bir oda listesi yoksa oda listesine eklenmektedir. Tüm yönlerdeki duvarların değerlendirildiği algoritma Tablo

Algorithm 4 Raycast algoritması

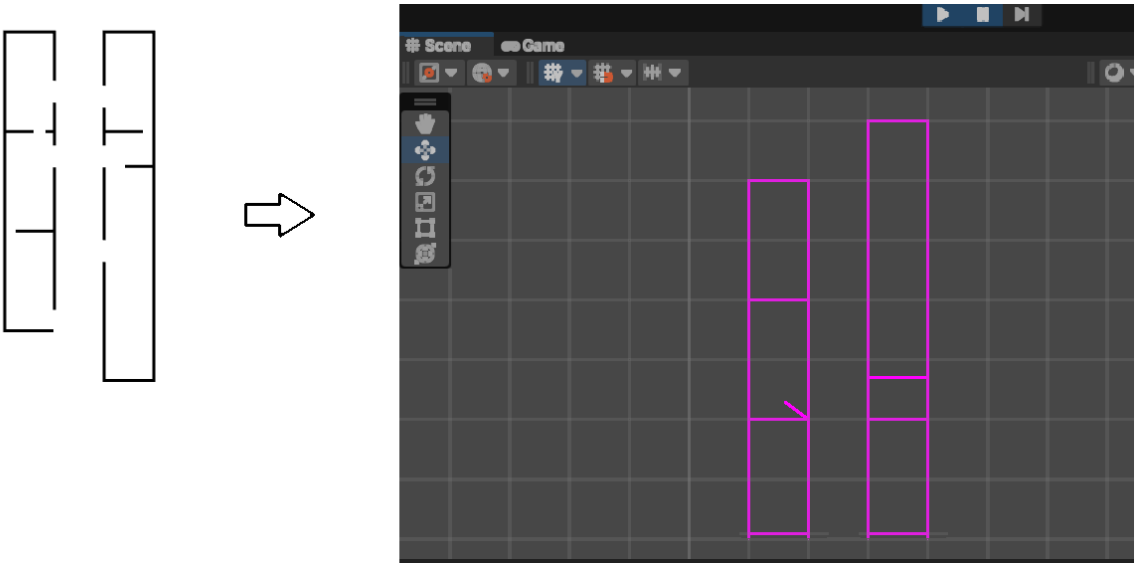
```
açı = 0
while açı < 360 do
    açıyı 10 arttır
    if ışın çizgiye çarpıyorsa then
        listeye al
        if listede 4 farklı yönden çizgi var ise then
            burası bir odadır
            if bu odadan listede yoksa then
                oda bilgilerini geçici listeye ekle
                çarpan açı değerleri ve oda bilgileri ile tuple oluştur
                oluşturulan tuple'ı listeye ekle
            end if
        end if
    end if
end for
if 4 farklı yönden çizgiye çarpmıyorsa then
    devam et
end if
```

Tablo 4.4 Raycast algoritması.

4.4'de gösterilmiştir. Bir oda içerisinde Raycast noktasının konumuna göre eksik duvar görünümü olabilmektedir. Bunu engellemek için Raycast noktaları artırılmıştır. Her noktanın gördüğü duvarlar; her Raycast sonucu oda listesi ile karşılaştırılıp eksik duvarlar giderilmeye çalışılmıştır. Işın çarpan her duvarın bilgileri '(açı,duvar)' şeklinde listeye eklenmektedir.

Her Raycast merkezinden ışınların 360° gönderilmesinden sonra bu süreç tekrarlanmaktadır. Dış duvar, ortak duvar listeleri algoritmaya göre yeniden oluşturulmaktadır. Açı durumlarına göre önceden dış duvar olarak listelenmiş duvar, sonraki Raycast fonksiyonunda ortak duvar ya da koridor duvarı olarak listelenmesi gerekebilmektedir. Raycast yapılacak nokta sayısının artırılması daha doğru sonuç için gereklidir. Odalar artırılan numaralar şeklinde kaydedilmektedir. Eğer halihazırda bulunan bir oda duvarlarını kapsayan yeni bir oda verisi varsa eski oda silinmektedir. Kapı görevi gören boşlukların olması Raycast'in çalışmasında doğru sonuç elde etmeyi engellemektedir. Örneğin; kapı olarak düzenlenmiş boşluklardan çıkan Raycast ışınları oda ile ilgili yanlış duvar bilgileri elde edilmesine

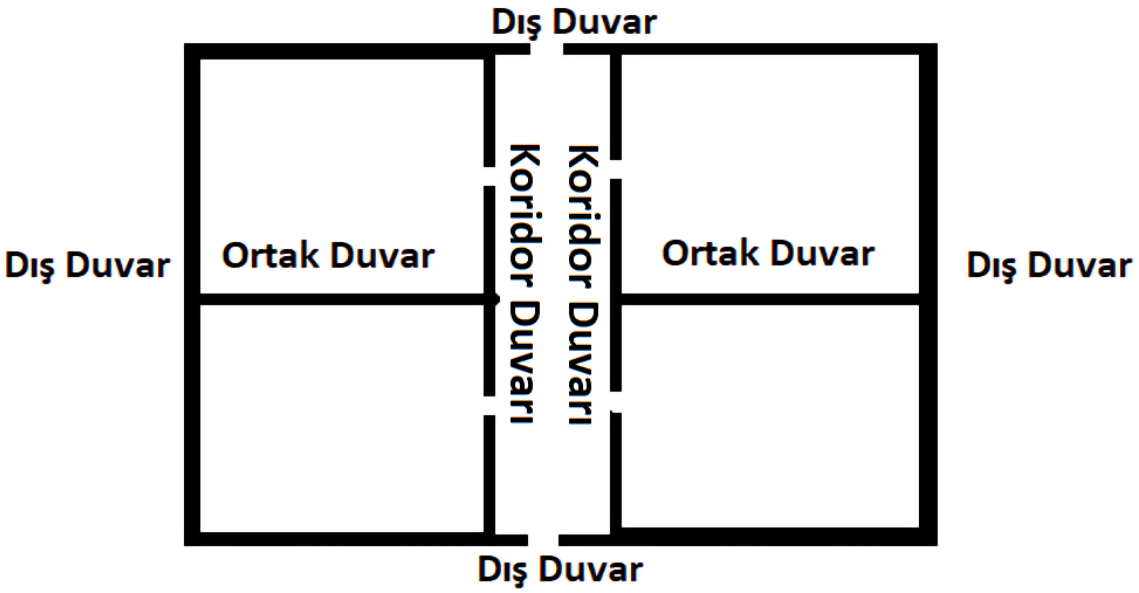
neden olacaktır. Raycast'in daha sağlıklı yapılması için belli uzunluktaki aralıklar kapı olarak değerlendirilmekte ve bu boşluklar çizgiler oluşturularak kapatılmaktadır. Duvarlar arasındaki belli uzunlukta boşlukları kapatan bu algoritma, ayrıca muhtemel kapı ve aralıklarıda göstermektedir.



Şekil 4.5 SVG planın Unity'de kapı boşlukları doldurularak tekrar oluşturulması.

Burada önemli bir husus aynı oda içerisindeki iki Raycast merkezi varsa bu iki nokta aynı duvarları görecektir. Bu, bir dış duvarın yanlış değerlendirme ile ortak duvar listesine alınmasına neden olabilecektir. Böyle durumları engellemek için Raycast verileri Tuple olarak hem duvar bilgisi hem de açısı ile alınmaktadır. Bir duvar farklı iki Raycast tarafından gönderilen ışın tarafından gözükse bile bunların açıları arasındaki fark duvarın ortak ya da dış duvar olduğunu belirtir. Örneğin; bir duvar iki oda tarafından da dışa bakan bir duvar olarak değerlendirilebilir. Eğer bu şekilde bir duvarsa iki merkezden gönderilen farklı ışın tarafındaki açılar yaklaşık değerde olacaktır. Farklı olarak, duvar iki odayı ayıran bir duvar ise iki merkezden çıkan ışının açıları arasındaki fark 90°'den büyük olacaktır. Bu değerlendirmeyi yapan algoritma Tablo 4.5'de gösterilmiştir. Ortak duvar, dış duvar farkı noktadan Raycast işlemi ile tekrar tekrar yenilenerek yapılmaktadır.

Burada diğer bir önemli nokta ise koridora bakan duvarların diğer duvarlardan ayrımı konusudur. Koridor için geliştirdiğimiz algoritmada, koridorun yapısı gereği kendisine açılan



Şekil 4.6 Duvar tiplerinin sınıflandırılması.

Algorithm 5 Duvar tipi algoritması

```
for all oda : tüm odalar do
    for all duvar : oda do
        if duvar daha önce görülmüşse then
            if açılar arası fark 90 dereceden büyükse then
                duvarı ortak duvar listesine ekle
                duvarı dış duvar listesinden çıkar
            end if
        end if
    end for
end for
```

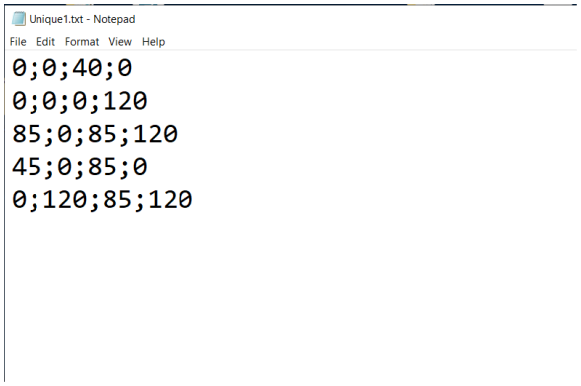
Tablo 4.5 Duvar tipi algoritması.

bir çok kapı olması beklenmektedir. Bundan dolayı bir oda içerisinde kendisi açılan dört ya da beşten fazla kapı ya da aralık raycast tarafından görülüyorsa bu oda koridor olarak değerlendirilmektedir. Ayrıca her oda için 180° farklı açılarda ışınların uzaklık farkına göre bir en boy oranı elde edilerek koridor için bir sabit değer elde edilmektedir.

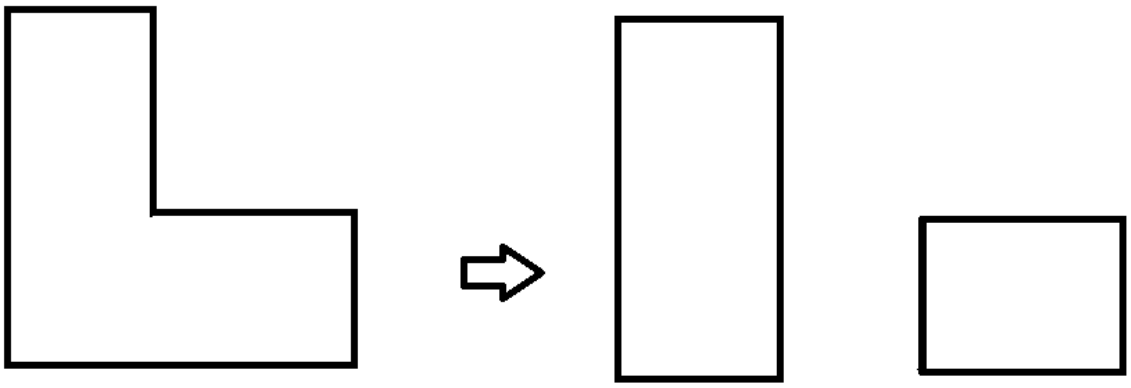
Ev ve alışveriş merkezi için dörtgensel çatı konseptleri geliştirilmiştir. SVG planlarının karmaşık bir yapıda olması çatı yapımını zorlaştırmaktadır. Karmaşık yapıdaki planlar için

algoritma geliştirilmiştir. Bu algoritma şekli dörtgenlere bölerek ev için üçgen prizma, alışveriş merkezi için dörtgensel alan şeklinde çatı oluşturulmasına imkan sağlamaktadır.

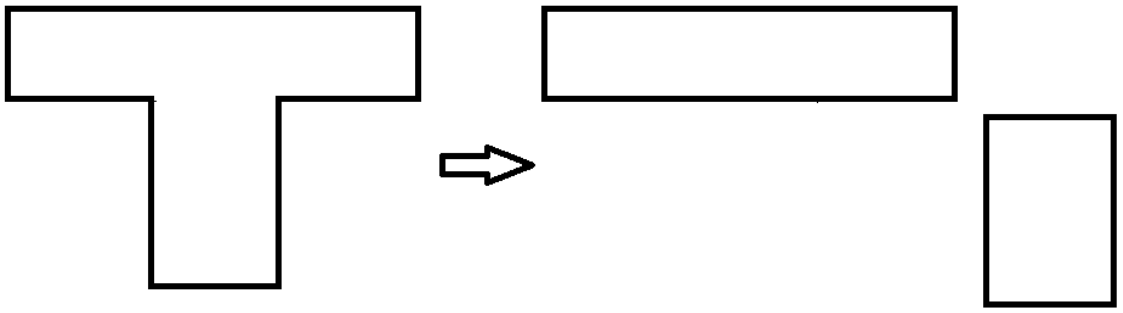
Duvar koordinatlarına ayrılmış konum bilgileri Şekil 4.6 de gösterilmiştir. Konum koordinatları ortam analizi sonucu her satırda x1, y1, x2, y2 noktaları olacak şekilde noktalı virgül ile ayrılarak dosyaya kaydedilmektedir.



Şekil 4.7 Ortam analizi sonucu elde edilen duvar konum bilgileri.



Şekil 4.8 L şeklindeki kapsayıcı alanın çatı algoritmasıyla dörtgenlere ayrılması.



Şekil 4.9 T şeklindeki kapsayıcı alanın çatı algoritmasıyla dörtgenlere ayrılması.

Algorithm 6 Çatı algoritması.

```
1:  noktalarla oluşturulabilecek tüm farklı köşegenleri bul
2:  for all köşegen : tüm köşegenler do
3:      if en uzun köşegen ise then
4:          if köşegen kendi dikdörtgensel alanı içerisindeki bir köşegen tarafından
            kesilmiyorsa then
5:              kendini kesen eş köşegeni bul
6:              bu iki köşegenden dörtgen oluştur
7:              oluşan bu dörtgeni çatı listesine ekle
8:          end if
9:          if kendini kapsamayan bir köşegen tarafından kesiliyorsa then
10:             kendini kesen eş köşegeni bul
11:             bu iki köşegenden dörtgen oluştur
12:             oluşan bu dörtgeni çatı listesine ekle
13:         end if
14:     end if
15: end for
```

Tablo 4.6 Çatı algoritması.

4.3. Planın Üç Boyutlu Oluşturulması

İki boyutlu ortam analizi sonucu belirlenen dış duvar, ortak duvar, koridor duvarı, oda merkez noktaları ve çatı koordinatları dizine kaydedilmektedir. Değerlendirme sonucu belli kategorilerde sınıflandırılmış bu listelerdeki koordinatlar okunmaktadır. Duvarlar, listelerden alınan '(nokta1, nokta2)' şeklinde okunan değerlerden elde edilmektedir. Elde edilen '(nokta1, nokta2)' Tuple'ı oluşturulan CreateWall() fonksiyonuna girerek iki nokta arasında duvar oluşturulmaktadır. CreateWall() fonksiyonu bir Prefab kullanır. Bu fonksiyon, Unity'nin Instance() fonksiyonunu kullanarak girilen Prefab'a göre iki nokta arasında Prefab nesnesini çoğaltır. İki nokta arasındaki duvarın özellikleri, yüksekliği, genişliği ve dokusu fonksiyon aracılığı ile ayarlanabilmektedir.

Algorithm 7 Üç boyutlu duvar oluşturma algoritması

```
1:  for all tüm duvarlar do
2:      başlangıç noktasını al
3:      bitiş noktasının al
4:      oyun nesnesi oluştur
5:      başlangıç ve bitiş noktası arasındaki uzunluğu hesapla
6:      başlangıç ve bitiş noktası arasını prefab ile doldur
7:      bitişe göre dönme katsayısını ayarla
8:      oyun nesnesine mesh renderer ekle
9:      mesh renderer'i başlangıç ve uzunluğa göre ayarla
10: end for
```

Tablo 4.7 Üç boyutlu duvar oluşturma algoritması

Yapıların duvarının temelini oluşturacak Prefab'a giydirilecek doku için araştırma yapılıp konsepte uygun birçok doku eklenmiştir. Şato duvarını oluşturmak için şato resimlerinden dokular oluşturulmuştur. Alışveriş merkezini simüle etmek için dış duvar ve koridor duvarında şeffaf cam dokular kullanılmıştır. Evin dış duvarı ise tuğla dokuları ile üç boyutlu olarak oluşturulmuştur.

Çatı olarak iki boyutlu şekli tamamen kaplayacak şekilde algoritmadan gelen dörtgensel bir çatı kullanılmıştır. Ev oluşumunda dış duvarların pencereler içerecek şekilde bölünmesi

Algorithm 8 Merkez noktası bulma algoritması

```
1: for all nokta: tüm Raycast merkez noktaları do
2:     yatay yönde uzaklıkları hesapla
3:     yatay konumu ve iki uzaklığı kullanarak odanın yatay merkezini bul
4:     dikey yönde uzaklıkları hesapla
5:     dikey konumu ve iki uzaklığı kullanarak odanın dikey merkezini bul
6:     x ve y koordinatlarını merkez noktası listesine ekle
7: end for
```

Tablo 4.8 Merkez noktası bulma algoritması.

amaçlanmıştır. Bundan dolayı dış duvar ikinci aşamada her beş metrede bir pencere olacak şekilde bölünerek pencere ve duvar parçalarına ayrılmıştır. Dış duvar yapısı aralıklı parçalar okunarak oluşturulmaktadır.

Pencere ve pencere yanı duvar dokusu klasik pencere figüründen ilham alınarak tasarlanmıştır. Şekil 5.5'de pencerenin çerçeveleri gözükmektedir. Pencerenin çerçeveleri Uv haritalandırma yöntemiyle oluşturulmuştur. Evin ortak duvarı da tuğla dokusu ile tasarlanmıştır.

Üç farklı konseptte kullanmak için birçok farklı eşya araştırması yapılmıştır. Uygun eşya kütüphaneleri Unity'ye eklenmiştir. Bu eşya kütüphanelerdeki seçilen eşyalar Gameobject olarak projeye eklenmiştir. Oda koordinatlarına göre merkez noktaları ve duvar kenarları baz alınarak konsept içerisine yerleştirilmektedir.

Her konsept için konsepti yansıtan eşya kütüphaneleri araştırılıp eklenmiştir. Şato için şato ortamıyla ilgili, ortaçağ dönemini yansıtan eşyalar bulunmuştur. Ev için ev ortamını yansıtabilecek eşyalar bulmuştur Alışveriş merkezi için beyaz eşya dükkanlarında genel olarak karşılaştığımız eşyalar seçilmiştir. Bu eşyaları yerleştirirken yapının odalara ayrılmasından faydalanmıştır. Odaların merkez noktaları bulunulmuş, merkez noktası ve diğer duvar çeşitlerinden faydalanarak bu eşyalar odaya yerleştirilmiştir. Bu işi yapan yerleştirme algoritmasında tekdüzelikten kaçınmak için duvar ve merkez noktaları birlikte kullanılmıştır.

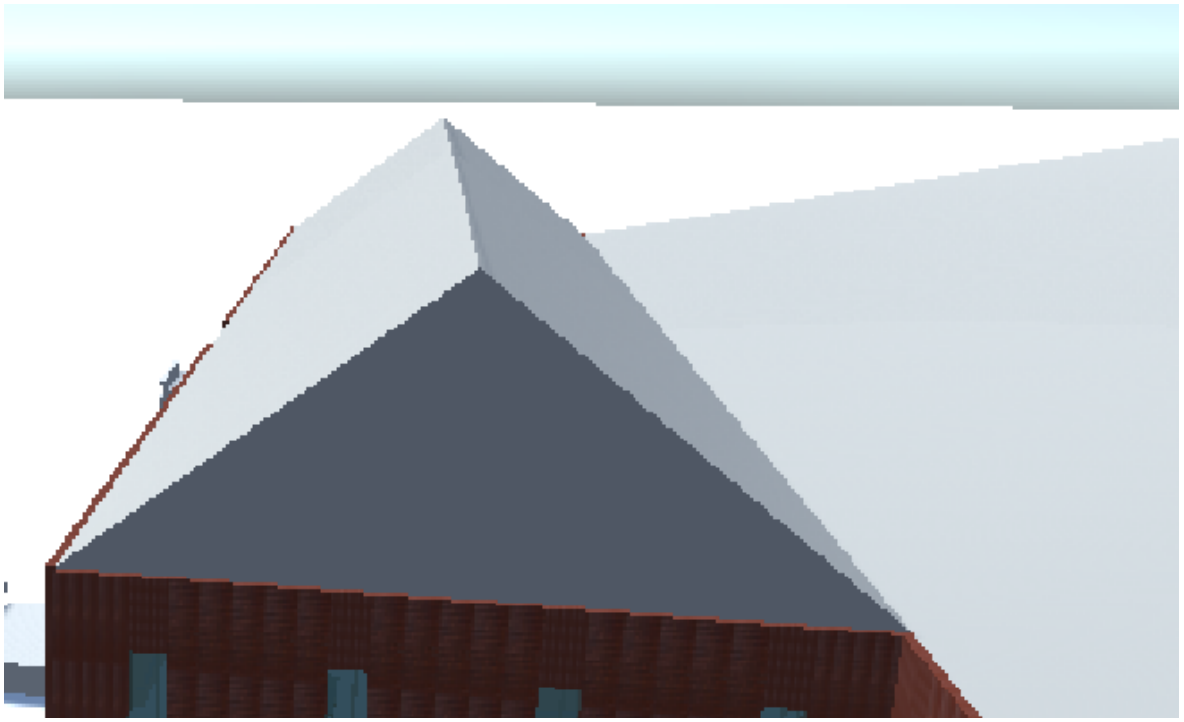
5. Sonuç

Bu tezin en önemli amaçlarından biri kullanıcı tarafından verilen iki boyutlu SVG planı en ideal şekilde üç boyutlu ortama yansıtmaktır. Oda, koridor ve kapı için geliştirilen algoritmalar bunu hedeflemektedir. Bu sınıflandırma sayesinde oyun ortamımız gerçekçi bir ortama dönüşerek oyuncuya ideal oyun ortamı hissi verecektir. Duvarları dış duvar, ortak duvar ve koridor duvarı şeklinde ayırmamızın nedeni her duvarın yapısının üç boyutlu modellemede farklı yapıda olmasından dolayıdır. Dış duvarlar konseptlere göre kapı ve pencere içerecektir. Ortak duvarlar ise penceresiz tek bir dokudan oluşacaktır.

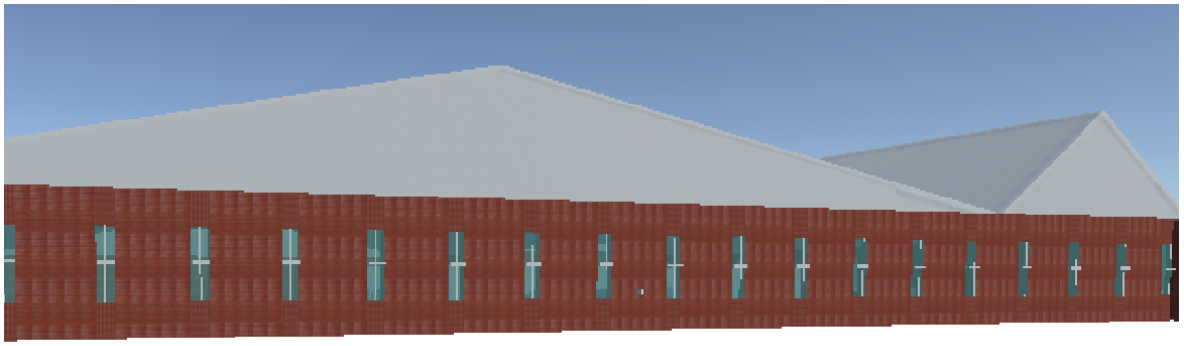
Yapı planlarının iki boyutlu yerine üç boyutlu olarak kullanıcıya sunulması etkin kullanım sağlamaktadır. Planın simüle edilmesi, kullanıcının görmek istediği yapının içini bire bir tecrübe etmesini sağlar. Bu tezdeki amaçlardan biri de kullanıcının daha az zaman harcayarak bir oyun ortamı oluşturmasına olanak sağlamaktır. Bu tezde önerilen metotlar ile geliştiricinin seçimine göre üç farklı konseptte oyun ortamı sağlanmaktadır. Bunlar; alışveriş merkezi, ev ve şatodur. Konseptlerde üç farklı yapının iç duvarları, koridor duvarları ve dış duvarları farklı dokulardan oluşmaktadır. Bu yapıların duvarları Raycast sonucu sınıflandırma ile oluşan dış duvar, koridor duvarı ve iç duvar verileri kullanılarak oluşturulmuştur.

5.1. Ev Konsepti

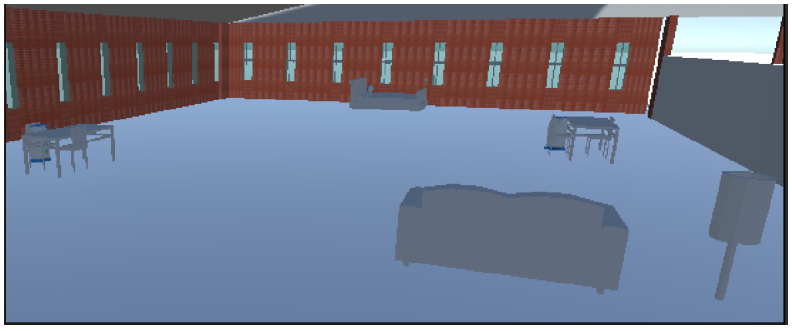
Dış duvarlarda her beş metrede bir pencere gelecek şekilde ayarlanmıştır. Çatı kısmı düz dörtgensel olarak minimum dörtgensel bölge kapanacak şekilde tasarlanmıştır. Koridor duvarları beyaz desenlidir. Ortak duvarlar için tuğla kullanılmıştır. Pencerelerin alt ve üstü duvarlardan oluşmaktadır. Pencere için Uv kullanılarak çerçeve yapılmıştır. Evin çatısı; çatı algoritmasından gelen koordinatlarla minimum alanı kapsayacak şekilde üçgen prizmadır. Üçgen prizmanın dokusu beyaz renklidir. Şekil 5.1' de evin çatı yapısı gösterilmektedir. Çatının yüksekliğini sağlayan üçgen prizmada her alan için yükseklik 8 metredir.



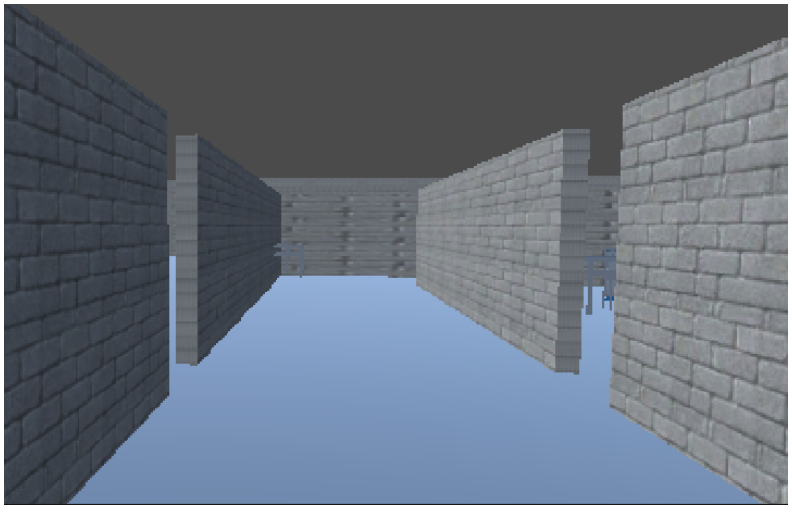
Şekil 5.1 Ev çatısı



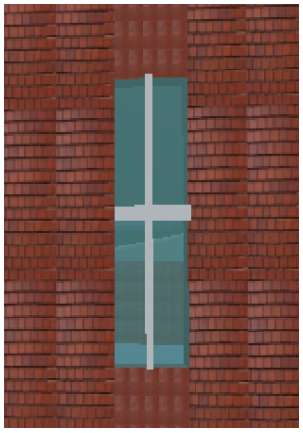
Şekil 5.2 Ev dış duvarlar



Şekil 5.3 Ev odası



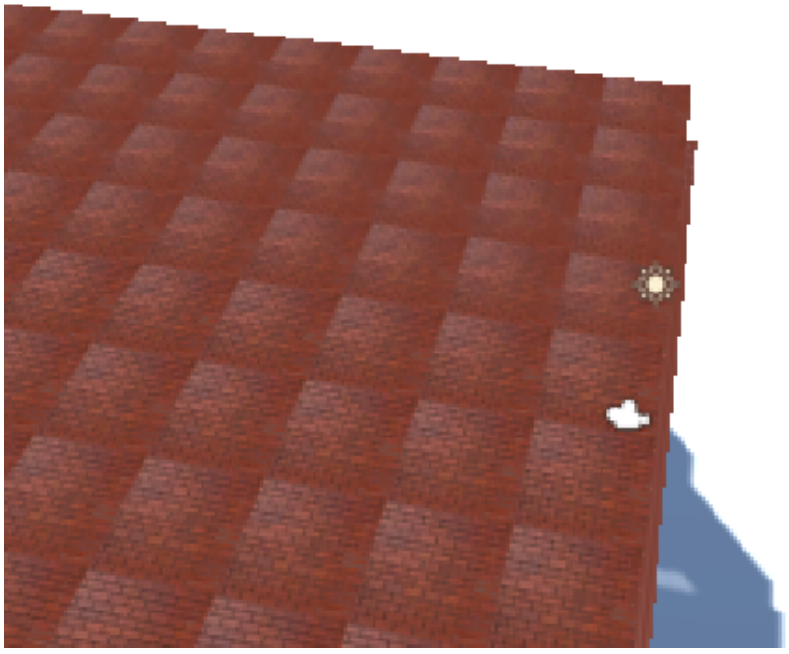
Şekil 5.4 Ev koridoru



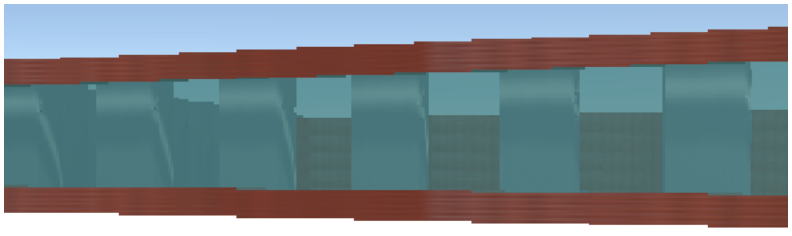
Şekil 5.5 Ev penceresi

5.2. Alışveriş Merkezi Konsepti

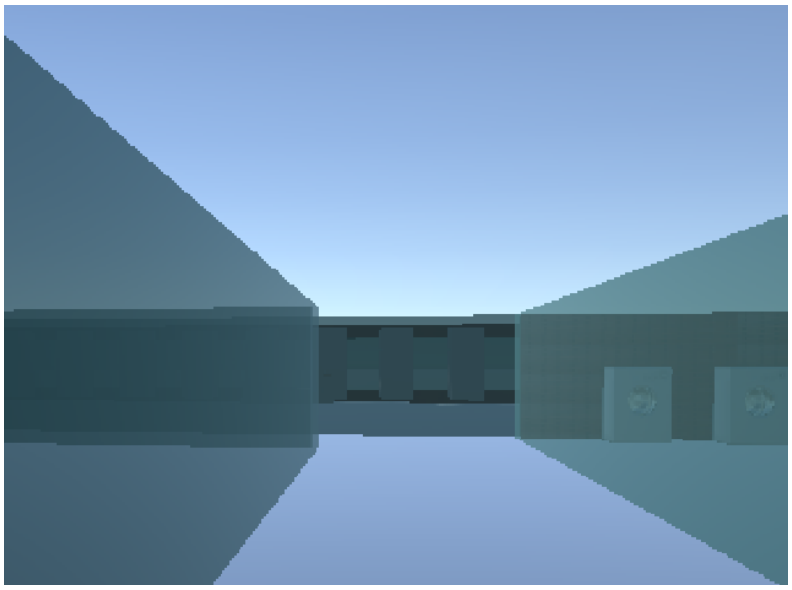
Dış duvarlar; duvar kenarları tuğla dokuyla kaplanarak, kenarların ortası ise şeffaf cam dokuyla kaplanarak oluşturulmuştur. Çatı kısmı düz dörtgensel olarak minimum alanla kaplanacak şekilde tasarlanmıştır. Koridor duvarları beyaz desenlidir. Ortak duvarlar için tuğla kullanmıştır. Alışveriş merkezinde koridora bakan duvarlar ise şeffaf ve cam dokulu olacaktır.



Şekil 5.6 Alışveriş merkezi çatı



Şekil 5.7 Alışveriş merkezi dış duvarlar

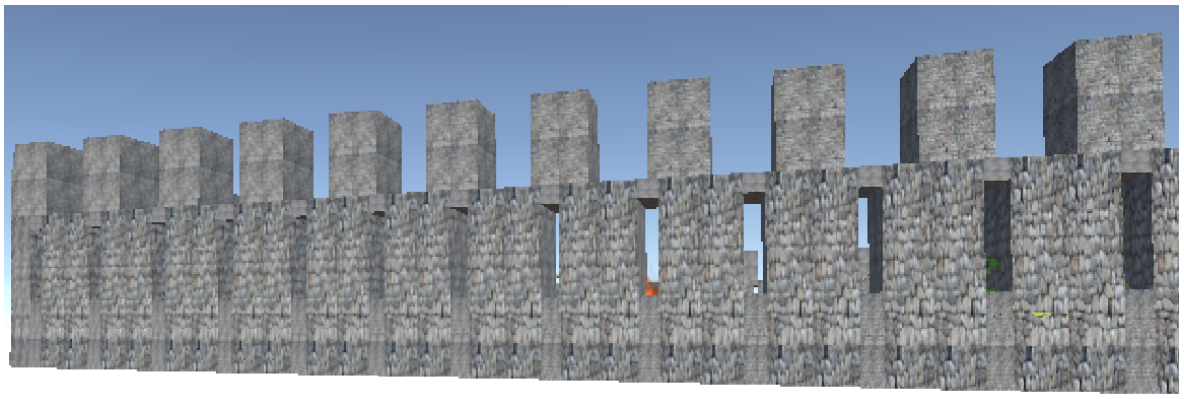


Şekil 5.8 Alışveriş merkezi koridor

Şekil 5.9 Alışveriş merkezi içerisi

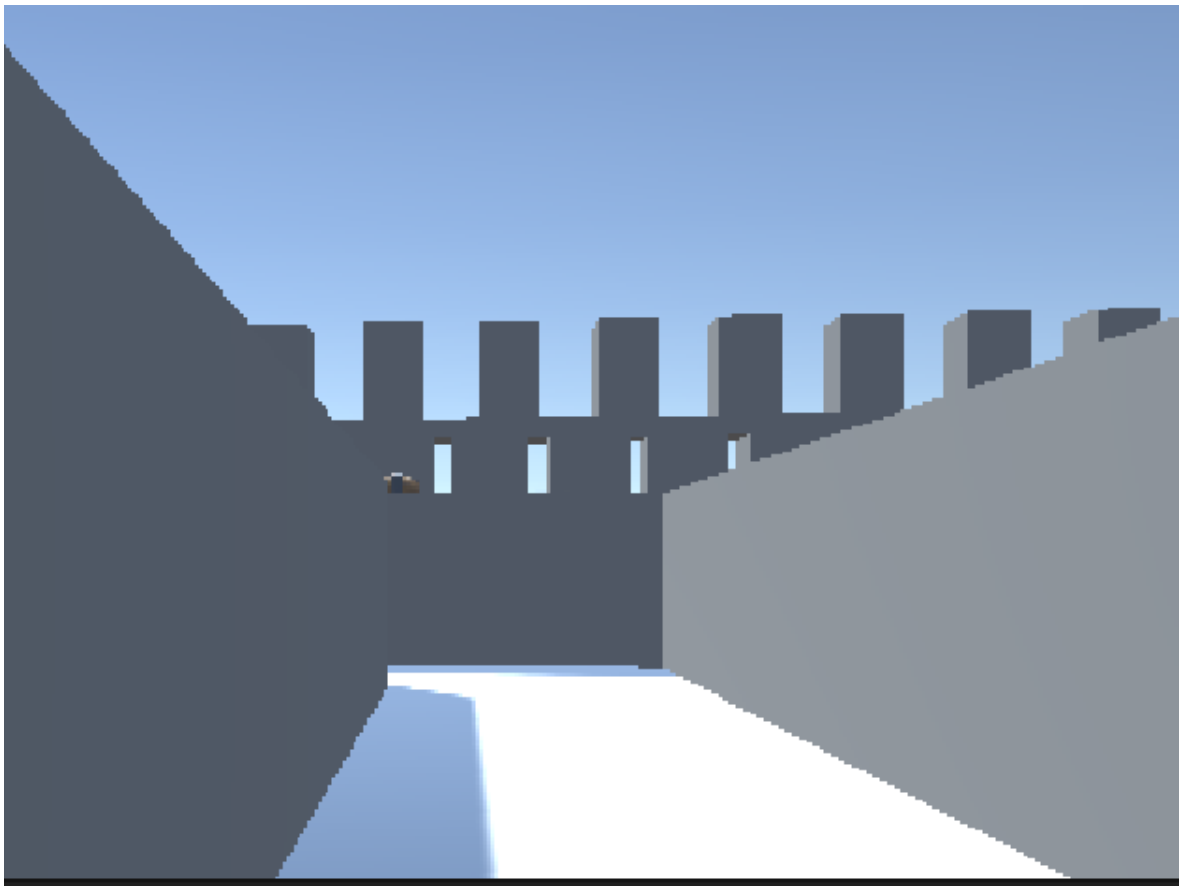
5.3. Şato Konsepti

Dış duvar, kayalardan oluştuğunu göstermek için genişliği kalın ayarlanmıştır. Dış duvarların üst kısımlarında üç metre aralıklı surlarla kaplanmasını sağlayan bir algoritma geliştirilmiştir. Koridor duvarları beyaz desenlidir. Ortak duvarlar için tuğla deseni kullanmıştır. Şato konsept gereği çatı içermemektedir. Bundan dolayı içerisine ağaç ve ateş animasyonu nesneleri eklenmiştir.

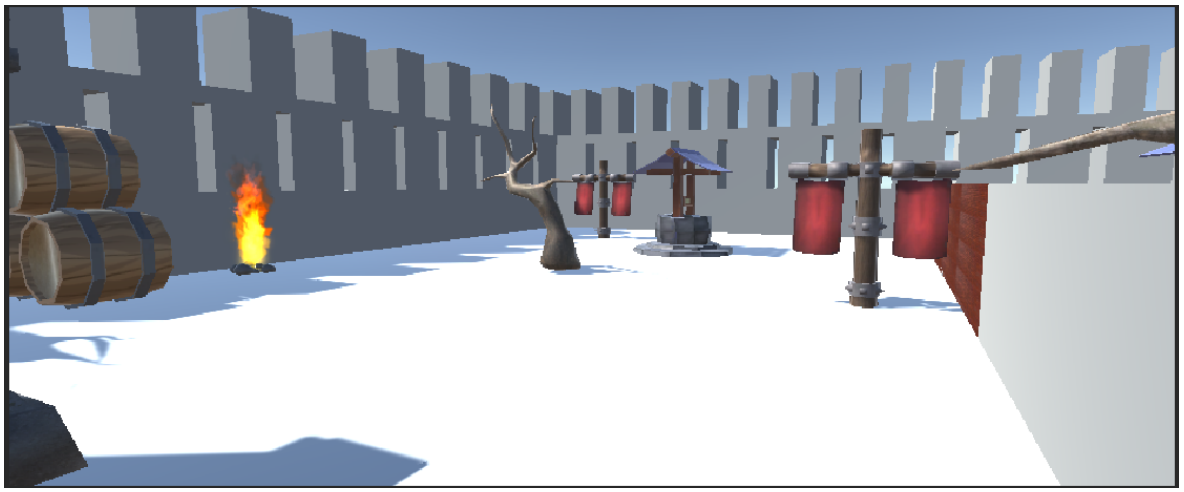


Şekil 5.10 Şato dış duvarları

Şekil 5.11 Şato penceresi



Şekil 5.12 Şato koridor



Şekil 5.13 Şato içi

6. Tartışma

Bu tezde SVG çizimleri üç boyutlu olarak farklı konseptlerde oyun ve simülasyon ortamı olarak oluşturulmuştur. Buradaki oyun ortamı farklı oyun mekanikleri ile desteklenerek kapsamlı bir oyun elde edilebilecektir. Bu tezde daha fazla konseptte oyun ortamı sunulabilirdi. Sunulan konseptler daha fazla eşya ve dokuya sahip olabilirdi. Bu tezde sadece dörtgensel planların üç boyutlu ortamlara çevrilmesi ile ilgili yöntem önerilmiştir. Dairesel ve eğrisel çizimlerde üç boyutlu ortama aktarılabilmesi için de yöntem önerilebilirdi. Çatı ve pencerelerin farklı konseptlerde kullanıcının isteğine göre seçilebilmesi sağlanabilir. Svg ortam analizinde plandaki her nokta için, noktanın belirli mesafe altında ve üstünde iki defa Raycast fonksiyonu çağrılmaktadır. Her Raycast fonksiyonu sonucuyla listedeki duvar tipleri tekrar değerlendirildiğinden SVG ortam analizi fazla zaman almaktadır. Yapılan Raycast yazılımı daha verimli bir fonksiyonla oluşturulabilir. Yüksek donanım özelliklerine sahip bir bilgisayarda çalıştırılması yazılımın daha etkili kullanılması için gereklidir. Bu tezin önerdiği metot, üç farklı aşama olarak çalıştırılmaktadır. SVG çiziminin etiketlere ayrımı Python kütüphaneleri ile yapılmaktadır. Bu noktaların okunması ve Raycast fonksiyonunun uygulanması iki boyutlu Raycast algoritması tarafından yapılmaktadır. En son elde edilen duvarların üç boyutlu farklı konseptlerle oyun ortamına dönüştürülmesi ise üç boyutlu Unity projesinde yapılmaktadır. Bu üç projenin tek istekte çalışması sağlanabilir.

EK-1 Ray Fonksiyonu

```
public void  raycastGlobe(float CoordinatXaxis, float CoordinatYaxis)
{
int RaysToShoot = 60;
float angle = 0;
bool[] Isdirection = new bool[] { false, false, false, false };
var Corridor = new List<float>();
var CoordinatesList = new List<float>();
bool ExecuteOneTime = true;
WallOfPositionList.Clear();
IntialTupleList.Clear();

for (int i = 0; i < RaysToShoot; i++)
{
float x = Mathf.Sin(angle);
float y = Mathf.Cos(angle);
angle += 2 * Mathf.PI / RaysToShoot;

Vector3 dir = new Vector3(transform.position.x + x, transform.position.y + y, 0);


RaycastHit hit;


if (Physics.Raycast(new Vector3(CoordinatXaxis, CoordinatYaxis, 0), dir, out hit))
{
if (hit.rigidbody != null)
{




}
}

Debug.DrawLine(new Vector3(-2, -2, 0), dir, Color.red, 10.0f);
if (Physics.Raycast(new Vector3(CoordinatXaxis, CoordinatYaxis, 0), dir, out hit))
{
if (hit.collider.gameObject.name == "Door")
{
```

```
DoorNumber++;
if (DoorNumber>2)
{
CooridoorDict.Add(DoorNumber,true);
}


}
//4 direction distance
if (i%15==0)
{
FOUR_BROTHERS.Add(hit.distance);
}
if (FOUR_BROTHERS.Count==4)
{
}


if (hit.collider.gameObject.name!="Door")
{
InitialFlaotArray = hit.collider.gameObject.name.Split('*');
FloatWallArray = Array.ConvertAll(InitialFlaotArray, float.Parse);
}
foreach (var item in InitialFlaotArray)
{
}
Debug.DrawLine(new Vector3(1, 4, 0), hit.point, Color.white);

CheckFourDirection(angle, Isdirection, InsideOfFourWalls);

CheckIfWallExistOnce(WallOfPositionList, FloatWallArray);
CheckIfWallExistOnceTuple(IntialTupleList, FloatWallArray, angle);


if (CheckFourDirection(angle, Isdirection, InsideOfFourWalls))
{

RoomWallDictCopy= new Dictionary<int, List<List<float[]>>>(RoomWallDict);
if (i == RaysToShoot - 1)
{


//Ayn  oday  birden fazla kez eklemeyoruz
IsListAreSame = false;
```

```
Debug.Log("CountSee" + RoomWallDictCopy);
//RoomDictList.Add(new Dictionary(IncreaseRoomNumber(RoomNumber),
WallOfPositionList));
foreach (var item5 in WallOfPositionList)
{
Debug.Log(RoomNumber + "---JJJ" + string.Join("\t", item5) + "JJJJ");
}

float k = WallOfPositionList[0][0];
WallOfPositionList = WallOfPositionList.OrderByDescending(x =>
(x[0]+x[1]*x[2]*+x[3])).ToList();



foreach (var item in WallOfPositionList)
{
if (IsItemInTheListString(DoorList,item))
{
    Debug.Log(RoomNumber+" heye " + item);
}

}
if (WallOfPositionList.Select(i => IsItemInTheListString(DoorList, i)).Count()>2)
{
Debug.Log(RoomNumber + " hoho " + WallOfPositionList.Select(i =>
IsItemInTheListString(DoorList, i)).Count());

foreach (var item in WallOfPositionList)
{
    if (IsItemInTheListString(SharedWallList,item))
    {
    }
}

}
if (DoorList.Intersect(WallOfPositionList.ToList()).Count() > 0)
{
}



if (RoomNumber > 0) {
foreach (KeyValuePair<int, List<List<float[]>>> item in RoomWallDictCopy)
{
    if (TwoListAreSame(item.Value[0], WallOfPositionList.ToList()))
```

```
    { IsListAreSame = true; }

    if (TwoListAreIntersect(item.Value[0], WallOfPositionList.ToList()))
    {
        IntersectionList.Add(item.Key);
        InterKey = item.Key;
        IsListAreIntersect = true;

    }
}
if (!IsListAreSame )
{



    RoomWallDict.Add(RoomNumber, new List<List<float[]>>
    { WallOfPositionList.ToList() });
    RoomWallDictTuple.Add(RoomNumber, IntialTupleList.ToList());
    IncreaseRoomNumber(RoomNumber);
    RoomNumber++;
    WallOfPositionList.Clear();
    IntialTupleList.Clear();
    ExecuteOneTime = false;
}


if (IsListAreIntersect)
{
    foreach (var item in IntersectionList)
    {
        RoomWallDict.Remove(item);
        RoomWallDict.Add(item, new List<List<float[]>>
        { WallOfPositionList.ToList() });

        RoomWallDictTuple.Remove(item);
        RoomWallDictTuple.Add(item, IntialTupleList.ToList());

    }


    WallOfPositionList.Clear();
    IntialTupleList.Clear();
    ExecuteOneTime = false;
}
```

```
}
else
{
RoomWallDictSingle.Add(RoomNumber, new List<float[]>
( WallOfPositionList.ToList() ));
RoomWallDict.Add(RoomNumber, new List<List<float[]>>
{ WallOfPositionList.ToList() });
RoomWallDictTuple.Add(RoomNumber, IntialTupleList.ToList());
IncreaseRoomNumber(RoomNumber);
RoomNumber++;
WallOfPositionList.Clear();
IntialTupleList.Clear();
ExecuteOneTime = false;
}
IntersectionList.Clear();
WallOfPositionList.Clear();
IntialTupleList.Clear();
ExecuteOneTime = false;
RoomWallDictCopy.Clear();

}
}
}
if (hit.collider !=null)
{
CheckIfItemOccursOnce(PositionList, (hit.transform.position));

}

if (hit.collider == null)
{
Corridor.Add(angle);
}
}
int ListItemComparingValue = 0;
var keyList = new List<int>(RoomWallDict.Keys);
if (keyList.Count== RoomWallDict.Count)
{
for (int h = 0; h < keyList.Count; h++)
{


foreach (var item in RoomWallDict[keyList[h]][0])
{
foreach (var item3 in RoomWallDictTuple.ElementAt(h).Value)
```

```
{

for (int k = 0; h > k; k++)
{
if (IsItemInTheListStringTuple(RoomWallDictTuple.ElementAt(k).Value,
item3.Item2, item3.Item1) && !IsItemInTheListString(SharedWallList,
(item3.Item2)))
{
    SharedWallList.Add(item3.Item2);
}

}
}


if (RoomWallDictTuple.ElementAt(h).Value.Select(i => IsItemInTheListString(DoorList,
i.Item2)).Count() > 7)
{

foreach (var item2 in RoomWallDictTuple.ElementAt(h).Value)
{

if (!IsItemInTheListString(SharedWallCoordList, item2.Item2))
{
    SharedWallCoordList.Add(item2.Item2);
}
}

}



if (CombinedPositionList.Count == 0)
    {
        Debug.Log(item.GetType());
        CombinedPositionList.Add(item);
        Debug.Log(CombinedPositionList.Count);

    }


    else if (CombinedPositionList.Count != 0 && !IsItemInTheListString
    (CombinedPositionList, item))
    {
        Debug.Log("HHHH"+h+ RoomWallDict.Any(i=>(i.Value[0].Contains
```

```
        (item)&& i.Key!=h)) + "IIIII");
        if (RoomWallDict.Any(i => (i.Value[0].Contains(item)
        && i.Key != h)))
        {
        }


        CombinedPositionList.Add(item);

    }

    else
    {
        Debug.Log(string.Join("\t", item) + "tttt");
        Debug.Log("LLLLL" + RoomWallDict.FirstOrDefault(i => i.Value[0].Contains(item)).Key
        + "DDDDD");
        Debug.Log("JTJTJJTJTJTJTTJTJTJTJTJJTJTJTJJTJTJTJTJJTJTJTJJ");
    }

    (CombinedPositionList.Count != 0 && string.Join("\t", item) == (string.Join("\t", trr)))



    foreach (var itemCombined in CombinedPositionList)
    {

        Debug.Log(h + "aa" + string.Join("\t", itemCombined) + ":::");

        if (itemCombined==item)
        {

        }

    }


}

foreach (var UniqueItem in CombinedPositionList)
{
    if (IsItemInTheListString(SharedWallList, UniqueItem))
    {

        UniqueOuterWallList.Remove(UniqueItem);
        UniqueOuterWallList.Remove( UniqueItem);
```

```
        UniqueOuterWallList.RemoveAll(i => i == UniqueItem);
        UniqueOuterWallList.RemoveAll(i => string.Join("\t", i) == string.Join("\t"
        ,UniqueItem));

}
else
    {
        UniqueOuterWallList.Add(UniqueItem);
    }
}

foreach (var Unique in UniqueOuterWallList.Distinct())
{
    if (IsItemInTheListString(SharedWallList, Unique) || SharedWallList.Contains(Unique))
    {
        UniqueOuterWallList.RemoveAll(i => string.Join("\t", i)
        == string.Join("\t", Unique));
        UniqueOuterWallList.Remove(Unique);
        UniqueOuterWallList.RemoveAll(i => i == Unique);
    }
}
}



using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\Documents
\RoomWallDictTuple.txt", false))
{
foreach (var item in RoomWallDictTuple)

{
stream.WriteLine(item.Key);
foreach (var item2 in item.Value)
{
stream.WriteLine(item2.Item1);
stream.WriteLine(string.Join(";", string.Join(";", item2.Item2)));
}
}

}
CenterPointList.Clear();
using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\
Documents\RoomWallDict.txt", false))
{
```

```
foreach (var item in RoomWallDict)

{

stream.WriteLine(item.Key);


foreach (var item2 in item.Value)

{
if (item2.Count!=0)
{
AverageRoomX = item2.Average(z => (z[2] + z[0]) / 2);
AverageRoomY = item2.Average(z => (z[3] + z[1]) / 2);
CenterPointList.Add(new float[] { item2.Average(z => (z[2]+z[0])/2)
, item2.Average(z => (z[3] + z[1]) / 2),
item2.Min(k=> Math.Min(Math.Abs(k[0]-AverageRoomX),Math.Abs(k[0]
- AverageRoomX))),
item2.Min(k=> Math.Min(Math.Abs(k[0]-AverageRoomY),
Math.Abs(k[0] - AverageRoomY))) });
}
if (RoomWallDict.Any(i => (!i.Value.Equals(item.Value) &&
!i.Value.Contains(item2))))
{
foreach (var item3 in item2)
{

stream.WriteLine(string.Join(";", string.Join(";", item3)));

}
}
}
}

}

using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\
Documents\CenterPointList.txt", false))
{
foreach (var item in CenterPointList)

{
stream.WriteLine(string.Join(";", string.Join(";", item)));
}
}
```

```
using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\
Documents\Combined1.txt", false))
{
foreach (var item in ListCL.ToList())

{
foreach (var item2 in item)
{
stream.WriteLine(string.Join(";", string.Join(";", item2)));

}
}
}
foreach (var item in UniqueOuterWallList.Except(SharedWallList).ToList())
{
if (!IsItemInTheListString(SharedWallList, item) &&
!IsItemInTheListString(UnUniqueOuterWallList, item))
{
UnUniqueOuterWallList.Add(item);
}
}
foreach (var item in DoorList)
{
UniqueOuterWallList.RemoveAll(i => i.Equals(item));
}

using (StreamWriter stream = new StreamWriter(@"C:\Users\makay
\Documents\Unique1.txt", false))
{
foreach (var item in UniqueOuterWallList.Except(DoorList).ToList())

{
if (!IsItemInTheListString(DoorList, item))
{
stream.WriteLine(string.Join(";", string.Join(";", item)));
}
}
}
UnUniqueOuterWallList.Clear();
UnUniqueOuterWallList = UniqueOuterWallList.Except(SharedWallList)
.ToList();
WindowListForShop(UnUniqueOuterWallList);

//checkSameLine(SharedWallListFormatted, SharedWallList);
foreach (var item in SharedWallList.Except(SharedWallCoordList))
```

```
{
if (!IsItemInTheListString(SharedWallCoordList, item) &&
!IsItemInTheListString(UnSharedWallList, item) &&
!IsItemInTheListString(DoorList, item))
{
UnSharedWallList.Add(item);
}
}

using (StreamWriter stream = new StreamWriter(@"C:\Users\makay
\Documents\Shared1.txt", false))
{
foreach (var item in UnSharedWallList.Except(DoorList).ToList())

{
stream.WriteLine(string.Join(";", string.Join(";", item)));
}
}
foreach (var item in SharedWallListFormatted)
{
if (!IsItemInTheListString(UnSharedWallListFormatted, item))
{
UnSharedWallListFormatted.Add(item);
}

foreach (var item in ShopWindowList)
{
if (!IsItemInTheListString(UnShopWindowList, item))
{
UnShopWindowList.Add(item);
}
}

using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\Documents\line3.txt", false))
{
foreach (var item in LineListFormatted)
{
//(IsItemInTheListString(LineListFormatted, item)
stream.WriteLine(string.Join(";", string.Join(";", item)));

}
}
using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\Documents\windowss4.txt", false))
{
int count;
```

```
foreach (var item in ShopWindowList)
{
if (IsItemInTheListString(ShopWindowList, item))
{
stream.WriteLine(string.Join(";", string.Join(";", item)));
}

}
}
using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\Documents\DiscrateOuterWall.txt", false))
{
foreach (var item in ShopOuterWall)

{
stream.WriteLine(string.Join(";", string.Join(";", item)));
}
}
using (StreamWriter stream = new StreamWriter(@"C:\Users\makay\Documents\Cooridor1.txt", false))
{
foreach (var item in SharedWallCoordList)

{
if (!IsItemInTheListString(DoorList, item) && IsItemInTheListString(SharedWallList, item))
{
stream.WriteLine(string.Join(";", string.Join(";", item)));
}
}
}




}
```

EK-2 ÇATI ALGORİTMASI

```
public List<float[]> DivideRoof(List<float[]> RoofPointList, List<List<(float, float)>>
ListCL, CoordReadClass GetCoordsFromFile2, List<float[]> UniqueOuterWallList)
{
bool CheckInside = false;
bool BreakOuter = true;
```

```
bool outside=true;
int PointIterator = 1;
int cntr = 0;

foreach (var item in RoofPointList.Distinct())
{
Debug.Log( string.Join("\t", item));
}
CoordReadClass akka=new CoordReadClass();


foreach (var item in ListCL.ToList())

{
foreach (var item2 in item)
{
if (!IsItemInTheListString(RoofPointList, new float[2] { item2.Item1, item2.Item2 }))
{
RoofPointList.Add(new float[2] { item2.Item1, item2.Item2 });

}

}
}
int iterator = 1;
float diff, diff2, PmaxX, PmaxY, PminX, PminY, RmaxX, RmaxY, RminX, RminY;
Point p1 = new Point(2, 4);
Point p2 = new Point(2, 4);
Point p3 = new Point(2, 4);
Point p4 = new Point(2, 4);

Point pd1 = new Point(2, 4);
Point pd2 = new Point(2, 4);
Point pd3 = new Point(2, 4);
Point pd4 = new Point(2, 4);


Dictionary<float, float> RoofRectangle = new Dictionary<float, float>();


List<float[]> DiagonalPointList = new List<float[]>();
List<float[]> DiagonalPointList2 = new List<float[]>();

List<float[]> InitDiagonalPointList = new List<float[]>();
Dictionary<float, Tuple<float, float[]>> RoofPointListDict =
```

```
new Dictionary<float,Tuple<float, float[]>>();
float max = 0;
float InitDistance;
float MaxKey;
float[] InitArr = new float[4];
//GetCoordsFromFile2.Point a= new GetCoordsFromFile2.Point();
foreach (var item in RoofPointList.Distinct())
{
foreach (var item2 in RoofPointList.Distinct())
{
diff = item[0] - item2[0];
diff2 = item[1] - item2[1];
Debug.Log("Anayasa " + string.Join("\t", item) + " by this " + string.Join("\t", item2));
Debug.Log("aaa" + item[0].GetType() + " kkk" + item2[0]);
InitDistance = Convert.ToSingle(Math.Sqrt(diff * diff + diff2 * diff2));


Debug.Log("Iterator " + iterator);
InitArr = new float[4] { item[0], item[1], item2[0], item2[1] };
string KeyNme = Convert.ToString(item2[0]) + Convert.ToString(item[1]) + Convert.ToString(item[0]) +
Convert.ToString(item2[1]);
if (KeyNme.Length > 6)
{
KeyNme.Remove(0, (KeyNme.Length % 7) + 1);
}
if (KeyNme.Length < 6)
{
KeyNme = KeyNme + '1'+'1';
}
foreach (var itemU in UniqueOuterWallList.Except(DoorList))
{

if (akka.IsIntersect(ConvertPoint(itemU[0], itemU[1]),
ConvertPoint(itemU[2], itemU[3]),
ConvertPoint(item[0], item[1]), ConvertPoint(item2[0], item2[1]))
&& !Enumerable.SequenceEqual(new float[2] { itemU[0], itemU[1] },
new float[2] { item[0], item[1] })
&& !Enumerable.SequenceEqual(new float[2] { itemU[0], itemU[1] },
new float[2] { item2[0], item2[1] })
&& !Enumerable.SequenceEqual(new float[2] { itemU[2], itemU[3] },
new float[2] { item[0], item[1] })
&& !Enumerable.SequenceEqual(new float[2] { itemU[2], itemU[3] },
new float[2] { item2[0], item2[1] })
|| !UniqueOuterWallList.Any(i => akka.onSegment(
ConvertPoint(i[0], i[1]), ConvertPoint(item[0], item[1]),
```

```
ConvertPoint(i[2], i[3])))
|| !UniqueOuterWallList.Any(i => akka.onSegment(
ConvertPoint(i[0], i[1]), ConvertPoint(item2[0], item2[1]),
ConvertPoint(i[2], i[3])))
|| !UniqueOuterWallList.Any(i => akka.onSegment(
ConvertPoint(i[0], i[1]), ConvertPoint(item2[0], item[1]),
ConvertPoint(i[2], i[3])))
|| !UniqueOuterWallList.Any(i => akka.onSegment(
ConvertPoint(i[0], i[1]), ConvertPoint(item[0], item2[1]),
ConvertPoint(i[2], i[3])))


)
{

BreakOuter = false;
}
else
{

}
}

if (BreakOuter && !RoofPointListDict.ContainsKey(
Convert.ToInt32(KeyNme.Substring(0, 6)))
&&  item[0]!= item2[0] && item[1] != item2[1]
)
{

RoofRectangle.Add(Convert.ToInt32(KeyNme.Substring(0, 6)),
InitDistance);
//{ item[0], item[1], item2[0], item2[1] };
RoofPointListDict.Add(Convert.ToInt32(KeyNme.Substring(0, 6)),
new Tuple<float, float[]>(InitDistance,
InitArr.Concat(new float[4] { item2[0], item[1], item[0], item2[1] })
.ToArray()));
}

BreakOuter = true;

if (iterator<RoofPointList.Count-1)
{
iterator++;

}
```

```
}
}
RoofPointListDict = RoofPointListDict.OrderBy(obj => obj.Value.Item1).
ToDictionary(obj => obj.Key, obj =>
obj.Value);
foreach (KeyValuePair<float, Tuple<float, float[]>>
item in RoofPointListDict)
{
p1.x = item.Value.Item2[0];
p1.y = item.Value.Item2[1];
p2.x = item.Value.Item2[2];
p2.y = item.Value.Item2[3];
p3.x = item.Value.Item2[4];
p3.y = item.Value.Item2[5];
p4.x = item.Value.Item2[6];
p4.y = item.Value.Item2[7];

foreach (var ites in RoofPointList)
{
if (akka.IsIntersect(ConvertPoint(ites[0], ites[1]),
ConvertPoint(RoofPointList[PointIterator][0],
RoofPointList[PointIterator][1]),
p1, p2)
)
{
}
if (PointIterator < RoofPointList.Count - 1)
{
PointIterator++;
}
}




if (DiagonalPointList.Count > 0)
{
outside = true;
foreach (KeyValuePair<float, Tuple<float, float[]>>
item5 in RoofPointListDict)
{


PmaxX = new[] { item.Value.Item2[0], item.Value.Item2[2],
```

```
item.Value.Item2[4],
item.Value.Item2[6] }.Max();
PmaxY = new[] { item.Value.Item2[1], item.Value.Item2[3],
item.Value.Item2[5],
item.Value.Item2[7] }.Max();
PminY = new[] { item.Value.Item2[1], item.Value.Item2[3],
item.Value.Item2[5],
item.Value.Item2[7] }.Min();
PminX = new[] { item.Value.Item2[0], item.Value.Item2[2],
item.Value.Item2[4],
item.Value.Item2[6] }.Min();

pd1.x = item5.Value.Item2[0];
pd1.y = item5.Value.Item2[1];
pd2.x = item5.Value.Item2[2];
pd2.y = item5.Value.Item2[3];
pd3.x = item5.Value.Item2[4];
pd3.y = item5.Value.Item2[5];
pd4.x = item5.Value.Item2[6];
pd4.y = item5.Value.Item2[7];

RmaxX = new[] { item5.Value.Item2[0], item5.Value.Item2[2], item5.Value.Item2[4],
item5.Value.Item2[6] }.Max();
RmaxY = new[] { item5.Value.Item2[1], item5.Value.Item2[3], item5.Value.Item2[5],
item5.Value.Item2[7] }.Max();
RminY = new[] { item5.Value.Item2[1], item5.Value.Item2[3], item5.Value.Item2[5],
item5.Value.Item2[7] }.Min();
RminX = new[] { item5.Value.Item2[0], item5.Value.Item2[2], item5.Value.Item2[4],
item5.Value.Item2[6] }.Min();



Debug.Log("emek "+ UniqueOuterWallList.Any(i => akka.onSegment(ConvertPoint(i[0],
i[1]),
ConvertPoint(140,110), ConvertPoint(i[2], i[3]))));

MaxKey = RoofPointListDict.FirstOrDefault(i => i.Value.Item1
.Equals(RoofPointListDict.
Max(k => k.Value.Item1))).Key;

CheckInside = new[] { RoofPointListDict[MaxKey].Item2[0],
RoofPointListDict[MaxKey].Item2[2] }
.Max() >= RmaxX
&& new[] { RoofPointListDict[MaxKey].Item2[0], RoofPointListDict[MaxKey].Item2[2] }
.Min() <= RmaxX
```

```
&& new[] { RoofPointListDict[MaxKey].Item2[1], RoofPointListDict[MaxKey].Item2[3] }
.Min() <= RminY
&& new[] { RoofPointListDict[MaxKey].Item2[1], RoofPointListDict[MaxKey].Item2[3] }
.Max() >= RmaxY;


if (PmaxX >= RmaxX && PminX <= RminX && PmaxY >= RmaxY && PminY <= RminY
)
{

//En uzun k egenlerden ba layarak e er bu k egenleri
//kesen varsa onlar  0'la ve sil!!
if (RoofRectangle[item.Key] != 0 &&
(akka.IsIntersect(ConvertPoint(RoofPointListDict[MaxKey].Item2[0],
RoofPointListDict[MaxKey].Item2[1]), ConvertPoint(RoofPointListDict[MaxKey]
.Item2[2], RoofPointListDict[MaxKey].Item2[3]), pd1, pd2) ||
akka.IsIntersect(ConvertPoint(RoofPointListDict[MaxKey].Item2[0],
RoofPointListDict[MaxKey].Item2[1]), ConvertPoint(RoofPointListDict[MaxKey]
.Item2[2], RoofPointListDict[MaxKey].Item2[3]), pd3, pd4)) &&
!DiagonalPointList.Contains(item.Value.Item2)
&& !RoofPointListDict.Max(i=>i.Value.Item1).Equals(item.Value.Item1)
|| !UniqueOuterWallList.Any(i => akka.onSegment(ConvertPoint(i[0], i[1]), pd1,
ConvertPoint(i[2], i[3])))
|| !UniqueOuterWallList.Any(i => akka.onSegment(ConvertPoint(i[0], i[1]), pd2,
ConvertPoint(i[2], i[3])))
|| !UniqueOuterWallList.Any(i => akka.onSegment(ConvertPoint(i[0], i[1]), pd3,
ConvertPoint(i[2], i[3])))
|| !UniqueOuterWallList.Any(i => akka.onSegment(ConvertPoint(i[0], i[1]), pd4,
ConvertPoint(i[2], i[3])))
|| (CheckInside && !RoofPointListDict.Max(i => i.Value.Item1)
.Equals(item.Value.Item1))
)
{
    RoofRectangle[item5.Key]=0;
}
}

else
{
outside = false;
}

}
if (outside)
{
```

```
DiagonalPointList.Add(item.Value.Item2);

}


InitDiagonalPointList.Clear();
}
else
{
DiagonalPointList.Add(item.Value.Item2);



}
}
foreach (KeyValuePair<float, Tuple<float, float[]>> item4
in RoofPointListDict)
{
Debug.Log("eski " + string.Join("\t", item4.Value.Item2));

if (RoofRectangle[item4.Key]!=0)

{
DiagonalPointList2.Add(item4.Value.Item2);
}
}
return DiagonalPointList2;
}
```

EK-3 SVG AYRIŞTIRMA ALGORİTMASI

```
from lxml import etree
import shutil
import pysvg
import re
from shapely import geometry
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from shapely.ops import unary_union
import shapely.ops as so
import pickle
```

```
import json


def copy_rename(old_file_name, new_file_name): #to make a new file that
    can be manipulated easily
    src_dir = os.curdir
    dst_dir = os.path.join(os.curdir, new_file_name)
    src_file = os.path.join(src_dir, old_file_name)
    shutil.copy(src_file, dst_dir)


tree = etree.parse(open('svg.svg', 'r')) #parser for the svg file

listOfPaths= [] #array holding every paths d value

listOfLines= [] #array holding every paths d value

listOfCurves= [] #array holding every paths c value

listOfPolyLines= [] #array holding every paths d value


for element in tree.iter():

    if element.tag.split('}')[1] == 'path': #checking and taking paths and their values


        listOfPaths.append(element.get("d"))
#print(listOfPaths)

result=[] #list of lists to hold every path and it's d values as a
list that is separated with respect to endpath 'z' value
resultUnmod=[] #same but not separated by z value

for path in listOfPaths: #separate all the paths with z, adding
    z to the ends because split removes z
    text=path
    sep= "z"
    result.append( [x+sep for x in text.split(sep)])
    print("formatted paths are here")
    #print([x+sep for x in text.split(sep)])
    resultUnmod.append(path)

for m in result:
    if "z" in m:
```

```
        m.remove("z")

# #create a copy to test(later deemed unnecessary)

fillList=[] #list that holds the fillers for every shape

for element in tree.iter(): #get the fillers of paths

    if element.tag.split('}')[1] == 'path':

        fillList.append(element.get("fill"))

#understand=re.split('([a-zA-Z])',result[1][1]) #trial code


"""for nice in result[1]:
    data=re.split('([a-zA-Z])', nice) #splits with respect to letters
    so that it can be understood what commands are given in paths
    #and coordinates found accordingly

    if "" in data:
        data.remove("")
    if " " in data:
        data.remove(" ")
    if "   " in data:
        data.remove("   ")

"""
polygon_points=[]


polygons=[]

pol_array=[]

temp_array=[]
cellList=[]
for z in result:
    for t in z:
        data=re.split('([a-zA-Z])', t)#splits the smallest
        lines which start with "M" and end with "z" with respect to letters

        if "" in data:
            data.remove("")
        if " " in data:
```

```
    data.remove(" ")
if "   " in data:
    data.remove("   ")

previousX=""
previousY=""
print("data is here")
print(data)
init_poly=[]
#data=data.replace("P","l")
for d,i in enumerate(data):
#for i in data:
    print(i,"max i is here")
    print(data.index(i),"data index of i is here")
    print(data[d])

    if i=="M" or i=="m":

        indexVal=data.index(i)
        print(data.index(i))
        x=data[indexVal+1].split(",")[0]
        y=data[indexVal+1].split(",")[1]
        init_poly.append(geometry.Point(float(x),float(y)))

        previousX=float(x)
        previousY=float(y)


    elif i=="l":
        indexVal = data.index(i)

        try:
            if("," in data[d + 1]):
                print(data[d + 1])
                x = data[d + 1].split(",")[0]

                y = data[d + 1].split(",")[1]

                if ("-") in x:
                    x = -1*float(x.split("-")[1])

                if ("-") in y:
                    y =-1*float(y.split("-")[1])
```

```
            else:
                print(data[d + 1],"x is here")
                x = data[d + 1].split("-")[0]
                y = data[d + 1].split("-")[1]
                y=-1*float(y)

        except:
            x=re.split(' ', data[d + 1])


        print(x)
        if x != '':
            init_poly.append(geometry.Point(float(previousX)+float(x),float(previousY)
            +float(y)))


            previousX = previousX+float(x)
            previousY = previousY+float(y)
    elif i=="L":
        indexVal = data.index(i)

        try:
            if("," in data[d + 1]):
                x = data[d + 1].split(",")[0]

                y = data[d + 1].split(",")[1]

                if ("-") in x:
                    x = -1*float(x.split("-")[1])

                if ("-") in y:
                    y =-1*float(y.split("-")[1])

            else:
                x = data[d + 1].split("-")[0]
                y = data[d + 1].split("-")[1]
                y=-1*float(y)
                #y = "-"+ data[indexVal + 1].split("-")[1]

        except:
            x=re.split(' ', data[d + 1]) #this part is a
            problem because of how the coordinates are written with no comma
            #and possible multiple minus signs

        print(x)
```

```
    if x != '':
        init_poly.append(geometry.Point(float(x),float(y)))


        previousX =float(x)
        previousY = float(y)

elif i=="c":
    #print(list("44,33 33,44".split(' '))[1])
    indexVal = data.index(i)

    try:
        if("," in data[d + 1]):
            print(data[d + 1])
            x = data[d + 1].split(",")[0]

            y = data[d + 1].split(",")[1]

            if ("-") in x:
                x = -1*float(x.split("-")[1])

            if ("-") in y:
                y =-1*float(y.split("-")[1])

        else:
            x = data[d + 1].split("-")[0]
            y = data[d + 1].split("-")[1]
            y=-1*float(y)
            #y = "-"+ data[indexVal + 1].split("-")[1]

    except:
        x=re.split(' ', data[d + 1])

    print(x)
    if x != '':
        init_poly.append(geometry.Point(float(x),float(y)))


        previousX =float(x)
        previousY = float(y)
    print(x,"x is here")
elif i == "h" :
    indexVal = data.index(i)
    x=data[d + 1]
    if ("-") in x:
```

```
                print(data[d + 1].split("-"))
                x = -1*float(data[d + 1].split("-")[1])


            init_poly.append(geometry.Point(previousX+float(x),float(previousY)))
            previousX=previousX+float(x)
        elif i=="H":
            indexVal = data.index(i)
            x=data[d + 1]
            if ("-") in x:
                print(data[d + 1].split("-"))
                x = -1*float(data[d + 1].split("-")[1])


            init_poly.append(geometry.Point(float(x),float(previousY)))
            previousX=float(x)

        elif i == "v" :
            y=data[d + 1]
            if ("-") in y:
                y = data[d + 1].split("-")
            indexVal = data.index(i)
            y=data[d + 1]
            init_poly.append(geometry.Point(float(previousX),previousY + float(y)))

            previousY = previousY+float(y)
        elif  i=="V":
            y=data[d + 1]
            if ("-") in y:
                y = data[d + 1].split("-")
            indexVal = data.index(i)
            y=data[d + 1]
            init_poly.append(geometry.Point(float(previousX),float(y)))

            previousY=float(y)
        #print(init_poly)
    if(len(init_poly)>2):
        temp_array.append([(p.x, p.y) for p in init_poly])
        pol_array.append(geometry.Polygon([[p.x, p.y] for p in init_poly]))
    #print(geometry.Polygon([[p.x, p.y] for p in init_poly]))

pointList=[]

edgeList=[]
poly = geometry.Polygon([[p.x, p.y] for p in pointList])
```

```
iterVal=0

fig, ax = plt.subplots()

ax.set_ylim(3000, 0)  # decreasing time
ax.set_xlim(0, 3000)  # decreasing time

#unary_union(pol_array)

for i in temp_array:
    print(i)

with open('outfile2.txt', 'w') as fp:
    json.dump(temp_array, fp)
for i in pol_array:
    print(i)

    print("look here")
    plt.plot(*i.exterior.xy)
    xs, ys = i.exterior.xy
    ax.fill(xs, ys, alpha=0.5, fc='r', ec='none')
    new_shape = so.cascaded_union([new_shape,i])
plt.plot(polygon1)

ply2=Polygon([(100,200),(0,50),(0,50),(0,50),(0,50),(110,210),(0,50),
(0,50),(0,50),(0,50),(100,200)])
polygon2 = Polygon([(0,5),                    (1,1),        (3,0),])
polygon3 = Polygon([(0,0),(100,0),(100,100),(90,10)])

plt.plot(*unary_union(pol_array).exterior.xy)

fig, axs = plt.subplots()

ax.set_aspect('equal', 'datalim')


for geom in new_shape.geoms:
    print(geom)
    if not geom:
        print("sad")
    else:
        xs, ys = geom.exterior.xy
        ax.fill(xs, ys, alpha=0.5, fc='r', ec='none')
```

```
plt.show()


print(unary_union(pol_array))
```

KAYNAKLAR

[1] Web. https://blog.sciencemuseum.org.uk/eight-unusual-facts-about-super-mario. **14/7/2022**.

[2] Web. https://www.nintendoblast.com.br/2021/02/analise-super-mario-3d-world-bowsers-fury-review.html. **14/7/2022**.

[3] Web. https://conceptartempire.com/uv-mapping-unwrapping/. **14/5/2022**.

[4] Web. https://editor.method.ac/. **14/5/2022**.

[5] Web. https://en.wikipedia.org/wiki/computer_graphics. **14/5/2022**.

[6] Web. https://en.wikipedia.org/wiki/history_of_video_games. **14/5/2022**.

[7] Web. https://www.netinbag.com/tr/internet/what-is-a-2d-computer-graphic.html. **14/5/2022**.

[8] Web. https://steamdb.info/graph/?tagid=5379. **14/5/2022**.

[9] Anurag Sarkar and Seth Cooper. Towards game design via creative machine learning (gdcml). In *2020 IEEE Conference on Games (CoG)*, pages 744–751. IEEE, **2020**.

[10] Web. https://mobidictum.biz/tr/unity-raycast-yapimi/. **14/5/2022**.

[11] Web. https://docs.unity3d.com/scriptreference/physics.raycast.html. **14/5/2022**.

[12] Web. https://tr.wikipedia.org/wiki/c_sharp. **14/5/2022**.

[13] Web. https://docs.unity3d.com/scriptreference/physics.spherecast.html. **14/5/2022**.

[14] Web. https://karadotgames.com/unity-collider-nedir. **14/5/2022**.

[15] Web. https://akagamess.blogspot.com/2017/12/unity-3d-prefabs-nedir.html. **14/5/2022**.

[16] Web. https://www.fibiler.com/divisions/ehil/mahzen/xml/xmlsimple /txt/html/document_xmlparser.html. **14/5/2022**.

[17] Web. https://developer.mozilla.org/en-us/docs/web/svg/tutorial/paths. **14/5/2022**.

[18] Web. http://haberbin.com/lexical-analysis-nedir. **14/5/2022**.

[19] Web. https://www.scrapingbee.com/blog/data-parsing. **14/5/2022**.

[20] Web. https://shapely.readthedocs.io/en/stable/manual.html. **14/5/2022**.

[21] Corey H Walsh and Sertac Karaman. Cddt: Fast approximate 2d ray casting for accelerated localization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3677–3684. IEEE, **2018**.

[22] Ömer Faruk Çangir. *Yeni Nesil Kinect Kamera ile 3B Ortamlarda Nesne Seçme*. Master's thesis, Hacettepe Üniversitesi Bilişim Enstitüsü.

[23] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33:16351–16361, **2020**.

[24] Khouloud Salameh, Joe Tekli, and Richard Chbeir. Svg-to-rdf image semantization. In *International Conference on Similarity Search and Applications*, pages 214–228. Springer, **2014**.

[25] Mohamad Nurfalihin Mohd. Othman. *Agent-based pathfinding algorithm in partially observable environment using raycasting and navigation Mesh.* Master's thesis, Universiti Teknologi Malaysia, Faculty of Computing.

[26] K Kapetanakis, P Spala, P Sympa, G Mamakis, and AG Malamos. A novel approach in converting svg architectural data to x3d worlds. In *Proceedings of the international conference on telecommunications and multimedia, TEMU*, pages 14–16. **2010**.

[27] Horst Eidenberger. Smil and svg in teaching. In *Internet imaging V*, volume 5304, pages 69–80. SPIE, **2003**.

[28] Sebastiano Battiato, Gianluca Barbera, Gianpiero Di Blasi, Giovanni Gallo, and Giuseppe Messina. Advanced svg triangulation/polygonalization of digital images. In *Internet Imaging VI*, volume 5670, pages 1–11. SPIE, **2005**.

[29] Youngsoo Byun and Bong-Soo Sohn. Abgs: A system for the automatic generation of building information models from two-dimensional cad drawings. *Sustainability*, 12(17):6713, **2020**.

[30] Lucile Gimenez, Sylvain Robert, Frédéric Suard, and Khaldoun Zreik. Automatic reconstruction of 3d building models from scanned 2d floor plans. *Automation in Construction*, 63:48–56, **2016**.

[31] Xuetao Yin, Peter Wonka, and Anshuman Razdan. Generating 3d building models from architectural drawings: A survey. *IEEE computer graphics and applications*, 29(1):20–30, **2008**.