PETRİ AĞLARI KULLANARAK OYUN GELİŞTİRME

GAME DEVELOPMENT USING PETRI NETS

FİKRET AVCI

PROF. DR. HAŞMET GÜRÇAY

Tez Danışmanı

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin

Bilgisayar Grafiği Anabilim Dalı için Öngördüğü

YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

Eylül 2022

Fikret Avcı'ın hazırladığı **"PETRİ AĞLARI KULLANARAK OYUN GELİŞTİRME"** adlı bu çalışma aşağıdaki jüri tarafından BİLGİSAYAR GRAFİĞİ ANABİLİM DALI'nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Dr. Serdar Arıtan
Başkan . . . . . .

Prof. Dr. Haşmet Gürçay
Danışman . . . . . .

Doç. Dr. Emre Sümer
Üye . . . . . .

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,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

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

Fikret Avcı

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.

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

Fikret Avcı

ÖZET

PETRİ AĞLARI KULLANARAK OYUN GELİŞTİRME

Fikret Avcı

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Prof. Dr. Haşmet Gürçay

Eylül 2022, 71 sayfa

Bu tez kapsamında, oyun dinamiklerinin Petri Ağları üzerinde tanımlandığı kullanıcı etkileşimi gerçek zamanlı yapılan bir oyun geliştirilmiştir. Oyunun geliştirilmesinde zamanlı ve rastgele Petri Ağları kullanılmıştır. Böylece, oyun sektöründe kullanılabilecek biçimsel bir modelleme dili tanıtılmıştır. Tanıtılan modelleme dili, basit grafik gösterimi sayesinde geliştiriciler arasındaki iletişimi sağlamak amacıyla kullanılabilir. Ayrıca, Petri Ağlarının sadece modelleme dili olarak değil, aynı zamanda analiz yöntemi olarak da oyun geliştirme alanında kullanılabileceğini göstermek amacıyla, Petri Ağlarına log tutma özelliği eklenmiştir. Kavram ispatı amaçlı geliştirilen oyunda modellenen Petri Ağı, detaylı bir şekilde açıklanmaktadır.

Keywords: Petri Ağları, Oyun Geliştirme, Oyun Analizi, Renklendirilmiş Petri Ağları, Modelleme

ABSTRACT

GAME DEVELOPMENT USING PETRI NETS

Fikret Avcı

Master of Science , Computer Animation and Game Technologies

Supervisor: Prof. Dr. Haşmet Gürçay

September 2022, 71 pages

Within the scope of this thesis, a real-time user interaction game has been developed in which game dynamics are defined on Petri Nets. Time-Augmented and Stochastic Petri Nets has been used in the development of the game. Thus, a formal modeling language that can be used in the game industry has been introduced. The introduced modeling language can be used for communication between game developers due to its simple graphical representation. Furthermore, logging feature has been added to Petri Nets to show that Petri Nets can be used in game development not only as a modeling language but also as an analysis method. The Petri Net, which has been modeled in the game developed for proof of concept, is explained in detail.

Keywords: Petri Nets, Game Development, Game Analysis, Colored Petri Nets, Modelling

İÇİNDEKİLER

Sayfa

TABLOLAR

Sayfa

ŞEKİLLER

Sayfa

ÇEVİRİLER

English	**Türkçe**
Place	Yer
Transition	Geçiş
State	Durum
Token	Jeton
Arc	Yay
Reset Arc	Sıfırlama Yayı
Inhibitor Arc	Durdurucu Yayı
Fire	Yanma
Marking	Durum
Colored Petri Nets	Renklendirilmiş Petri Ağları
Low-level Petri Nets	Düşük seviye Petri Ağları
Initial marking	Başlangıç durumu
Delay	Gecikme
Variable	Değişken
Mutual Exclusion	Karşılıklı Dışlama
Time-Augmented Petri Nets	Zamanlı Petri Ağları
Timed Places Petri Nets	Yer Zamanlı Petri Ağları
Timed Transitions Petri Nets	Geçiş Zamanlı Petri Ağları
Stochastic Petri Nets	Rastgele Petri Ağları
Exponential Distribution	Üstel Dağılım
Hard Coding	Sabit Kodlama

1. GİRİŞ

Oyun sektöründe çeşitli disiplinlerden personeller arasında görsel iletişim çok önemlidir. Bu amaçla UML, akış şeması gibi yöntemler kullanılmaktadır. Ancak, bu yöntemler, eş zamanlı sistemler, detaylı model analizi gibi çeşitli açılardan problem taşımaktadır [5]. Bu nedenle, sistemlerin tasarımı ve analizi için kullanılan bir modelleme dili olan Petri Ağları, kısıtlı sayıda da olsa birçok araştırmacı tarafından oyun geliştirme çalışmalarında kullanılmıştır [5–9]. Fakat, yapılan tüm araştırmalarda Petri Ağları, ya sıra tabanlı basit oyunların geliştirilmesinde kullanılmış, ya da oynanabilir bir oyun geliştirilmeyip oyunun doğrulanması, otomatik kod üretimi gibi kavram ispatı amaçlı çalışmalarda kullanılmıştır ve Petri Ağları kullanılarak, kullanıcı ile gerçek zamanlı etkileşebilen bir oyun geliştirilmemiştir. Ayrıca, yapılan araştırmaların çoğu Petri Ağlarının en basit biçimi olan Düşük Seviye Petri Ağları olup, zamanlama ve rastlantısal özelliklere sahip karmaşık Petri Ağlarının oyun alanında kullanımı ile ilgili çalışmalar da mevcut değildir. Bu tez kapsamında karmaşık özelliklere sahip gelişmiş Petri Ağları kullanılarak geliştirilen bir oyun ile oyun sektöründe kullanılabilecek biçimsel bir modelleme dili tanıtılmıştır.

Petri Ağları yalnızca modelleme amacıyla değil, aynı zamanda sistemlerin analizi amacıyla da kullanılabilir. Yapılan araştırmalarda Petri Ağları üzerinde birçok analiz yöntemleri geliştirilmiştir [1]. Bu tez kapsamında bu yöntemler üzerinde çalışılmasa da Petri Ağlarının analiz kabiliyetinden ilham alınarak, geliştirme sonrası oyunun analizi için Petri Ağlarına log tutma özelliği eklenmiştir.

1.1. TEZİN AMACI

Bu tezde, oyun dinamiklerinin ve oyunun başarılı ya da başarısız bitirilmesi gibi durumların Petri Ağları üzerinde tanımlandığı gerçek zamanlı bir oyun geliştirilmiştir. Oyunda kullanılan Petri Ağları gecikme ve rastgelelik gibi özelliklere sahiptir. Bu sayede, bu gibi özelliklerin ve oyundaki bağlantıların biçimsel gösterimi için bir yöntem ortaya konulmaktadır.

1.2. KATKISI

Bu tez kapsamında, oyun sektöründe kullanılmak için biçimsel bir modelleme dili tanıtılmıştır. Yapılan katkılar aşağıdaki gibi özetlenebilir:

- Önceki çalışmaların aksine, Petri Ağları gerçek zamanlı bir oyunda kullanılmıştır.
- İlk defa, gecikme ve rastgelelik gibi özellikler taşıyan Petri Ağları, oyun tasarımında denenmiştir.
- Oyun içi log dosyası tutma için uygun bir metot tanıtılmıştır.

1.3. ORGANİZASYON

Tezin organizasyonu aşağıdaki gibidir:

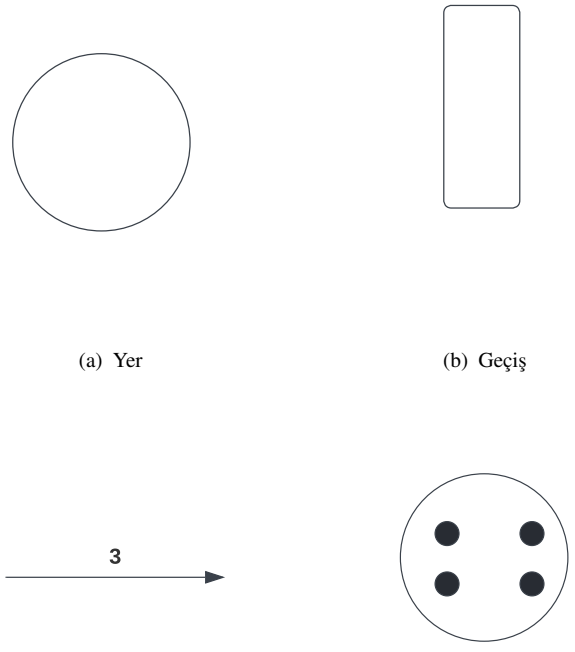
- Bölüm 1'de tezin kapsamı, katkısı, amacı özetlenmiştir.
- Bölüm 2'de veri toplanarak yapılan oyun analizi çalışmaları sunulmuştur.
- Bölüm 3'te Petri Ağlarının tanımı, çeşitli Petri Ağı türleri ve analiz yöntemleri anlatılmıştır.
- Bölüm 4'te Petri Ağlarının oyun sektöründe kullanılan örnekleri verilmiştir.
- Bölüm 5'te tez kapsamında geliştirilen Petri Ağı ve oyunun detayları açıklanmaktadır.
- Bölüm 6'da tezin sonuçları ve gelecek çalışmalar için öneriler verilmektedir.

2. PETRİ AĞLARI

Bir Petri Ağı çok bileşenli sistemlerde ortaya çıkan eşzamanlı süreçlerin tanımlanması ve analizi için kullanılan grafiksel bir araçtır. Petri Ağları, Carl Adam Petri tarafından keşfedildiğinden beri [10], üretim, yazılım, iletişim gibi çeşitli alanlarda sistemlerin modellenmesinde kullanılmaktadır [11]. Aynı zamanda biyoenformatik, kimyasal enformatik gibi bilimsel alanlarda Petri Ağlarının kullanımı için çalışmalar mevcuttur [12]. Petri Ağları modellemesine yönelik CPN Tools, Yasper, Woped gibi çeşitli araçlar ücretsiz olarak kullanıma sunulmuştur [13–15].

2.1. Düşük Seviye Petri Ağları

Petri Ağları, sistemleri dizayn etmek, simülasyonunu yapmak ve doğrulamak için kullanılan grafik modelleme dilidir. Petri Ağları üzerindeki bileşenler ile sistem tanımlanmaktadır. Bu bileşenler yer, geçiş, yay ve jeton olmak üzere dört adettir. Yerler, sistemin içinde bulunduğu durumu temsil eder ve çember ile gösterilir. Geçişler, aksiyon almaya imkan verir ve dikdörtgen simgesiyle gösterilir. Yerleri ve geçişleri birbirine bağlayan yaylar, ok ile ifade edilir ve yaylar, ağırlık ile işaretlenir. Jetonlar, yerler üzerinde bulunur ve sistemin sahip olduğu kaynakları gösterir ve yerler içerisinde küçük bir daire ile tarif edilir [16]. Şekil 2.1'de tüm bu bileşenlerin grafik gösterimi yapılmıştır. Petri Ağları üzerindeki tüm bu bileşenlerin kabiliyetleri, hangi amaçla ve nasıl kullanılacağı ihtiyaca göre uyarlanabilir [7]. Örneğin, 2.2. kısmında anlatıldığı gibi bilgi transferini mümkün kılan Renklendirilmiş Petri Ağları, 2.3. bölümünde açıklanan gecikme parametresinin eklendiği Zamanlı Petri Ağları, 2.4. kısmında anlatılan Petri Ağlarına rastgelelik özelliğinin eklendiği Rastgele Petri Ağları gibi modifikasyonlar mevcuttur.



(a) Yer (b) Geçiş

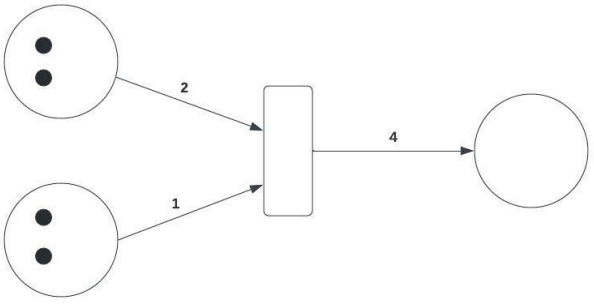
(c) Ağırlığı 3 olan bir yay (d) Yer üzerinde 4 adet jeton

Figure 2.1 Petri Ağı Bileşenleri

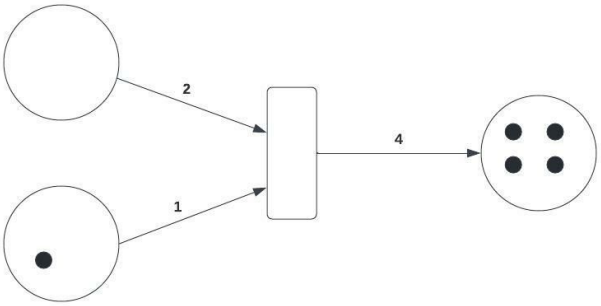
Petri Ağları üzerindeki durum değişimleri geçişlerin gerçekleşmesi ile mümkündür. Petri Ağları üzerinde bir jetonun bir yerden başka bir yere geçmesine yanma adı verilir. Bir yanmanın gerçekleşebilmesi için geçişlerin bağlı olduğu yerlerdeki jeton sayısının bağlı olduğu yayın ağırlığından büyük ya da ağırlığına eşit olması gerekmektedir. Yanma gerçekleştikten sonra girdi yerlerindeki jeton sayısı eksiltilip çıktı yerlerine bağlı olduğu yayın ağırlığı kadar jeton eklenir. Şekil 2.2(a)'de görüleceği üzere yayların ihtiyaç duyduğu jeton sayısı üstten alta doğru 2 ve 1'dir. Ayrıca geçişe bağlı yerlerde 2'şer adet jeton vardır. Bu sayı yayların ihtiyaç duyduğu sayıya eşit ve bu sayıdan büyük olduğundan yanma gerçekleşebilmektedir. Şekil 2.2(b)'de gösterildiği gibi yanma gerçekleştikten sonra girdi yerlerindeki jeton sayısı yay ağırlığı kadar azaltılmıştır, yani üsten alta doğru sırasıyla 2 ve 1 azaltılmıştır. Ayrıca, çıkış yayının ağırlığı kadar jeton bağlı olduğu yere eklenmiştir. Yanma gerçekleştikten sonra çıkış yayının ağırlığı 4 olduğundan 4 adet jeton çıkış yerine eklenmiştir.

Aynı anda birden fazla yerde jeton olmasına imkan verdiği için durum makinesinin aksine paralel çalışabilen eylemlere izin verir. Çünkü Petri Ağlarının üzerinde eş zamanlı geçişler mümkündür [1]. Bu yüzden, Petri Nets eş zamanlılık, senkronizasyon, karşılıklı dışlama ve anlaşmazlık olan sistemler için daha uygundur [17].

Petri Ağlarının bir avantajı grafiksel formel gösterime izin vermesidir. Bu sayede oyun tasarımcısı ile oyun geliştiricisi arasındaki iletişimi kolaylaştırmak için kullanılabilir [7].



(a) Yanmadan önce



(b) Yandıktan sonra

Figure 2.2 Geçiş aşaması

Petri Ağlarının bileşenlerinin isimlendirilmesi İngilizce literatürde farklılık gösterebilmektedir. Tablo 2.1'de Petri Ağlarının genel kullanımdaki bazı isimleri gösterilmiştir. Petri Ağlarının grafik gösteriminin yanında Tablo 2.2'de gösterildiği gibi matematiksel tanımı vardır [1]. Bu matematiksel ifadenin yanında sistemin üzerindeki

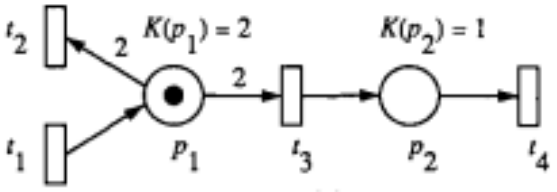


Figure 2.3 Petri Ağı Örneği[1]

yerlerin sahip olduğu jetonlar, Petri Ağının durumunu belirtir [18] ve M fonksiyonu ile gösterilir [1]. M fonksiyonunun $p$ninci elemanı p yerindeki jeton sayısıdır. Şekil 2.3'de gösterilen Petri Ağında bunun bir örneği verilmiştir. Başlangıç durumu $M_0 = (1, 0)$, t_1 geçişi yandıktan sonra $M_1 = (2, 0)$ durumuna geçilir. Bundan sonra, t_2 ve t_3 geçişleri de yandıktan sonra sistem $M_2 = (0, 1)$ durumuna geçiş yapar ve t_4 geçişinin yanmasıyla beraber sistem $M_3 = (0, 0)$ durumuna gelir. Aynı şekil üzerinde gösterilen $K(p)$ fonksiyonları yerlerin kapasitelerini belirtir [1]. Bu tezde matematiksel tanım yerine grafik gösterim kullanılmıştır.

Giriş Yerleri	Geçiş	Çıkış Yerleri
Input places	Transition	Output places
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Task or job	Resources released
Conditions	Clause in logic	Conclusion(s)
Buffers	Processor	Buffers

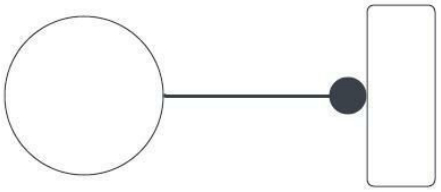
Table 2.1 Geçiş ve Yerlerin Genel Kullanımdaki Bazı İsimleri

Bir Petri Ağı 5 değişkenden oluşmaktadır, $PN = (P, T, F, W, M_0)$:
$P = p_1, p_2, ..., p_m$ yerlerin sonsuz kümesi,
$T = t_1, t_2, ..., t_n$ geçişlerin sonsuz kümesi,
$F \subseteq (P \times T) \cup (T \times P)$ yayların kümesi (akış ilişkisi),
$W : F \rightarrow 1, 2, 3, ...$ ağırlık fonksiyonu,
$M_0 : P \rightarrow 0, 1, 2, 3, ...$ başlangıç durumu,
$P \cap T = \emptyset$ ve $P \cup T \neq \emptyset$.
Başlangıç durumu olmadan N ile simgelenmiş bir Petri Ağı $N = (P, T, F, W)$ ile gösterilir.
Başlangıç durumu verilmiş bir Petri Ağı (N, M_0) ile ifade edilir.

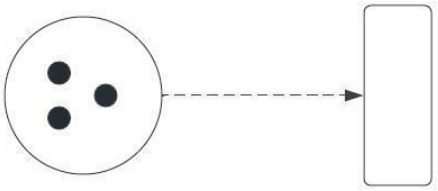
Table 2.2 Petri Ağlarının Biçimsel Tanımı

Petri Ağları yazılımlarının gösterim şekilleri değişiklik gösterse de literatürde daha özel bileşenler de tanımlanmıştır. Durdurucu yayı ve sıfırlama yayı buna örnek olarak

gösterilebilir. Şekil 2.4(a)'deki gibi ucunda bir nokta ile tarif edilen durdurucu yayı, girdi yerinde jeton olmaması halinde aktive olur ve jeton olması halinde yanma gerçekleşmez. Sıfırlama yayı ise, geçiş olduktan sonra ilgili yerdeki tüm jetonları siler [19]. Bu tezde de kullanılan sıfırlama yayı, Şekil 2.4(b)'de gösterildiği gibi kesik çizgi ile gösterilmiştir.



(a) Durdurucu yayı

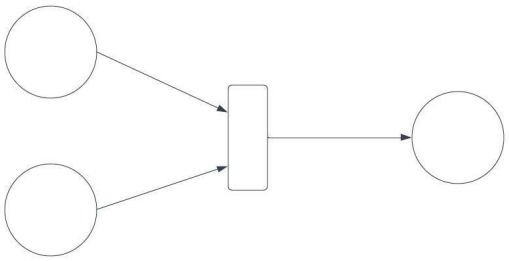


(b) Sıfırlama yayı

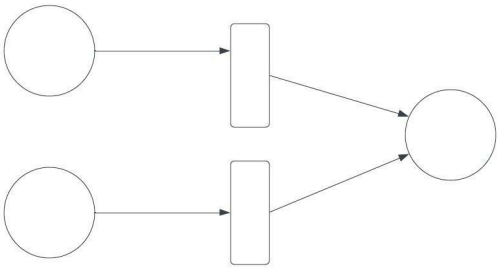
Figure 2.4 Yay türleri

Petri Ağları ile mantıksal kapılar da modellenebilmektedir. Şekil 2.5'de AND, OR, XOR ve NOT kapıları gösterilmiştir. Çeşitli sistemlerde kullanılabilecek Petri Ağları modelleri Şekil 2.6'de belirtilmiştir. Bu konfigürasyonlar sayesinde senkron ya da asenkron iletişim, seçim yapma, karşılıklı dışlama gibi çeşitli sistemler modelenebilmektedir [2]. Şekil 2.6'de "Sequential" olarak belirtilmiş Petri Ağında, eylemler sırasıyla gerçekleşir, "Concurrent" olarak ifade eilmiş Petri Ağında iki farklı eylem, aynı anda birbirinden bağımsız gerçekleşir, "Choise" olarak etiketlenmiş Petri Ağında *P7* yerine bağlı iki farklı çıkış yayı ile kullanıcıdan seçim yapmsı istenebilir, "Synchronous Communication" isimli Ağda iki farklı yerden aynı anda mesaj gelmektedir, "Asynchronous Communication" isimli Ağda tam tersine *T10* geçişinin yanabilmesi için *P24* yerinde jeton olmalıdır, *P24* yerine

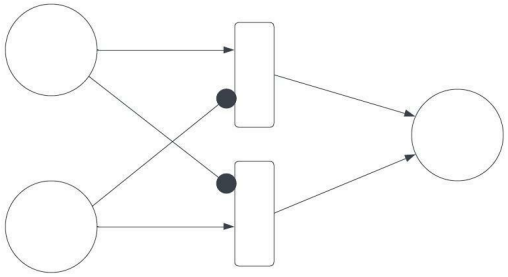
gelecek jetonlar *P20* yerindeki jetonların tüketilmesi ile mümkün olduğu için *P20* ve *P24* yerinden gelen mesajlar eş zamanlı olarak gelemezler, ”Mutual Exclusion” Ağında ise *T11* ve *T13* geçişleri *P25* yeri üzerinde bulunan kaynakları aynı anda tüketememektedir, böylece karşılıklı dışlama sağlanır, ”Fork” olarak gösterilmiş Petri Ağında *T6* geçişiyle beraber kaynaklar dallanma yapılabilmektedir, ”Join” isimli Ağda ise bunun tam tersine kaynaklar geri toplanabilmektedir.



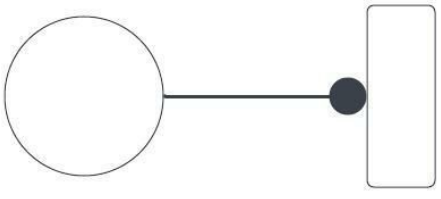
(a) AND-kapısı



(b) OR-kapısı



(c) XOR-kapısı



(d) NOT-kapısı

Figure 2.5 Mantıksal kapılar

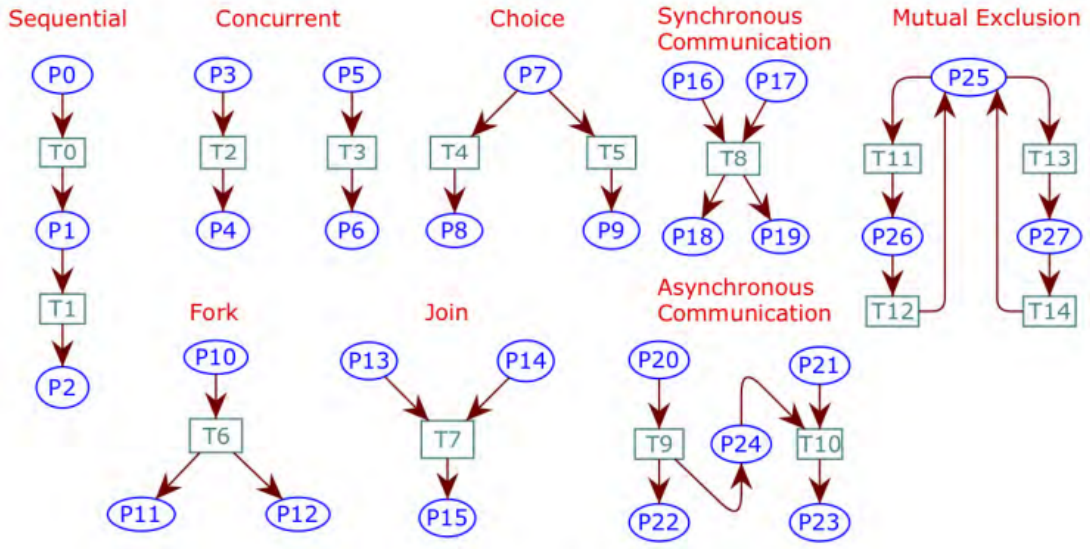


Figure 2.6 Petri Ağları Modelleri [2]

2.2. Renklendirilmiş Petri Ağları

Büyük ölçekli sistemlerin modellenmesi için düşük seviye Petri Ağlarının fazla pratik olmamasından [20] dolayı Jensen tarafından Renklendirilmiş Petri Ağları geliştirilmiştir [16]. CPN kısaltması ile de bilinen Renklendirilmiş Petri Ağları, Düşük seviye Petri Ağlarından farklı olarak jetonlar üzerinde bilgi taşımaya imkan vermektedir. Jensen'in sunduğu Petri Ağında, renklendirilmiş jetonlar bir türe aittir. Bu tür, int, string gibi ilkel türler olacağı gibi bir int ve bir string'ten oluşan daha karmaşık INTxDATA gibi bir tür olarak da tanımlanabilir. Bir yanmanın gerçekleşebilmesi için bağlı olduğu yerlerdeki aynı türe ait jetonların değerinin eşleşmesi gerekmektedir. Şekil 2.7'deki gösterimde başlangıçta 'Sonraki Paket' yerinde değeri '1' olan bir jeton vardır, 'Gönderici' yerinde ise INT değerleri '1'den '7'ye kadar değişen sırasıyla 'M', 'E', 'R', 'H', 'A', 'B', 'A' içeriğindeki DATA değerlerini taşıyan yedi adet jeton vardır. Bu durumda '1' değerleri eşleştiği için ilk başta 'M' verisi gönderilir. Sonrasında 'Sonraki Paket'e gelen 'n' değeri ile jetonun değeri artırılmaktadır ve öteki jetonlar sırasıyla da gönderilmektedir. Ayrıca yayın üzerinde bulunan doğru-yanlış fonksiyonu tanımlanabilmektedir. Bu durumda, yanmanın gerçekleşmesi için ek olarak bu fonksiyonun değerinin de doğru olarak dönmesi zorunludur [16].

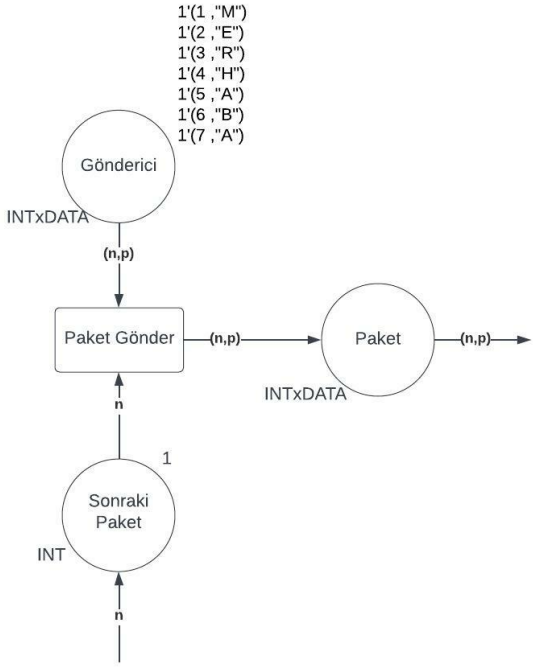


Figure 2.7 Renklendirilmiş Petri Ağları

2.3. Zamanlı Petri Ağları

Petri Ağları üzerinde gecikme uygulamak için zamanlayıcı da yerleştirilebilmektedir. Yere eklenen zamanlanmış jeton sayesinde belirli bir zaman geçmeden, jeton geçişe katılamamaktadır [21]. Şekil 2.8'de, yanmanın gerçekleşmesi için geçmesi gereken minimum ve maksimum süreler geçişler üzerinde gösterilmiştir. Bu durumda, Tablo 2.2'de belirtilen yerlerin sonsuz kümesi $PN = (P, T, F, W, M_0)$ fonksiyonuna $I : T \rightarrow [min, max]$ olmak üzere I fonksiyonu eklenebilir [3].

Zamanlı Petri Ağları, zamanlama özelliği verilen bileşene göre iki şekilde yapılabilir. Eğer jeton belirlenen zaman geçtikten sonra, yere ekleniyorsa Yer Zamanlı Petri Ağları, eğer geçiş etkinleştirildikten sonra yanma için erteleme yapılıyorsa Geçiş Zamanlı Petri Ağları adı verilir. Geçiş Zamanlı Petri Ağları da kendi içerisinde önseçim modeli ve yarış modeli olmak üzere iki modele ayrılmaktadır. Önseçim modelinde, geçiş etkinleştikten sonra yanma olmaksızın girdi yerindeki bütün jetonlar tükekilir. Bu sebeple, jetonlar öteki geçişler tarafından kullanılamamaktadır. Yarış modelinde ise geçiş yanma olmadan jetonlar

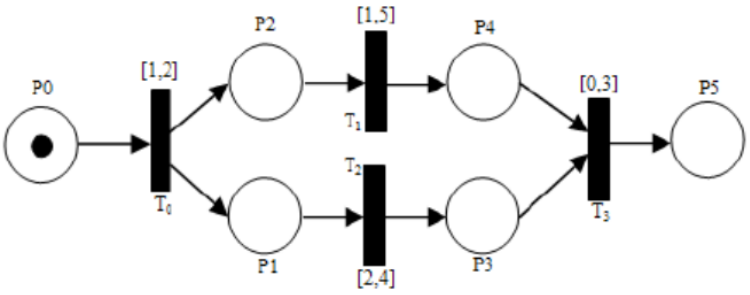
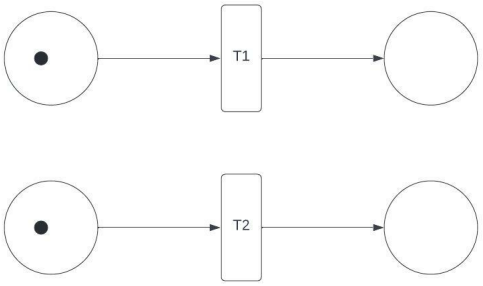


Figure 2.8 Zamanlı Petri Ağı [3]

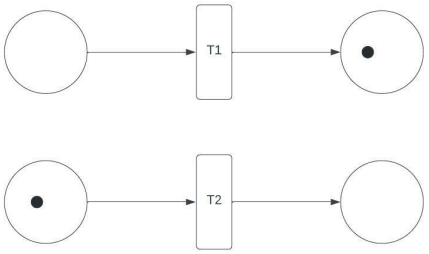
tüketilemez, bu sayede jetonlar başka geçişlerin kullanımına sunulabilir. Bu sayede, ilk yanma gerçekleşen geçiş jetonları kullanır [22].

2.4. Rastgele Petri Ağları

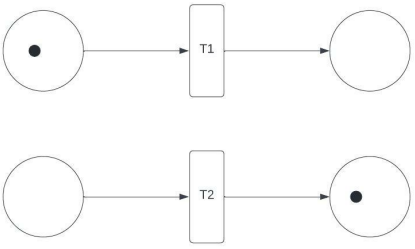
Rastgele Petri Ağları [23, 24], geçişlere rastgele yanma gecikmelerinin verildiği Petri Ağlarıdır [25]. Petri Ağları $PN = (P, T, F, W, M_0)$ fonksiyonuna $\Lambda = (\lambda_1, ..., \lambda_m)$ oran parametresi kümesinin eklenmesiyle Rastgele Petri Ağları elde edilir. λ, yanma zamanının üstel dağılım $F_{x_i}(x) = 1 - e^{-\lambda_i x}$ formülünde kullanılan parametredir ve x_i, t_i geçişinin yanma zamanıdır [22]. Örneğin, başlangıç durumu Şekil 2.9(a) gibi olan bir Rastgele Petri Ağında T1 geçişinin T2 geçişinden önce yanması halinde Şekil 2.9(b)'deki durum, T2 geçişinin T1 geçişinden önce yanması hali ise Şekil 2.9(c)'deki durum oluşmaktadır.



(a) Başlangıç zamanı



(b) $x_1 < x_2$



(c) $x_2 < x_1$

Figure 2.9 Rastgele Petri Ağları

2.5. Petri Ağları Analizi

Petri Ağları için tanımlanmış, bir takım analiz yöntemleri mevcuttur. Bu sayede sistemin belirli şartlar altında istenen davranışı sergileyip sergilemediği, sistemin üzerinde tanımlanmış bir hata olup olmadığı gibi sorulara cevap verilebilir. Petri Ağları analizi için

Erişebilirlik, *Sınırlılık*, *Canlılık*, *Tersine çevirme ve ev durumu*, *Kapsama*, *Kalıcılık*, *Senkron uzaklığı*, *Eşitlik* gibi davranışsal özellikler tanımlanmıştır [1].

2.5.1. Davranışsal Özellikler

Erişebilirlik: Petri Ağı üzerinde verilen M_0 bağlangıç durumundan, M_r durumuna erişilebilen herhangi bir yanma dizisi varsa M_r, M_0'dan erişilebilir olarak belirtilir. M_0'dan erişmeyi mümkün kılan tüm yanma dizileri kümesine *erişilebilirlik kümesi* denir ve $R(M)$ ile gösterilir [26].

Sınırlılık: Petri Ağı üzerindeki her bir yerin alabileceği en fazla jeton sayısı Petri Ağının sınırını belirler. Örneğin, her bir yer en fazla bir token alıyorsa sınırlıdır. Aynı zamanda, sınırsız Ağlar sonsuz bir şekilde jeton bulundurabileceğinden dolayı, bellek sızıntısı, aritmetik taşma gibi sorunlara yol açabilir. Bu yüzden sınırlı Ağlar daha güvenli olarak ifade edilebilir [1].

Canlılık: Canlılık, Petri Ağı üzerindeki bir geçişin hangi sıklıkla yanma gerçekleştirmesidir. Eğer geçişi yakabilecek herhangi bir durum yoksa, geçiş ölü olarak ifade edilir. Geçişin, ne kadar sıklıkla yandığına göre canlılık derecesi verilebilir. Bir t geçişinin canlılık derecesi, yanma sıklığına göre şu şekilde ifade edilir [1]:

LO : Eğer t geçişi hiç bir zaman yanma gerçekleştiremiyorsa *(Ölü)*
$L1$: Eğer t geçişi en az bir kere yanabiliyorsa
$L2$: Eğer verilen herhangi bir k pozitif tam sayısı için, t geçişi en az k kadar yanabiliyorsa
$L3$: t geçişi sonsuz kere yanabiliyorsa
$L4$: t geçişi tüm durumlar için yanma gerçekleştiriyorsa

Table 2.3 Canlılık Seviyeleri [1]

Tersine çevirme ve yuva durumu: Sistem üzerinde başlangıç durumuna geri dönülebiliyorsa tersine çevrilebilir olarak tanımlanır. Tersine çevirme işlemi her zaman başlangıç durumuna gelmek zorunda değildir, sistemin yuva olarak tanımlanan bir duruma geri dönebilir olması da tersine çevrilebilir olarak tanımlanabilir [1].

Kapsama: Eğer Petri Ağı üzerindeki bütün yerler için $M'(p) \geq M(p)$ koşulunu sağlayan herhangi bir $M'(p)$ durumu varsa, $M(p)$ durumu kapsamalı olarak ifade edilir [1]. Bunun anlamı, $M'(p)$ durumunda bütün yerlerdeki jeton sayısı, $M(p)$ durumundaki tüm yerlerdeki jeton sayısından fazla ya da jeton sayısına eşitse $M'(p)$ durumu, $M(p)$ durumunu kapsar.

Kalıcılık: Petri Ağı üzerinde bir geçişin yanması başka bir geçişin yanmasını etkilemiyorsa Ağ kalıcı olarak ifade edilir. Bunun anlamı eğer bir yer birden fazla geçişin girdisi olarak kullanılıyorsa sistem kalıcı değildir [1].

Senkron uzaklığı: Senkron uzaklığı, iki geçiş arasındaki karşılıklı bağımlılık derecesidir [1].

Eşitlik: Bir geçiş yanma gerçekleştirmiyorken ikinci geçişin yanma sayısı sınırlandırılmışsa eşit olarak ifade edilir [1].

2.5.2. Analiz Yöntemleri

Petri Ağları için üç adet analiz yöntemi mevcuttur: Kapsama (Erişebilirlik) ağacı metodu, matris denklemi metodu, küçültme tekniği. Bunlardan ilki olan Kapsama (Erişebilirlik) ağacı metodu için bütün durumların ağaca dahil olması gereklidir, bu yüzden büyük Petri Ağları için ağacın büyümesi söz konusu olduğundan sadece küçük sistemler için kullanışlıdır. Ancak, diğer yöntemler büyük sistemler için de kullanılabilir [1].

Kapsama (Erişebilirlik) Ağacı: Kapsama (Erişebilirlik) Ağacı, başlangıç durumundan yola çıkarak mevcut durumdan, olası geçişler ile oluşacak yeni durumların ağaca eklenmesiyle büyüyen tüm olası durumları kapsayan durum ağacıdır [1].

Sistemin sınırsız olduğu durumda ağaç sonsuz bir şekilde büyüyeceğinden ω sembolü kullanılmaktadır. ω sonsuz olarak düşünülebilir. Kapsama Ağacı Tablo 2.4'de açıklanan algoritma ile oluşturulabilir [1].

Tablo 2.4'de açıklanan kapsama grafiğinin bir örneği, Şekil 2.10'deki Petri Ağı için Şekil 2.11'de çizilmiştir. Şekil 2.10'de, $M_0 = (1, 0, 0)$ başlangıç durumu için t_1 ve t_3 geçişleri

Adım 1: M_0 Başlangıç durumunu kök olarak al ve "yeni" olarak işaretle
Adım 2: "yeni" durumlar var iken aşağıdakileri uygula:
Adım 2.1: Yeni M durumunu seç
Adım 2.2: Eğer M durumunu başka bir durumu ile özdeş ise
M durumununu "eski" olarak işaretle ve bir sonraki yeni duruma geç
Adım 2.3: Eğer M durumunu üzerinde bir geçiş yoksa "ölü-son" olarak işaretle
Adım 2.4: M durumunu üzerinde geçiş var iken aşağıdakileri her bir t geçişi işin uygula:
Adım 2.4.1: M durumundan t geçişi yanmasıyla M' durumununu oluştur
Adım 2.4.2: Eğer sistem üzerinde bütün yerler için $M'(p) \geq M''(p)$ koşulunu sağlayan herhangi bir $M''(p)$ durumu varsa ve $M'(p) \neq M''(p)$ ise
$M'(p) > M''(p)$ koşulunu sağlayan $M'(p)$ durumunu ω ile değiştir
Adım 2.4.3: M' durumunu yeni bir düğüm olarak ağaca ekle,
t geçişi ile M düğümüne bağla ve M' durumunu "yeni" olara işaretle

Table 2.4 Kapsama (Erişebilirlik) Ağacı Algoritması [1]

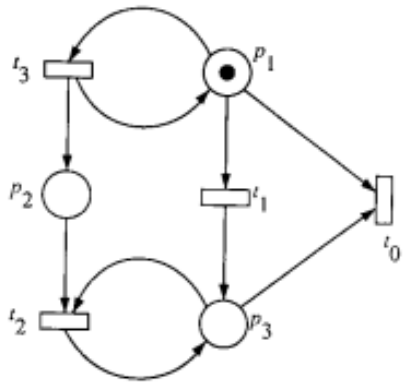


Figure 2.10 Petri Ağı [1]

aktiftir. t_1 geçişi yandığında oluşan $M_1 = (0, 0, 1)$ durumunda hiçbir geçiş mümkün olmadığı için "ölü-son"dur. t_3 geçişinin yanması ile ortaya çıkan $M_3' = (1, 1, 0)$ geçişi, $M_0 = (1, 0, 0)$ geçişini kapsamaktadır. Bu yüzden yeni durum $M_3 = (1, \omega, 0)$ olarak belirtilir ve t_1 ve t_3 geçişleri yine geçişleri aktiftir. t_1 geçişinin yanması $M_4 = (0, \omega, 1)$ geçişini meydana getirir ve t_2 geçişi de artık aktiftir. t_2 geçişinin yanmasıyla oluşan M_5, M_4 ile özdeştir ve eski olarak işaretlenir [1].

Matris Denklemi Metodu: Petri Ağının davranışı, matris denklemleriyle gösterilebilir.

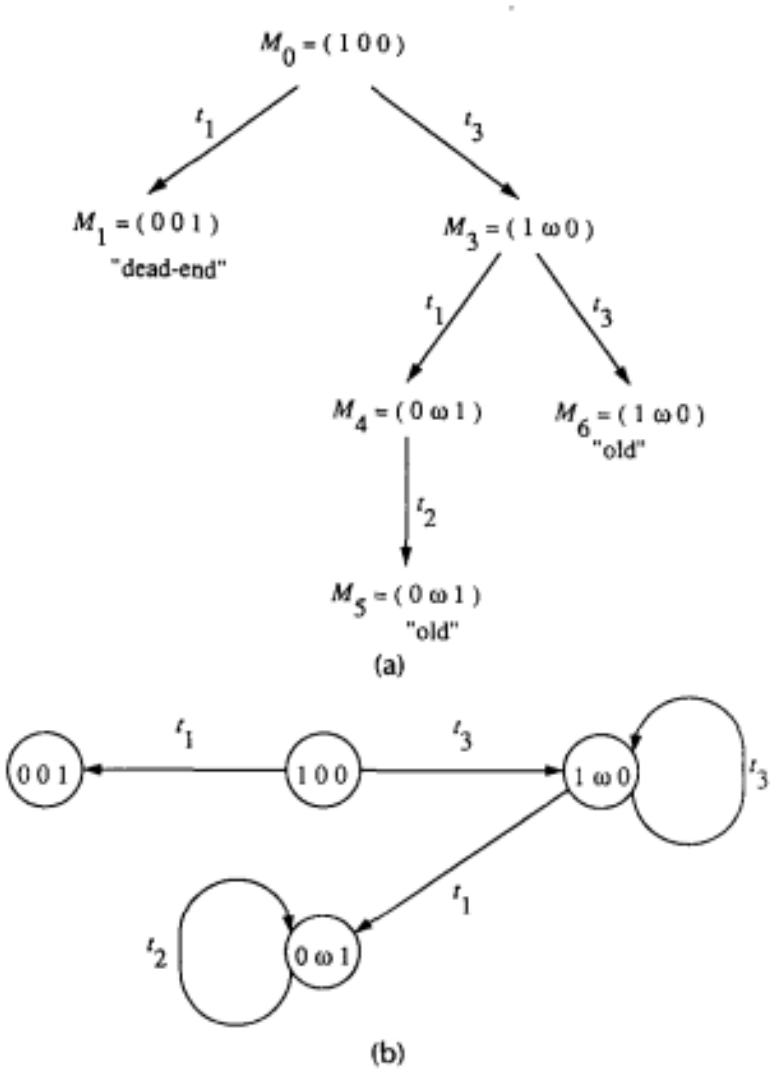


Figure 2.11 Kapsama ağacı ve Kapsama grafiği [1]

İlişki Matrisi: n adet geçişi ve m adet yeri olan bir Petri Ağının nxm'lik A ilişki matrisi aşağıdaki formülle gösterilebilir:

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$$

$a_{ij}^{+} = w(i,j)$ i geçişinden j çıkış yerine bağlanan yayın ağırlığı, $a_{ij}^{-} = w(j,i)$ i geçişinden j giriş yerine bağlanan yayın ağırlığıdır.

Aşağıdaki durum oluştuğunda yanma gerçekleşebilir:

$$a_{ij}^{-} \leq M(j), \quad j = 1, 2, ..., m.$$

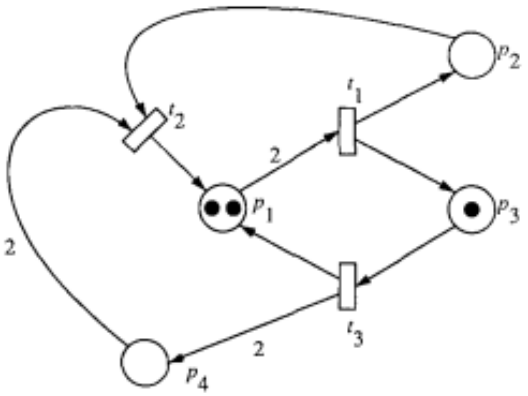


Figure 2.12 Petri Ağı [1]

Durum Denklemi: M_k durum matrisi $mx1$'lik sütun matrisi olarak tanımlanır. M_k'nin j'ninci elemanı k'ninci yanma gerçekleştikten sonraki j yerinde bulunan jeton sayısıdır. Aynı zamanda, u_k kontrol vektörü $nx1$'lik matris olup i'ninci eleman, i geçişinin k'ninci yanmada yanmasıdır. Bu matrisler bir araya getirildiğinde aşağıdaki formül ortaya çıkmaktadır:

$$M_k = M_{k-1} + A^T u_k, \quad k = 1, 2, ...$$

Gerekli Erişebilirlik Matrisi: M_d durumunun, M_0 durumundan $u_1, u_2, ..., u_d$ yanma sırasıyla erişebilir olduğunu varsayarsak aşağıdaki formül yazılabilir:

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k$$

Yukarıdaki formülün çözümü yoksa M_d durumu, M_0 durumundan erişilemez.

Şekil 2.12'de görülen Petri Ağının $M_0 = (2, 0, 1, 0)^T$ durumundan $M_1 = (3, 0, 0, 2)^T$ durumuna geçmesi için $M_d = M_0 + A^T \sum_{k=1}^{d} u_k$ formülü uygulandığında aşağıdaki denklem elde edilir:

$$\begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

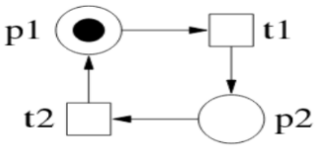


Figure 2.13 Petri Ağı [4]

Yukarıdaki denklemin çözümü ile $\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ elde edilir. Bunun anlamı üçüncü geçiş yandığı zaman M_0 durumundan M_1 durumuna geçilebilir.

Şekil 2.13'de görülen Petri Ağının $M = (1, 1)^T$ geçişi mümkün değildir, çünkü aşağıdaki denklemin çözümü yoktur [4]:

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Küçültme Tekniği: Büyük Petri Ağlarının analizini yapmak için sistemin canlılık, sınırlılık özelliklerini bozmadan, sistem daha basit olacak şekilde küçültülebilir. Şekil 2.14'de küçültme dönüşümlerinin en basit örnekleri gösterilmektedir [1].

Şekil 2.15'deki örnekte soldaki grafik kücültme kurallarıyla sağdaki şekle dönüştürülmüştür [1].

Not: Analiz metotlarıyla kısmında açıklanan davranışsal özelliklerin analizi yapılabilir. Örneğin, sistemin istenen duruma erişebilip erişemeyeceği kontrol edilebilir. Ancak, metotların getirdiği kısıtlar hesaba katılmalıdır. Örneğin, Kapsama (Erişebilirlik) Ağacı tekniğinde ω ile bilgi kaybolduğundan ötürü canlılık özelliği incelenemez [1]. Petri Ağı için kullanılan CPN Tools, AlPiNA, GreatSPN, LoLA, MARCIE, CHARLIE gibi analiz araçları mevcuttur [27].

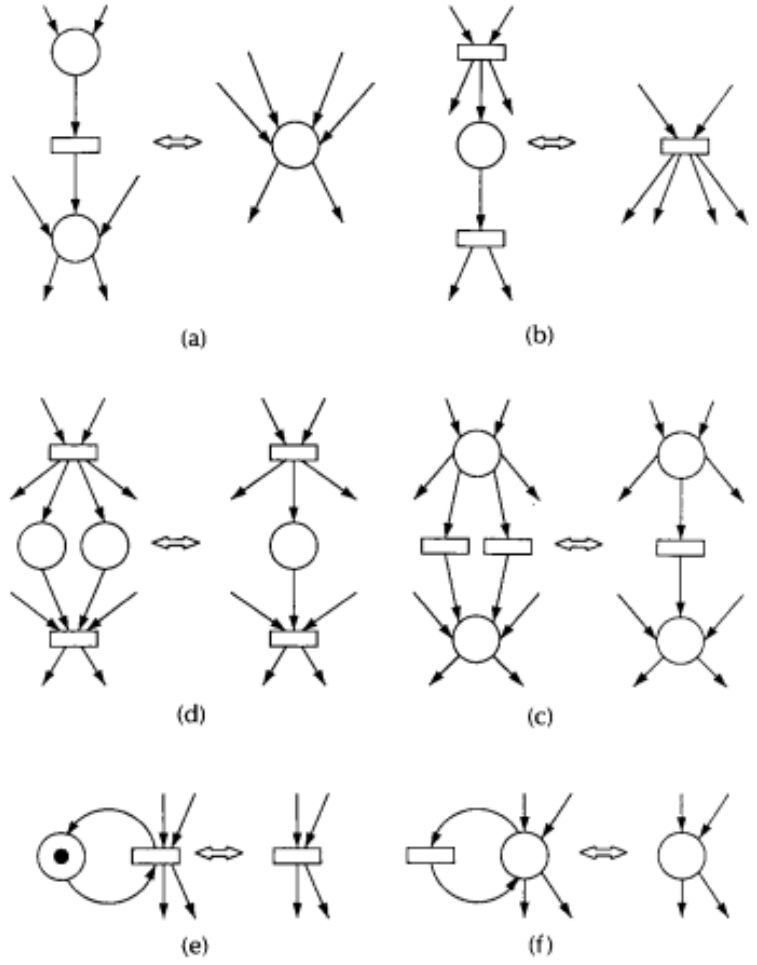


Figure 2.14 Küçültme dönüşümleri [1]

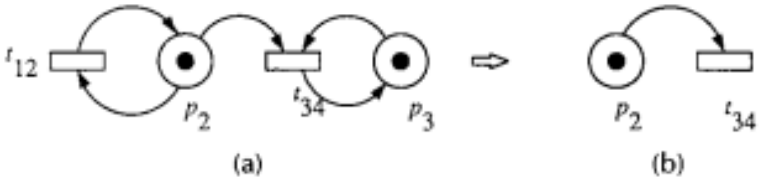


Figure 2.15 Küçültme örneği [1]

3. ALAKALI ÇALIŞMALAR

3.1. Oyun Analizi

Gerçek zamanlı log alma, oyun analizi için kullanılan etkili bir yöntemdir. Özellikle kullanıcı tarafından seçim yapılan durumlarda tasarımcı ya da geliştiriciler tarafından tahmin edilememiş bazı durumlar ortaya çıkabilir veya kullanıcı kuralları ihlal edebilir. Bu tarz tahmin edilemeyen bir durum ya da hata tespiti için log alma, kullanışlı bir metottur. Ayrıca, oyunun farklı parçaları teker teker analiz edilip oyunun daha geniş bir perspektiften bakmaya yardımcı olabilir [28]. Günümüzde oynanan çoğu oyun çevrimiçi olup, geliştiriciler için kullanıcıların tüm hareketlerini kaydetme imkanı vardır. Bu durum, oyuncuların davranışlarının ve ihtiyaçlarının detaylı analizi büyük bir fırsattır [29].

Farooq ve arkadaşları geliştirdikleri mobil oyunda, ivme ölçer ve dokunmatik ekran sensörlerinden gelen veriler ile, oyuncuların oyunu oynarken fiziksel olarak hareketli ya da durağan pozisyonda mı olduklarını ve beceri düzeylerini ölçmüşlerdir. Ek olarak, oyun içi logların, oyuncuların geliştirdikleri stratejiyi yorumlamak için kullanılabileceğini belirtmişlerdir [30]. Bölüm geçme ve oynama zamanı gibi bilgiler, oyuncuların bölüm bitirme zamanlarının, hangi yolları seçtiklerinin ve oyunun verimliliğinin analizi açısından önemlidir. Ayrıca, veri logunun otomatikleştirilmiş halde yorumlanması, özellikle büyük veri kümelerinin olduğu durumlarda hem zaman hem insan maaliyetininin düşürülmesinde oldukça etkilidir [31]. Bertens ve arkadaşları, bir hayatta kalma tahmin etme metodu olan Kaplan-Meier yöntemini kullanarak Age of Ishtaria isimli oyunda, oyuncuların oyunu ne zaman ve hangi seviyeye ulaştıklarında terk edeceklerini tahmin etmeye çalışmışlarıdır. Bu tahmin için satın almalar, toplam oynama zamanı, seviye gibi bilgiler kullanılmıştır ve tıklama sayısı, ilerlenen mesafe, elde edilen deneyim gibi eylem loglarının da kullanılabileceği belirtilmiştir [29].

Oyun verisi toplama ayrıca makine öğrenmesi alanında da kullanılmaktadır. Chimera isimli dijital kart oyununda oyun dengesini sağlamak amacıyla yapay sinir ağı geliştirilmiştir. Bu sayede, oyunun dengesini bozan bir kart olup olmadığı, oyun esnasında yeterli sayıda

stratejinin geliştirilip geliştirilemediği gibi sorunlar tespit edilmiştir ve alınan oyun verileri sayesinde güçlü kartların gücünün azaltılması, oyuncuların başlangıçtaki canının artırılması gibi düzenlemeler yapılmıştır [32]. Bergsten ve Öhman, tezlerinde yapay sinir ağlarını kullanarak devasa çok oyunculu çevrimiçi rol yapma oyunu olan Mortal Online isimli oyunda kaybeden oyunculardan alınan veriler ile oyuncuların ne zaman kaybedeceğini tahmin etmeye çalışmışlardır. Her bir veri tipi için oluşturulan log dosyalarında oyun içi veriler ve kullanıcı girdileri kaydedilmektedir [33]. Hodge ve arkadaşları, DotA 2 oyununda karşılaşmanın sonuç tahminini, oyunun ilk 5 dakikasından alınan gerçek zamanlı verileri, standart makine öğrenmesi modelleri üzerinde kullanarak gerçekleştirmişlerdir. Oyundan alınan gerçek zamanlı veriler öldürme, kule yıkımı, altın ve deneyim kazanımı gibi bilgilerdir [34]. Shen, League of Legends oyununun ilk 10 dakikasından alınan verileri oyunun sonucunun tahmini için kullanmıştır. Bir önceki makalede olduğu gibi iki takımın öldürme, kule yıkımı, altın ve deneyim kazanımı gibi verileri, AdaBoost, GradientBoost, RandomForest, ExtraTree, SVM, Naïve Bayes, KNN, LogisticRegression, DecisionTree gibi farklı makine öğrenmesi algoritmaları ile işlenmiştir [35]. Sevenov ve arkadaşları, DotA 2 oyununun kazananının tahminini, oyundan alınan veri kümesi ile Naive Bayesclassifier, Logistic Regression ve Gradient Boosted Decision Trees yöntemlerini kullanarak gerçekleştirmişlerdir [36].

3.2. Petri Ağlarının Oyunlarda Kullanımı

Petri Ağlarının oyun alanında kullanıldığı örnekler mevcuttur. M. Araújo, L. Roque, Coğrafi Keşifleri simüle eden bir oyun tasarlamışlardır. Kullanıcı, oyuncu olmayan karakterler (NPC) ile Petri Ağları aracılığıla etkileşim kurmaktadır. Örneğin, kullanıcıya dostça, düşmanca, tarafsızca olmak üzere üç adet kara verme imkanı sunulmuştur ve kullanıcının etkileştiği oyuncu olmayan karakterlerin (NPC) de halihazırda aynı şekilde dostça, düşmanca, tarafsızca olmak üzere üç adet tutumu vardır. Kullanıcının verdiği karar ve oyuncu olmayan karakterlerin tutumu Petri Ağlarına girdi olarak verilip, dostça, düşmanca, tarafsızca gibi cevaplar çıktı olarak verilmektedir. Aynı zamanda, korsan ve kaya gibi engelleri girdi olarak kullanan bir ticaret gemisinin limana girmek ya da açık denize

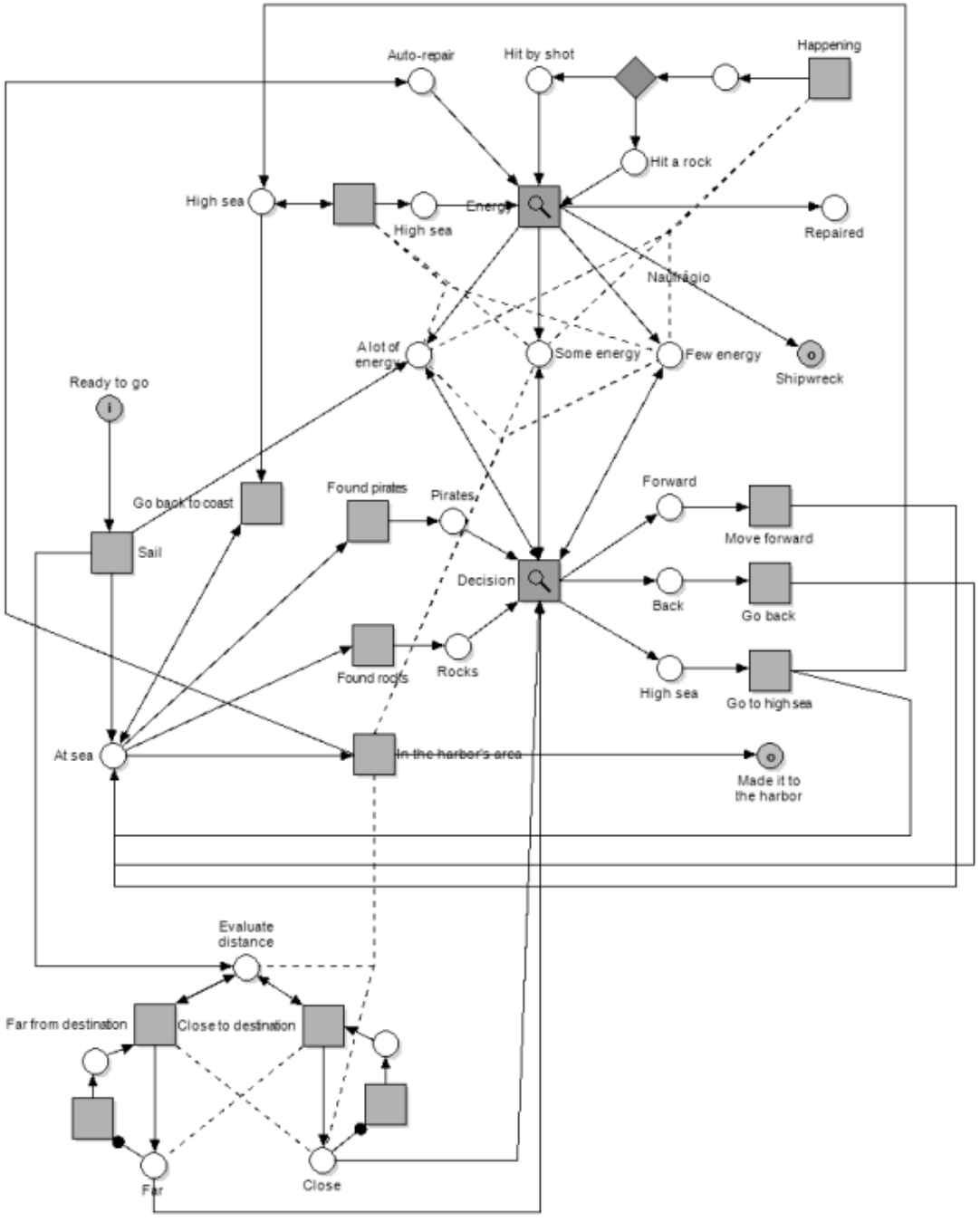


Figure 3.1 M. Araújo, L. Roque tarafından tasarlanan Petri Ağı[5]

geri dönmek gibi kararların algoritması da Petri Ağları ile yapılmıştır. Şekil 3.1'de oyun için tasarlanan Petri Ağı görülmektedir. Oyunda kullanılan modellerle, Petri Ağlarının, doğrulama, simülasyon, oyun dengesi kurma gibi çeşitli açılardan UML, Akış şeması gibi diğer grafik araçlarına kıyasla daha verimli bir yöntem olabileceğini vurgulamaktadır [5].

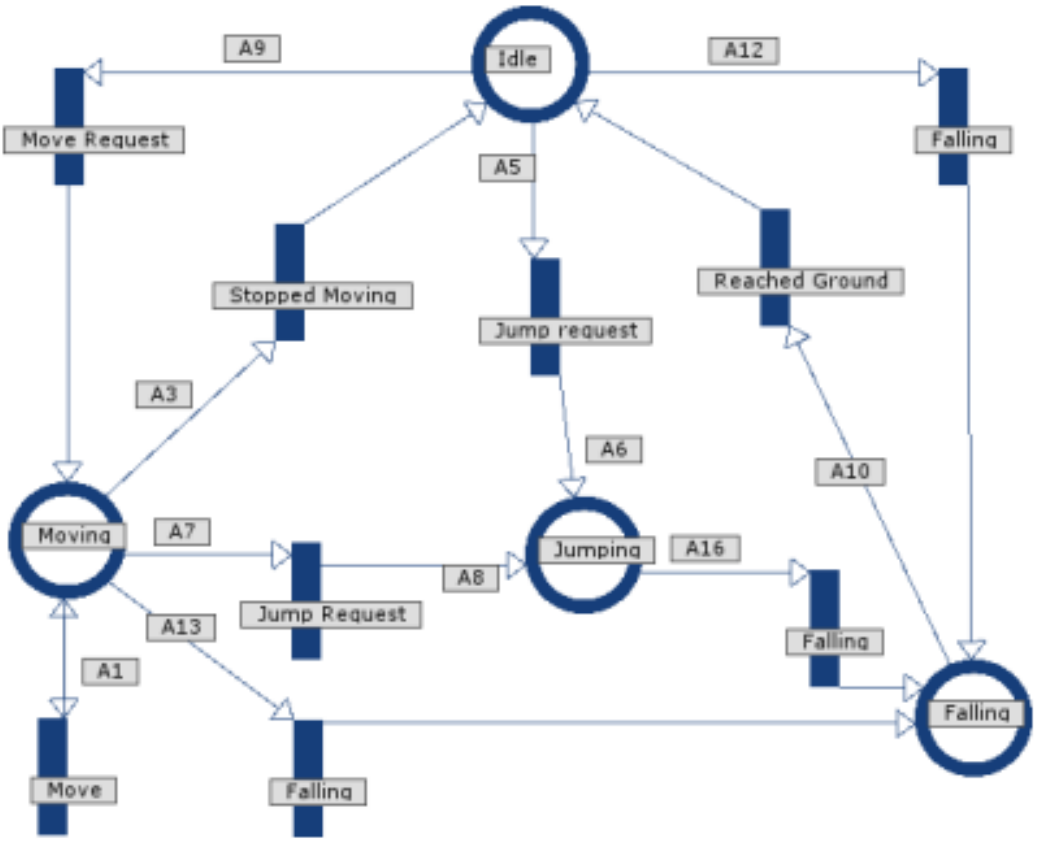


Figure 3.2 Carlsson tarafından tasarlanan karakter kontrol modeli[6]

Carlsson tezinde Petri Ağları kullanarak Unity3D üzerinde biri Petri Ağları ağı tanımlamaya diğeri otomatik kod üretmeye yarayan iki adet editör geliştirmiştir. Ancak, Petri Ağları üzerinde sadece jeton bulunmaktadır. Bunun anlamı, durum makinesi mantığındaki gibi jeton aynı anda yalnızca bir durumda kalacağından dolayı Petri Ağlarının en önemli özellikleri olan jetonlar ile paralel zamanlılık sağlanmamıştır. Ayrıca, geleneksel durum makinesinin gerektirdiği gibi, geçişler bir giriş ve bir çıkış yayı ile sınırlandırılmıştır. Bu yüzden, üretilen kod, durum makinesinden pek farklı değildir. Buna ek olarak, dizayn edilen dört adet model ile otomatik kod üretilmesine rağmen oynanabilir düzeyde bir oyun geliştirilmemiştir. Şekil 3.2'de tasarlanan modellerden biri olan bir karakter kontrol modeli ve Şekil 3.3'de bu modelin kodu gösterilmiştir. Tüm bunlara rağmen Petri Ağları benzeri bir ağ üzerinde otomatik kod üretme, hem oyun geliştirme esnasında zaman kazanımı hem de test edilebilirlik açılarından ötürü değerli bir fikir olarak göze çarpmaktadır [6].


```
Arc 1:  rb.velocity =new Vector2(Input.GetAxis("Horizontal")
*  movementSpeed * Time.fixedDeltaTime, rb.velocity.y);
Arc 3:  if (Input.GetAxis("Horizontal") == 0)
Arc 5:  if (Input.GetKeyDown(KeyCode.Space))
Arc 6:  rb.AddForce(new Vector2(0, 1) * jumpHeight);
Arc 7:  if (Input.GetKeyDown(KeyCode.Space))
Arc 8:  rb.AddForce(new Vector2(0, 1) * jumpHeight);
Arc 9:  if (Input.GetAxis("Horizontal") != 0)
Arc 10:  if (Physics2D.OverlapBox(
            rb.position + groundBox.offset,
            groundBox.size, 0, layerMask) != null)
Arc 12:  if (rb.velocity.y < -0.1f)
Arc 13:  if (rb.velocity.y < -0.1f)
Arc 16:  if (rb.velocity.y < 0)
```

Figure 3.3 Model kodu[6]

Brom ve arkadaşları tarafından geliştirilen eğitim amaçlı ”Europe 2045” isimli bir ’ciddi’ oyun bulunmaktadır. Ekonomi, politik, medya araştırmaları alanlarında eğitim vermeyi amaçlayan online multi player oyunun hikaye anlatımı kendileri tarafından modifiye edilen Petri Ağları kullanılarak yapılmaktadır. Sıra tabanlı olan bu oyunda oyuncuların karar almalarını yönetmek amacıyla oy kullanabildikleri ”ballot” isminde geçiş eklenmiştir. Kullanıcıların kullandıkları oya göre bağlı yerlere jeton eklenmektedir. Jetonlar da ”age” olarak belirtilen sıra bilgisine sahiptir ve bu özelliğiyle renklendirilmiş olarak tanımlanabilir. Jetonlar, yerlere yerleştiklerinden itibaren geçen sıra bilgisini tutarlar ve belirlenen sıra sayısı dolmadan tetiklenmezler. Geçiş ile jeton kazanan yerlere ek olarak ilgili sıra geldiğinde jeton eklenen (token-generating trigger) yerler de tanımlanmıştır. Aynı zamanda, jeton ile tetiklenen klasik tetiklemeler ile birlikte bir geçişte birden fazla yer bağlı olması durumunda çıkan çakışmayı çözmek için tetiklenen özel tetiklemeler tanımlanmıştır, ancak tasarlanan oyunda bu özellik kullanılmamıştır. Şekil 3.4’de Brom ve arkadaşları tarafından tasarlanan Petri Ağında kullanılan bileşen sembolleri, Şekil 3.5’de ise geliştirilen oyunun bir sahnesinde

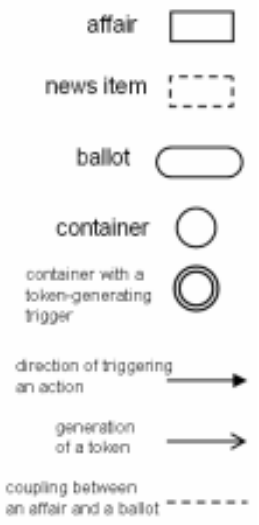


Figure 3.4 Bileşen sembolleri [7]

kullanılan Petri Ağı gösterilmektedir. Geliştirilen oyundan bir kare, şekil 3.6'de görülebilir [7].

Sinclair ve arkadaşları geliştirdikleri ciddi oyunun doğrulamasını, oyuna entegre ettikleri Petri Ağları ile gerçekleştirmişlerdir. CPN Tools ile tasarlanan oyun, Unity 3D ortamında geliştirilmiştir ve CPN Tools ile iletişim halindedir. Müze ziyaretçilerinin kullanıcı deneyimlerini artırmaya yönelik geliştirilen bu oyun, Petri Ağlarından aldığı emirleri uygulayıp, Petri Ağlarına geri bildirimde bulunmaktadır. Bu sayede, oyunun tasarlanan davranışı sergileyip sergilemediği yine Petri Ağları ile kontrol edilmektedir. Şekil 3.7'de geliştirilen oyundan bir görüntü verilmiştir [8].

Moh. Syufagi ve arkadaşları oyuncuların motivasyonlarını ölçmek için geliştirdikleri ciddi oyunda Petri Ağlarını kullanmışlardır. Motivasyon davranışı ölçümü için kullanılan yapay sinir ağları, Petri Ağları ile modellenmiştir. Şekil 3.8'de Moh. Syufagi ve arkadaşları tarafından modellenen Petri Ağı, Şekil 3.9'de ise Ağ üzerindeki geçiş ve yerlerin anlamları gösterilmektedir. Şekil 3.8'de belirtildiği gibi, veriler hazırlandıktan sonra, P302, P303, P304 yerlerindeki sinir ağlarına girmektedir. Bu yerlerin çıkışı ise oyuncunun sahip olduğu motivasyondur [9].

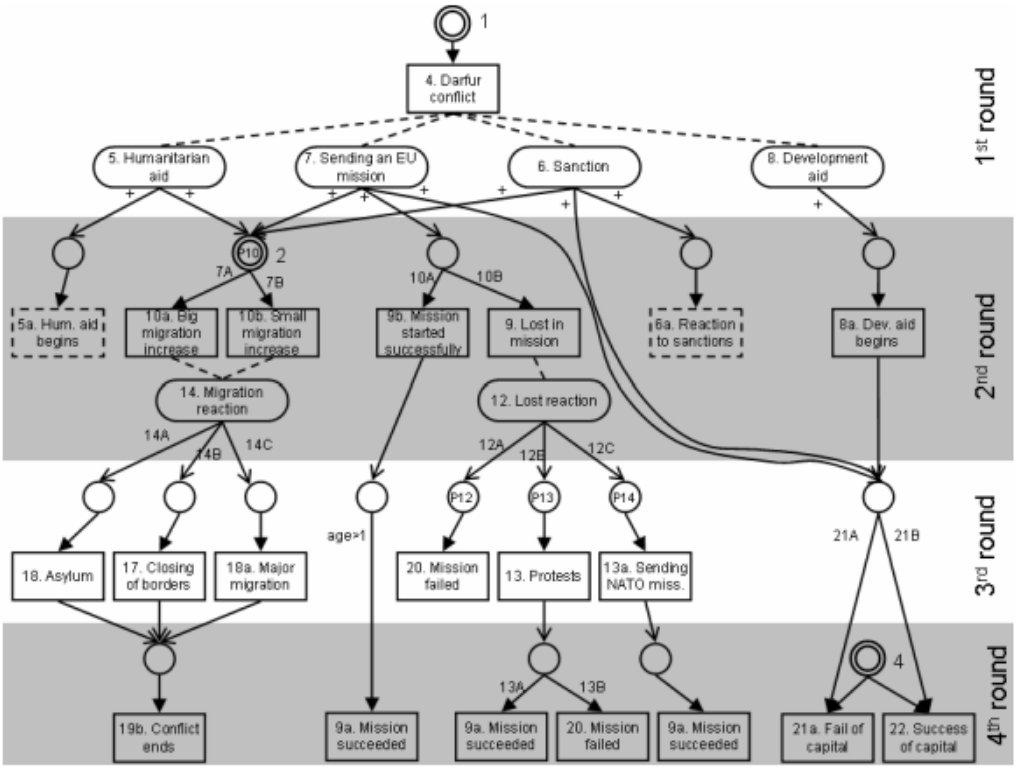


Figure 3.5 Brom ve arkadaşları tarafından modellenen Petri Ağı [7]

Figure 3.6 Brom ve arkadaşları tarafından geliştirilen oyundan kareler [7]

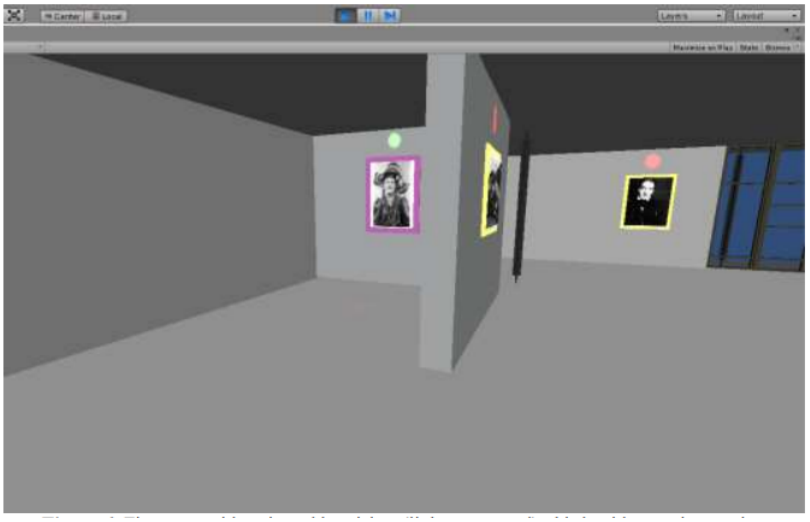


Figure 3.7 Sinclair ve arkadaşları tarafından geliştirilen oyundan bir sahne [8]

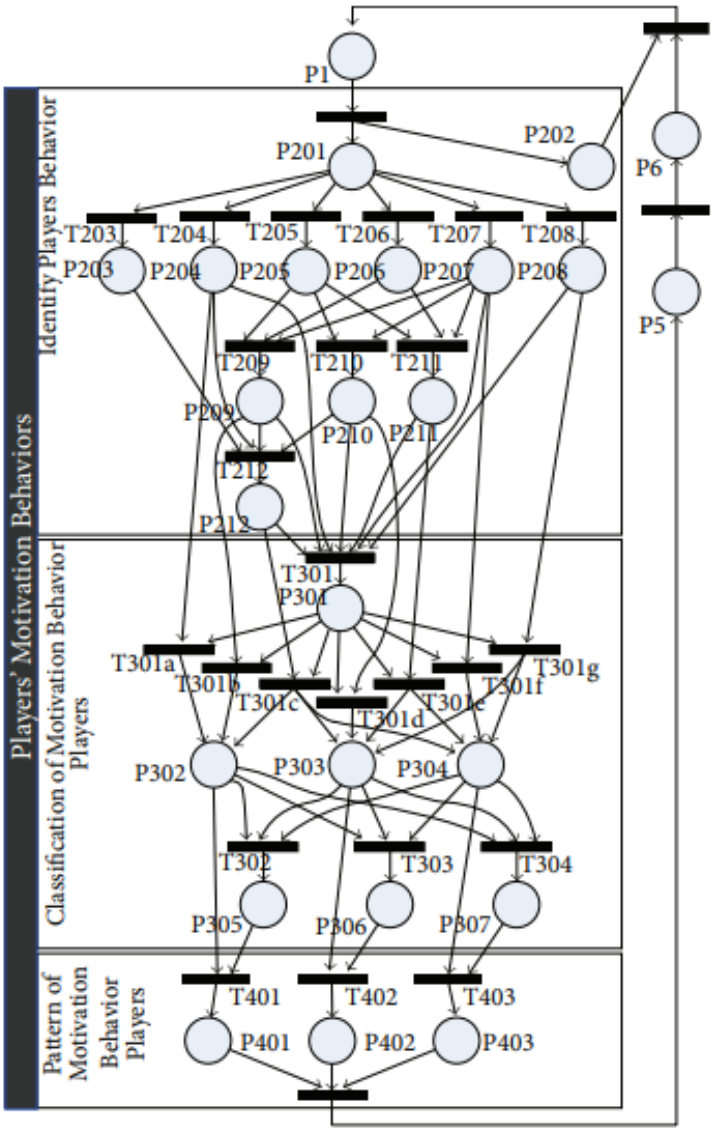


FIGURE 2: Details of Motivation Behavior Game Model.

Figure 3.8 Moh. Syufagi ve arkadaşları tarafından modellenen Petri Ağı [9]

Table 5: Details of transition on Motivation Behavior Game Model.

Transition	Interpretation
T203	The result of the frequency of neglected in tests or avoided in games
T204	The result of how many times the players search info
T205	The result of wrong in tests or losses in games
T206	The result of the players is uncertainty (cancel)/to decline (escape)
T207	The result of true/win
T208	The count of how much the players use time to finish the job
T209	Average of loss (m), cancel (c), and victory (b) values
T210	Average of loss (m) and victory (b) values
T211	Sum of 30% loss (m), 20% cancel (c), and 50% victory (b) values
T212	Average of avoiding (o), searching info (i), picking question (q), and trying to answer (tr) values
T301	Obtain the highest value of the m, b, c, q, or tr (max. value)
T301a	Divide the i value by the max
T301b	Divide the q value by the max
T301c	Divide the st value by the max
T301d	Divide the tr value by the max
T301e	Divide the e value by the max
T301f	Divide the b value by the max
T301g	Divide the t value by the max
T302	Set one value if then value of high active choice (ac_3) in LVQ method is higher than value of high persistence (ps_3) or value of high mental effort (me_3), or else reset (zero value)
T303	Set one value if then value of high persistence (ps_3) in LVQ method is higher than value of active choice (ac_3) or value of high mental effort (me_3), or else reset (zero value)
T304	Set one value if then value of high mental effort (me_3) in LVQ method is higher than value of high persistence (ps_3) or value of active choice (ac_3), or else reset (zero value)
T401	To multiply
T402	To multiply
T403	To multiply

Table 4: Details of place on Motivation Behavior Game Model.

Place	Interpretation
P1	Problems arise in the game
P201	Players resolve the problem
P202	Players avoid/leave the problem
P203	Number of how many times the players neglected the tests or avoided the games (o)
P204	Number of how many times the players search info (i)
P205	Number of wrong in tests or losses in games (m)
P206	Number of the players is uncertainty/to decline (escape) (c)
P207	Number of True in games or win in games (b)
P208	Number of how much the players use time to finish the job (t)
P209	Fixes the value of pick question/playing the game (q)
P210	Fixes the value of trying to answer/to finish (tr)
P211	Fixes the value of self-efficacy/Ability (e)
P212	Step report of player at some stage (st)
P301	Fixes the value of maximal (max)
P302	LVQ method to classify the players motivation behavior of active choice (ac) into; low active choice (ac_1), semi-active choice (ac_2), or high active choice (ac_3)
P303	LVQ method to classify the players motivation behavior of Persistence (ps) into low persistence (ps_1), semi-persistence (ps_2), or high persistence (ps_3)
P304	LVQ method to classify the players motivation behavior of mental effort (me) into low mental effort (me_1), semi-mental effort (me_2), or high mental effort (me_3)
P305	Value is one or zero
P306	Value is one or zero
P307	Value is one or zero
P401	Value is active choice (ac) or zero
P402	Value is persistence (ps) or zero
P403	Value is mental effort (me) or zero
P5	Motivation leveling algorithm
P6	Responds to the players level of motivation behavior as the reference to selection of problem in game

Figure 3.9 Geçiş ve yerlerin anlamları [9]

4. YÖNTEM

Bir arcade video oyunu olan Road Fighter[37] oyununun uyarlaması, Petri Ağları kullanılarak Unity 3D oyun motorunda geliştirilmiştir. Düz bir yol üzerinde oynanan oyunda amaç, araçlara ve yan bariyerlere çarpmadan bitiş çizgisini geçmektir. Aynı zamanda aracın benzini de zamanla bitmektedir, bu yüzden yol üzerinde giden benzin istasyonlarına dokunarak benzin doldurulur. Oyun başladığında 'e' tuşuna basarak motor çalıştırılır, aynı şekilde 'q' tuşuna basarak motor durdurulur. Motor çalışır haldeyken 'w' tuşuna basarak araç hızlandırılır ve 's' tuşuna basarak araç yavaşlatılır. Araç, 'a' ve 'd' tuşları ile sırasıyla sola ve sağa kaydırılır, "space" tuşuyla ani olarak durdurulur. Benzin bitmeden benzin istasyonlarına dokunarak ve engellerden kaçınarak bitiş çizgisi geçildiğinde oyun bitmiş olur. Bunlara ek olarak, yol üzerinde buzlanma ve çamur bulunmaktadır. Araç buzlu yolla temas ettiğinde sağa ya da sola doğru kaydıktan sonra kendini düzeltmektedir. Çamurun üzerinden geçtiğinde araç, yavaşlamaktadır. Ayrıca, yol üzerinde temas edildiğinde kalkan görevi gören araçlar vardır. Bu araçlarla temas ettikten sonra belirli bir süre boyunca yoldaki diğer araçlara çarpılsa bile "game over" olunmamaktadır. Böylece bir kış ayının mevsimsellik etkisi Petri Ağları ile modellenerek oyuncu ile gerçek zamanlı etkileşebilen bir oyun tasarlanmıştır. Oyundan alınan ekran görüntüleri resim 4.1'de görülebilir.

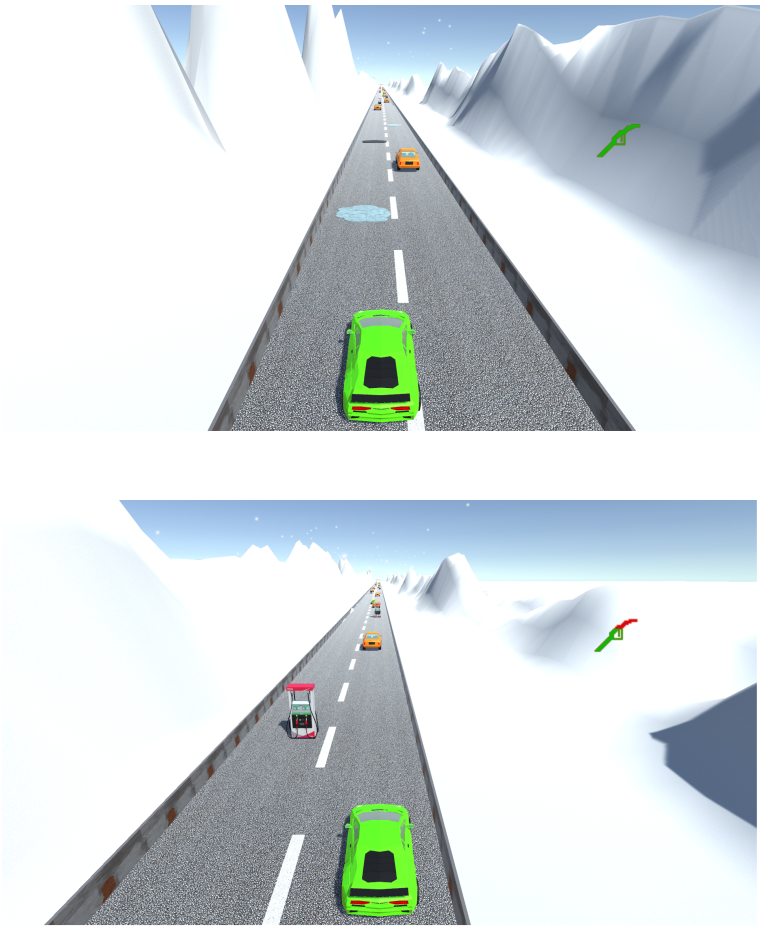


Figure 4.1 Oyundan görüntüler

4.1. Petri Ağları

Oyun için tasarlanan Petri Ağları geçiş, yer, jeton, giriş yayı ve çıkış yaylarından oluşmaktadır. Jetonlar üzerinde tür bilgisini taşıdığından dolayı Petri Ağı, Renklendirilmiştir. Ayrıca, gecikme, rastlantısallık gibi özellikler de Petri Ağı üzerinde tanımlanmaktadır.

4.1.1. Yerler ve Jetonlar

Yerlerin özel bir kabiliyeti olmayıp sadece jeton bilgisini taşımaktadır. Jetonlar, *State*, *Energy* ve *Objects_Move*, *Ice_Slip_Left*, *Ice_Slip_Right*, *Protector* olmak üzere altı türdedir.

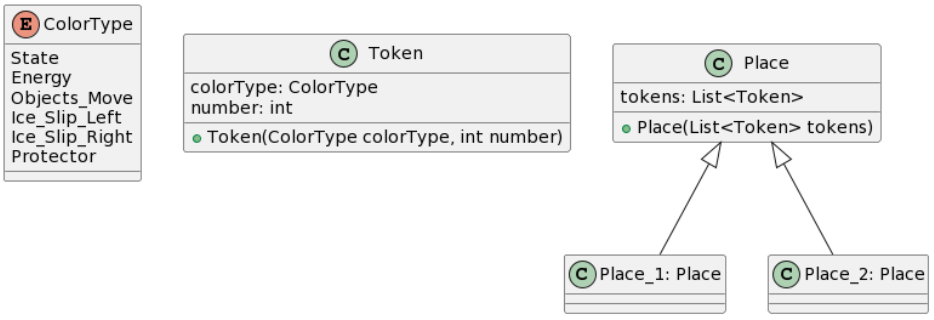


Figure 4.2 Yer ve jeton sınıf şeması

Giriş ve çıkış yayları da, aynı şekilde bu türlerden birine aittir. Şekil 4.3'de görüleceği üzere, Petri Ağı üzerinde durum türü mavi, enerji türü kırmızı, *Objects_Move* türü ise sarı ile gösterilmiştir. *Ice_Slip_Left* ve *Ice_Slip_Right* türleri şekil 4.9'de gösterildiği gibi sırasıyla yeşil ve mor renktedir. *Protector* jetonu ise şekil 4.7'de belirtildiği gibi turuncu renktedir. *State* jetonu, genel olarak aracın hareket etmesi, kullanıcı girdisi, oyunun bitişi gibi durumlar için kullanılmaktadır. *Energy* jetonu aracın benzin durumu için kullanılmaktadır. *Objects_Move* jetonu ise oyundaki engel yaratan araçların, benzin istasyonlarının ve kalkan görevi gören araçların sabit hızda hareketini sağlar. *Ice_Slip_Left* ve *Ice_Slip_Right* jetonları ise araç buzlu yüzey ile temas ettikten sonra sırasıyla sola doğru ve sağa doğru kayıp tekrardan toparlamasını sağlar. *Protector* türündeki jeton koruyucu araçlarla temas ettikten sonra belirli bir süre boyunca araca kalkan olma görevi için kullanılır. Şekil 4.2'de gösterildiği gibi sistem üzerinde tanımlanan yerler, "Place" sınıfından türeyip üzerinde renk bilgisi de bulunan jetonları barındırmaktadır. Jetonlar, "Token" sınıfı ile belirtilmiştir.

4.1.2. Giriş yayları

Giriş yaylarına türle birlikte koşul fonksiyonu tanımlanabilmektedir. Şekil 4.3'de soldaki yayın türü mavidir ve bir koşul tanımlanmamıştır. Sağdaki yayın türü ise kırmızıdır ve *if (Input.GetKeyDown("e"))* koşulu tanımlanmıştır. Bir yanmanın gerçekleşebilmesi için bağlı olan tüm giriş yaylarının karşılaması gereken iki şart vardır. Öncelikle giriş yayında tanımlanan koşul varsa koşulun sağlamalıdır, ardından bağlı olduğu yerde kendisiyle aynı türden yeterli sayıda jeton olmalıdır. Örneğin, Şekil 4.3'de soldaki mavi türündeki yayın

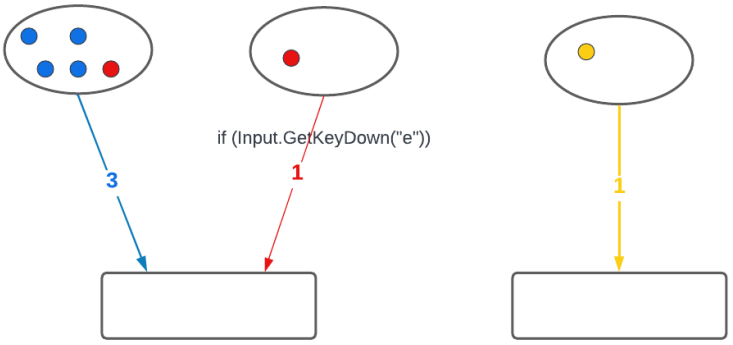


Figure 4.3 Giriş yayları

yanma işlemine geçebilmesi için bağlandığı yerden 3 adet mavi türünden jeton tüketmesi gerekmektedir. Aynı anda, sağdaki kırmızı türüne ait yayın ise hem kırmızı türünden bir jetonu tüketmesi hem de tanımlanan koşulun sağlanması gereklidir. Bu iki yayın şartları karşılandığı taktirde yanma gerçekleşebilmektedir. Şekil 4.5'de belirtildiği üzere, "IncomingArc" olarak ifade edilen giriş yayları için tür ve jeton sayıları tanımlanmaktadır ve bir girdi yerine bağlıdır. Ayrıca, "IncomingArc"tan türeyen her bir giriş yayı nesnesi için doğru-yanlış bilgisini döndüren "condition()" fonksiyonu tanımlanmalıdır. Örneğin, şekil 4.3'de *if (Input.GetKeyDown("e"))* koşulu "condition()" fonksiyonu içinde tanımlanmıştır.

Giriş yayının bir türü olarak kullanılan sıfırlama yayı, bağlı olduğu yer üzerinde, kendi türünden en az bir adet jeton olduğu taktirde yanmaya imkan vermektedir. Sıfırlama yayının bir ağırlığı yoktur, yanma olması halinde yer üzerinde jeton sayısından bağımsız olarak kendi türündeki bütün jetonları tüketir. Şekil 4.4'de olduğu gibi kesik çizgiyle gösterilir ve şekil üzerinde yanma olduktan sonra kırmızı renkteki bütün jetonları tüketir.

4.1.3. Çıkış yayları

Çıkış yaylarının bağlandığı yere jeton ekleme şekline göre çeşitli türlere ayrılmıştır: düşük seviye çıkış yayları, gecikmeli çıkış yayları, rastlantısal çıkış yayları.

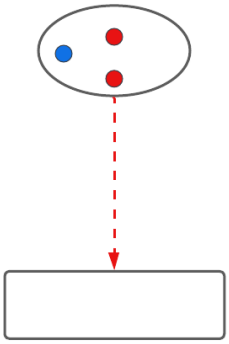


Figure 4.4 Sıfırlama yayı

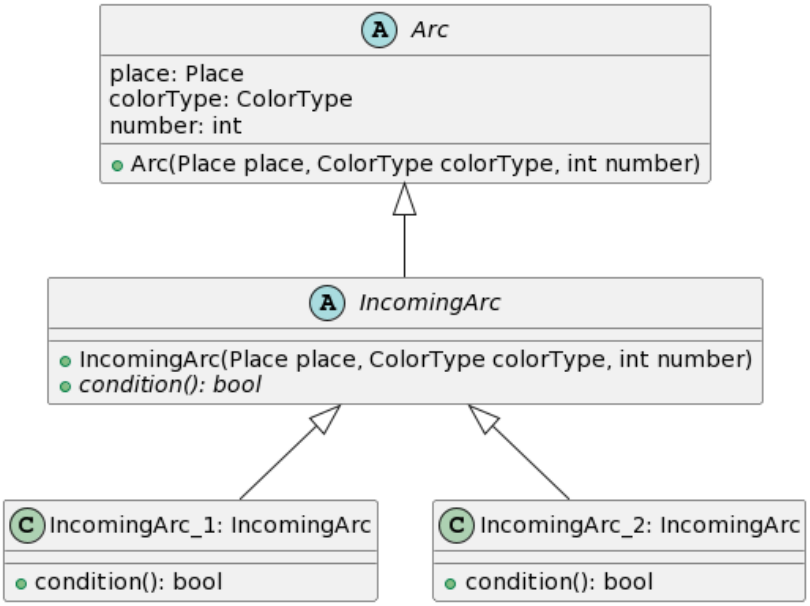
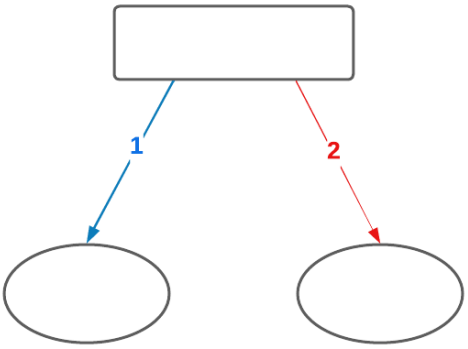
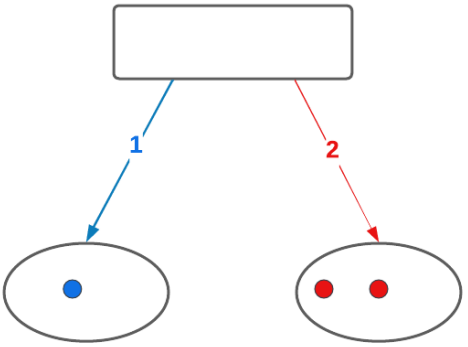


Figure 4.5 Giriş Yayı sınıf şeması

Düşük seviye çıkış yayları, yanma gerçekleştikten sonra ilgili geçişin bağlandığı çıkış yaylarının bağlı olduğu yerlere kendisi ile aynı türden ve kendisinin ağırlığı kadar jeton ekler. Şekil 4.6'de yanma gerçekleştikten sonra soldaki mavi türündeki çıkış yayı bağlantılı olduğu yere 1 adet mavi türündeki jeton eklerken, soldaki kırmızı türündeki çıkış yayı ise bağlı olduğu yere 2 adet kırmızı türünde jeton eklemektedir. Şekil 4.8'de görüldüğü gibi "Outgoing" olarak ifade edilen çıkış yayları için tür ve jeton sayıları tanımlanmaktadır ve bir çıkış yerine bağlıdır.



(a) Geçişten önceki durum



(b) Geçişten sonraki durum

Figure 4.6 Çıkış yayları

Gecikmeli çıkış yayları, bağlı olduğu yere tanımlanan gecikme süresi sonra ağırlığı kadar kendisiyle aynı renkte jeton ekler. Şekil 4.7'de gösterilen Petri Ağında yanmadan sonra 5 saniye sonra bağlanılan yere turuncu türünde 1 adet jeton eklenmektedir. Şekil 4.8'de gösterilen sınıf şemasında Gecikmeli çıkış yayları, Düşük seviye çıkış yaylarından türemektedir ve "DelayedOutgoingArc" ismiyle ifade edilmektedir. Düşük seviye çıkış yaylarına ek olarak saniye cinsinden gecikme süresi bilgisini tutar.

Gecikmeli çıkış yayları, bölüm 2.3.'de açıklanan Zamanlı Petri Ağlarının bir uygulamasıdır. Bu tezde kullanılan gecikme, geçişten yere jeton eklenirken verildiği için ilgili bölümde açıklanan Yer Zamanlı Petri Ağlarından uyarlanmıştır.

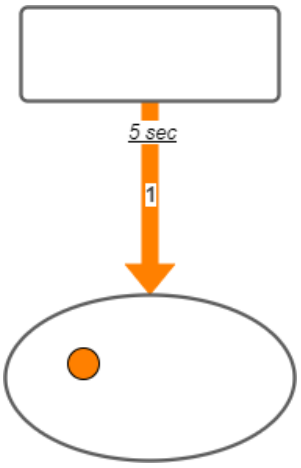


Figure 4.7 Gecikmeli çıkış yayı

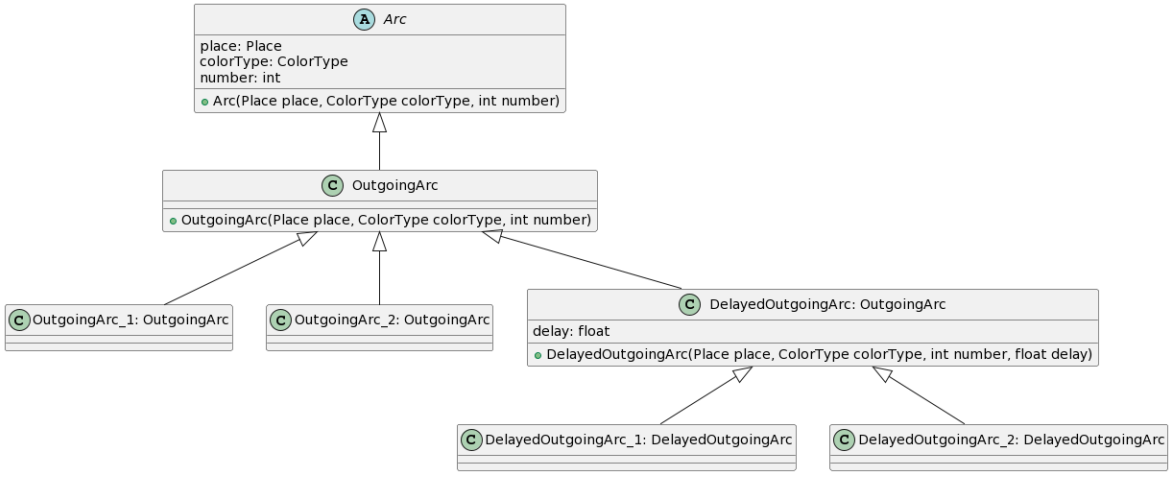


Figure 4.8 Düşük Seviye ve Gecikmeli Çıkış Yayları sınıf şeması

Rastlantısal çıkış yayları, yanma gerçekleştikten sonra bağlı olduğu yere rastgele renkte jetonun ağırlığı kadar jeton eklenmesini sağlar. Rastlantısal çıkış yayı üzerinde, rastgele eklenecek jeton renklerinin hangi oranda ekleneceği tanımlanmaktadır. Örneğin, şekil 4.9'de bulunan rastlantısal çıkış yayı, yanma gerçekleştikten sonra 1 adet ya yeşil ya da mor renkli jetondan üretmektedir. Bu renkteki jetonların üretilme oranları da *<<1>>* ile verilmiştir. *<<1>>* ile gösterilen rakam, oranın ağırlığıdır. Her iki renkten aynı oranda verildiği için yüzde 50 ihtimalle yeşil, yüzde 50 ihtimalle mor renkli jeton, bağlanılan yere eklenir. Eğer bu oran bir jeton için *<<2>>* olsaydı, bu renkteki jetondan yüzde 66 ihtimalle üretilecekti. Şekil 4.10'de rastgele çıkış yayının sınıf şeması gösterilmektedir. "RandomOutgoingArc" olarak ifade edilen rastgele çıkış yayları üzerinde bağlı olduğu yerin ve tanımlanan jeton

Figure 4.9 Rastlantısal çıkış yayı

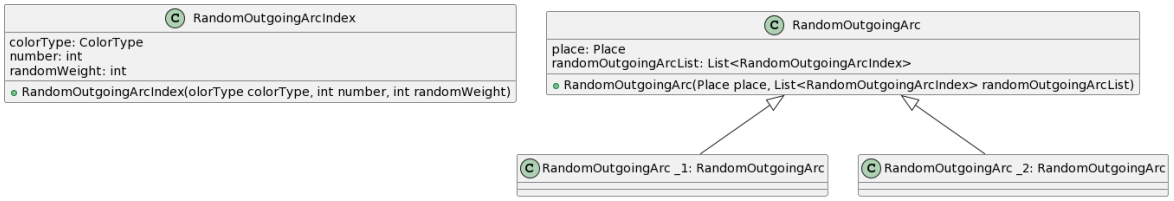


Figure 4.10 Rastlantısal Çıkış Yayı sınıf şeması

türlerinin bilgisini tutar. Jeton türleri de üzerinde renk, ağırlık ve oran ağırlığı bilgilerini içerir.

Rastlantısal çıkış yayları, bölüm 2.4.'de anlatılan Rastgele Petri Ağlarının bir uyarlamasıdır. İlgili kısımda anlatılan Petri Ağlarında rastlantısallık, geçişlere rastgele yanma gecikmelerinin verilmesiyle gerçekleşmektedir. Ancak, bu tezde önerilen rastlantısallık, gecikme olmaksızın bağlanılan yere rastgele bir şekilde jeton eklenmesiyle elde edilmektedir.

4.1.4. Geçişler

Geçiş üzerinde eylem tanımlanabilmektedir. Bu sayede, yanma gerçekleşirken tanımlanan eylem yerine getirilmektedir. Örneğin, Şekil 4.11'de yanma gerçekleşmesi halinde geçiş üzerinde tanımlı *Car.speed–* eylemi icra edilmektedir. Şekil 4.12'de gösterildiği üzere, "Transition" olarak adlandırılan geçişlerden türeyen geçiş nesneleri için "action()" fonksiyonu tanımlanmalıdır ve geçişler üzerinde giriş yayları ve çıkış yayları bilgilerini tutar. Örnek olarak, şekil 4.11'de geçiş üzerinde *Car.speed–* eylemi "action()" fonksiyonu içerisinde tanımlanmıştır.

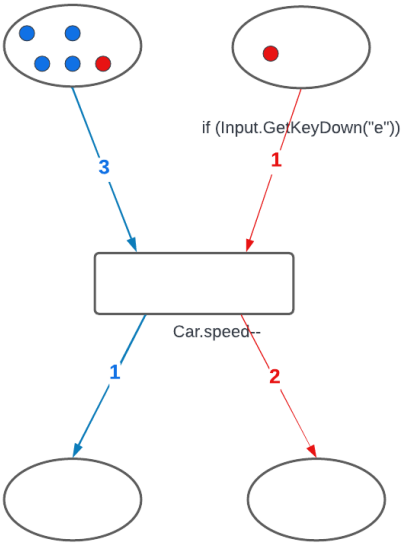


Figure 4.11 Geçiş

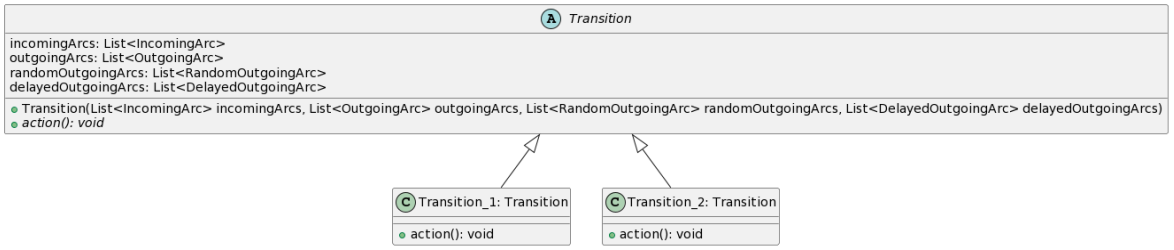


Figure 4.12 Geçiş sınıf şeması

4.1.5. Petri Ağı Algoritması

Tablo 4.1'de Petri Ağı altyapısında çalışan algoritma kısaca açıklanmıştır.

4.2. Oyun için Tasarlanan Petri Ağı

Bu kısımda, oyun için tasarlanan sistem anlatılmaktadır. Tez üzerinde anlatımı kolay olması açısından Petri Ağı üzerindeki geçişlere, yerlere, giriş yaylarına, çıkış yaylarına isim verilmiştir. Geçişler "transition", yerler, "place", giriş yayları "incomingArc", çıkış yayları "outgoingArc" ön-ekleriyle başlar. Yerlerin başlangıç durumları da şekillerde belirtilmiştir, içerisinde renkli jeton olan yerler başlangıçta bu jetonlarla başlar. Giriş ve çıkış yaylarının ağırlıkları parantez içerisinde verilmektedir. Petri sistemine ek olarak geçişlerin eylemleri

Adım 1: HER BİR Geçiş aşağıdaki koşul geçerli İSE yanma mümkündür:
Adım 1.1: Geçiş üzerindeki HER BİR GirişYayına bağlı Yerin sahip olduğu
GirişYayı ile aynı renkteki Jetonun sayısı,
Geçiş üzerindeki Giriş Yayının ağırlığına eşit ya da büyük İSE VE
Geçiş üzerindeki HER BİR GirişYayının koşulu doğru İSE
aşağıdaki adımları uygula
Adım 1.1.1: Geçişin eylemini gerçekleştir.
Adım 1.1.2: Geçiş üzerindeki HER BİR GirişYayına bağlı Yerin
GirişYayı ile aynı renkteki Jetonun sayısını GirişYayının ağırlığı kadar azalt.
Adım 1.1.3: Geçiş üzerindeki HER BİR DüşükSeviyeÇıkışYayına bağlı Yere
DüşükSeviyeÇıkışYayına ağırlığı kadar
DüşükSeviyeÇıkışYayı ile aynı renkte Jeton ekle.
Adım 1.1.4: Geçiş üzerindeki HER BİR GecikmeliÇıkışYayına bağlı Yere
GecikmeliÇıkışYayına ağırlığı kadar
GecikmeliÇıkışYayı ile aynı renkteki Jetonu, belirlenen gecikme sonra ekle.
Adım 1.1.5: Geçiş üzerindeki HER BİR RastlantısalÇıkışYayına bağlı Yere
RastlantısalÇıkışYayının rastgele seçilmiş jetonunun ağırlığı kadar
RastlantısalÇıkışYayının rastgele seçilmiş jetonunu ile aynı renkte Jeton ekle.

Table 4.1 Petri Ağı Algoritması

ve giriş yaylarının koşulları da Unity 3D ortamında olduğu gibi verilmektedir. Petri Ağı bir sayfaya sığmayacak kadar büyük olduğundan bölümlere ayrılmıştır ve bölümleri birbirine bağlayan geçişlerin ve yerlerin içerisi gri renge boyanmıştır.

Şekil 4.13'de kullanıcının kullandığı araba hariç, diğer araçların, benzin istasyonlarının ve koruyucu araçların sabit hızda ileri yöndeki hareketini sağlayan ağ görülmektedir. Tablo 4.2'de ağın eylemi verilmiştir. *transition_objects_move_continuous* geçişi her döngüde yanma gerçekleştirip arabaların ve benzin istasyonlarını hareket ettirir.

transition_objects_move_continuous:
Obstacles.GetComponent<Transform>().position += new Vector3(0, 0, 0.1f);
Fuels.GetComponent<Transform>().position += new Vector3(0, 0, 0.1f);
Protectors.GetComponent<Transform>().position += new Vector3(0, 0, 0.1f);

Table 4.2 Nesne hareketi ağı koşul ve eylemleri

Şekil 4.14'de Petri Ağı üzerinde kullanıcıdan alınan girdilerin işlendiği ve kullanıcının kontrol ettiği arabanın sabit hızla gitmesini sağlayan ağ gösterilmektedir. *incomingArc_stop_halt*, *incomingArc_halt_stop*, *incomingArc_halt_move*,

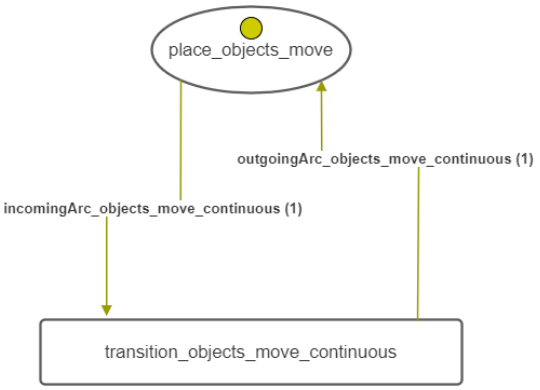


Figure 4.13 Nesne Hareketi Ağı

incomingArc_move_halt, *incomingArc_turn_left*, *incomingArc_turn_right*, *incomingArc_speed_up*, *incomingArc_speed_down* giriş yayları, klavyeden ”e”, ”q”, ”w”, ”s”, ”space”, ”a”, ”d” girdilerini alır ve bağlı oldukları geçişlerdeki eylemler kullanılarak sırasıyla motoru çalıştırma, motoru durdurma, hızlandırma, yavaşlatma, durdurma, sola kayma ve sağa kayma fonksiyonlarını yerine getirilir. Ayrıca, *transition_move_continuous* sürekli aktif olduğundan kullanıcı elini çekse de araç sabit hızla gider. Ayrıca, araba çamurlu yoldan geçtiğinde *incomingArc_mud* aktif olur ve arabanın hızını yavaşlatır. Tablo 4.3’de ağ üzerinde tanımlanan eylemler ve koşullar verilmiştir.

Şekil 4.15’de arabanın enerji seviyesi yönetilmektedir. 3 adet enerji seviyesi vardır ve *place_energy_level1* yerinde mevcut bulunan jeton ile oyuna en yüksek enerji seviyesinden başlanır. *outgoingArc_move_continuous_to_level1* ve *outgoingArc_energy_up_to_energy_level1* çıkış yaylarının ağırlığı sırasıyla 1 ve 3’tür. Bu sayede, araba sabit hızla gittiğinde her devirde 1 adet jeton, kullanıcı her ”w” tuşu ile hızlandığında 3 adet jeton, *place_energy_level* yerine eklenir. Böylece, hızlanan arabanın enerji tüketiminin sabit hızla giden arabanın enerji tüketiminden yüksek olmasının simülasyonu sağlanmış olur. Aynı şekilde, yavaş giden araba benzin istasyonuna ulaşamayacağından dolayı, oyuncunun arabayı belirli bir hızın altına düşürmemesi de bu yolla sağlanır. *place_energy_level* yerinde bağlı tüm giriş yaylarının ağırlığı 300 olduğundan dolayı, bu yerdeki jeton sayısı 300’e ulaşınca bir alt enerji seviyesine geçilir. *place_energy_level1*, *place_energy_level2* ve *place_energy_level3* yerlerinde aynı anda

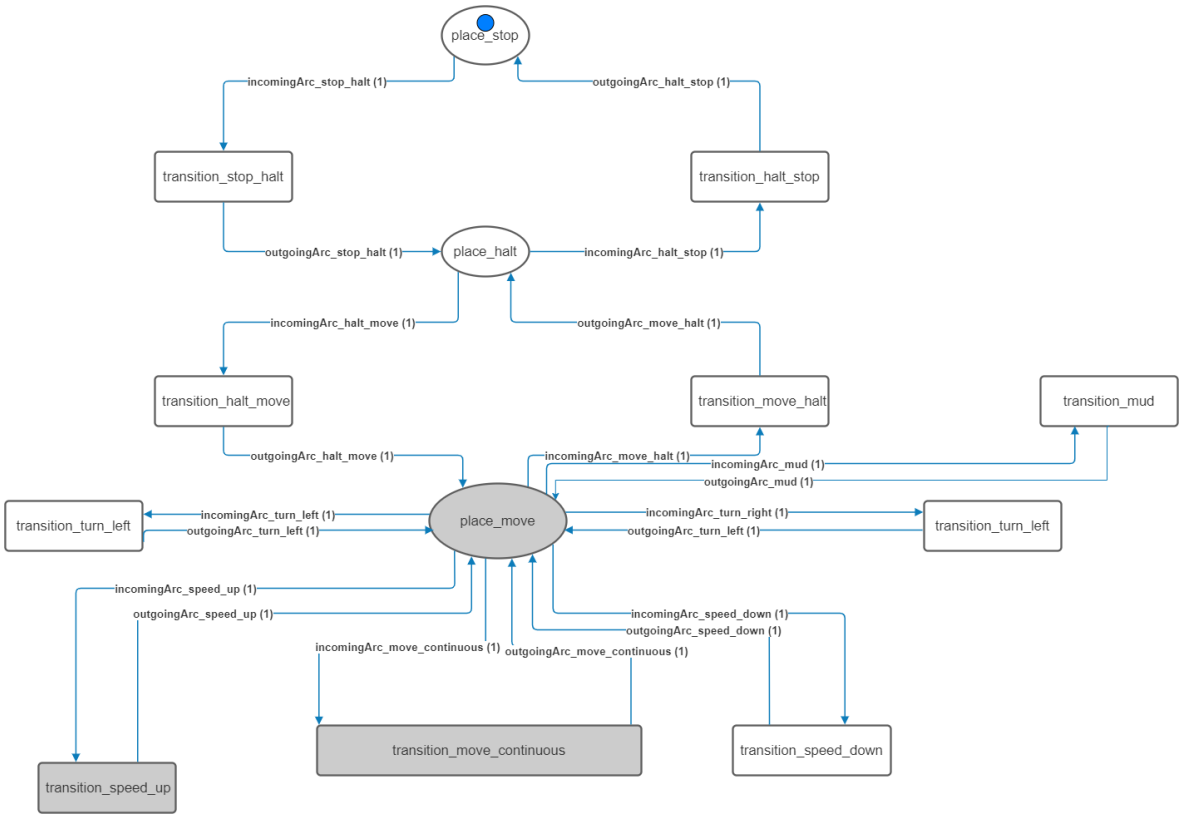


Figure 4.14 Durum Ağı

yalnızca birinde jeton bulunabilmektedir ve jetonun bulunduğu yer enerji seviyesini belirtir. Son enerji seviyesi olan *place_energy_level3* yerindeyken benzin ikmali yapılmaz ise şekil 4.16'de gösterildiği üzere *place_game_over* yerine geçileceğinden "game over" durumuna geçilir. Kısacası, son seviyeden sonra benzin tamamen biter. Benzin istasyonuna dokunulup benzin ikmali yapıldığında ise *transition_fuel_up_level1*, *transition_fuel_up_level2*, *transition_fuel_up_level3* geçişlerinin yardımıyla bir üst enerji seviyesine tekrardan geçilir. Burada dikkat edilmesi gereken bir detay, *transition_fuel_up_level1* geçişi üzerindedir. Sistem, *place_energy_level1* enerji seviyesinden iken benzin ikmali yapılırsa bundan daha üst bir enerji seviyesi bulunmadığından benzini doldurmanın tek yolu *place_energy_level* yerindeki jetonları sıfırlamaktır. Bu yüzden, *incomingArc_delete_fuel_level1* yayı, sıfırlama yayı olarak kullanılmaktadır. Tablo 4.4'de enerji yönetiminde kullanılan eylemler ve koşullar verilmektedir.

Şekil 4.16'de oyunun başarılı ya da başarısız bir şekilde sonlandırmayı sağlayan yönetim

transition_halt_move:
Car.GetComponent<CarComponent>().speed = 0;
transition_move_halt:
Car.GetComponent<CarComponent>().speed = 0;
transition_move_continuous:
Car.GetComponent<Transform>().position +=
new Vector3(0, 0, Car.GetComponent<CarComponent>().speed * 0.05f);
transition_turn_left:
Car.GetComponent<Transform>().position += new Vector3(-0.5f, 0, 0);
transition_turn_right:
Car.GetComponent<Transform>().position += new Vector3(0.5f, 0, 0);
transition_speed_up:
Car.GetComponent<CarComponent>().speed++;
transition_speed_down:
Car.GetComponent<CarComponent>().speed–;
transition_mud:
Car.GetComponent<CarComponent>().speed = 2;
incomingArc_stop_halt:
if (Input.GetKeyDown(”e”))
incomingArc_halt_stop:
if (Input.GetKeyDown(”q”))
incomingArc_halt_move:
if (Input.GetKeyDown(”w”) ||Input.GetKeyDown(”s”))
incomingArc_move_halt:
if (Input.GetKeyDown(”space”))
incomingArc_turn_left:
if (Input.GetKeyDown(”a”))
incomingArc_turn_right:
if (Input.GetKeyDown(”d”))
incomingArc_speed_up:
if (Input.GetKeyDown(”w”))
incomingArc_speed_down:
if (Input.GetKeyDown(”s”))
incomingArc_mud:
if (Car.GetComponent<CarComponent>().isMud)

Table 4.3 Durum Ağı koşul ve eylemleri

ağı görülmektedir. Benzin bittiğinde *transition_energy_level3* üzerinden, yol üzerindeki diğer araçlarla çarpışmada *transition_crash* üzerinden, yan bariyerlerle çarpışmada *transition_barrier* üzerinden ”game over” durumuna geçilir. Bitiş çizgisine ulaşıldığında oyun başarılı bir şekilde bitirilir. Kullanıcı ”esc” tuşuna bastığında *incomingArc_game_exit*

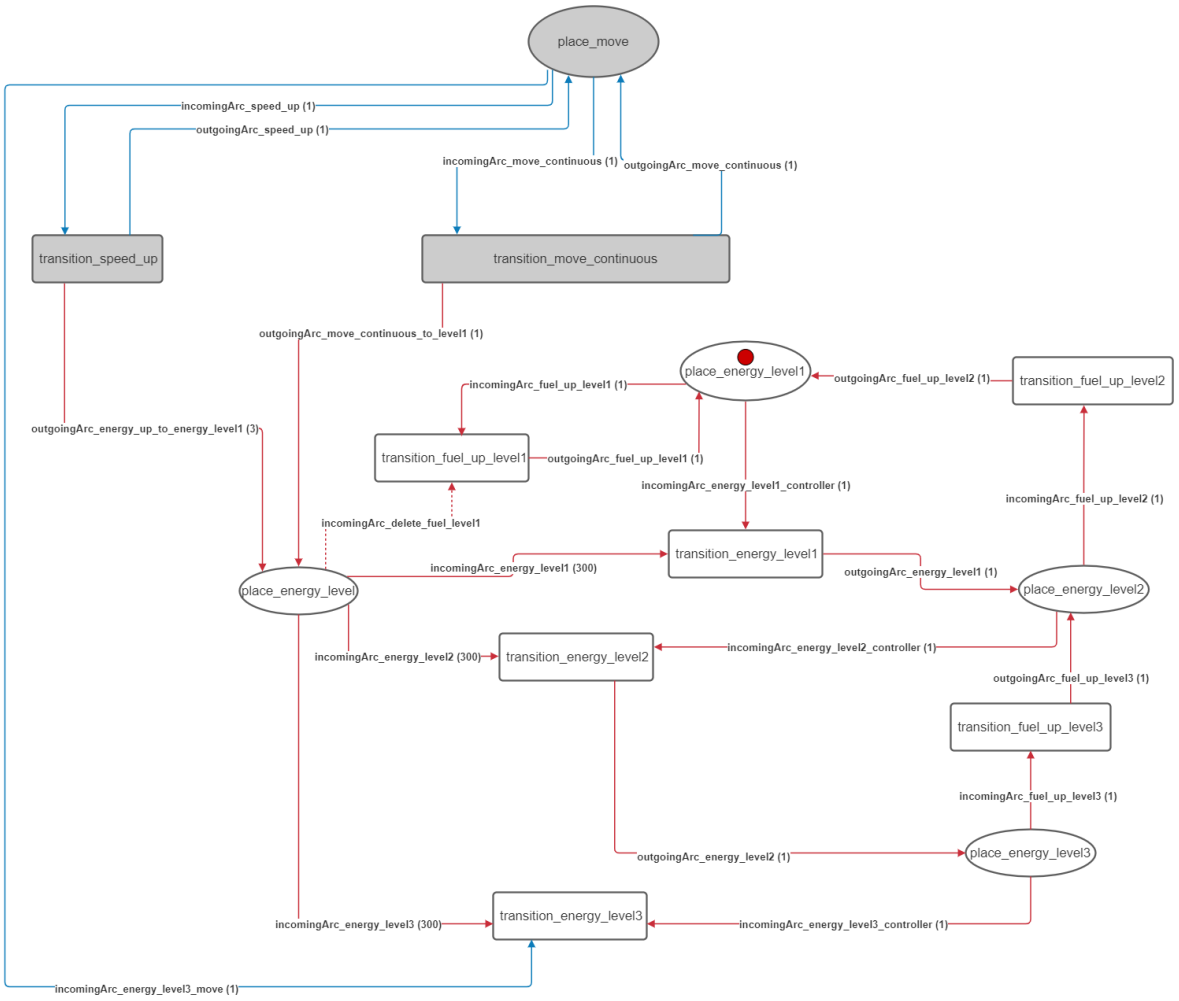


Figure 4.15 Enerji Ağı

giriş yayı ile oyundan çıkılır. Tablo 4.5'de oyun durumunu yönetmeyi sağlayan koşullar ve eylemler verilmiştir.

Şekil 4.17'de kalkan görevi gören koruyucu araba ile temas edildiğinde belirli bir süre boyunca diğer arabalara çarpmadan hareket etmek amacıyla tasarlanan Petri Ağı görülmektedir. Koruyucu aracla temas edilmesi, *incomingArc_protector* giriş yayı ile algılanır ve *place_protector* yerinde başlangıç zamanında bulunan *Protector* jetonunu tüketir. Bu sayede, "game over" durumunu sağlayan *transition_crash* geçişinin *Protector* jetonunu tüketmesi gereken bir giriş yayına sahip olmasından dolayı "game over" durumuna geçilmesi engellenir. *transition_protector* geçişinde arabanın rengi değiştirilir ve eski haline geri gelmesi için *delayedOutgoingArc_protector_delay* gecikmeli çıkış yayı aracılığıyla

transition_speed_up:
Car.GetComponent<CarComponent>().speed++;
transition_move_continuous:
Car.GetComponent<Transform>().position +=
new Vector3(0, 0, Car.GetComponent<CarComponent>().speed * 0.05f);
transition_energy_level1:
Car.GetComponent<CarComponent>().isFuel = false;
healthImage.fillAmount -= 0.33f;
transition_energy_level2:
Car.GetComponent<CarComponent>().isFuel = false;
healthImage.fillAmount -= 0.33f;
transition_energy_level3:
Car.GetComponent<CarComponent>().speed = 0;
healthImage.fillAmount -= 0.33f;
transition_fuel_up_level2:
healthImage.fillAmount += 0.33f;
transition_fuel_up_level3:
healthImage.fillAmount += 0.33f;
incomingArc_speed_up:
if (Input.GetKeyDown(”w”))
incomingArc_fuel_up_level1:
if (Car.GetComponent<CarComponent>().isFuel)
incomingArc_fuel_up_level2:
if (Car.GetComponent<CarComponent>().isFuel)
incomingArc_fuel_up_level3:
if (Car.GetComponent<CarComponent>().isFuel)

Table 4.4 Enerji Ağı koşul ve eylemleri

belirli bir süre bekler. Bu süre içerisinde *place_protector* yerinde jeton olmadığından dolayı araç, yol üzerindeki diğer arabalara çarpmadan ilerleyebilir. Süre dolduğu anda *transition_protector_delay* geçişi kullanılarak *Protector* jetonu *place_protector* yerine geri bırakılır. Böylece, öteki araçlarla çarpışma durumu yeniden sağlanır. Tablo 4.6'de koruyucu ağının tanımlandığı koşullar ve eylemler verilmektedir.

Şekil 4.18'de buzlanma etkisinin verildiği Petri Ağı görülmektedir. Buz ile temas, *incomingArc_ice_collide* giriş yayı ile tespit edilip *randomOutgoingArc_ice_collide_random* rastlantısal çıkış yayı ile yüzde 50 ihtimalle *Ice_Slip_Left*, yüzde elli ihtimalle *Ice_Slip_Right* jetonları *place_ice_collide* yerinde yaratılır. Eğer yaratılan jeton, *Ice_Slip_Left* türünden ise araç sola doğru kayar, *Ice_Slip_Right* türünde ise araç sağa doğru kayar. Bir

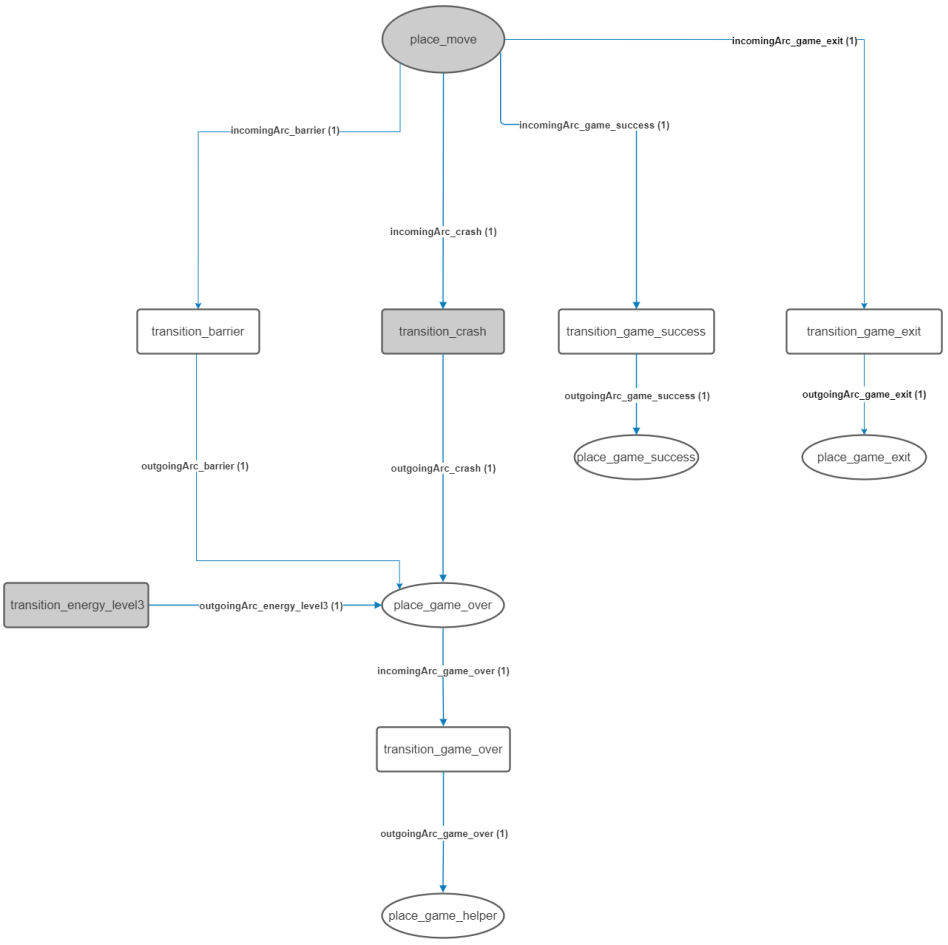


Figure 4.16 Oyun Durum Yönetim Ağı

süre kayıldıktan sonra araç geri düzeltilir. Sola kayma için bakacak olursak, *outgoingArc_ice_left_slip* çıkış yayıyla *place_ice_left* yerine 20 adet jeton eklendikten sonra her döngüde *transition_ice_left_helper* geçişi ile araç biraz sola doğru kaydırılır ve *place_ice_left* yerindeki jetonlar birer adet azaltılır. Aynı şekilde, her döngüde *place_ice_left_correction* yerine birer jeton eklenir. 20 döngü sonra *place_ice_left* yerindeki jetonlar tükendiği için sola kayma durur ve *place_ice_left_correction* yerinde 20 adet jeton biriktikten sonra *place_ice_left_correction_helper* yerine 20 adet jeton birden eklenir. Son olarak, *transition_ice_left_correct_car* geçişi aracılığıyla araç, *place_ice_left_correction_helper* yerindeki jetonları birer birer tüketerek, 20 döngü boyunca yavaş yavaş düzeltildikten sonra eski haline getirilir. Aynı yöntem sağa kayma için de geçerlidir. Buzlanma ağının koşul ve eylemleri tablo 4.7'de görülebilir.

transition_energy_level3:
Car.GetComponent<CarComponent>().speed = 0;
GameManager.GetComponent<GameManager>().UpdateHealth(false);
transition_game_over:
GameManager.GetComponent<GameManager>().GameOver();
transition_game_success:
GameManager.GetComponent<GameManager>().GameSuccess();
transition_game_exit:
Application.Quit();
incomingArc_barrier:
if (Car.GetComponent<CarComponent>().isBarrier)
incomingArc_crash:
if (Car.GetComponent<CarComponent>().isCrash)
incomingArc_game_success:
if (Car.GetComponent<CarComponent>().isFinishLine)
incomingArc_game_exit:
if (Input.GetKeyDown("escape"))

Table 4.5 Oyun Durum Yönetim Ağı koşul ve eylemleri

transition_protector_delay:
SportsCar.GetComponent<MeshRenderer>().materials[0].SetColor
("_Color", new Color(0x4D, 0xFF, 0x00, 0xFF));
Car.GetComponent<CarComponent>().isCrash = false;
transition_protector:
SportsCar.GetComponent¡MeshRenderer¿().materials[0].SetColor
("$_Color$", $newColor(0xFF, 0x00, 0x00, 0xFF)$);
incomingArc_protector_crash:
if (Car.GetComponent<CarComponent>().isCrash)
incomingArc_protector:
if (Car.GetComponent<CarComponent>().isProtector)

Table 4.6 Koruyucu Ağı koşul ve eylemleri

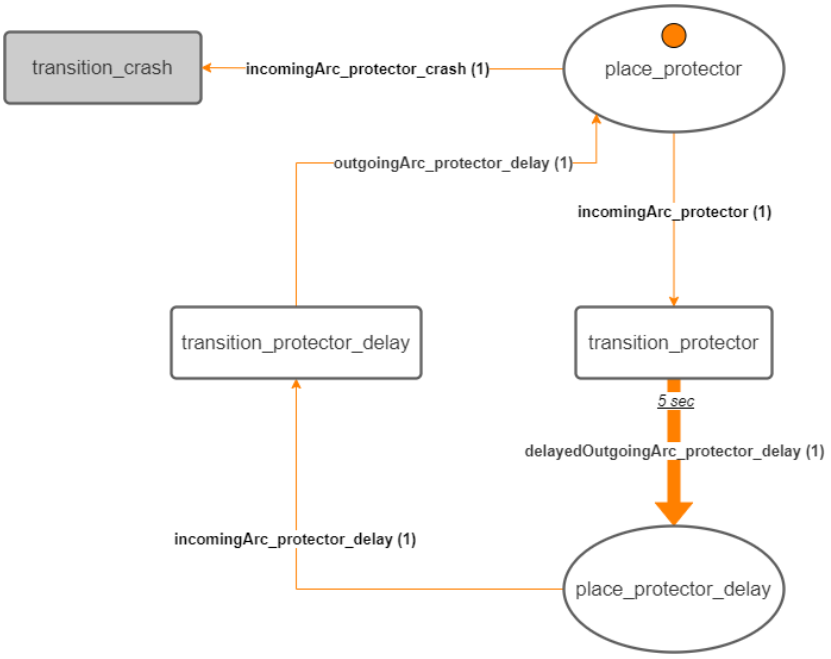


Figure 4.17 Koruyucu Ağı

transition_ice_left_helper:
Car.GetComponent<Transform>().Rotate(0.0f, -1.0f, 0.0f, Space.Self);
transition_ice_right_helper:
Car.GetComponent<Transform>().Rotate(0.0f, 1.0f, 0.0f, Space.Self);
transition_ice_left_correct_car:
Car.GetComponent<Transform>().Rotate(0.0f, 1.0f, 0.0f, Space.Self);
transition_ice_right_correct_car:
Car.GetComponent<Transform>().Rotate(0.0f, -1.0f, 0.0f, Space.Self);
incomingArc_ice_collide:
if (Car.GetComponent<CarComponent>().isIce)

Table 4.7 Buzlanma Ağı koşul ve eylemleri

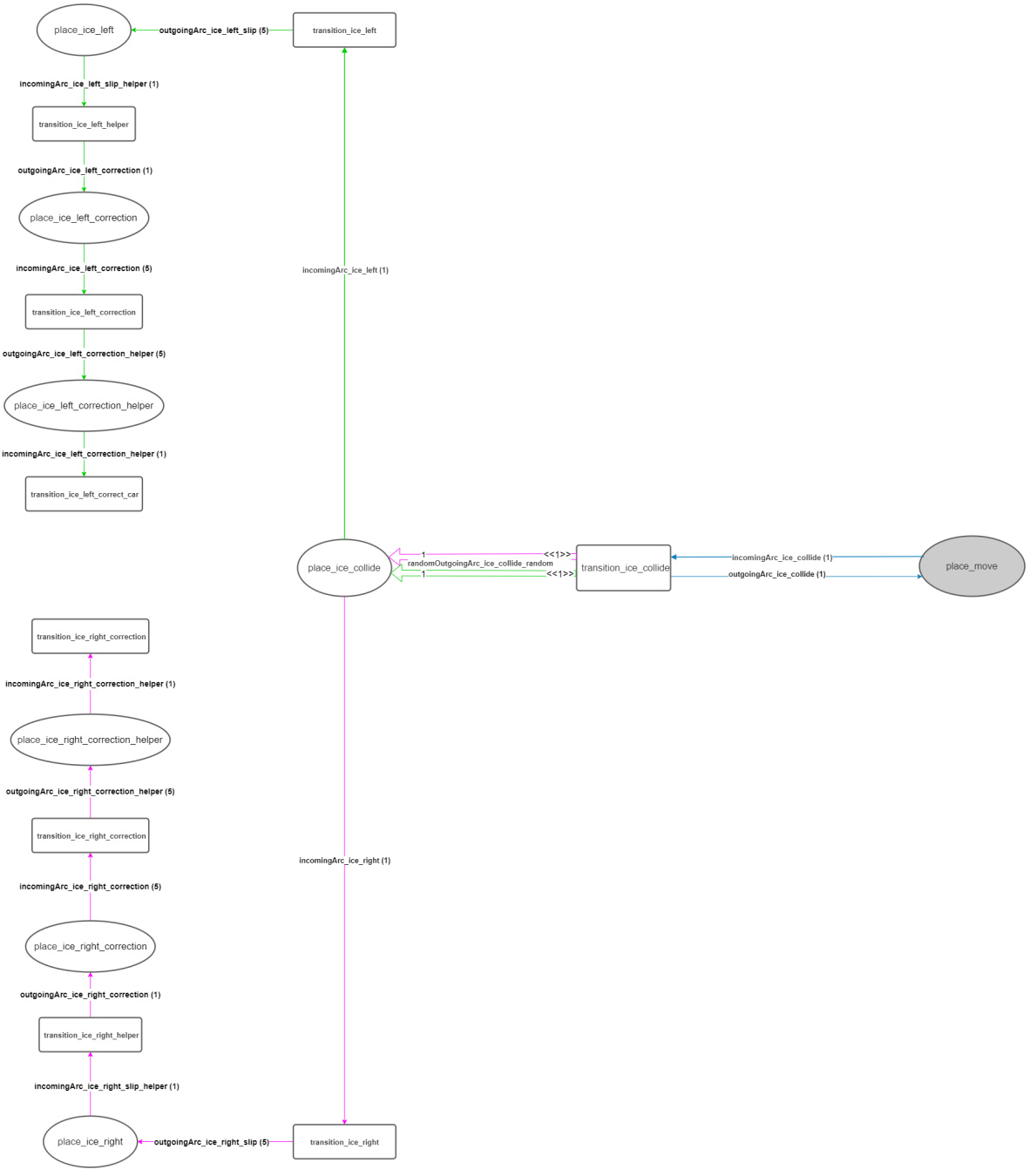


Figure 4.18 Buzlanma Ağı

4.3. Oyun içi Log Alma

Petri Ağları üzerinde log alma özelliği eklenmiştir. İstenilen geçişe veri kaydetme fonksiyonu eklenebilir. Ayrıca, oyun geliştiricisi geçişler gerçekleştiğinde hangi verilerin tutulacağını yönetebilir. Böylece oyun esnasında kullanıcının girdileri ve oyun esnasında gerçekleşen olayların verileri otomatik bir şekilde kaydedilir. Tablo 4.8'de oyun içinde tutulan bir logun örneği verilmiştir. Verilen örnekte *transition_crash* geçişinde, arabanın hız ve pozisyon bilgileri, *transition_speed_up*, *transition_speed_down* geçişlerinde arabanın pozisyon bilgisi kaydedilmiştir. Bu sayede, kullanıcının yol üzerinde nerede hızını artırıp azalttığının ve çarpışma esnasında aracın hızının ve yol üzerinde hangi noktada olduğunun bilgileri otomatik olarak kaydedilmiştir.

Bu yöntemle, oyun içi analiz ya da makine öğrenmesi gibi konularda kullanılabilmek için veri tutmayı otomatik hale getiren bir araç geliştirilmiştir. Petri Ağları kullanılarak, oyun geliştiricileri, oyun yazılımında sabit kodlama uygulamasına başvurmadan daha hızlı ve güvenli bir yolla verileri toplayabilir.

speed up,(0.0, 1.0, -20.0)
speed up,(0.0, 1.0, -19.4)
speed up,(0.0, 1.0, -18.2)
speed up,(0.0, 1.0, -16.6)
speed up,(-1.0, 1.0, -8.2)
speed up,(-1.0, 1.0, -4.9)
speed up,(-1.0, 1.0, -1.6)
speed up,(-1.0, 1.0, 3.6)
speed up,(-1.0, 1.0, 8.4)
speed up,(-1.0, 1.0, 16.5)
speed up,(-1.0, 1.0, 23.0)
speed up,(-1.0, 1.0, 26.0)
speed up,(-1.0, 1.0, 28.0)
speed up,(-1.0, 1.0, 30.6)
speed up,(0.0, 1.0, 39.3)
speed up,(0.0, 1.0, 43.2)
speed up,(0.0, 1.0, 47.4)
speed up,(0.0, 1.0, 52.6)
speed up,(0.0, 1.0, 58.9)
speed down,(-1.0, 1.0, 103.5)
speed down,(-1.0, 1.0, 109.3)
speed down,(-1.0, 1.0, 114.1)
speed down,(-1.0, 1.0, 122.5)
speed up,(0.5, 1.0, 137.2)
speed up,(0.5, 1.0, 142.8)
crash,8,(1.5, 1.0, 163.6)

Table 4.8 Oyun Logları

5. SONUÇLAR

Petri Ağları oyun geliştirme alanı içerisinde birçok yönden umut vadeden bir yöntemdir. Örneğin, oyunun tasarlama aşamasında tasarımcılar, geliştiriciler ve gözden geçirenler arasındaki iletişim, Petri Ağlarının basit yapısı sayesinde daha kolay hale getirilebilir. Bu tez kapsamında otomatik kod üretme yapılmasa da oyunda kullanılan Petri Ağları otomatik kod üretmeye uygun olarak tasarlanmıştır. İleride yapılacak çalışmalarda tasarlanan Petri Ağı üzerinde otomatik kod üretilerek daha karmaşık bir oyun geliştirme, bir yöntem olarak denenebilir. Daha önce sıra tabanlı oyunlarda kullanılan Petri Ağları farklı oyun türleri üzerinde denenip, çıkan sonuçlar üzerinde çalışılabilir. Bu tezde tasarlanan oyun, yalnızca Petri Ağları kullanılarak geliştirilmiştir. Ancak, daha karmaşık oyunlar için, oyunun tamamı için Petri Ağlarını kullanmak yerine, oyun içerisindeki durum yönetimi gibi belirli özelliklerin uygulaması ya da oyun içerisindeki karakterlerin yönetimi için kullanılabilir. Böylece, hibrit yöntemler elde edilebilir. Örneğin, belirli bir miktar puan kazandıktan sonra bölüm geçme ya da harita açma işlemi Petri Ağları ile tasarlanabilir. Puanlar Petri Ağları üzerinde tanımlanıp, geçilen bölümler ya da açılan haritalardaki oyun dinamikleri Petri Ağları kullanılmadan geliştirilebilir. Bu tez kapsamında, oyun üzerinde tanımlanan Petri Ağı üzerinde taşınan tek bilgi, renk olarak ifade edilen jetonların türüdür. Bölüm 2.2.'da anlatılan Renklendirilmiş Petri Ağı örneğiyle daha karmaşık bilgilerin sistem üzerinde transferi de ileride yapılacak çalışmalarda eklenebilir. Bunu yanı sıra, bölüm 2.5.'da anlatılan yöntemlerle oyunun daha geliştirme aşamasında gelmeden tasarlama aşamasında analizinin yapılması da mümkündür. Örneğin, belirli bir durumdan belirlenen bir duruma geçişin mümkün olup olmadığı ve yanma gerçekleştirmesi mümkün olmayan gereksiz geçişlerin var olup olmadığı gibi soruların cevapları analiz yöntemleriyle bulunabilir. Böylelikle, oyun analizi için hem biçimsel hem zaman ve maaliyet verimli bir yöntem tasarlanabilir. Ayrıca, geliştirme sonrası oyun analizi için Petri Ağları, oyun içi verileri toplama amacıyla da kullanılabilir. Bu veriler sayesinde kullanıcıların hangi yolu izlediği, hangi bölümde takıldığı gibi sorular, makine öğrenmesi gibi yöntemlerle bulunabilir. Ayrıca, Petri Ağına eklenen rastgelelik, gecikme gibi özellikler sayesinde oyuna yeni özellikler kazandırılmıştır. Bu

tarz özelliklerin kullanılıp kullanılmayacağı Petri Ağının esnek yapısı sayesinde geliştirici tarafından belirlenebilir.

Oyun geliştirilirken Petri Ağlarının avantajlarının yanı sıra bir takım sorunlar da tespit edilmiştir. İlk olarak, oyun karmaşıklaştıkça Petri Ağları da karmaşık hale gelmektedir ve Petri Ağını tasarlayanlar gibi gözden geçirenlerin de bu konuda uzmanlığı gereklidir. Örneğin, şekil 4.15'de *incomingArc_energy_level3_move* yayının amacı, benzin bittikten sonra *place_move* yerinde bulunan jetonu silmektir, çünkü aksi taktirde araç "game over" surumuna girdikten sonra da hareketine devam etmektedir. Uzman olmayan bir gözün bu durumu tespit etmesi güç olabilir. Bazı özelliklerin eklenmesi için ek bazı önlemlere ihtiyaç duyulmaktadır. Örneğin, aracın hareketinin sürekliliği için *transition_move_continuous* geçişi eklenmiştir. Yine aynı sebepten, *transition_ice_collide*, *transition_turn_left*, *transition_turn_right* gibi geçişlerden *place_move* yerine geri dönen çıkış yayları vardır. Eklenen çıkış yaylarının sebebi, bu geçişler *place_move* yerindeki jetonu tükettiklerinden dolayı hem hareketin sürekliliği hem de diğer özelliklere engel olmamaktır. Bu yüzden, bağlı oldukları yere aynı şekilde jeton bırakmaktadırlar. Aynı şekilde, ağ büyüdükçe öngörülemeyen durumlar ortaya çıkabilir. Örnek olarak, şekil 4.15'de kullanılan *incomingArc_delete_fuel_level1* sıfırlama yayı en başta düşünülmemiştir, ancak ağ oluşturuldukça sıfırlama yayının gerekliliği anlaşılıp sonradan eklenmiştir. Geliştirme aşamasında tespit edilen diğer bir sorun ise *place_move* yerine bağlı geçişlerin sağladığı özellikler sadece bu yerde *State* jetonu olması halinde çalışmaktadır. *State* jetonu *place_stop* ya da *place_halt* yerlerindeyken "game over" olmak, esc tuşuyla oyundan çıkış yapmak gibi oyun içerisinde sürekli olması gereken özellikler kullanılamamaktadır. Bu durumu engellemek adına *transition_crash* ve *transition_game_exit* geçişlerinin muadilleri *place_stop* ya da *place_halt* yerlerine eklenebilir. Bu yüzden, bu geçişleri kopyalamak gerekmektedir. Bunun gibi, her durumda geçerli olması gereken durumlar için kullanışlı değildir.

Özet olarak, Petri Ağları oyun tasarlama ve geliştirme aşamalarında kullanım için uygun bir araçtır. Ayrıca, esnek yapısı sayesinde oyunun yapısına göre rastgelelik, gecikme, bilgi taşınımı gibi özellikler eklenebilir ya da çıkartılabilir. Ancak, büyük çaptaki oyunlar için Petri Ağının karmaşıklaşması, geliştirme aşamasında ek önlemlere ihtiyaç duyulması, farklı

durumlarda geçerli olan özellikler için kod kopyalamak gerekliliği gibi nedenlerden dolayı bir takım dezavantajları da bulunmaktadır. Bu dezavantajlarına rağmen Petri Ağları, oyun sektöründe otomatik kod üretme, oyun analizi gibi çalışma alanları üzerinde kullanılabilecek bir grafiksel modelleme aracı olarak göze çarpmaktadır.

REFERANSLAR

[1] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, **1989**.

[2] Vijay Gehlot. From petri nets to colored petri nets: A tutorial introduction to nets based formalism for modeling and simulation. pages 1519–1533. **2019**. doi:10.1109/WSC40007.2019.9004691.

[3] Aymen Louati, Kamel Barkaoui, and Zohra Sbaï. Formal verification of uml2 timing diagrams based on time petri nets. volume 8. **2015**. doi:10.4018/IJISSS.2016040107.

[4] Mohamed Torky. Petri nets: Properties, analysis and applications, **2018**. doi:10.13140/RG.2.2.24938.16328.

[5] Manuel Araújo and Licinio Roque. Modeling games with petri nets. *Breaking New Ground: Innovation in Games, Play, Practice and Theory - Proceedings of DiGRA 2009*, **2009**.

[6] Martin Carlsson. Automatic code generation from a colored petri net specification for game development with unity3d, **2018**.

[7] Cyril Brom, Vít Sisler, and Tomas Holan. Story manager in 'europe 2045' uses petri nets. 4871:38–50, **2007**.

[8] Kirsten Sinclair and Daniel Livingstone. Accurate complex systems design: Integrating serious games with petri nets. *International Journal of Serious Games*, 3, **2016**. doi:10.17083/ijsg.v3i1.105.

[9] Moh Syufagi, Mochamad Hariadi, and Mauridhi Hery Purnomo. Petri net model for serious games based on motivation behavior classification. *International Journal of Computer Games Technology*, 2013, **2013**. doi:10.1155/2013/851287.

[10] C. Adam Petri and W. Reisig. Petri net. *Scholarpedia*, 3(4):6477, **2008**. doi:10.4249/scholarpedia.6477. Revision #91647.

[11] Richard Zurawski and Mengchu Zhou. Petri net and industrial application: A tutorial. *Industrial Electronics, IEEE Transactions on*, 41:567 – 583, **1995**.

[12] Ralf Hofestädt. Advantages of petri-net modeling and simulation for biological networks. *International Journal of Bioscience, Biochemistry and Bioinformatics*, 7:221–229, **2017**. doi:10.17706/ijbbb.2017.7.4.221-229.

[13] Cpn tools. `https://cpntools.org/`. Accessed: 2022-06-15.

[14] Yasper. `https://www.yasper.org/`. Accessed: 2022-06-15.

[15] Woped. `https://woped.dhbw-karlsruhe.de/`. Accessed: 2022-06-15.

[16] Kurt Jensen. A brief introduction to coloured petri nets. pages 203–208, **1997**.

[17] M. Thirumaran, P. Dhavachelvan, D. Aishwarya, and K. Rajakumari. Conventional usage of finite state machine over petri net inweb service change management framework. *IERI Procedia*, 4:99–109, **2013**. ISSN 2212-6678. doi:https://doi.org/10.1016/j.ieri.2013.11.016. 2013 International Conference on Electronic Engineering and Computer Science (EECS 2013).

[18] Jörg Desel and Wolfgang Reisig. *Place/transition Petri Nets*, pages 122–173. Springer Berlin Heidelberg, Berlin, Heidelberg, **1998**. ISBN 978-3-540-49442-3. doi:10.1007/3-540-65306-6_15.

[19] H.M.W. Verbeek, M.T. Wynn, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Reduction rules for reset/inhibitor nets. *Journal of Computer and System Sciences*, 76(2):125–143, **2010**. ISSN 0022-0000. doi:https://doi.org/10.1016/j.jcss.2009.06.003.

[20] Vijay Gehlot. A tutorial introduction to colored petri nets framework for model-driven system design and engineering. In *2021 Annual Modeling*

and Simulation Conference (ANNSIM), pages 1–12. **2021**. doi:10.23919/ANNSIM52504.2021.9552063.

[21] Søren Christensen, Jens Bæk Jørgensen, and Lars Michael Kristensen. Design/cpn — a computer tool for coloured petri nets. pages 209–223, **1997**.

[22] Falko Bause and Pieter Kritzinger. *Stochastic Petri Nets -An Introduction to the Theory*. **2013**. ISBN 3-528-15535-3.

[23] Michael Karl Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. Ph.D. thesis, **1981**. AAI8201138.

[24] S. Natkin. *Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systemes Informatiques*. Ph.D. thesis, **1980**.

[25] M. Marsan. Stochastic petri nets: An elementary introduction. volume 424, pages 1–29. **2006**. ISBN 978-3-540-52494-6. doi:10.1007/3-540-52494-0_23.

[26] A. Venkateswarlu and Abhishek Halder. A study of petri nets modeling , analysis and simulation. **2006**.

[27] Weng Jie Thong and Mohamed Ameedeen. A survey of petri net tools. *Lecture Notes in Electrical Engineering*, 315:537–551, **2015**. doi:10.1007/978-3-319-07674-4_51.

[28] Mia Consalvo and Nathan Dutton. Game analysis: Developing a methodological toolkit for the qualitative study of games. *Game Studies*, 6, **2006**.

[29] Paul Bertens, Anna Guitart, and África Periáñez. Games and big data: A scalable multi-dimensional churn prediction model. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 33–36. **2017**. doi:10.1109/CIG.2017.8080412.

[30] Sehar Shahzad Farooq, Jong-Woong Baek, and KyungJoong Kim. Interpreting behaviors of mobile game players from in-game data and context logs. In

2015 IEEE Conference on Computational Intelligence and Games (CIG), pages 548–549. **2015**. doi:10.1109/CIG.2015.7317895.

[31] Joey R. Fanfarelli. Assessing computational thinking pedagogy in serious games through questionnaires, think-aloud testing, and automated data logging. In *2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall)*, pages 149–152. **2021**. doi:10.1109/ICISFall51598.2021.9627365.

[32] Ji Hun Kim and Richard Wu. Leveraging machine learning for game development, **2021**.

[33] John Bergsten and Konrad Öhman. Player analysis in computer games using artificial neural networks, **2017**.

[34] Victoria J. Hodge, Sam Devlin, Nick Sephton, Florian Block, Peter I. Cowling, and Anders Drachen. Win prediction in multiplayer esports: Live professional match prediction. *IEEE Transactions on Games*, 13(4):368–379, **2021**. doi:10.1109/TG.2019.2948469.

[35] Qiyuan Shen. A machine learning approach to predict the result of league of legends. In *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*, pages 38–45. **2022**. doi:10.1109/MLKE55170.2022.00013.

[36] Aleksandr Semenov, Peter Romov, Sergey Korolev, Daniil Yashkov, and Kirill Neklyudov. Performance of machine learning algorithms in predicting game outcome from drafts in dota 2. volume 661. **2016**. ISBN 978-3-319-52919-6. doi:10.1007/978-3-319-52920-2_3.

[37] Road fighter. `https://en.wikipedia.org/wiki/Road_Fighter`. Accessed: 2022-07-06.